A Brief Survey of Techniques for Fault Tolerance in Field Programmable Devices

Grant Mackey*
*Department of Electrical Engineering & Computer Science
University of Central Florida, Orlando, Florida 32816–2450
Email:gmackey@eecs.ucf.edu*

***Abstract*—The focus of this work is on how field programmable devices handle faults. More specifically, how physical constraints may hamper fault tolerance techniques. This paper analyzes representative work in exhaustive BIST, voting, FPTA fault tolerance, competitive runtime reconfiguration, and embryonics. Its aim is to examine how the presented techniques perform with added constraints such as space/weight, time, power, et cetera.**

I. Introduction

Field programmable(FP) devices are employed by many different disciplines such as mechanical engineering, high speed low-latency supercomputing networks, signal processing, and many others. The greater appeal of these devices however, is for operation in hazardous and/or unreachable environments to humans, such as nuclear environments or space [3], [9]. While FP devices sitting in server racks or mounted to industrial equipment can suffer the occasional fault due to physical impact damage or heat stress, the likelihood that they fail due to a radiation induced fault is very low; extreme environments pose more challenges to hardware reliability.

The biggest issue facing a FP device is faults brought on by exposure to excessive amounts of radiation. The types of radiation induced faults that a FP device can encounter while operating in an extreme environment are Total-Dose Effects(TDE) and Single-Event Effects(SEE). The effects of TDEs for transistor based FP (FPTA) devices can include N_{OT} annealing, threshold shifts caused by: switching oxide traps, interface state buildup and annealing, and rebound or super-recovery [7]. Wherein the effects of TDEs in FP Gate Arrays (FPGA) are usually seen as loss of throughput due to an increasing propagation delay caused by differences in output rise and fall times for the programmed logic gates [7] or physical damage to the logic of the IC due to voltage surges. TDEs are considered as what a FP device experiences during its lifetime of operation.

In contrast, Single-Event Effects are instantaneous events of failure. SEEs can be described as destructive and non-destructive [10]. Non-destructive failures of FP devices are transient faults that are reflected in the logical operation of the FP device. For example, a high-energy particle interacts with the FP device causing an output being flipped from a 1 to a 0, producing an erroneous output. The device has experienced a fault, but no components have been damaged and the particle which caused the error is gone, in the next calculation the error will not occur. However, destructive SEEs cause permanent damage to the device. Some examples of destructive SEEs are Noise, leakage current, transconductance, Single-Event Latchups resulting in high operating current, etc. [1], [7].

The process of Radiation-hardening does increase a FP devices overall reliability, but this process does not ensure against all faults.

Hence, active fault recovery mechanisms exists. These techniques vary in approach, but ultimately provide reconfigurability mechanisms to recover from faults. Because these fault tolerance mechanisms vary in approach, their physical requirements of the hardware also differ. In devices which operate in extreme situations (such as space) time, weight, size and power requirements all become very important to the design of a FP device. The aim of this paper is to analyze when one technique is better than others for the given constraints.

In sections II-VI this paper discusses representative fault handling techniques in exhaustive BIST, voting, FPTA fault tolerance, competitive runtime reconfiguration, and embryonics. In addition, they are evaluated for performance based on how well they should theoretically perform when additional constrains such as weight, FP device size, power requirements and time are added. In VII, the best case performers for the varying degrees of constraints are discussed.

II. Roving STARs and Exhaustive BIST

A. Discussion

For section II we evaluate a work done by Emmert and Stroud as a representative of exhaustive built in self-testing (BIST) [2]. Roving STARs takes the self-testing areas concept and applies it to FPGAs to provide online fault testing, diagnosis and tolerance. Because the STARs are 'moving' objects within the design, fault detection is handled intrinsically, removing the need for dedicated fault detection.

Shown in Figure 1, there are two roving STARs the H STAR and the V star. These STARs sweep through the FPGA constantly, testing only the regions of the FPGA they cover, while leaving the rest of the device online and functional. While roving the FPGA these regions are performing very intensive BIST, covering both the logic and interconnect portions of the FPGA. If a fault does occur and is unrepairable, a STAR can be 'stolen' and its resources used to repair the faulty are of the FPGA. Interestingly enough, because the STARs BIST is so exhaustive and fine grained, partially damaged areas of the FPGA can be used to perform other tasks, rather than simply being discarded and no longer utilized. Figure 2 shows a 4x2 area BIST. Within it there are two spare logic blocks (SPARE), Three test pattern generator blocks (TPG), two blocks under test (BUT), and an output response analyzer(ORA). In this approach, two BUT are configured in the same manner and the three TPG send input to them which tests all modes of operation for

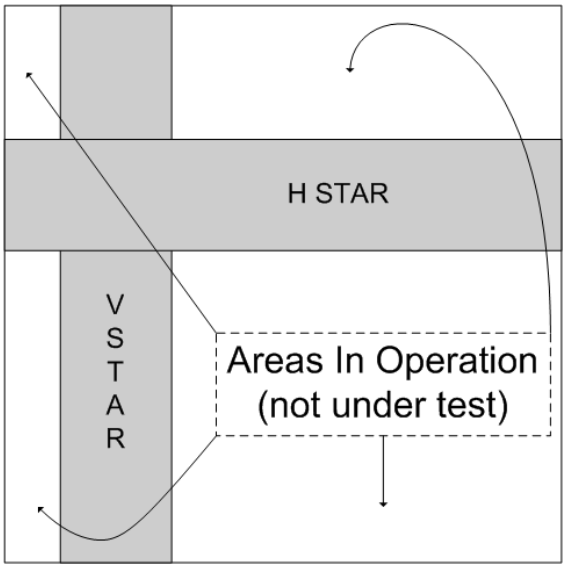

Fig. 1. Roving Example

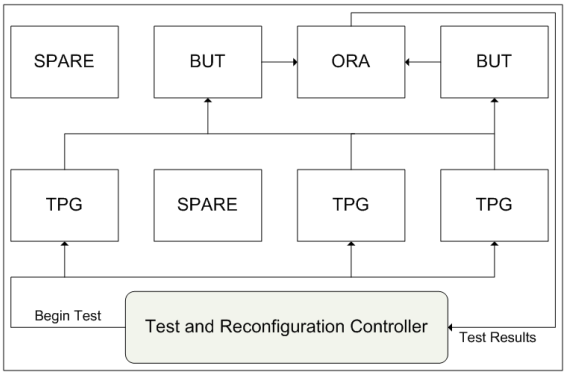

Fig. 2. Roving BiST

the BUT. both BUT then send their output to the ORA. Each incongruent output is marked as a failure by the ORA and the results of the test are sent to the test and reconfiguration controller (TREC). Once the test patterns have been completed for the two BUT, the 4x2 region is reconfigured again so that different PLBs are put under test. This process is repeated for the are until all of the PLBs have been a TPG, spare, BUT and ORA. The results recorded from this process are all sent to the TREC, which ultimately decides whether or not a fault has occurred and which PLB is the faulty resource. Because the fault testing process of roving STARs is independent of the FPGA logic, the component resolution of this BIST can be coarse or fine grained, it is left to the one implementing it.

Having two STARs (H STAR and V STAR) it is possible with this approach to detect and isolate interconnect problems. Wire segments in an FPGA are connect-able via configuration interconnect points (CIP). The BIST process for testing wires is the same as the BIST process for logic blocks, but instead of BUT you have wires under test (WUT). The rotation process continues for each area and the TREC evaluates where the fault has occurred. The hindrance to testing interconnects is that they are utilized by the FPGA, not just a particular group of PLBs. Hence, only a subset of CIPs are tested at any one STARs sweep as to not impede the functionality of the rest of the FPGA. Utilizing two STARs allows for signal re-routing while CIPs are under test.

The test and reconfiguration controller shown in 2 is a microprocessor outside of the FPGA. This external resource is a single point of failure for this approach. The TREC is responsible for controlling the sweep of the STARs across the FPGA as well as controlling the BISTs. It initiates a test area, assigns roles to the PLBs and analyzes output sent to it by the ORA. Without the TREC, there is no fault tolerance.

B. Evaluation

For the roving STARs approach several factors need to be considered when evaluating its fault tolerance ability. Given no restraints on time, space/weight, or power this approach provides complete fault tolerance. Both the logic and interconnect resources are considered in its fault handling mechanisms. The only immediate drawback to this approach is that the mechanism which drives the fault tolerance of the approach is not fault tolerant. If the external micro controller fails, there is no mechanism to keeps the STARs roving and hence the approach fails.

When adding constraints, there are several things to consider. 1) There is no fault detection mechanism in this approach, the STARs are always roving the FPGA looking for faults which may or may not exist. This in turn requires a consistent power draw to ensure that Emmert's approach functions correctly. 2) For each STAR, there must be spare resources and the TREC must exist. This adds additional space and weight concerns when designing a tool. The TREC adds additional weight concerns because it is a necessary chip for this approach, while the needs spares for the STARs decrease the effective space of the FPGA for other functionality. Finally 3) The detection latency for faults is non-determinant. That is, the detection latency for this approach depends on the size of the FPGA it is implemented on.

In a study by Parris, this latency is quantified in a relation [10]. While the latency for the larger of the Xilinx boards (XC4VLX200) is about 17 seconds per half of the FPGA, or $\approx$34 seconds is one STAR has stopped roving, the actual scaling of detection latency is sub linear for the increase in FPGA size (about 1/4th). Hence, it is possible that this approach will scale as FPGAs continue to grow in component size and operational speed. As far as whether or not the detection latency is small enough

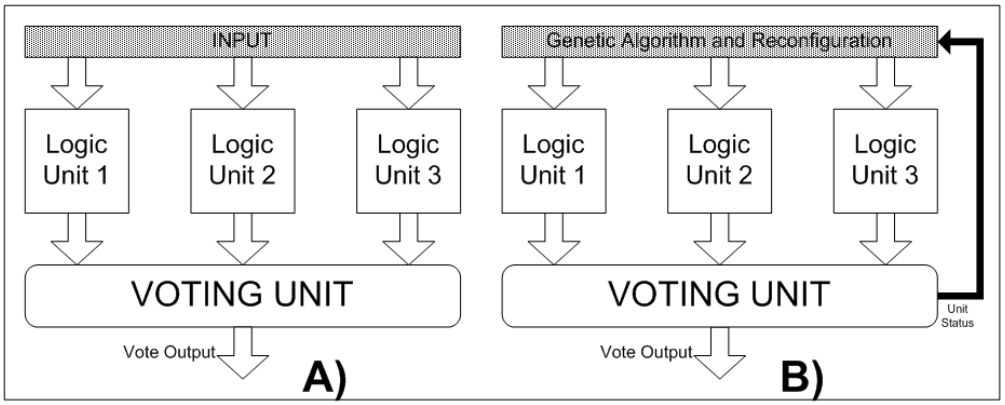

Fig. 3. A) Voting B) Voting with GA

or not depends upon the application the FPGA will be used for.

III. Triple Modular Redundancy

A. Discussion

This section discusses triple modular redundancy from the aspect of hardware spares and a single module recover approach approach by Vigander [11]. Figure 3, A) denotes the basic TMR approach. While this is a straight forward approach, basic TMR is limited in its ability to provide fault tolerance. An input is sent to three identical logic blocks, the output from those three logic blocks is sent to the voting unit, which chooses the correct output by majority vote. This approach cannot tolerate more than one failure, and has no means of correcting itself because it is a passive fault handling mechanism.

In an approach by Zhang, basic TMR is revisited with the addition Standby [8]. In this approach, the basic TMR structure of figure 3A exists, but with standby spare configurations (created at design time) and a means to detect if one of the logic units has experienced a fault. TMRSB utilizes the properties of TMR to remain online with one failed unit while a spare configuration is loaded, thus providing better reliability to TMR. The process of repair continues until all pre-designed configurations have been exhausted.

Vigander offers another approach to strengthen TMR, applying a genetic algorithm (GA) to the recovery of faulty logic units in a TMR configuration. Figure 3B shows the flow of adding a GA to the datapath of TMR. When output from the three logic blocks are collected, so is the state of component health. If a fault has been detected, the faulty logic unit is taken offline and undergoes evolution to repair itself. In his work, Vigander finds that obtaining 100% recovery rates for damaged logic in a voting scheme often takes too long. Instead, he finds that allowing for imperfect repair of damaged components provides for acceptable results. For example, in his experiments Vigander uses a 4 bit multiplier with voting. Trying to repair the multiplier for all inputs takes too long and is not always possible with the available hardware. However, accepting that the multiplier will produce correct results for most but not all of the possible inputs yields faster GA times and reintroduction of function to the TMR.

B. Evaluation

With the basic TMR approach there is no real fault tolerance, only fault masking. This approach requires three times as much hardware to implement a design. Hence, with this

approach there are size concerns as well as power concerns. However, the biggest advantage of TMR is that its fault handling is *in situ* and instantaneous, qualities that other fault tolerant methods do not have.

Zhang's addition to TMR, TMRSB, adds *even more* hardware overhead, 200% more. The FP device in question now requires 6 times as much hardware and power than before. However, unlike traditional TMR this method will continue to function after experiencing more than one fault (but only one fault at a time). The sacrifice for this approach is that by gaining extra fault tolerance, throughput is sacrificed for module failure detection.

Finally, in Vigander's approach there are still the power and size concerns of employing TMR. But, by utilizing a GA for the recovery of logic units this approach has better fault tolerance than TMR and potentially TMRSB. The drawback is that repair of damaged modules relies on a GA, which takes more time to complete than loading a new configuration like in TMRSB. In addition, Vigander expresses concern that because the GA is also present on the FPGA that it too might become faulty, producing improper reconfigurations. He proposes to mitigate this by sending corrective messages to the device periodically. This adds a distance constriction upon this approach. The author states himself that this approach will work best for a system that is staying local to Earth. Devices too far from a base station will wait too long to receive corrective information.

IV. Competitive Runtime Reconfiguration

A. Discussion

A more recent fault tolerant work (and its followup) introduced by DeMara is competitive runtime reconfiguration(CRR) [5], [6]. This fault tolerance mechanism utilizes a bimodal voting scheme rather than other online GA repair voting like N-MR. This approach uses

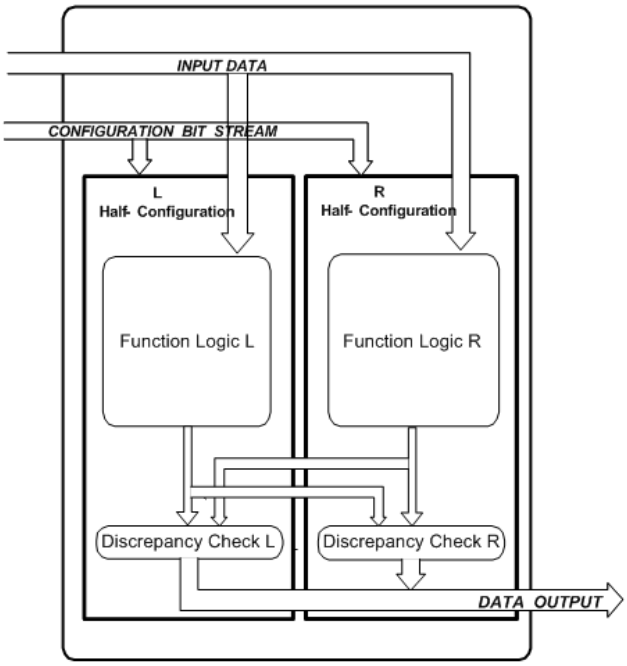

Fig. 4. Tandem CRR

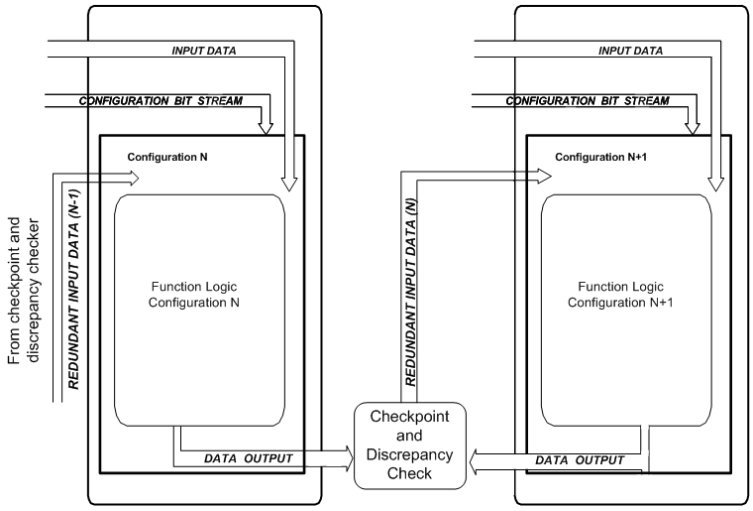

Fig. 5. Bounding CRR

traditional concurrent error detection in two different situations shown in figures 4 and 5 to ensure mutually exclusive resources.

In figure 4, the FPGA is split into halves L and R. A population of 'pristine' individuals are generated at design time and stored to the device. At runtime, two of these functionally equivalent physically different configurations are loaded into halves L and R. Throughout the course of operation random samplings of outputs are cross compared between L and R until a discrepancy arises. Once this occurs, the fitness of both 'pristine' individuals are lowered and new configurations are loaded at random (with favor of pristine individuals) and the process continues.

The detection phase for the bounding approach in figure 5 differs from the tandem approach. In bounding, the entire FPGA is utilized rather than half and half. In bounding CRR one configuration is loaded onto the FPGA and a random sampling of its output is recorded as a 'checkpoint' then a different configuration is loaded on to the FPGA. For this approach there must exist a checkpoint and discrepancy check module. This module will provide the next configuration to be loaded with the prior configuration's evaluation window. Before the new configuration can begin processing new input, it must redundantly compute the evaluation window data. If there are no discrepancies between the two data sets, then the new configuration continues until it is time to switch with a different configuration, wherein the process of checkpoint restart continues.

With the CRR approach there is no predetermined fault granularity for refurbishment. As configurations are rotated through the FPGA, faulty configurations fall below a set threshold and are refurbished by a GA, while the working configurations, which were falsely downgraded, are gradually exonerated. Hence this temporal voting approach does not need a dedicated fault isolation mechanism. With time, the faults which have occurred in the device become apparent. In this manner, CRR continues to refurbish the device configurations via a GA until there ceases to be enough resources on the FPGA to complete the given task.

B. Evaluation

In DeMara's work, tandem CRR is the only approach that was evaluated. This approach requires no additional hardware, all of the fault tolerance is contained within the FPGA, resulting in no additional weight. Tandem CRR noticed only an 12.6% reduction in device throughput while completing regeneration in their experiments, while yielding $\approx$ 96% correctness in design. Hence the time and performance of this approach are not drawbacks. The limitation of this approach is that it has a 2x overhead to implement a design.

Conceptually, bounding CRR removes tandem CRR's space issues. However, with bounding you sacrifice time for space. The reconfiguration time of the entire device posed to great of an overhead for this approach to merit analysis in the presented work [4]. If reconfiguration time could be mitigated, this approach would be a comparable trade-off with tandem CRR, space for time.

V. Fault Tolerance in Field Programmable Transistor Arrays

Figure 6 is a diagram of a field programmable transistor array (FPTA) cell. This is the functional equivalent of a BLE in FPGAs. For this section this paper looks to a work by Keymeulen as a representative work for FPTAs(cite Keymeulen).

A. Discussion

In [**?**], the author argues that while FPTA technology is not as mature as other FP devices such as field programmable gate or analog arrays(FPGA/FPAA), they are more useful in

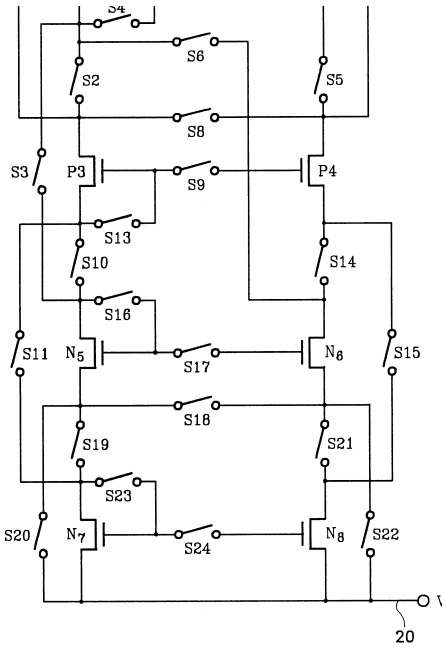

Fig. 6. FPTA Cell

some fields, such as robotics, where the granularity of the other devices is too coarse. The ultimate benefit of FPTAs is that they can be programmed to handle both analog and digital input with transistor level granularity.

An FPTA cell is an array of transistors which are interconnected via programmable switches. These programmable switches are also implemented with transistors which function as T-gate switches. Thus, the topology is represented by an on-off switch topology that can be represented as a binary state. While an FPTA cell is immutable, it is possible to build arrays of FPTA cells via FPTA cell cascading (a. stoica toward evolvable hardware chips: experiments with a programmable transistor array).

Figure 6 show an FPTA cell which has 24 bits to determine the state of the 24 present switches. Applying evolution to an FPTA is similar to FPGA techniques. Once the FPTA cell has been loaded with a configuration bit stream, the circuit is tested and its output rms value is compared against the target value to determine the configuration fitness. After all configurations are ranked, the best among them are selected and used to generate the next generation. In this evolution step, some individuals remain unchanged, while others will undergo crossover and/or mutation operators. The only difference between FPTA evolution and FPGA is the fitness metric, one is voltage level, the other is bitstream.

In this paper, Keymeulen presents two different styles of evolvable algorithms as well as intrinsic and extrinsic evaluation of those methods. Keymeulen explores a ”population-evolution” approach and a ”fitness-evolution” approach. The population based approach is a familiar online repair genetic algorithm, it aims to adapt the design of the FPTA at runtime in the face of faults. The fitness-based approach aims to determine, at design time, fault tolerant FPTA configurations that are robust enough to handle faults and continue to function. To properly generate new, genetically superior, individuals in a fitness-based model, the algorithm must have a wealth of knowledge about the possible failure conditions the FPTA might experience over the course of its life.

B. Evaluation

The first experiment they conduct is an extrinsic experiment of an analog multiplier. The population-based fault tolerance approach outperforms fitness-based approach in this set of experiments. There is no discussion on the time the GA used for the Population-based evaluation only the number of generations that were necessary to complete recovery. The population based approach yields better FPTA performance and converges to a solution faster than the fitness based GA. For the intrinsic experiment, Keymeulen models an XNOR gate.

Again, the population-based evaluation outperforms the fitness based GA. The detection and repair time of both methodologies was between 7 and 17ms.

The appeal of an FPTA is that it can obtain performance like an FPGA, but also has the ability to handle both analog and digital signals within the same device. Hence, in terms of space and weight, this is a good solution. While power concerns are not discussed, the response times of the FPTA are in the low milliseconds. Keymeulen also discusses the possibility of combining both methods to first model robust fault tolerant designs that recover from faults without reconfiguration with a caveat of being able to adapt to failure situations which were not modeled at design time. While this has the possibility to reduce the latency on failure repair even further for FPTAs, Keymeulen leaves this for future work.

VI. Embryonics

A. Concept

The goal of embryonics is to provide a highly scalable and fault tolerant FP device scheme. From [**?**], "[Embryonics] is ... an approach to improve fault tolerance in evolvable hardware by using a cellular architecture presenting dynamic self-repair and reproduction properties." Embryonics strives to obtain extreme fault tolerance in FP devices while needing little redundant components(spares).

The main design focus of embryonics is the artificial stem cell concept. That is, these 'cells' are designed with the ability to differentiate at any time into a needed functionality without being told how to design that needed functionality. That being said, embryonics has two design tenets:

- Each cell within the design space contains the whole genome. The cell is to be understood as the smallest atomic unit in the design, and the genome is the complete set of rules(functions) required for the design to work. The cell must also be able to decide and differentiate, via interactions with its neighboring cells, itself into any function which is needed to fully express the genome (the cell must be totipotent).
- The design must possess self organization properties. Each cell must be able to monitor its neighbors to decide if a fault has occurred. Then, based on the fault, decide how it must again differentiate into a different functionality do that the genome is expressed. For this tenet, spares are needed.

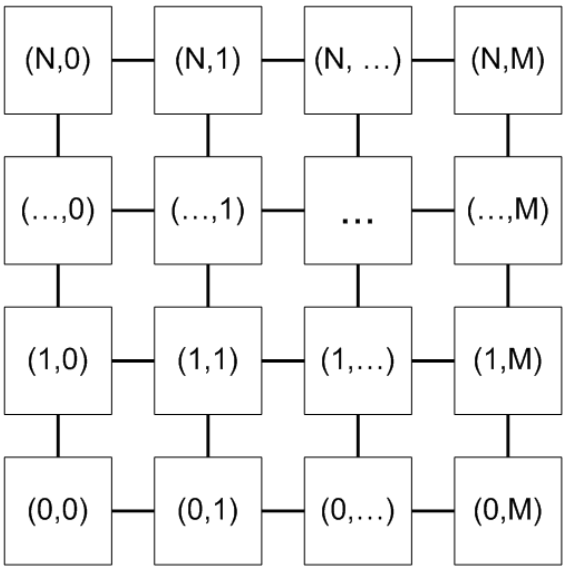

Fig. 7. Network

Figure 7 represents a genome. In this figure, (0,0) represents the 'mother' cell which is responsible for populating a space with all of the required functionality. This is embryonics' definition of cell division. The mother cell replicates itself to the top and right daughter cells at time 1, then the subsequent 'left-edge' cells replicate up and right while all other cells replicate right until the edges of the genome space are reached [**?**].

The functionality of each cell is identical. Figure 8, shows the basic function blocks of each cell. The Memory is the DNA of the

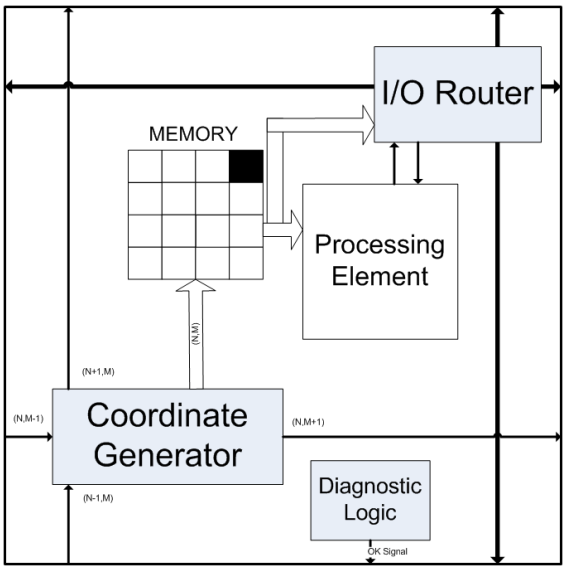

Fig. 8. Cell

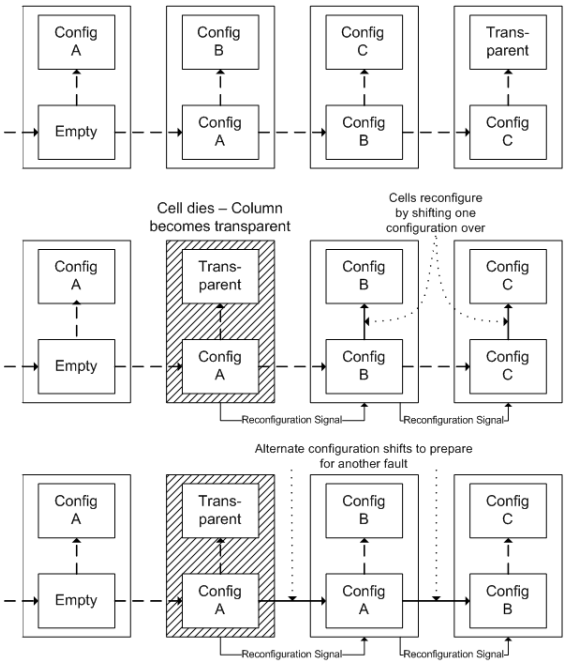

Fig. 9. Fault recovery strategy for [**?**]

embryonics cell. It contains all of the necessary configurations for a genome to be properly realized within a FP device. Based on input from the coordinate generator, the Memory sends the appropriate configuration to the processing element to establish the genome. The I/O router is responsible for passing all traffic within the cells of a genome. The diagnostic logic is responsible for reporting whether or not the cell has experienced an error/is no longer functional. In the event of a fault, it is also the responsibility of the I/O Router to pass traffic through the non-functioning cell if possible, and to reroute traffic around the dead cell if pass through is not possible.

B. Fault Tolerance

A work by Mange serves as one of the first attempts at implementing an embryonics approach on a FP device. [**?**]. In this work, fault tolerance is achieved through a binary decision forest. When a fault occurs within a cell, the cell is marked dead. Then the remaining cells consult the decision tree to then decide how they should organize and re differentiate. The shortcoming of this approach is that the configurations are predetermined, rather than dynamically generated based on the remaining resources. That is, once this approach runs out of spare cells it can no longer provide fault tolerance

Figure 9 shows an instance of cell failure discussed in [**?**], [**?**] which allows an array to handle a single fault. In this figure, each cell stores a neighboring configuration, as well as its own. In the event of a failure, the cell 'column' becomes transparent via the I/O routing resources and configurations are shifted over one cell in the array. In the example, configuration B fails. To mitigate this failure, configuration C differentiates into configuration B and a spare cell differentiates into configuration C. Finally, to prepare for another failure condition, each cell copies configuration data from its lefthand neighbor. This approach leverages partial cells rather than full artificial stem cells, and hence can only suffer one fault at a time per row. However to mitigate inter row issues of multiple failures, a similar process to column fault tolerance exists.

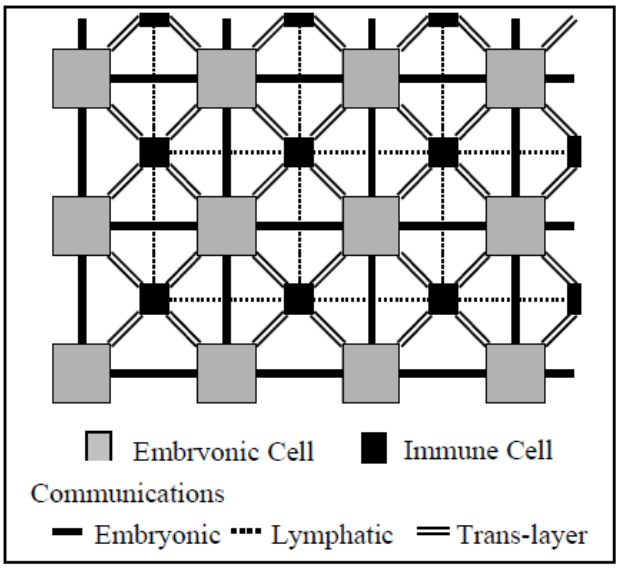

Fig. 10. Antibody cell monitors four closest neighbors. Each embryonic cell monitored by four surrounding antibody cells

A work by Bradley discusses fault handling in embryonics from an immunotronics perspective [**?**]. This approach uses self-healing/learning mechanisms within the cellular structure of the FP device to provide for greater resources utilization within an embryonics approach. Shown in figure 10, this approach adds an immune system to the genome configuration and an additional communication network to allow for global message passing of device health. With the addition of antibodies and the lymphatic communication network, this approach is able to use partially functional cells as opposed to other approaches where the cells are simply discarded.

C. Evaluation

In Mange, an embryonics approach is just realized. The main focus of this work was first establishing a homogeneous cell approach on an FPGA. They succeed, however no mention of performance or overhead is mentioned. From the paper it is possible to discern that the approach is fault tolerant. The cells are able to self organize and differentiate, meaning that the genome area is able to detect and isolate faults on a cellular level, rather than needing an overarching 'global' FD and FI mechanism. As this is a first approach it is possible to assume that the space implementation costs are higher as to provide this mechanism with the necessary spare cell resources, requiring larger FPGAs needing more power. However, conceptually this device needs no external hardware, adding no additional weight beyond the FPGA. Time also seems to be of benefit for this approach. While fault correction time was not discussed in this paper, the time to detect error among cells and reconfigure should be constant regardless of FP device size. Hence, as long as the time to detect faults among neighbors is minimal, fault recovery for this approach is superb because it is independent of the physical device.

The work by Ortega and the follow up by Canham show the possibility of a reduced set genome for embryonics approaches. This work still prescribes to the tenets of the field, interneighbor self organization and fault detection, as well as cell differentiation. The difference is that this approach requires less space than an approach like Mange. By having a reduced or 'local' copy of the genome, more space can be utilized for spare cells in the FPGA. In addition, this technique has a very simple but deterministic reorganization property. Unlike Mange, this approach is dynamic. Cells do not need to consult a decision tree to reorganize themselves, they simply 'move' a configuration to the right. Therefore this approach should have faster reconfiguration times than the approach Mange discusses. Unfortunately this approach has not been implemented in hardware yet. Hence there are no guarantees on its time performance. The bigger drawback of this approach is that there is no mechanism to reuse partially faulty cells. When this approach runs out of spare cells, there is no mention in the paper as to how this approach

deals with faults.

The work by Bradley discusses an important shortcoming of a pure embryonics approach. If a cell experiences a transient fault, it can still be marked as permanently faulty and discarded. With this approach, the antibody cells and lymphatic network allow for continual monitoring of the faulty resources to determine whether or not they can be reintroduced to the cell array. What is also possible with this approach is partial repair and cross genome cell function. That is, this approach has a global communication array that can determine if a partially functioning cell that was marked dead in one area of an array can be reintroduced in its original cell array, or if it is possible to be of use in a different cell array. However the author has not yet implemented this approach. If fruitful, this approach will add additional resources needed by the FPGA for the lyphatic system, however this approach gains space because there will no longer be a need for an excess of spare cells (which can also fail while dormant). This approach is a better fault tolerant methosd than the other two embryonic approaches discussed because it still has the same benefits, but has the added of benefit of recycling damaged cells.

In all, Embryonics is still a fairly new field. The papers being published are more interested in simply getting the concepts applied to physical hardware, rather than performance testing. As this field progresses and FP technology refines even further, Embryonics should prove to be a very powerful fault tolerance scheme for field programmable devices.

VII. Conclusion

Finally we will discuss the overall drawbacks of each approach as constraints are added to each approach and decide, conceptually, which approach is best.

A. Weight

In this section, weight is to be defined as any additional hardware required by the FP device in order to function properly.

1) Roving STARs and Exhaustive BIST: For this approach, it is important to consider the extra weight required by the external fault tolerance mechanism (TREC). Besides this module, there are no extra components.

2) Triple Modular Redundancy, Competitive Runtime Reconfiguration, Embryonics: These three approaches do not garner any additional weight for their FPGA implementations.

3) Field Programmable Transistor Arrays: What is interesting about the fault handling discussed in Keymeulen's FPTA work is that an FPTA can provide for functionality of analog and digital circuits. With this in mind, implementing a fault tolerant strategy on an FPTA device would potentially safe weight in a device which needs fault tolerance for digital and analog circuit because it can perform the job of an FPGA and FPAA. Overall, when this condition is applied, FPTAs are the best choice in regards to weight constraint.

B. Implementation Size

1) Triple Modular Redundancy: The largest device overhead of the discussed approaches, as the name implies this approach has at minimum a 3x implementation overhead. In the approach by Zhang, it is a 6x implementation overhead.

2) Competitive Runtime Reconfiguration: With the exception of bounded CRR, which sacrifices time for space, CRR has the second largest implementation overhead with a 2x overhead which is immutable.

3) Roving STARs and Exhaustive BIST, Field Programmable Transistor Arrays, Embryonics: These approaches do have space overheads from just simply implementing a design, that is the nature of fault tolerant hardware, somewhere there need to be spares. However, the exact conceptual overhead of these approaches is not discussed in the representative works with enough depth to draw specific conclusions among the three areas. It can be determined though that in relation to tandem CRR and

TMR, these approaches require less FP device implementation overhead.

C. Power

1) Roving STARs and Exhaustive BIST: This approach is a fairly power hungry fault tolerance strategy. Instead of a fault detection mechanism, the device FT strategy simply continuously and exhaustively tests the entirety of the FP device looking for faults. In addition, it also has a external piece of hardware which must also be powered.

2) Triple Modular Redundancy: This is also another power hungry strategy because three times as much hardware must be powered in order to have proper fault handling, regardless of whether the approach is TMR, TMRSB or Vigander's approach.

3) Competitive Runtime Reconfiguration, Field Programmable Transistor Arrays, Embryonics: These three approaches do not have any conceptual heavy power consuming traits. Among these three it is not really possible to discern which is 'best' on power consumption. Tandem CRR runs two functionally indentical physically different configurations simultaneously, looking for faults. This requires a 2x power draw for the functional implementation. Bounding CRR utilizes the entire FPGA, but must spend an evaluation window of redundant computation to discern whether or not the new configuration is faulty or not, wasting power.

For FPTAs whether or not the device level components are more power efficient than FPGAs or FPAAs depends more on manufacturers, rather than device components. However, as discussed in a case above, this approach would remove then need to power an FPGA and an FPAA if the mission called for such fault handling. The power consumption of the device though ultimately resides in the fault tolerance mechanism applied to it, which can vary.

Finally, in the case of embryonics, it seems that the field is too immature to draw conclusions. The current approaches seem to illustrate a large need of FP resource size, meaning a larger board to power the approach as compared to other more mature FT tolerance strategies. In the future as concepts like Bradley's work become more mature, the larger overhead of FPGA implementation should go down, reducing an embryonic systems' power needs.

D. Time

1) Roving STARs and Exhaustive BIST: This approach has a constant detection error detection and repair time. It is completely deterministic in that aspect. However, the detection time is dependant on the device that Roving STARs is implemented on. While it is constant for the device, it scales linearly as the device size increases. Hence, for a very large FPGA, this approach may have a very large fault tolerance latency.

2) Competitive Runtime Reconfiguration: The fault detection latency for the tandem CRR approach is pretty immediate. Once there is a discrapancy between the two sides, new configurations are loaded onto the FPGA. In the case of bounded CRR, while the entire device can now be utilized for a configuration, it sacrifices detection time in the form of a redundant checkpoint replay.

3) Field Programmable Transistor Arrays: There are no real benefits of a FPTA over a FPGA or FPAA in terms of recovery time. It is ultimately up to the fault tolerance mechanism being used. Keymeulen discusses an approach which he did not implement which would utilize precomputed fault tolerant designs with a population-based fault handing method. This approach could potentially save reconfiguration time becuase the precomputed fault tolerant designs would last through several device faults before needing a GA to reconfigure the device.

4) Embryonics: The fault detection time for embryonics approaches is fairly immediate and is also completely independent of device size. The repair times for this approach have not

yet fully been examined. Assumming that this approach matures, embryonics will become a very fast, and powerful fault handling method.

5) Triple Modular Redundancy: The clear winner for the least time needed to properly handle a fault is any TMR approach. TMR is immediate and in situ, nothing else is close. The only approach which might catch TMR in the future is embryonics. However, for the time being this is the fastest fault tolerance approach.

REFERENCES

[1] http://radhome.gsfc.nasa.gov/radhome/see.htm.

[2] Miron Abramovici, John M. Emmert, and Charles E. Stroud. Roving stars: An integrated approach to on-line testing, diagnosis, and fault tolerance for fpgas in adaptive computing systems. *Evolvable Hardware, NASA/DoD Conference on*, 0:0073, 2001.

[3] D. Alberto, E. Falletti, L. Ferrero, R. Garello, M. Greco, and M. Maggiora. Fpga implementation of digital filters for nuclear detectors. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 611(1):99 – 104, 2009.

[4] Ronald DeMara. Personal Correspondence at University of Central Florida.

[5] Ronald F. Demara, Senior Member, Kening Zhang, and Student Member. Autonomous fpga fault handling through competitive runtime reconfiguration. In *in Proceedings of the NASA/DoD Conference on Evolvable Hardware (EH05*, 2005.

[6] Ronald F. DeMara, Kening Zhang, and Carthik A. Sharma. Autonomic fault-handling and refurbishment using throughput-driven assessment. *Applied Soft Computing*, In Press, Corrected Proof:–, 2010.

[7] H. L. Hughes and J. M. Benedetto. Radiation effects and hardening of MOS technology: devices and circuits. *IEEE Transactions on Nuclear Science*, 50:500–521, June 2003.

[8] R. F. DeMara K. Zhang, G. Bedette. Triple modular redundancy with standby (tmrsb) supporting dynamic resource reconfiguration. *IEEE Systems Readiness Technology Conference*.

[9] D.S. Katz and R.R. Some. Nasa advances robotic space exploration. *IEEE Computer*, 2003.

[10] Matthew G. Parris, Carthik A. Sharma, and Ronald F. DeMara. Progress in autonomous fault recovery of field programmable gate arrays. *ACM computing surveys*, V(N), 2010.

[11] Sverre Vigander. Evolutionary fault repair of electronics in space applications. Master's thesis, Department of Computer Science, NTNU, 2001.