N-Modular Redundancy Techniques for Fault Tolerance in Reconfigurable Logic Devices

Juan C. Carcheri
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816-2450
jcarcheri@knights.ucf.edu

***Abstract*—The popularity, effectiveness and proven success of exploiting modular redundancy in fault mitigation techniques is well known. N-Modular redundancy (NMR) is a common fault mitigation approach for Field Programmable Gate Arrays (FPGAs) that has been successfully implemented and demonstrated by several works, particularly triple-modular redundancy (TMR). In this paper, we visit a number of previous N-Modular redundancy based fault tolerance techniques. All methods and schemes are compared and some of the more promising approaches are highlighted.**

Keywords: Fault-tolerance, FPGA, Modular redundancy, performance, reliability.

I. INTRODUCTION

The concept of using redundancy as a fault mitigation scheme for FPGAs was introduced many years ago. Modular redundancy is now widely used primarily for detecting faults and even complementing evolutionary repair mechanisms, as we will see later. More importantly, modular redundancy is a common technique used to alleviate against failures caused by single event upsets (SEU).

Field Programmable Gate Arrays (FPGA) are sensitive to radiation-induced single event upsets. This is especially true for space-based applications that are more likely to be exposed to high-energy charged particles (radiation). Single event upsets may cause sporadic flips of configuration bits causing connectivity and logic errors. This is of utmost importance since the functionality of an SRAM-based FPGA relies on data that is stored in millions of configuration latches. Given that, an SEU can have undesirable effects if not handled properly.

Modular redundancy is one of the more common techniques for mitigating against failures caused by SEUs, particularly Triple Modular Redundancy (TMR) [13]. Modular redundancy is a technique where a functional block is replicated N times and the outputs are then compared. For example, given the case of two modules, a difference in their outputs indicates that one of them is faulty and further analysis can be done to figure out which is the faulty module. In the case of a three-module setup (i.e., TMR), given a single fault, a discrepancy in the outputs will cause one module to differ from the other two. This passive technique essentially enables the system to mask the fault and provide the correct response by continuing to use the majority output. Hence, the threshold value of the system is the output of the majority voter. In its most simplest setup using voting, the application will work correctly as long as more than (N+1)/2 modules are without faults.

As noted by the three-module example system, it has been shown that modular redundancy provides very small detection latency and is one the main reasons for its wide deployment in applications. A fault is uncovered as soon as the error reaches the voting/validation logic. One should also note that using redundancy as a detection scheme has little effect on the timing performance of the system. This is due to performance overhead being only dependent on the latency of the voting logic.

The primary drawback of NMR-based systems is the resource overhead. For example, in the case of TMR, there would be at least three times the area overhead needed to replicate functionality. In addition to the resource overhead cost, the power increase due to a modular redundancy scheme must be considered. In fact, results from the analysis done by [10] demonstrate that power consumption rises in the range of 3x to 7x when TMR is applied to design. Work in the field has been done to alleviate some of these overheads, as we will see in the following sections.

In this paper, we will visit some previously proposed N-Modular Redundancy based fault tolerance (FT) techniques. In all the covered approaches, modular redundancy plays an essential role within the entirety of the fault tolerant system in one way or another. Some proposed approaches have decided to implement redundancy in a duplex mode (where N=2), while others have preferred to use the popular triple module mode implementation. Additionally, certain fault tolerance schemes have employed more than three redundant modules for increased reliability. However, fault tolerance techniques are not restricted to deployment of static redundancy, where N is a predetermined number that does not change. FT methods proposed in [1 and 11] have taken a more dynamic route by reorganizing the redundancy configuration of a system as determined by the needs of a mission at a particular moment in time.

This paper is organized as follows. Section II visits and reviews various triple module redundancy based fault tolerance

techniques. Section III examines dual modular redundancy based techniques. In Section IV, we take a closer look at techniques where the number of redundant modules is not restricted to one value. A comparative analysis is done in section V and a conclusion given in Section VI.

II. Triple Modular Redundancy Techniques

Triple Modular Redundancy (TMR) was first introduced by Von Neuman as a common technique to provide design hardening [7]. The proposed approach is similar to Figure 1, where the desired circuit is replicated three times and voting performed on the single output. This is considered to be a passive redundancy scheme where faults are masked as they occur without the need to isolate the fault area.

Given a single fault, if one module fails, the other two functioning modules can take precedence over the incorrect output and maintain a correct overall output of the system. It is obvious to see that in this basic scheme, TMR can tolerate up to one faulty module. An overall faulty functional output may be produced if a fault occurs in a second module. Therefore, TMR is limited in its fault tolerance capacity as a passive method. However, there are methods that have been proposed to increase system reliability, as we will see in the upcoming subsections.

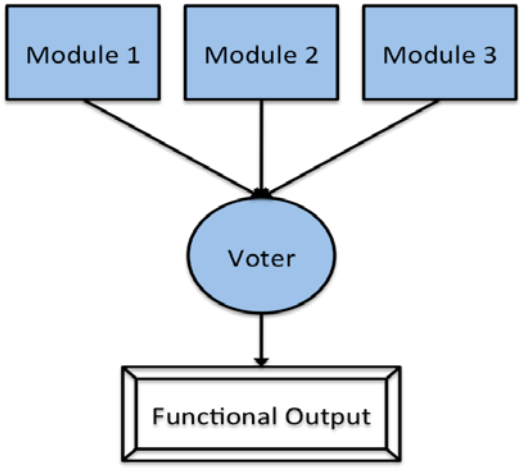

Figure 1 - TMR with One Voter

TMR can be implemented on a design in different basic ways. In addition to the conventional TMR scheme where a single voter is used, another commonly used architecture contains triplicate voters as well. The 3-voter architecture is similar to Figure 2. Using just one voter inconveniently provides a single point of failure in the system. The use of three voters improves the reliability of the system by removing the single point of failure. However, this technique does consume more logic resources and is slightly slower. Improved reliability is traded off with slower operating speed and additional resources.

As mentioned earlier, TMR is quick to detect and recover from faults through redundancy. Nonetheless, this speed does come at a cost to both resource overhead and power consumption. At least three times the area overhead and power consumption is needed to replicate functionality [10]. The overhead for continuous redundancy is still at hand even when the system is running without any faults. None of these costs have denied TMR from being deployed in mission critical applications such as space-based systems. TMR is arguably the most widely used technique to mitigate against design failures caused by SEUs.

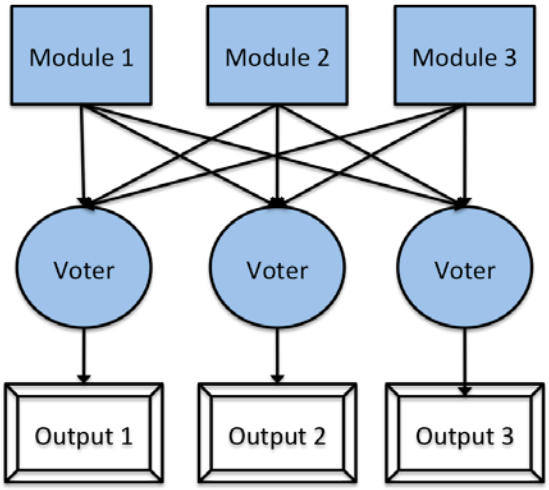

Figure 2 - TMR with Triplicate Voter

A. TMR with Standby (TMRSB)

Since a fault occurring in a second module of the conventional TMR system (similar to Figure 1) can produce an erroneous functional output, it is only able to handle one fault. In order to improve the reliability, [8] proposed a fault tolerance method where TMR is combined with the other popular fault tolerant technique of Standby (SB). The proposed method is conveniently referred to as Triple Modular Redundancy with Standby (TMRSB).

The fault tolerant method of Standby is where a component of interest has one or more identical backup components offline. A switch mechanism is then used to select one of the backup components and deploy it as one of the active components. This is done when the original active component fails. The benefit of the standby technique is that the system is allowed to operate properly with a momentary execution decay that is only affected by the overhead of the switching mechanism.

As displayed in Figure 3, each module of the TMR scheme contains standby configurations that are available at run-time. When a fault occurs within one of the currently active modules, a fault-free standby configuration is then chosen and put into service in its place. It should be noted that the standby configurations are created at design-time. This process continues until all standby configurations for the faulty module have been used up and no others are readily available.

The advantage of this technique with regards to the conventional TMR technique is that a fault is allowed to occur in a second module without affecting the functional output. The functional output remains correct while the repair (i.e. replacement of the faulty module with a standby configuration) is in progress. As one can see, the ability of TMR to remain

online with two modules is exploited by this method as well as exploiting the benefits of standby.

TMRSB is however still at the mercy of vulnerability to faults occurring within the standby configurations. Wherever the standby configurations are stored in memory, they can still be exposed to faults caused by radiation just like the active modules can. Hence, a faulty standby configuration can cause an unexpected functional output even with the presence of a flawless switching mechanism. The faulty standby configuration will also not be detected until it is placed online as one of the active modules. Consequently, the switching mechanism will continue loading backup configurations until a fault-free one is found and loaded. The system likelihood is determined by the number of successful fault-free standby configurations. In order to reduce complexity, the analysis done in [8] made the assumption that there is always at least one fault-free standby configuration within the standby pool.

One must also consider the correct number of *m* standby configurations to deploy as displayed in Figure 3. The results from the analysis demonstrate that a higher value of *m* does not linearly increase the reliability of the system. The values of *m* used in the study were 1, 2, 4, 10 and 15. The best results occur when *m*=2 or *m*=4. In fact, having very high values of *m* causes the switching mechanism to slow, causing a larger performance overhead. Better results are demonstrated when *m*=4 as compared to *m*=2 since more standby configurations can be utilized as the mission gets longer in time.

Overall, the TMRSB technique proposed can improve system reliability. The mathematical models and simulation used in the aforementioned paper demonstrate that TMRSB provides higher reliability for a system over TMR and SB alone.

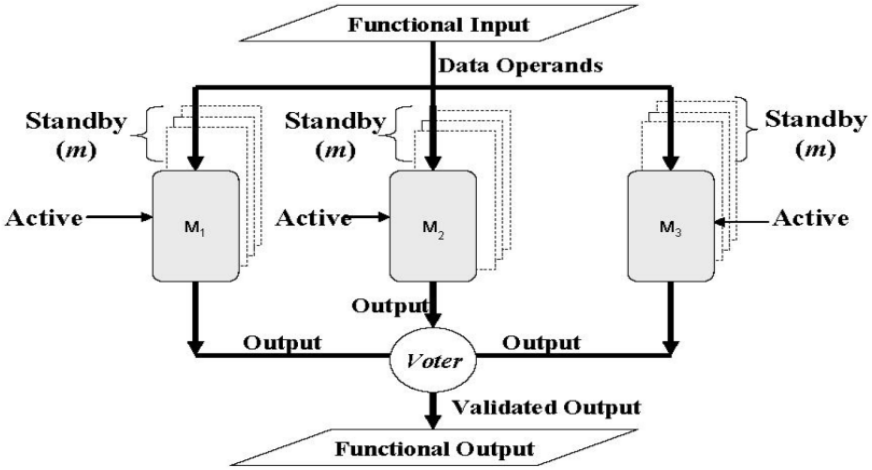

Figure 3 - TMR with Standby [8]

B. Jiggling

Garvie and Thompson [9] proposed a method to tolerate transient and permanent faults in Field Programmable Gate Arrays. The proposed method exploits and combines the benefits of TMR and scrubbing, in addition to introducing the concept of Jiggling. TMR is used in this method to keep the system online in the presence of a fault while repair work is undermined. Scrubbing is used to do away with single event upsets in the FPGA configuration data by reprogramming. The proposed method uses 'lazy scrubbing' as opposed to traditional scrubbing, in which the whole TMR configuration on an FPGA is regularly reprogrammed. Instead, lazy scrubbing just reconfigures a module when its respective output is different from the other two active modules. When scrubbing the faulty module, the majority vote of the three configurations is used as the original. This results in consuming less power and less single point of failures. After the first Jiggling mend, lazy scrubbing cannot be used due to the alterations of the original configurations by the genetic algorithm (GA) used to tolerate a permanent fault.

Jiggling is a technique where the resources in the other two fault-free active modules are used to repair the faulty module affected by permanent faults. Permanent faults, or local permanent damage (LPD), are considered to be within a module if the module fails to recover after lazy scrubbing. Once a permanent fault is located, the Jiggling process begins. This process uses an (1+1) evolutionary strategy introduced by [5] which is a minimal GA. A (1+1) ES has one parent or individual and one offspring or mutant. Whenever the genetically modified mutant presents a better fitness than the parent, that parent individual is replaced. Otherwise, the ES will keep producing mutants until one is produced that has a higher evaluated fitness. Mutations are single bit flips in the configuration stream of the faulty module. Any part of the Configuration Logic Block (CLB) within the FPGA, such as the routing or Look-up Table (LUT), can be affected by the mutation.

A history window of past mutations is preserved in order to overcome the possibility that a bad mutant gets a lucky high fitness due to noise issues. This enables rollback to the greater individual whenever that case would occur. Normal mission

operation is taken advantage of when evaluating the fitness of an individual. The test vectors employed to evaluate the fitness of the newly created mutants are provided by the normal FPGA operational inputs. A score (either 0 if incorrect or 1 if correct) is collected for every output under every input vector. The output is considered correct if it is the same as the output of the voter, otherwise it is considered incorrect. The fitness evaluation process is ended when all possible input combinations have been encountered. The resulting fitness is then simply the sum of the scores.

Similar to the TMRSB method discussed above, the Jiggling technique was analyzed and implemented with a simulation model. The reported repair times were short, since they usually took under two minutes on average. Approximately 90% of successful repairs were complete within the 11 minutes of simulation timeout used in the experiments. This is considerably promising as permanent damage is only likely to occur once every 6 months. The argument is then brought up that the approach would succeed in nearly every opportunity given the large time limit of the real world.

In summary, the proposed approach is a respectable fault mitigation technique. The use of lazy scrubbing deployed in this approach allows the system to use less power than when using traditional scrubbing to mitigate transient faults. It also does not depend on fault-free memory to hold configuration data. The jiggling portion of this approach also does not rely on any fault-free memory holding design-time configurations for repairing modules exposed to permanent faults.

III. Dual Modular Redundancy Techniques

While some approaches are mainly concerned with providing an adequate scheme to SEU fault correction, the methods deploying a dual modular redundancy (DMR) scheme put a stronger emphasis on attaining a quicker and lower resource-consuming fault detection scheme. DMR has been shown to be an effective way of detecting errors in FPGA-based systems [6]. As the name implies, DMR techniques operate by duplicating resources and comparing the results with some sort of comparator as shown in Figure 4. DMR is able to consume less power and area within the FPGA than TMR by only utilizing two redundant components.

Full mitigation of SEUs at the redundancy level may not be necessary for some systems. Instead, some approaches are more interested in using some other system-level mechanism to mitigate the fault such as repairing the configuration bitstream or using some sort of evolutionary approach. The main goal of DMR is to achieve similar single event transient fault mitigation as TMR with fewer redundant modules. In fact, DMR logic requires up to 33% less area than TMR. Consequently, similar to conventional TMR characteristics, DMR is not able to tolerate multiple faults since the approach is primarily affective when only one module is faulty. However, as seen through examples in the following subsections, various fault tolerance schemes have benefited from the concept of dual modular redundancy.

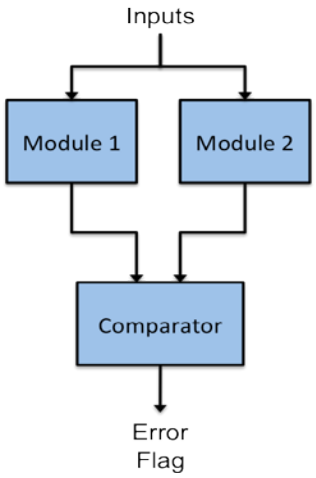

Figure 4 - Dual Modular Redundancy with Comparator

A. Hybrid Time and Component Redundancy Technique

The approach proposed in [3] can be described as a hybrid approach to tolerate transient faults such as SEUs for SRAM-based FPGAs. The approach combines Dual Modular Redundancy with Comparison (DWC) and Concurrent Error Detection (CED), referred to as the DWC-CED technique. The CED portion, which is based on time redundancy, allows the system to detect permanent faults. Time and hardware redundancy is combined to cope with permanent faults caused by SEUs, while reducing the area and number of pins overhead in the combinational logic caused by other hardware redundancy techniques such as TMR. In addition to detecting and coping with permanent and transient faults in the combinational logic of the FPGA, this approach is also able to detect physical faults. Physical faults can be described as permanent faults that cannot be corrected by reconfiguration.

The Dual Modular Redundancy with Comparison (DWC) technique alone is only able to detect transient faults, but not those in which become permanent faults. As such, a CED block is inserted in each redundant module of this duplex approach as depicted in Figure 5. This allows the system to determine which module is faulty. An important characteristic of the DWC-CED approach is that under a single fault assumption, it is able to detect which module is faulty and therefore selects the correct output amongst the two modules.

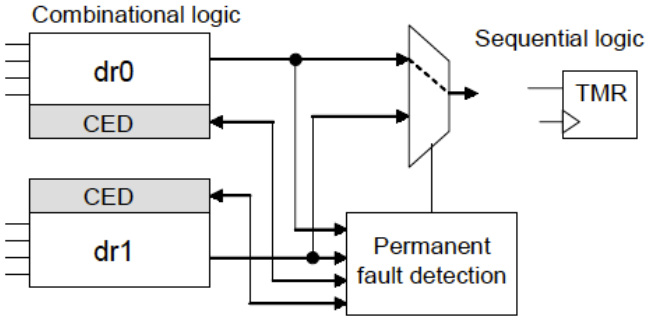

Figure 5 - DWC Combined with CED [Lima-2005]

In order to avoid increasing the area overhead, CED was implemented based on time redundancy instead of hardware redundancy. The concept of time redundancy is where computations are repeated in ways that allows detection of errors by taking advantage of the pulse characteristics of transient faults. The repeated computations are performed differently and the signals are later compared to allow redundancy to detect permanent faults. This is referred to as an encode and decode process. During the first computation, operands are applied to the CLB directly and the result is stored in a register. During the second computation, the operands are altered and then used on the configuration logic. The result of the second computation is compared with the result of the first one. A discrepancy between them indicates that there is a permanent fault within the module.

It is important to note that two clock cycles are needed to detect the permanent fault within the combinational logic of the module. However, the output of the DMR scheme can be exploited to determine whether the extra computation is needed for permanent fault detection. This means that the extra time does not come about every clock cycle. The recomputation is only necessary when a discrepancy occurs between the outputs of the two modules. The output register will hold its original value for one extra clock cycle when a discrepancy occurs and allow CED to detect the permanent fault. This scheme allows the system to not take a hit on performance during a fault-free or single fault operation.

Faults in the combinational logic are mitigated by scrubbing and faults in the sequential logic are corrected by the TMR scheme used in the CLB flip-flops. To help insure that only one fault occurs between reconfigurations, the scrubbing is performed continuously and with a fast enough rate. Detection and voting circuit faults do not obstruct correct execution of the system since it is already triplicated. As long as only one redundant module is faulty, any voter output corrupted by a fault is not of importance. This approach can also deal with faults in the routing as long as there is no fault on the routing that can connect signals from both modules. This is why it is necessary to use a dedicated region for each module.

A fault injection experiment with a Virtex prototype board using emulation was employed to validate the proposed DWC-CED technique. The fault coverage was measured by exhaustively employing all possible combinations of vectors and faults inside the multipliers. The results of the proposed approach demonstrate that 99.97% of the faults can be detected and corrected within one clock cycle delay. Experimental results for area overhead were also demonstrated with comparison to results of a no fault tolerance and TMR technique. The DWC-CED, TMR and a no fault tolerant techniques were implemented on 8x8 and 16x16 multipliers. The reported results clearly demonstrate that the DWC-CED technique reduces both the area and number of input and output pins of the user's combinational logic. While the proposed approach reduces the area overhead from three to two (in comparison with TMR), it still possesses the area overhead of an encoder and decoder circuit. Most importantly, the technique can only tolerate a single fault and may not achieve 100% fault coverage for all designs.

B. Competitive Runtime Reconfiguration

The concept of DMR supports the goal for increased fault tolerance efficiency in different types of techniques. The proposed approach in [12] uses DMR to grade the fitness of competing configurations in an evolutionary approach. The approach developed provides autonomous self-repair for SRAM-based FPGAs. Faults are handled through a comparison of pair-wise functional outputs. The two individuals are physically unique configurations but contain identical functionality. This means that they have the same input-output behavior but different implementations or designs.

At design time, a population of 'Pristine' individuals (as referred to by [12]) are produced and used as the initial population. CRR divides the FPGA into two mutually exclusive configurations, resulting in a left half and a right half configuration as shown in Figure 6. CRR then assigns the left half configuration to one individual and the right half to another individual in the population. Throughout the process, two alternate configurations, loaded from the left and right populations respectively, are compared at any point in time. As time goes by, alternate pairings are considered. It is clear from this comparison process that CRR uses a temporal voting approach for fault detection. Therefore, similar to TMR and other redundancy-based approaches, the detection latency in this approach is negligible as the fault is detected once the first discrepant output occurs amongst the two competing configurations.

Once a discrepant output occurs, the fitness value of each competing configuration is demoted and placed in the 'Suspect' pool. At this point, no verdict is reached on which individual is actually the faulty one. Alternatively, in the case of matching outputs, the fitness value for each competing individual is increased. The faulty individual is later realized over time, as more pairings are made with other individuals in a diverse set of configurations. An advantage to note here is that this process occurs as part of the normal processing throughput of the FPGA, thus exploiting the normal operational FPGA input parameters. Hence, no additional test vectors are need.

The evolutionary part of this approach is kicked off based on the fitness value of the individuals in the population. If an individual's fitness becomes less than the repair threshold, the individual is marked as 'Under Repair' and genetic operators are applied once with a random pristine individual. The genetic operators used in this approach include two-point crossover, mutation, and cell swapping. The two-point crossover genetic operator is used to replace functional units with others from pristine configurations. The mutation operator reconfigures suspect CLBs with alternatives and cell swapping repositions CLBs within a configuration. The individual remains marked as 'Under Repair' until its fitness increases through application of genetic operators and repeated pairings to a value that is greater than or equal to the operational threshold. At this point, the individual is then marked as 'Refurbished' and will remain as such unless its corresponding fitness decreases below the operational threshold, where it is again marked as 'Under Repair'.

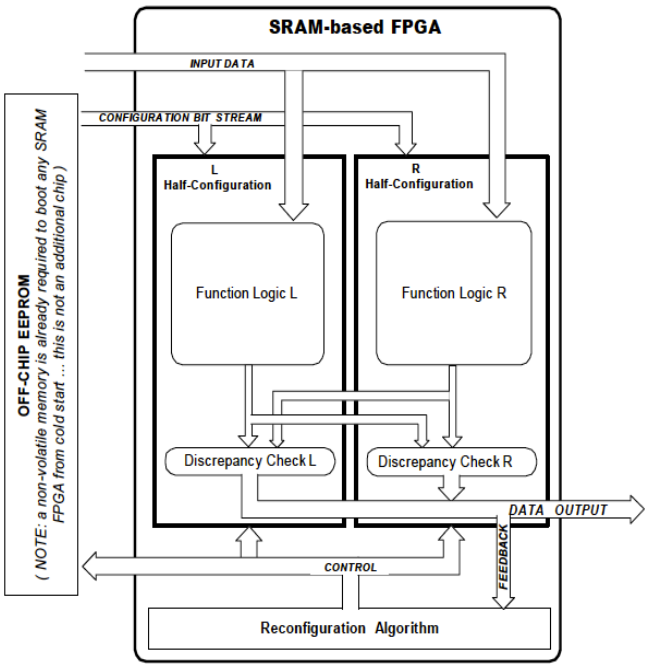

Figure 6 - CRR Arrangement in SRAM-based FPGA [12]

A simulated FPGA consisting of a 3x3 multiplier circuit was used for implementation and experimentation. Results from the experiments show that a complete repair was realized within a few thousand evaluations for about half of the runs while remaining partially online. In addition, the recovery process allowed an average data throughput of 87.4%, only requiring an average of 13.6% of the computations to be redundant.

As a suitable option for a fault tolerance scheme in SRAM-based FPGAs, CRR has again demonstrated the benefits of applying a redundancy concept in part of the whole FT system. In this case, the DMR approach was used for fault detection and in grading the fitness of the competing configurations. It is also worth noting that SEUs will be automatically scrubbed by CRR's alternate pairing process. As the authors of [12] mention, the paring process will result in the configuration fitness to decrease initially in the presence of transient faults, but will later rise. Thus, resulting in transients being gradually diminished over time.

IV. Dynamic Redundancy Based Techniques

So far, the techniques presented have been based on a predetermined number of redundant modules, which cannot change. The methods were intended to either only run with two or three redundant components. This section will present proposed fault-tolerant methods that are based upon or extended from a N-Modular Redundancy (NMR) technique, where the number of modules can vary. In addition, we will take a closer look as to whether or not deploying higher redundancy values actually benefit the overall fault tolerant system, and if so, in what way. While it is obvious that higher redundancy values can actually negatively affect the system by incurring a higher area and performance overhead, the following subsection will demonstrate that the benefits can sometimes outweigh the incurred overhead costs.

A. Enhancement of Evolutionary Repair through N-Plex Voting

It has been known for some time that using Genetic Algorithms (GA) for fault recovery has been an effective approach for mitigation against faults [14]. GAs are used to locate fitting alternative configurations for dealing with a fault. Functionally equivalent, but physically different configurations are used to define the initial population for the GA. The GA looks around the search space and eventually identifies the solution as the individual with the highest fitness. Each individual configuration is given a fitness value based on the number of correct outputs from a set of inputs. In addition, genetic operators such as mutation and crossover are applied to alternative configurations during the evolution process. This process is continued until a perfect configuration is found or the maximum number of generations determined a-priori has been exhausted.

Although the use of GAs has demonstrated success for autonomous fault tolerance in FPGAs, the GA is not always able to form and establish a perfect individual. This is sometimes true even after a large number of generations. Given that, a study is done in [1] to enhance the evolutionary repair mechanism by extending the n-plex voting scheme.

In the proposed approach, three or more GAs are used rather than one. The overall functional output is then considered to be the majority output of the GAs. In other words, the majority output is determined by the resulting comparison of the best-fit individual from each GA. An arrangement of both three-plex and five-plex voting was used in the experiments. An example of the five-plex voting scheme is similar to Figure 7.

When using three-plex voting, the process involves running three separate GA runs in a generation. After each generation, the outputs of the three GA runs are voted on. The study clearly demonstrates that the extension of a voting arrangement to a GA attains better overall performance as opposed to using a single GA for the evolutionary repair process. In fact, it was shown that the triple voting scheme aided the GA in reaching a complete repair at an average of 113.86 generations. This is significantly lower than the average number of generations needed for best repair with a single GA. The reported numbers of generations needed by the standalone GA in this study were 934, 852 and 274. Complete repair with the triple voting scheme was achieved in 70% of the experimental runs (7 out of 10).

When using five-plex voting, a majority vote is determined by a "three-out-of-five" ruling instead as observed in Figure 7. Although the five-plex voting arrangement increases the resource overhead, experiments is this study have demonstrated that five-plex voting achieves better performance than even the three-plex voting scheme. In fact, the average number of generations needed to reach a complete repair was 48.33. This is significantly lower than the average 113.86 generations needed by the three-plex voting method as mentioned above. In

addition, complete repair was achieved 90% of the experiment runs (9 out of 10).

Similar to any n-plex voting technique, real-time fault handling is managed. However, similar to any static NMR based scheme, this comes at the cost of n-fold increase in power consumption during fault-free mission operation. A benefit demonstrated by the results is that improvement is shown in the combined repair performance from several incomplete repairs acquired by the GAs. Complete repair is achieved with far fewer generations than may be required with a standalone GA. It can be argued that the faster results and improved probability of complete repair validates the extra space and power needed to implement the voting scheme.

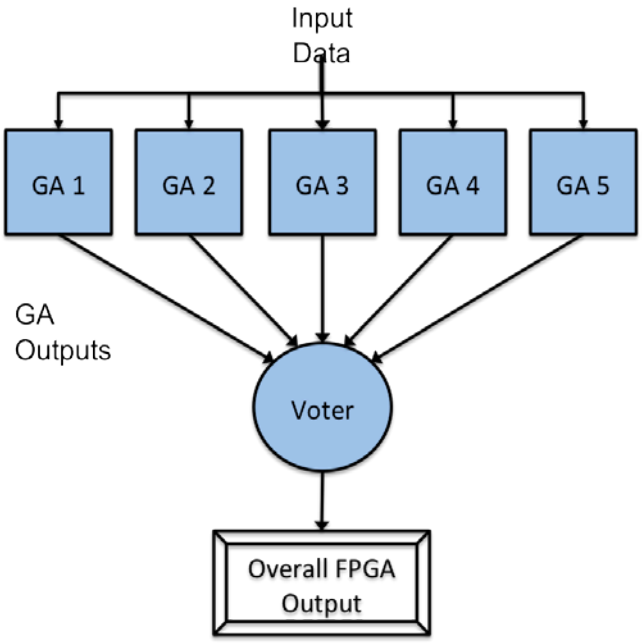

Figure 7 - Five-Plex Voting Arrangement

B. Reconfigurable Adaptive Redundancy System (RARS)

The methods analyzed thus far are considered one-layered fault tolerant systems, where the fault tolerance capabilities have been implemented in only one layer of the system, in this case being the hardware level. The approach presented in [11] demonstrates a self-sustainable adaptive fault tolerant system implemented as a two-layered system, consisting of a hardware layer and a software layer. The hardware layer provides self-repair by adjusting the configuration of a system through an approach called Reconfigurable Adaptive Redundancy System (RARS), which will be discussed more thoroughly later in this section. The software layer provides a level of supervision for FPGA activities, bestowing the capabilities of performance monitoring and active repair of the system through evolutionary repair techniques. Repair techniques included are scrubbing and evolutionary repair through the use of an Organic Genetic Algorithm (OGA).

As hinted by its use of an organic GA, the proposed FT system leverages on the concept of Organic Computing (OC), whereby providing a system that is able to adapt and promote self-reliance. Organic computing techniques are aimed at carrying out the goals of providing autonomous systems capable of adaptive and fault-tolerant behaviors. The OC model emphasizes biologically inspired life-like self-x properties of a system, such as self-healing and self-reorganization. The goal of systems demonstrating OC techniques is to autonomously provide those self-x properties, while concurrently alleviating undesirable behaviors, such as single event upsets. Due to its beneficial capability of dynamic reconfiguration, SRAM-based FPGAs have become popular hardware platforms for the implementation of OC systems [2 and 4].

The beneficial fault mitigation capabilities of modular redundancy are once again exploited by a fault tolerant system, this case by the general purpose approach RARS. RARS is implemented at the hardware layer of the OC system. However, unlike other redundant schemes such as TMR, RARS does not rely on a predetermined number of modules. Instead, the components within RARS can be reorganized at run-time, depending on the mission requirements at a specific time. RARS is an adaptive redundancy design aimed at power conservation by only reconfiguring to a more power-consummating configuration like TMR, when needed. This may occur when redundant instances of an application are required to identify or mask a fault as exemplified by TMR.

One of the biggest problems with conventional TMR is that the system runs in triplex mode even at the absence of any fault, consuming three times the power even when not needed during fault free operation. This is very costly as the system's mission time is mostly spent fault-free. RARS provides a more efficient way of employing the redundant scheme with a duplex-TMR-duplex dynamic reconfiguration technique. It begins with a duplex architecture to detect faults, reconfiguring to TMR to isolate the single fault, and back to duplex to allow two data paths while offline repair of the faulty module is underway. This provides stellar results in minimizing power consumption, as discussed further when we take a closer look at the experiment outcomes provided by the authors.

The RARS architecture is consisted of one controller function named the Autonomic Element (AE) and *n*-number of identical user defined redundant implementations containing the application named Functional Elements (FE). It is important to note that the AE is application-independent while the implementations of the FEs are application-dependent. Hence, FEs are the only components that must be altered in the system when deployed for new applications. The AE monitors the status of the redundant components (FEs) and contains the logic that allows the RARS system to adaptively reorganize the available FEs. Although the implementation in [11] was organized to a max of three redundant parts, There is nothing that prevents generalizing RARS and implementing 2N+1 FEs, where N>0. However, one must remain conscious of technical performance requirements dictated by their mission objectives, as more redundant components will increase space and power.

The approach used was to initially enable two of the available three FEs in order to instantly detect a fault occurrence, while the third module is kept offline as a cold spare. The status of the two active modules is monitored by the

AE. In the presence of a fault, a discrepancy occurs and the AE automatically reorganizes the system in a TMR mode. The third standby module is placed online and a voting system is activated. Given that a single FE is faulty, this allows the fault to be isolated and masked while the correct output is used due to the outcome of majority voting.

Figure 8 depicts the RARS architecture employed in more detail. The Autonomic Element (AE), which processes the outputs of the available FEs, is composed of five modules that allow the adaptive behavior of RARS. The Discrepancy Sensor (DS) component is used to detect discrepancies between any two-of-three enabled FEs and is only activated when running in duplex mode. The DS is disabled otherwise to save power. The Voter component performs bitwise voting of the three FEs and outputs the majority vote. This component is only active when running in TMR mode and disabled when not in order to save power. It can also report to the Redundancy Controller (RC) that all three FEs are faulty in the case of multiple faults or which particular module is faulty in the case of a single fault. The Output Actuator is what primarily allows the flexibility of the AE compared to other static redundancy techniques. The Output Actuator is a 4x1 multiplexer with the inputs provided by the outputs of the three FEs and the Voter, allowing the RC to choose between any simplex configuration and the majority vote output. The Performance Monitor (PM) component samples the DS or Voter, depending whether system in duplex or TMR mode respectively, to provide performance reports to the RC. The RC is the core component with the responsibility of interfacing with the software layer by providing performance status and receiving control commands. It is also responsible for controlling the different possible configurations in RARS through its output signals.

When RARS is in a simplex configuration, only one FE is enabled and the Output Selector propagates the output of the active FE. This configuration can be used when fault tolerance is not of critical importance to the mission, allowing the system to conserve the most power possible. In a duplex configuration, two FEs are enabled as well as the DS in order provide discrepancy reports to the RC. The system can run in duplex mode during the repair of a faulty module after discovery in TMR mode to allow temporary degradation in output. When in a TMR configuration, the Voter and all three FEs are enabled, while the DS is disabled. The Voter output is propagated by the output selector, which allows fully correct output under the single-fault scenario. The configurations mentioned above are all options for the hybrid mode feature supported by RARS as exemplified by the employed duplex-TMR-duplex approach. The target application can be reorganized as needed to meet mission reliability requirements.

Since RARS's fault tolerance capability is limited by the number of redundant modules to support alternative configurations for bypassing faults, the software layer provides a higher level of restoration. The first purpose of the software layer is to collect the hardware status reports from RARS and provide a human-readable performance-monitoring module through the use of a GUI. The second purpose is to provide active repair either through scrubbing to correct SEUs or through the use of an organic GA for dealing with permanent faults. The two layers are connected through a parallel cable that connects the Joint Test Action Group (JTAG) port on the FPGA to the parallel port of a PC. A C++ application is ran to send and receive messages using a special communication protocol specifically developed for this application. The software monitors RARS performance and performs an appropriate refurbishment procedure when the redundancy implementation of RARS is not enough to mask the present faults (e.g., multiple faults in a TMR mode).

The software layer exploits dynamic partial reconfiguration (PR) to improve the evolutionary repair techniques. Through the use of PR, configuration time is reduced as compared to full configuration since the partial bitstream size is smaller. The smaller bitstream sizes allows for faster reprogramming of the FPGA. PR also allows the system to remain online while under repair. This means that overall system availability is increased since the system is able to maintain functionality even during repair. Dynamic PR is used in both the scrubbing and GA repair stages of the repair cycle. The scrubbing stage is where a-priori solutions are repetitively configured on the FPGA. The faulty FE is repetitively reconfigured until the faulty resource is excluded from the logic path. Scrubbing is used as the fist attempt of recovery for faults that cannot be handled by RARS. Dynamic PR is used in the GA stage when individual configurations are reconfigured to perform fitness evaluations. Experimental results show that dynamic partial reconfiguration reduced the repair time to 27.48% of the full bitstream configuration approach's repair time. In 20 runs of the experiments, there was an average of 148 generations required to completely repair the fault, while 95% of the time, a repair happens in less than 173 generations.

Additional promising results were shown in the experimental work. To demonstrate the practicality and evaluate the capability of this approach, a prototype was built on real Commercial-Off-The-Shelf (COTS) hardware. A Xilinx Virtex-4 FPGA was used to build the prototype on and an edge detection application was used for evaluation. RARS demonstrated significant advantages in power consumption over TMR. Results show that while still providing protection against transient and permanent faults, RARS is able to save up to 30% of the power used by TMR. Reduction in power consumption went from 9.91 mWatts for TMR to 7.02 mWatts in the proposed approach.

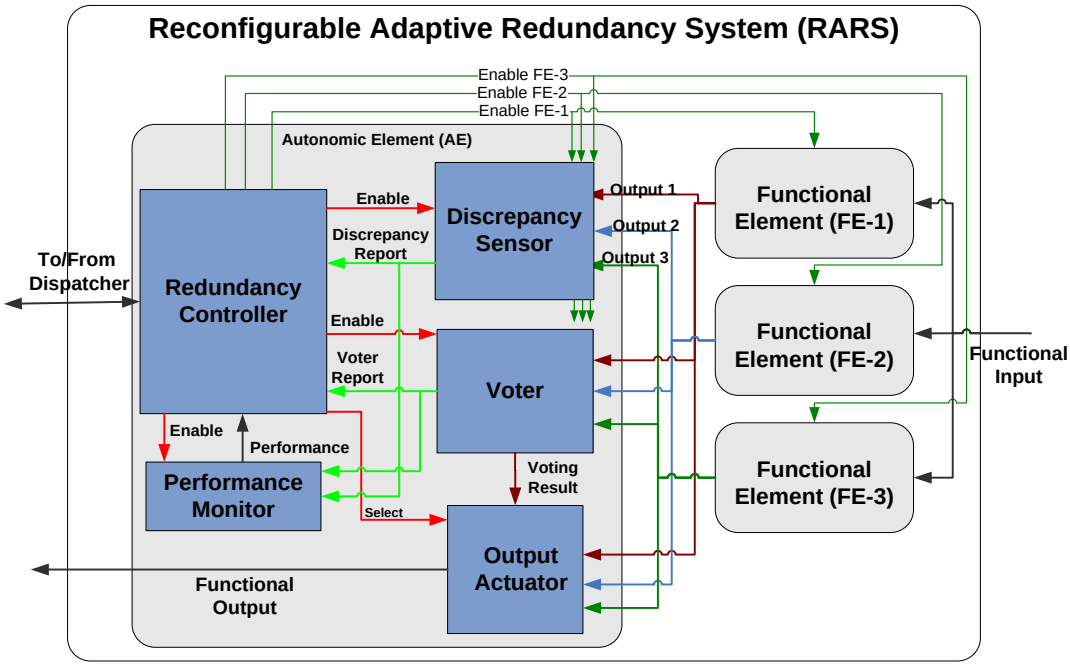

Figure 8 - RARS Architecture [11]

V. Comparative Analysis

Methods discussed in this study have certain advantages and disadvantages with regards to their fault tolerance capabilities. Their positive effects on the fault tolerant systems can significantly outweigh some negative characteristics present in the surveyed approaches. There are some characteristics that are shared amongst all surveyed methods. For instance, since all methods are based off of a redundancy approach, there is the penalty of incurring an increased physical resource overhead due to the implementation of replicated functionality. This section will highlight certain advantages and disadvantages of the surveyed methods and compare the approaches amongst one another.

While all methods are able to deal with both transient and permanent faults, some methods are limited in the amount of faults that can be handled correctly. TMR and DMR based techniques are subject to a degraded functional output in the presence of multiple faults. For example, the DWC-CED is only able tolerate a single fault properly. In the TMRSB approach, the system can tolerate a single fault and also capable of tolerating additional faults only as long as three fully functioning configurations exist in the pool. Meanwhile, techniques based on evolutionary repair, such as CRR and RARS, do not depend on the number of existing design-time alternate configurations. This is due to the fact that faulty modules are repaired through an evolutionary repair mechanism.

Most of the surveyed methods incorporate fault detection and isolation techniques in their fault handling system. Others, like the dynamic voting method discussed in [1], rely on an external fault detection mechanism. We can compare detection latency for those techniques that do provide fault detection schemes. Detection latency refers to the amount of time required for the fault-tolerant technique to detect and/or locate a fault. The TMRSB, Jiggling, CRR and RARS techniques have insignificant detection latency since faults are detected as soon as a discrepancy amongst the output of the active modules occurs. The DMR portion of the DWC-CED technique is able to detect transient faults in one clock cycle, but permanent faults in the combinational logic are detected in two clock cycles. Hence, the DWC-CED technique needs one clock cycle of latency to detect a permanent fault. This extra time does not come about every clock cycle though, since it is only necessary to perform the extra computation for detecting the permanent fault when the outputs of the two redundant modules do not agree.

Power consumption is another metric that should be considered when comparing the surveyed methods. Power consumption is one of the costs that come with quick detection in redundancy schemes. TMR-based techniques, such as the TMRSB and Jiggling technique discussed, automatically consume three times the power than techniques that do not utilize redundancy. Unfortunately, this is also true when the system is running fault-free. N-fold increase in power consumption during fault-free operation also applies to the dynamic voting scheme used to enhance evolutionary repair in [1], as described in Section IV. On other hand, DMR-based schemes, such as the DWC-CED and CRR techniques, have a lower resource-consuming fault detection scheme. Less power and area is consumed within the FPGA with these techniques than the above-mentioned TMR-based techniques by only using two redundant modules. Another reason for their high power consumption is that these techniques have also relied on static redundancy where the number of modules does not change during the mission lifetime. This is contrary to the dynamic behavior proposed by the RARS technique. RARS

uses adaptive redundancy to conserve power by only reorganizing to a more power-consummating configuration like TMR when necessary. The approach implemented only reorganizes itself to a TMR mode when needing to isolate a single fault due to a discrepancy in the outputs of the duplex configuration. Once the faulty module is fixed, it reorganizes itself to a duplex scheme. RARS has been able to demonstrate a 30% saving of power used in TMR-based methods.

In addition to power consumption, the surveyed methods also incur increased physical resource overhead. This is a worthwhile metric to compare against since not all user applications can tolerate the additional resources required to implement the fault-tolerance scheme. Physical resource overhead refers to the amount of resources that a user application must reserve, in addition to those required by the original application, in order to implement the fault-handling method. Techniques that employ TMR schemes incur a physical resource overhead of 200% of the size of the original application. At least three times the area is needed to replicate functionality. In contrast, with the DWR-CED technique discussed in Section III, permanent faults caused by SEUs are mitigated with less area and number of input and output pins in the combinational logic. Although the area overhead is reduced from three to two in this technique, it still contains the area overheard of an encoder and decoder circuit.

There are additional advantages and disadvantages to note that should be carefully considered depending on the application. If a user application is more heavily dependent on faster recovery times due to the criticality of the mission, the n-plex voting scheme for GAs discussed in Section IV should be considered for the repair mechanism. The evolutionary repair with dynamic voting scheme has been shown to improve efficiency, completeness and speed as compared to using a single GA. Also, if the application allows the increased area overhead, the use of the five-plex voting version should be considered since complete repair is reached in about 40% less than the average number of generations demonstrated in the three-plex version. This technique has demonstrated faster repair times than any other surveyed approach.

VI. Conclusion

Fault tolerance in Field Programmable Gate Arrays has been extensively studied with consideration to detection and repair performance. Various methods have been proposed that take advantage of the FPGAs ability to reconfigure autonomously. For many years now, redundancy has been one of the more widely used fault mitigation techniques and will continue to be so. This paper has sought to present some important publications in this field that propose fault-tolerant methods employing N-Modular Redundancy and compare the techniques that they present. This can be either to provide quick detection of faults, passive fault tolerance through masking, or to even extend evolutionary repair mechanisms. All surveyed methods provide runtime fault handling for transient and permanent faults in SRAM-based FPGAs.

The surveyed methods are appropriate to various applications, but not all. In addition, they each have differing strengths and weaknesses. It is up to careful analysis performed at design time to decide which method is best suited for a particular application's needs. Although some very promising methods have been presented here, there is no one-size-fits-all approach. There is still much room for future work in this field that can make more efficient use of FPGA resources. Certainly, the motivation and work in this field will continue to grow as long as the risk of corruption to FPGAs does too.

References

[1] C. J. Milliord, C. A. Sharma, and R. F. DeMara, "Dynamic Voting Schemes to Enhance Evolutionary Repair in Reconfigurable Logic Devices," Proceedings of the 2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig'05) on Reconfigurable Computing and FPGAs, 2005.

[2] C. Müller-Schloer, "Organic computing: on the feasibility of controlled emergence," Proc. 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, ACM, 2004, pp. 2-5.

[3] F. Lima, L. Carro, R. Reis: "Designing Fault Tolerant Systems into SRAM-based FPGAs", ACM/IEEE Design Automation Conference, 2003, pp. 650-655

[4] H. Schmeck, "Organic computing-a new vision for distributed embedded systems," Proc. Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05), 2005, pp. 201-203.

[5] H.P. Schwefel and G. Rudolph. Contemporary evolution strategies. In F. Mor´an, 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.

[6] J. Johnson, W. Howes, M. Wirthlin, D.L. McMurtrey, M. Caffrey, P. Graham, and K. Morgan, "Using Duplication with Compare for On-line Error Detection in FPGA-based Designs," Aerospace Conference, 2008 IEEE, pp. 1-11, 1-8 March 2008

[7] J. Von Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliable components," in Automata Studies, C. E. Shannon and J. McCarthy, Eds. Princeton, NJ: Princeton Univ. Press, pp.43-98, 1956.

[8] K. Zhang, G. Bedette, and R. F. DeMara, "Triple Modular Redundancy with Standby (TMRSB) Supporting Dynamic Resource Reconfiguration," In Proceedings of IEEE Systems Readiness Technology Conference AUTOTESTCON 2006, Anaheim, CA, September 2006, IEEE Computer Society, Washington, DC, USA, 690-696.

[9] M. Garvie and A. Thompson. "Scrubbing away transients and Jiggling around the permanent: Long survival of FPGA systems through evolutionary self-repair," Proceedings of 10th IEEE International On-Line Testing Symposium, pp. 155-160. IEEE Computer Society, 2004.

[10] N. Rollins, M. Wirthlin, and P. Graham, "Evaluation of power costs in applying TMR to FPGA designs," in Proceedings of the 7th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD), September 2004. 117

[11] R. Al-Haddad, R. Oreifej, R. Ashraf, and R. F. DeMara, "A Sustainable Organic Architecture Emphasizing Partial Reconfiguration for Reduced Power Consumption," unpublished.

[12] R. F. DeMara and K. Zhang, "Autonomous FPGA Fault Handling through Competitive Runtime Reconfiguration," in proc. NASA/DoD Conference on 29-01 June, 2005.

[13] R.E. Lyons, and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*April 1962, pp. 200-209.

[14] S. Vigander, "Evolutionary fault repair of electronics in space applications," Doctorate Dissertation, University of Sussex, Galmer, Brighton, UK, 2001.