Performance Analysis of Current Fault Management Techniques in FPGAs

Christopher E Giles
School of Electrical Engineering & Computer Science
University of Central Florida, Orlando, FL 32816, USA
Email: stardica@gmail.com

***Abstract*— Researchers are studying the reconfiguration capabilities of current Static Random Access Memory (SRAM) based Field Programmable Gate Arrays (FPGAs). Researchers are working on improving the sustainability, survivability, and availability of these FPGAs. This concept is called Evolvable Hardware. The goal of researchers is to develop and employ efficient techniques which provide an autonomous capability for self healing after the FPGA has experienced a permanent physical fault. The techniques in development are targeted towards systems where human interaction after a permanent system fault is typically not possible i.e. satellites and deep space probes. This paper focuses on the techniques in development for fault detection, isolation, diagnosis, and repair within the FPGA. Also after identification and description of the techniques currently used by researchers a theoretical application of these techniques is presented. In the theoretical application a description of how to employ these techniques together, as a system, to produce an efficient means of fault detection and isolation is demonstrated.**

***Index Terms*— Reconfiguration, Field Programmable Gate Array, Fault Detection, Fault Isolation, Recovery.**

I. Introduction

SRAM based FPGAs are used in a multitude of applications. FPGAs can be found in a range of technologies from word processors to vending machines to even being incorporated into mission critical components of space satellites. SRAM based FPGAs also have the unique capability in which they can be reconfigured at any time to perform any task within the FPGAs capability [14]. This leads to interesting and new territories for research into use of the FPGA's reconfiguration capability to establish self healing capabilities upon fault manifestation. A fault manifestation in these terms is considered to be the breakdown of functionality of the FPGA's logical resources. For example, FPGAs used in space satellites are deployed to harsh environments where they are regularly exposed to high energy bombardments. These deployed systems are out of reach of human hands and after a fault has manifested the system could be rendered inoperable with no means of repair. There are also other forms of faults which can occur from long use and manifest as a breakdown of the FPGAs internal logical components. But, for this paper we will consider faults from the point of initial manifestation and not delve into the mechanisms which trigger a fault's manifestation.

As identified in [1] and [2] researchers have discovered the capability and have been working on fault handling techniques to increase mission survivability, availability, and sustainability of these systems given the inevitable degradation of system capability over time due to permanent fault manifestation. These fault handling techniques support the phases of detection, isolation, diagnosis, and repair of the FPGA while still in service. In this case repair means that the system can autonomously reconfigure itself to attenuate faulty resources from its application data path or interconnect path. This is a novel idea which has been gaining momentum due to the increased use of FPGAs in today's computational systems. The repair capability in FPGAs has been proven to work in laboratories both extrinsically and intrinsically and to-date Field Programmable Devices (FPDs) are the only technology identified which can support this autonomous repair capability.

Other FPDs which show promise for use of self repair techniques are the Field Programmable Transistor Array (FPTA) [9] and Field Programmable Analog Array (FPAA) [3]. These technologies show similar capabilities of autonomous repair that can be found in SRAM FPGAs. The process of detection, isolation, diagnoses, and repair are the same, however, the approaches are slightly different in lower levels. Mainly the repair technique implementation needs to take into account the physical hardware characteristics. For example in an FPAA it is possible for faulty output to manifest in a voltage error variance. In the digital

implementation of the FPGA a single stuck at fault can occur where an output is stuck at a 1 or a 0 resulting in misleading results. The voltage variance in an FPAA could manifest in different ways making the task to detect much-much more difficult. The rest of the paper focuses on techniques for the SRAM based FPGA.

The remainder of the paper is formatted as follows. Section 2 provides a background to sum-up the work previously performed in the filed of FPD autonomous repair. Through regimented implementation and test the detection, isolation, diagnoses, and repair process has become a common theme in newer techniques which can be seen in techniques which try to provide solutions to cover the entire approach to the repair process. Section 3 highlights the strategies used in detection and isolation phases and then summarizes how repair can be achieved after isolation has taken place. For further information on the actual mechanism of repair and reconfiguration seek information on the use of Genetic Algorithms (GAs) for unique configuration generation for SRAM based FPGAs. Further information can also be found in [2] "on-line" repair techniques. Section 4 illustrates high performance applications and, finally, section 5 draws conclusions.

II. BACKGROUND

For each phase in the repair process there has been identified several techniques. These techniques share a common goal, but work differently and within their individual scope. Also each technique clearly has independent strengths and weaknesses. Tradeoffs can be made to determine which approach is best for the intended application. This section shows some of the current techniques in FPGA fault repair and breaks them down into their taxonomies. This section also shows the strengths and weaknesses for each identified technique. Techniques in the repair process are considered to be either user provided or manufacture provided. Furthermore, user provided techniques are considered to be either active or passive. This paper focuses on user provided techniques and no manufacture techniques are covered. However, it is important to mention that manufacture provided techniques are commonly referred to as "hardening" and typically included hardware-only techniques like radiation shielding to mitigate radiation damage from free radicals.

Fig. 2. Shows the process of system repair. The process includes the phases of fault detection, isolation, diagnoses, and repair. Normal operation is a part of the process, but it is not usually identified by researchers. The system is typically in normal operation prior to fault detection. After a fault manifest itself and is detected the system goes into either an off line or on line regeneration state. Depending on the technique(s) used this could include the substitution of prepositioned spares or the dynamic generation of a new system configuration which provides fault free behavior despite the presence of the fault. Dynamic configuration requires the use of on demand processing by Genetic Algorithms. To date there is no approach which uses an intelligence based method of regeneration. This is something that can be looked at in future work.

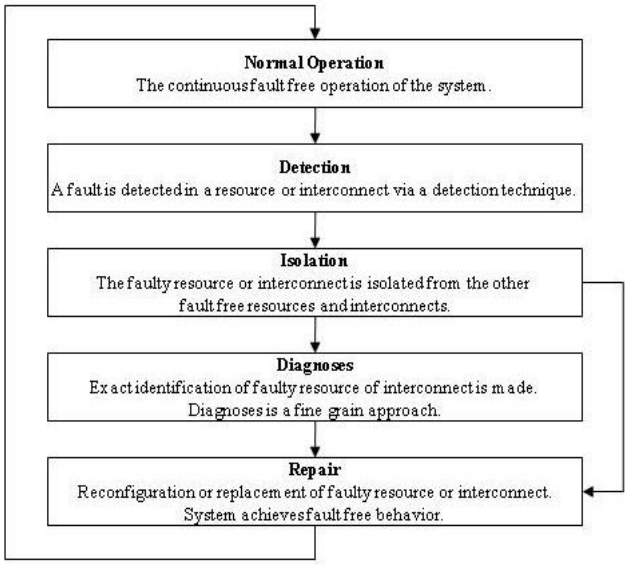


Fig. 2. System diagram for the reconfiguration repair process. The diagram illustrates the process of detection, isolation, diagnoses, repair, and finally return to normal operation.

The Detection phase includes the detection of a fault as a faulty output manifest itself. From the system's perspective

Technique/Capability	Detection	
	Fine Grain	Coarse Grain
CED		Concurrency
CRR [5]		Concurrency
TMR [7]		Concurrency
TMRSB [7]		Concurrency
Fault Filter [4]		Concurrency
STARs [6]	Exhaustive test	Concurrency

Table 1. This table contrast fault mitigation method detection capabilities.

the only indication that a fault has manifested in hardware is in the output of the hardware. It is currently not possible to detect a fault without observing the manifestation of the fault in some way. Thus a common attribute between each technique that can detect faults is that each technique is

designed around observation of the fault. Faults can manifest in output as either erroneous data or perhaps no output at all. It is also important to know that faults can be classified as either transient or permanent. Transient faults are data that has been corrupted as it travels through the data path. As the system finishes processing the transient fault, system output will resume fault free behavior. Permanent faults are the faults of hardware degradation and or destruction and which exhibit continuously faulty output. Table 1 contrasts fault detection techniques.

This means that techniques for fault detection are reliable given that a fault is manifested in data output and that the faulty output is articulated in the system. If the faulty output is never articulated then none of the techniques for detection will detect the faulty hardware. Furthermore, consider that if the system never articulates the fault even though the fault exists the system is still going to continuously provide fault free behavior.

Technique/Capability	**Isolation**	
	Fine Grain	**Coarse Grain**
CED		Very Coarse Grain
CRR [5]		Equivalent to CED
TMR [7]		Single Module
TMRSB [7]		Single Module
Fault Filter [4]		Single module
STARs [6]	Exact Resource	Equivalent to CED

Table 2. This table contrast fault mitigation method isolation capabilities.

The Isolation phase includes the identification of the faulty resource. Prior to entering the Isolation phase the only information known to the system is that there is a fault somewhere and that the fault manifested into faulty output. Identification can come in different levels of granularity depending on the technique used. The levels of granularity are classified as course grain and fine grain. Given the makeup of FPGAs fault identification can occur at the functional level, column level, Configurable Logic Block (CLB) level, or Look Up Table (LUT) level. There are also methods to isolate interconnection issues. However, it is possible to treat interconnection issues as CLB level and functional level faults. Isolation techniques can also be classified as active or passive. Active techniques search out the source of the fault and can produce fine grain identification of the faulty resource. Passive techniques can produce a course grain identification of the faulty resources. Course grain is typically no lower than the functional or modular level. Once a fault is identified it is up to the system to determine what to do with it. In practice the developer will put a single technique in place to perform the isolation depending on the level of granularity needed. Table 2 contrast fault isolation techniques.

Isolation can be complicated with fine grain identification techniques. It is common for isolation techniques which produce fine grain identification of faulty resources to require more resources for operation than techniques which produce course grain identification. Consumable resources include power, physical space on the FPGA, memory, and processor cycles. Fine grain isolation techniques also require off FPGA resources which are equally susceptible to the harsh environment. A common trait among isolation techniques is that, like in detection techniques, the system must articulate the fault. If the system never articulates the fault no identification can be made. To contrast fine grain and course grain isolation techniques fine grain techniques usually require multiple articulations as multiple articulations are required in the process of narrowing down suspect resources to fine granularity.

Technique/Capability	**Diagnosis**	
	Fine Grain	**Coarse Grain**
CED		
CRR [5]	Continuous testing	Functional Output
TMR [7]		
TMRSB [7]		
Fault Filter [4]		
STARs [6]	Exact Resource	Functional Output

Table 3. This table contrast fault mitigation method diagnosis capabilities.

It is also important to understand the relationship between isolation, diagnoses, and repair. As shown in figure 2 diagnoses can be bypassed and the process can go directly from isolation to repair. This quality is dependent on the level of granularity used in the techniques in which the repair process is employed. In coarse grain approaches it is not necessary to know exactly what the problem with the resource is. This means that just by the virtue of detecting the error the faulty resource is already identified and that within a given resource a bit is stuck or an interconnection is broken. In techniques which use spare allocations new spares will be substituted at the level of granularity in which the fault was detected. Furthermore, assume the designer wants to use the un-faulty portions of the faulted resource. This can not be implemented by course grain repair techniques. Reuse of faulty resource is a completely reasonable concept and the FPGA is capable of doing this. Reuse of faulty resources

would also help in long term sustainability and survivability. However, it would take very fine grain understanding of what the fault is and how the fault manifested in order to derive what future un-faulty capability a faulted resource could provide. Also further inspection would be required to identify how the faulty resource could be re-hosted within the current configuration. Table 3 contrast fault diagnosis techniques.

Technique/ Capability	Repair/Reconfiguration		
	Fine Grain	**Coarse Grain**	**Filter**
CED			
CRR [5]	100% restore	Swap for working Configuration	
TMR [7]			Can work with multiple errors using concurrency for output
TMRSB [7]		Swap a spare module	Can work with multiple errors using concurrency for output
Fault Filter [4]			Can work with multiple errors using concurrency for output
STARs [6]	Swap with working resource	Swap with spare module	

Table 4. This table contrast fault mitigation method repair capabilities.

Base on this understanding of granularity, in theory, the Diagnosis phase includes the algorithmic analysis of the fault itself. In practice diagnosis is more of a conclusion to the isolation phase given the technique uses a fine grain approach. In some diagnosis techniques resources are tested, at tedium, using exhaustive test. This means that resources are tested with every possible input multiple times. Some diagnosis techniques use only actual system generated input as it naturally propagates through the function. However, for both types of techniques regardless of the test, vector tests compare expected output with the actual output. Discrepancies in expected and actual output denote the fault. These discrepancies exactly determine what is wrong with the hardware. The diagnosis phase really determines in fine granularity what, where, and how the faulty hardware has developed. From this point, determination of path forward is technique based and the processes in this phase can vary considerably. In some course grain functional level techniques there is no diagnosis at all. The system simply truncates the faulted function at the functional level. Diagnosis techniques can be classified as fine grain, course grain, passive, active, exhaustive, and not exhaustive.

The Repair phase is the final phase of system fault repair. This phase includes the actual repair of the system's configuration. As mentioned earlier, currently SRAM based FPGAs are the only identified technology which can be reconfigured to mitigate permanent hardware faults. Aside from the actual process of reconfiguration of the FPGA, a determination of how to repair the system has to be made. Repair techniques can be classified as fine grain and course grain. There is also varying methods to restore normal operation to the system. Methods include swapping of predetermined spares or dynamic configuration generation using Genetic Algorithms (GAs). Of the GA operators for FPGA reconfiguration crossover and mutation are the most commonly used. Crossover works well because individuals in the population are comprised of actual configurations. When using crossover actual logical elements are being swapped which results in contiguous substitution of system resources. For optimal granularity resources to be swapped should be at the CLB or LUT level. It is important to note that neither substitution of predetermined spare resources or use of a GA will ever guarantee complete restoration of a system. To contrast spare allocation and use of a GA, spares have a predetermined finite swapping capability. GAs are not finite, so, in theory usage of GAs can potentially provide longer sustainability for remote systems as the system's resources could potentially be fully exhausted before the system is completely inoperable. Table 4 contrast fault repair techniques.

Technique/Capability	Overhead		
	Power	**Space**	**Complexity**
CED	Low	Low	Low
CRR [5]	Medium	Low	Medium
TMR [7]	High	High	Low
TMRSB [7]	Very High	Very High	Medium
Fault Filter [4]	Very High	Very High	High
STARs [6]	Very High	Very High	Very High

Table 5. This table contrast fault mitigation technique overhead.

Further background information on repair techniques and the application is given in [3]. The author elaborates on the indented usage and the requirements that FPGA repair techniques will have to overcome. Repair techniques will have to perform satisfactorily and take into account temporal fault arrival rates. This means the combined efforts of the repair techniques will have to be fast enough to support the mission of the system. Overcoming temporal arrival rates means that the system, as it goes through the process of

detection, isolation, diagnosis, and repair, will have to either be tolerant of multiple simultaneous faults or go through the repair process fast enough that repairs are solved before new faults arrive. The arrival rate of faults is an important concept to understand. Typically researchers implement an extrinsic test system which is a mixture of repair techniques, then, test are preformed on a simplified extrinsic prototype where only 1 fault is induced. The system is then tested to see how long it takes to attain 100% functionality again. No author tested a system that had a random fault arrival rate introduced. This means that laboratory tests are not representative of real world problems in remote systems.

Technique/Capability	Golden Elements
	Description
CED	Can be all on FPGA.
CRR [5]	Requires coordinator and GA processor of some type off FPA
TMR [7]	Can be all on FPGA.
TMRSB [7]	Can be all on FPGA.
Fault Filter [4]	Requires coordinator and GA processor of some type off FPA
STARs [6]	Very complex algorithms located off FPGA. System must work with stochastic processes.

Table 6. This table contrast fault mitigation technique "Golden Elements".

Another key major concept in the reconfigurable hardware repair process is that all of the techniques presented require some type of "Golden Elements". A golden element is a resource that is considered to be fault free for the correct operation of the repair techniques. This is a problem though, because the system in its entirety is susceptible to fault. There is no guarantee that the golden elements will remain fault free. Furthermore, the various repair techniques require varying size and complexity of golden elements. Most golden elements are located off of the FPGA in a microprocessor which controls test and runtime processing. Manufacturer based system hardening can help mitigate risk but it is never guaranteed that a system will not induce a fault in the golden element. Also there is no manufacturer hardening that can mitigate FPGA end of life fault manifestations. In best practice, repair techniques with golden elements that occupy the smallest foot print are less likely to incur faults in the golden element. It is important to minimize golden elements as much as possible.

It is a complex problem dealing with faults. Faults can arrive anywhere randomly. Fault problem solving is a stochastic process. Even in resources that are not utilized by the FPGA. These are called dormant faults. GAs that utilize predetermined configurations to conceive new working configurations would not even be aware that the new configuration has a dormant fault in it until the FPGA was reconfigured and the new fault manifested it self. Techniques that use spares also exhibit the same problem. If the spare isn't tested before rotating in there is no way to know if the spare is faulty. The point is that no technique presented within this paper is perfect. Each technique has strengths and weaknesses and each has an application. Some techniques only cover diagnosis while others cover all of the repair phases. It is possible, however, to look closely at what each technique is capable of and determine which technique will provide the longest sustainability, survivability, and availability within a given set of requirements. Tables 5 and 6 contrast fault technique overhead and "Golden Elements".

III. Fault Detection, Isolation, and Repair.

Fault detection, isolation, and repair is not a new concept in computer engineering. Prior to FPGAs Fault management schemes were implemented in Application Specific Integrated Circuits (ASICs). Unlike FPGAs, ASICs were fixed in design and could not be modified later on. Fault techniques were slightly different and focused on spares and redundancy being built in at design time. This means that whole functional blocks of hardware and associated software were implemented more than one time in order to increase survivability, availability, and sustainability. A single fault could render the spare or redundant module inoperable. So the more critical the system the more design time fault management would be worked into the system. Fault management schemes which used spares and redundancy incurred linear grows in overhead. It was a balancing act between Risk, Criticality, and Cost (overhead wise). However, with the recent capability to reconfigure an FPGA while in system, or during mission execution, fault management techniques are pushing the identification of working configurations as far out into the process of recovery as possible. This means that design time consideration for fault management is starting to become less-of-the-focus.

As identified in [4], [5], and [9] researchers have determined through analysis, simulation, and implementation that the use of Genetic Algorithms, post fault manifestation, is a very viable approach to increasing survivability, availability, and sustainability of their systems. Current research shows that Genetic Algorithms can evolve hardware with faults in place to obtain newly distinct, but functionally identical configurations for the FPGA. This has also been proven on other field programmable devices like the FPTA and FPAA.

After studying the current spectrum of fault management techniques, shown in the tables it is possible to see the evolution of these techniques. All of the techniques shown in

the tables incorporate a detection and isolation mechanism. But, only the CRR, TMRSB, and STARs techniques will individually detect, isolate, diagnosis, and attempt a reconfiguration or a repair. With the exception that the STARs technique relies on CED to overcome some of it's temporal fault detection weaknesses [6]. As mentioned earlier fault management techniques for FPGAs benefit from the work done with ASICs. Each of the identified techniques, shown in the tables, relies on these previously created approaches. The new addition to the previously completed work is the continuation of older concepts with the incorporation of Genetic Algorithms.

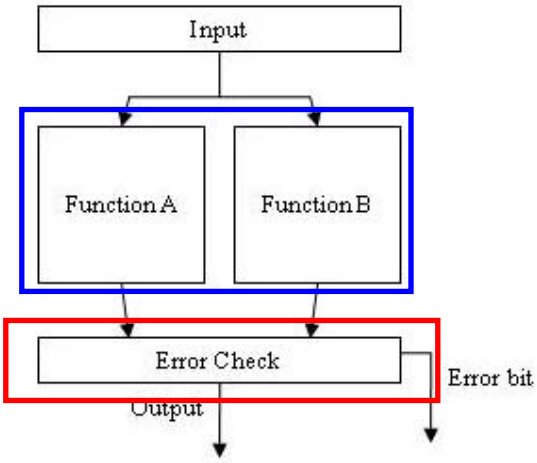


Fig. 3. This diagram illustrates the error detection and fault isolation capability within CED.

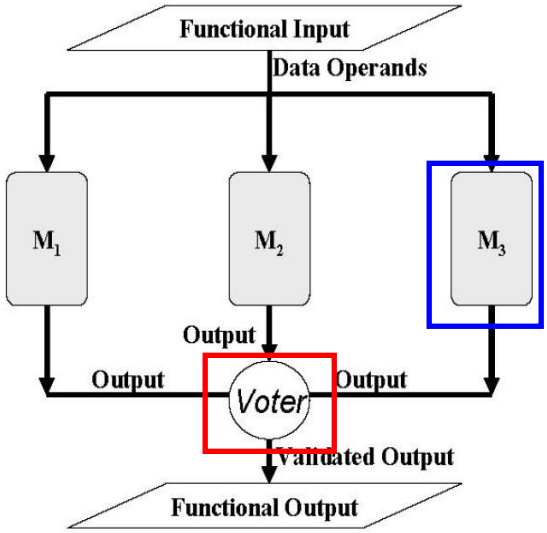


Fig. 4. This diagram illustrates the error detection and fault isolation capability within TMR [7].

The following subsections explore the detection, isolation and repair phases in further detail.

A. *Detection*

When comparing and contrasting the detection methods used in all of the techniques shown in the tables it is important to note how the detection method detects errors. This means two things; when can the error be detected and how can the error be detected. On further inspection, all of the techniques use CED or a conceptual form of CED, except for the STARs method. STARs is the only method that leverages CED to detect errors passively and provides its own active testing capability. This means that the STARs method also seeks dormant errors in the fabric of the FPGA by leveraging the Built in Self Testing (BIST) capability supplied by the manufacture and providing it's own extended test. This extended testing capability is called the BISTER. None of the other techniques do this and at this level of granularity. One might argue that STARs needlessly checks the FPGA for faults, but it depends on the application. Unlike the other techniques, STARs will actively check spares, resources, and interconnects which are not in use by the system, thus catching faults before the fault can manifest. Each of the other techniques has no knowledge of the current state of any spare in the system with exception of CRR. In CRR's case knowledge of a fault is only gained after a fault has manifested. Prior to manifestation CRR automatically assumes all spare resources and their applicable configurations are pristine and contain no fault, only after manifestation does a configuration and it's resources become suspect of containing faulty resources or interconnections.

The performance of CED is very good. The speed of which a fault is detected is almost negligible. CED has been used in multiple applications and each of the detection techniques after the advent of CED have used CED to detect errors and faults. CED's performance characteristics can be derived from [2], [10], [11], and [15]. The actual physical act of detecting a fault manifestation is nearly instantaneous. The red square shown in figures 3, 4, 5, 6, 7, and 8 show the mechanism which performs the detection of faults and errors. This mechanism can detect the manifestation of the faults and errors by comparing the output of two physically distinct functionally identical modules. Given a mismatch in output an error has manifested. CED can detect both transient faults and permanent faults.

The detection speed of CED goes with some caveats. While CED can detect a manifested fault it cannot detect any dormant faults. CED itself is also not capable of isolating faults effectively and there is no active component that is preventing faults from forming in the first place. When CED detects a fault the entire sub system for that instance of error

detection is suspect. This means that, as represented in figure 3, both functional modules are suspect as it is unknown which module is producing the fault. This also means that meaningful output has ended for the sub system. Later evolutions of CED can be seen in figures 4, 5, 6, and 7. Here TMR is introduced. By introducing a third module it is more likely that some kind of discrimination in the output can be made by the checker. This is why CED has been successfully iterated into TMR in later work; like the fault filter [4], CRR[5], and STARs [6]. In TMR, output can be maintained given two of the three modules have concurrency.

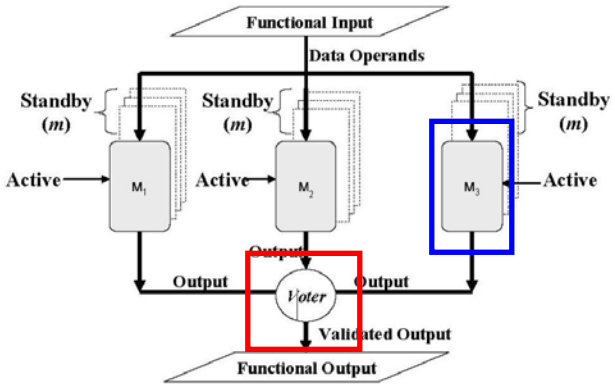


Fig. 5. This diagram illustrates the error detection and fault isolation capability within TMRSB [7]. TMRSB has the capability to repair the system using spares in standby. Some implementations use "hot" standbys.

"Golden Elements" are other areas of concern. Generally speaking, the fewer the number of "Golden Elements" the better. CED's golden element is the checker noted within the red box of figure 3, and for all other figures. If the checker is broken the entire detection capability is compromised. Not only that but depending on the configuration of the architecture all cascading output may be compromised resulting in an inoperable system. To combat this problem CRR incorporates the checker into the configuration of the system. This means that the checker is checked for concurrency with another checker. There is still a golden element here however it is less likely to produce faulty output later on as the output of the checker can reconfigured by a GA. As the techniques get more complicated in their detection methods more golden elements are introduced. This can be seen in CRR[5]. CRR proposes half configurations where functionally identical by physically distinct configurations compete in an elitism based paradigm. This technique is superior to other techniques, shown in the findings of the tables, but new forms of golden elements are introduced. Upon inspection of figure 6 it is apparent that the two modules must communicate output for comparison. The interconnect between the two modules is a golden element. Each modular configuration may not use the same resources for communication to the other half module. Also the interconnection goes between the drawn boundaries. This means that the interconnection is not a part of the scope of repair when a fault is detected. Thus for CRR it is paramount that the interconnection between modules be fault free.

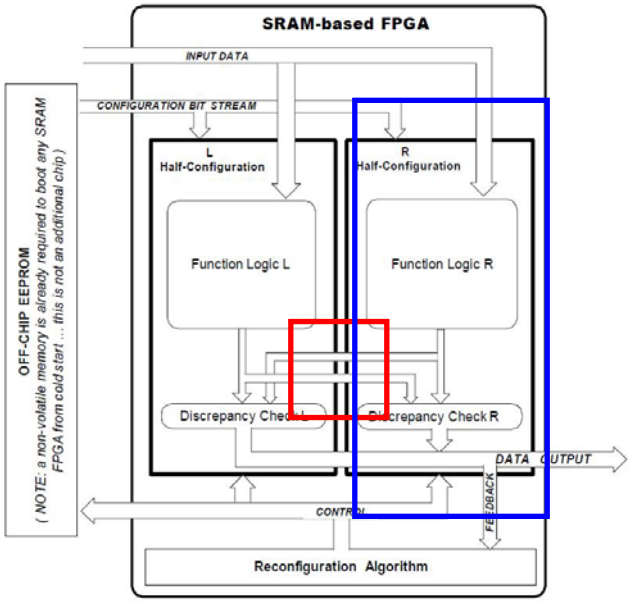


Fig. 6. This diagram illustrates the error detection and fault isolation capability within CRR [5].

STARs has the highest number of golden elements out of all of the techniques. STARs [6] is a very complex yet affect fault mitigation strategy. However the complexities introduce a number of golden elements. In STARs, all of the interconnections, TERC, off FPGA resources, and FPGA based BIST leveraged functionality are considered golden elements. STARs is best suited for terrestrial systems where the environment is harsh but radioactively benign.

B. Isolation

The isolation capabilities of each technique varies. These techniques can isolate faults at the fine grain level and at the coarse grain level depending on the design of the system and the level of incorporation of the technique, all of which is dependent on the acceptable level of over head. In each of the techniques shown in figures 3, 4, 5, 6, 7, and 8 the isolation capability is denoted by the blue box. It is important to understand the concept behind the level of incorporation of the technique. This does also relate to the level of granularity of the isolation. The blue box in each of the figures represents a module or function, of which the output of this module or function is compared to another physically distinct module or

function. For example, the module or function can be comprised of a single stage in a pipeline, a section of a pipeline, or even an entire pipeline [13]. The techniques scale well because the module is a black box. However, the higher level of granularity the error detection is incorporated at results in a lower understanding of the location of the fault. This means that in the TMR, configuration of each module is comprised of an entire 12 stage pipeline, when an error is detected one of the 12 stage pipelines is now compromised; this includes all of the pipeline's associated resources and interconnects. None of the techniques except for the Fault Filter, CRR, and STARs are capable or going back and trying to reclaim any of the faulted resources. This also implies that for the Fault Filter, CRR, and STARs extensive testing would be required.

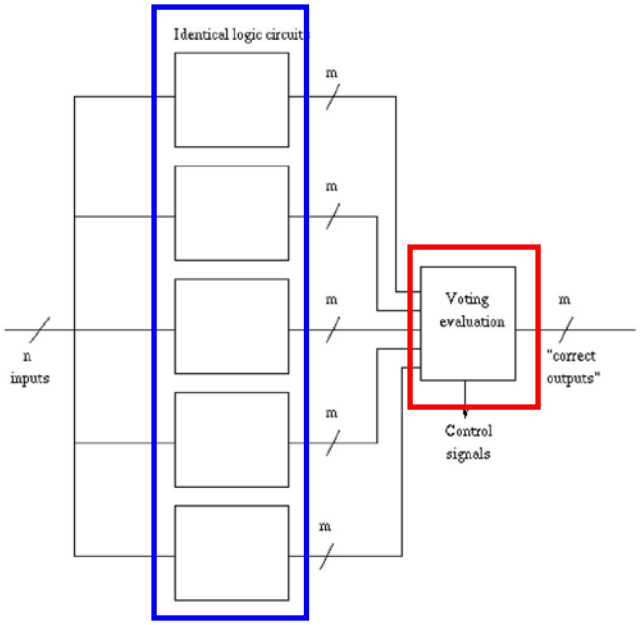


Fig. 7. This diagram illustrates the error detection and fault isolation capability within fault filtering [4].

CED doesn't intentionally isolate faults, however if implemented in lower levels of granularity at least a conclusion can be drawn about the two modules which conflict. The conclusion is that one of the two are faulty, thus in practice, they are both faulty. CED requires further logic to actually isolate a fault. Any of the techniques that provide more than two modules for redundancy provide this capability. More than two modules provides the capability to discriminate between each of the modules, then a single module can be identified or isolated. This is much better than CED however it requires 50% more resources than CED. This provides the means to determine in what functional module the error has manifested while also providing some output given that at least two modules still agree in their output.

The Fault Filter exploits the concept of multiple modules. By using multiple modules for redundancy a majority vote will provide consistent output even if none of the modules are 100% fault free. This concept, however, requires that there is still a majority, no matter how many faults exist, and even then cannot guarantee 100% fault free output. In the Fault filter the checker is a major component and if a fault occurs in the checker the entire system is compromised.

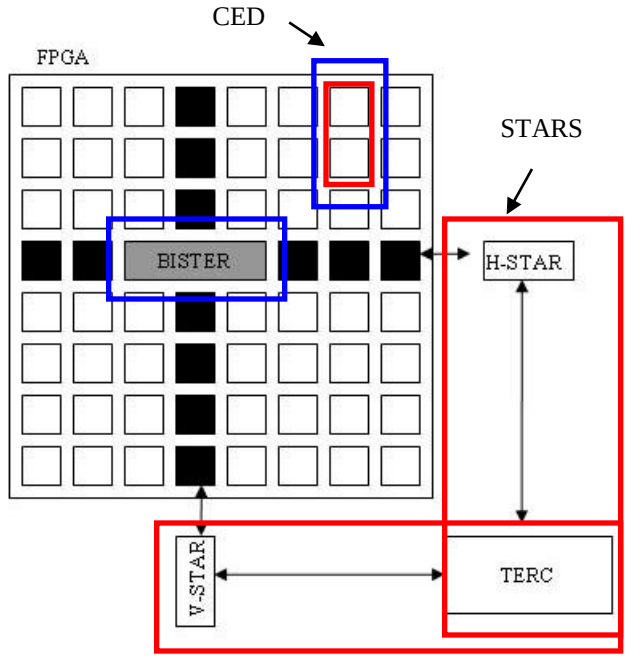


Fig. 8. This diagram illustrates the error detection and fault isolation capability within STARs [4].

There is another technique, called Combinational Group Based Testing (CGT) [12], that can be added in compliment of fault management techniques. This technique provides the capability to analyze suspect recourses in finer and finer detail until a single faulty resource is located. This technique relies on multiple reconfigurations of the FPGA and functional test of the reconfigurations to eliminate resources which are suspect. After a series of reconfigurations and functional test and given the system has a diverse enough configuration pool it is possible to isolate the single faulty resource. Typically in CGT the level of granularity is at the Programmable Logic Block (PLB) level, but it is possible to go to the Logic Look up Table (LUT) level. However, the lower the level of isolation the more diverse a set of configurations are required to isolate the fault. So at the LUT level much more granularity is required from the spare FPGA's possible configurations and more tests are required to figure out what resource is faulted within the PLB.

TMRSB, CRR, Fault Filter, and STARs do not require the

Technique/Capability	Performance			
	Temporal Fault Handling	**Spatial Fault Handling**	**Technique Speed**	**Long Term Survivability**
CED	1	Very limited	Negligible	Low
CRR [5]	Multiple	Multiple	Very High	Very High
TMR [7]	2	Very limited	Negligible	Low
TMRSB [7]	Multiple	Multiple	Very High	Moderate
Fault Filter [4]	Multiple	Multiple	High	High
STARs [6]	Multiple	Multiple	Slow	Moderate

Table. 7 Contrasts fault mitigation technique performance.

aid of CGT. Also it is important to note that any technique which does not utilize a GA to attempt repair will not benefit from the use of CGT. TMR and TMRSB are best suited for using CGT as a "bolt on" for isolation and diagnoses. An additional bolt on GA would be required to repair the FPGA. It does not make sense to use CGT in a fault management technique that does not attempt some repair to mitigate the faulty resource later on. CRR repairs it's configurations using a GA. When a configuration within CRR is proven faulty the entire configuration undergoes genetic modification. In TMRSB a "hopefully" working spare is used to replace the faulty module. In STARs and Fault Filter a brute force analysis of each resource is performed. These techniques use an exhaustive test approach in which the results will identify the faulty resource.

C. Repair

Repair of the FPGA constitutes the reconfiguration of the FPGA in a way such that the faulty resource is bypassed by the new data path and new interconnection path. Fancier techniques can reutilize the now degraded resource in a way that the output is still acceptable. CED and TMR alone are incapable of repairing the FPGA. As shown in figure 5, TMRSB relies on predetermined spares to replace the detected faulty resource. TMRSB is very expensive in overhead as a single set of spare modules doubles the space requirement for the technique and "hot" spares will double the power requirement by "N". All of this is in addition to the overhead of having three distinct modules running in parallel. CRR, Fault Filter, and STARs rely on the use of a genetic algorithm to identify a configuration for the FPGA that mitigates the faulted resource. Once the faulted resource is removed from the data path and interconnect path normal operation can resume. It is important to know that STARs is the only technique that can identify a faulty resource prior to manifestation. Upon identification STARs will reconfigure the FPGA to remove the fault from the data path and interconnection path. Once a physical fault has occurred the system can no longer reliably reuse that resource. In practice the resource is fully removed from the system.

After analysis of CED, CRR, TMR, TMRSB, Fault Filtering, and STARs the following is the derived overall usefulness of the technique in regards to survivability, sustainability, and availability of the system:

1) CED will no longer operate after 1 fault.
2) TMR will no longer operate after 2 spatially distinct faults.
3) TMRSB will no longer operate after enough spatially distinct faults manifest in a way such that no combination of modules produces consistent output between at least two modules.
4) CRR is comparable to TMRSB and will no longer operate after enough faults manifest in a way such that no combination of half modules can produce consistent output. This includes the caveat that CRR will attempt to repair a configuration to restore functionality in the configuration. However, 100% repair is not guaranteed. This means that with the addition of the repair capability the sustainability performance is open ended and the determination becomes a stochastic analysis which depends on the system's usage and age.
5) The Fault Filter will no longer work after enough faults manifest in a way such that no combination of modules produces consistent output between at least two modules. This is a scaled up version of TMR with the addition of a GA to repair faulted modules. Like CRR the repair performance is open ended and becomes a stochastic analysis which depends on the system's usage and age.
6) STARs, given continued operation of it's high quantity of golden elements, is also open ended. Assuming each of it's golden elements remain operational. STARs has a capability approximately equal to CRR in sustainability.

Provided the fault can be removed from the data path and interconnect path the system will resume normal operation until a new fault is detected. There are a finite number of faults the system can incur while operating normally. CRR, Fault Filter, and STARs currently provided the longest survivability as their probability of survivability is open ended.

IV. High Performance Applications

A common trait amongst all of the techniques, except for STARs, is that all of the techniques are based on N-Modular Redundancy (NMR). Instead of NMR STARs incorporates a completely different fault mitigation method and takes a system level approach to mitigating faults. This approach is specific to the application in which STARs is working for. The NMR approach is not specific to an application or system. NMR is only a simple technique which is applied to a data propagation path. It is important to understand the difference in these two approaches as their applications in high performance computing will be completely different; in good ways.

NMR is very scaleable and can be adapted to a hardware profile in multiple ways. The nature of the modular approach provides a scaleable fast mechanism to detect errors. In this method the module can consist of a single resource, a set of resources, a function, or an entire system. In a high performance computing application an NMR technique is best suited for application in a pipeline. Here, depending on allowable overhead and power consumption the modules can consist of single pipeline stages, groups of stages, or entire pipelines. NMR is based on concurrency which has the highest performance in detection speed. The detection speed is nearly negligible. Only after fault detection is system performance compromised.

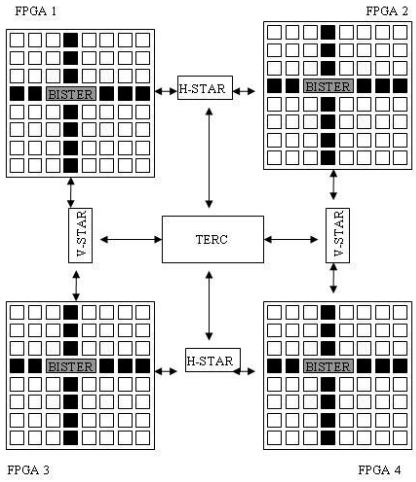
In CED system performance compromise is guaranteed upon fault detection because of the nature of pipelined processing. Each time the concurrency check fails the pipeline will have to be flushed. Thus it is not ideal that CED be used in a pipelined process. However, NMR schemes with more than 2 modules, like TMR, can provide adequate performance in a pipelined process. TMRSB implies that "hot" pipeline stages will be ready for swapping in given detection of a fault. CRR is better suited for configuration level items like the entire pipeline. Like CRR the Fault Filter is suite for single functions. In this case the Fault Filter uses concurrency to run as long as possible until concurrency is no longer obtainable.

Based on this understanding of the characteristics of NMR based fault mitigation techniques TMR and TMRSB are best suited for application into a pipelined process. Along with the high processing performance gains of the pipelined architecture the incorporation of TMR will also provide higher levels of sustainability, survivability, and availability.

Figure 9 shows a generic pipeline with a TMR application. Input is provided to the pipeline. Calculations are preformed in stages, as usual, except in this application the calculations are preformed in triplicate. There is negligible impact to the performance of the pipeline's speed up because work is performed in parallel. If the modules are comprised of distinct configurations, than the clock cycle will be based on the slowest module. The clock cycle, however, will also have to adjust for the propagation delay caused by the checker. Provided concurrency is met between 3 modules in any of the stages the checker will output to the latch. If a single error is detected output will still be provided to the latch with no throughput reduction. If two physically distinct faults are detected then though put will be reduced and a flush of the pipeline will be required.

TMRSB would be a benefit over TMR however the overhead would have to be taken into consideration. The most efficient way to deal with a detected fault would be to repair the faulty module during a pipeline flush. This would require that a GA operate on the module while the modules stage is configured in CED. Once the GA has found a successful configuration at the initiation of a pipeline flush then the module can be reconfigured and the stage can run in TMR again.

The draw back to an NMR high performance system is the overhead. TMR shown in figure 9 grows at approximately 3N where N is the number of pipelines. Implementing TMR in each stage of the pipeline would be expensive in overhead because of the triple redundancy, checkers, and power draw. It is possible, however, to configure only critical sections with TMR and leave the rest up to other less expensive techniques.



Fure. 10 illustrates a pipelined application using TMR. This configuration provides ultra high performance computing along with high survivability, sustainability, and availability.

In contrast to NMR systems STARs is suitable for system level applications. Applications of STARs require post FPGA design time knowledge for incorporation into STARs itself. Where NMR scales well at the modular level

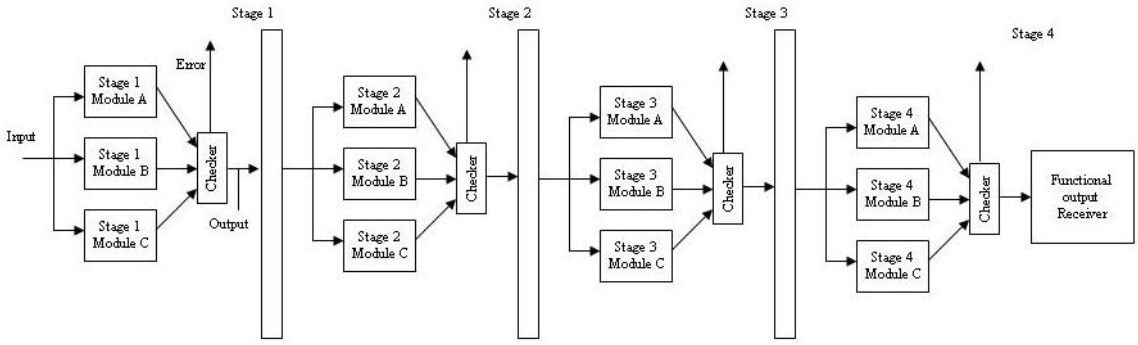


Figure. 9 illustrates a pipelined application using TMR. This configuration provides ultra high performance computing along with high survivability, sustainability, and availability.

STARs scales well at the core level. This, however, is an interesting and complimentary trait. While NMR is suitable for pipeline implementations STARs is suitable for multi core implementations. The foot print of STARs wouldn't grow significantly in a multi core implementation. There is, however, some extra required control logic that STARs would have to adapt to multi core implementations at the system level. Figure 10 shows STARs in a multi core configuration. This configuration would provide high performance in parallel processing throughput with the additional benefit of high performance in survivability, sustainability, and availability. The draw back to STARs in parallel, however, is that STARs is very complex to implement and the number of golden elements could be prohibitive in remote systems.

In the performance analysis the capabilities, strengths, and weaknesses for each technique are identified. The major driving factors are fault mitigation performance, overhead, and number of golden elements. It is observable that the more a single technique tries to handle the larger the overhead and number of golden elements.

Finally applications for high performance computing are identified. Each technique has applications which are better suited for it. NMR based techniques are best suited in pipelined or modular processing and STARs and configuration based techniques are better suited for functional and system level applications. Future work would include implementation of the concepts identified in the high performance applications section in the realization of a high performance ultra reliable processing component.

V. Conclusion

Newer FPGA fault mitigation techniques are becoming robust enough for inclusion into mainstream hardware architectures. For systems that must maintain ultra high availability, sustainability, and survivability techniques and applications using FPGAs are becoming more promising. Use of these techniques and leveraging the capability to reconfigure the FPGA on demand can greatly improve the performance of a system that must maintain high performance.

In this paper current fault mitigation techniques were compared and contrasted along with the process of implementing them. Normal operation, detection, isolation, diagnosis, and repair form the process in which to contrast current fault mitigation techniques. Some techniques cover the entire process and some techniques cover only one phase. No technique is perfect and each has strengths and weaknesses. Each technique has applications that the technique is better suited too.

References

[1] M. Parris, C. Sharma and R. Demara, "*Progress in Autonomous Fault Recovery of Field Programmable Gate Arrays*" ACM Computing Surveys, Vol. V, No. N, Article A, Pub. Date: April 2010.

[2] Stott, E., Sedcole, P., Cheung, P., "*Fault Tolerance and Reliability in Field Programmable Gate Arrays*" Computers & Digital Techniques, IET, vol. 4, no.3, pp.196-210, May 2010.

[3] Greenwood, G.W. "*Attaining Fault Tolerance Through Self Adaptation: The Strengths and Weaknesses of Evolvable Hardware Approaches*", In Zurada, J.M., Yen, G.G., Wang, J. (eds.) WCCI 2008. LNCS, vol. 5050, pp. 368–387. Springer, Heidelberg

[4] S. Vigander, "*Evolutionary Fault Repair and Electronics in Space Applications*". Ph.D. Dissertation University of Sussex, February 2001.

[5] R. F. Demara and K. Zhang, "*Autonomous FPGA Fault Handling through Competitive Runtime Reconfiguration*," in Proceeding of NASA/DoD Conference on evolvable hardware (EH'05), Washington D.C., U.S.A., June 29 – July 1, 2005.

[6] Abramovici, M.; Emmert, J.M.; Stroud, C.E.; , "*Roving STARs: an integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems*," Evolvable Hardware, 2001. Proceedings. The Third NASA/DoD Workshop on , vol., no., pp.73-92, 2001

[7] Kening Zhang; Bedette, G.; DeMara, R.F., "*Triple Modular Redundancy with Standby (TMRSB) Supporting Dynamic Resource*

Reconfiguration," Autotestcon, 2006 IEEE , vol., no., pp.690-696, 18-21 Sept. 2006

[8] Almukhaizim, S.; Bunian, S.; Sinanoglu, O.; , "*Reconfigurable low-power Concurrent Error Detection in logic circuits*," On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International , vol., no., pp.206-207, 5-7 July 2010

[9] Keymeulen, D.; Zebulum, R.S.; Jin, Y.; Stoica, A.; , "*Fault-tolerant evolvable hardware using field-programmable transistor arrays*," Reliability, IEEE Transactions on , vol.49, no.3, pp.305-316, Sep 2000

[10] Krasniewski, A.; , "*Concurrent error detection in sequential circuits implemented using FPGAs with embedded memory blocks*," On-Line Testing Symposium, 2004. IOLTS 2004. Proceedings. 10th IEEE International , vol., no., pp. 67- 72, 12-14 July 2004

[11] Almukhaizim, S.; Bunian, S.; Sinanoglu, O.; , "*Reconfigurable low-power Concurrent Error Detection in logic circuits*," On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International , vol., no., pp.206-207, 5-7 July 2010

[12] R. Demara, C. Sharma & A. Sarvi, "*Self-Healing Reconfigurable Logic using Autonomous Group Testing*". ACM Transactions on Autonomous and Adaptive Systems (TAAS) of Special Issue on Organic Computing. May,2007.

[13] Ezhilchelvan, P.D.; Mitrani, I.; Shrivastava, S.K.; , "*A performance evaluation study of pipeline TMR systems*," Parallel and Distributed Systems, IEEE Transactions on , vol.1, no.4, pp.442-456, Oct 1990

[14] Trimberger, S.; , "*A reprogrammable gate array and applications*," Proceedings of the IEEE , vol.81, no.7, pp.1030-1041, Jul 1993

[15] Krasniewski, A.; , "*Concurrent error detection in sequential circuits implemented using FPGAs with embedded memory blocks*," On-Line Testing Symposium, 2004. IOLTS 2004. Proceedings. 10th IEEE International , vol., no., pp. 67- 72, 12-14 July 2004