

# FPGA Based Fault Detection, Isolation and Healing for Integrated Vehicle Health

Ali Akoglu\*, Sonia Vohnout\*\*, and Justin Judkins\*\*

\*University of Arizona

Department of Electrical and Computer Engineering, Tucson, AZ 85721

[akoglu@ece.arizona.edu](mailto:akoglu@ece.arizona.edu)

\*\*Ridgetop Group, Inc.

6595 N. Oracle Road, Tucson, AZ 85704

[sonia@ridgetop-group.com](mailto:sonia@ridgetop-group.com) and [justin@ridgetop-group.com](mailto:justin@ridgetop-group.com)

## Abstract

Advances in VLSI technology have led to fabrication of chips with number of transistors projected to reach 10 billion in the near future. Affordable fault tolerant solutions transparent to applications with minimal hardware overhead in the micro architecture are necessary to mitigate component level errors for emerging system-on-chip (SoC) platforms. Ridgetop Group and the University of Arizona have developed innovative methods and systems for \_detection of anomalous conditions\_ that lead to faults in highly-complex electronic systems. Through built-in self-testing and fault detection, isolation and recovery capabilities we can offer 100% system availability and proactively avoid false or missed alarms, and estimate the remaining useful life of critical electronic components and their associated subsystems. A novel self-healing mechanism for SoC using field programmable gate array (FPGA) technology that localizes and isolates the faulty area and then replaces the functionality through partial configuration of the FPGA is introduced. Prognostic techniques are leveraged to address resource allocation and distribution to enable a more fault-tolerant, time efficient, and robust system. When prognostic detection methods are combined with reconfiguration strategies, system reliability and availability improve, reducing the probability of failure without compromise of either service quality or performance or requiring redundant components on the chip.

## Introduction

As computer systems become increasingly large and complex, their dependability and autonomy play a critical role at supporting next-generation science, engineering, and commercial applications. The design trends in the circuit technology with smaller die sizes and faster clock speeds have made the components more prone to random errors and crosstalk. On-board methods to not only monitor a system's health, but to also reconfigure systems to respond to hardware anomalies or failures automatically with recovery action are critical.

Autonomic computing holds the promise of handling this complexity by designing a system that is aware of its components and their characteristics inspired from the autonomous nervous systems of biological entities [1] [2] [3] [4]. Various ways have been proposed to achieve the fulfillment of this vision, from agent-based to policy-based self-management, from autonomic elements control loops to adaptive systems, self-stabilizing systems and many more. Support for self-healing system towards failure recovery requires extremely expensive manual labor. Fault tolerant services are provided with redundant hardware resources. Existing software based solutions are not transparent to the application, most of the times non-secure and requires powerful computers and introduces substantial amount of load and complexity for systems with heterogeneous software, hardware and network components [5] [6] [7].

In recent years, hybrid architectures embedding programmable logic core such as field programmable gate arrays (FPGAs) into a general Application-Specific integrated circuit (ASIC) SoC have started emerging [8] [9] [10]. Commercially available products such as Triscend's E5 and IBM's Cu-08 family expand SoC design flexibility, reduce design risk and lengthen product lifetimes significantly by allowing devices to adapt to changing standards over time [8] [11]. Several researchers have argued that using an FPGA core that is embedded in an SoC for testing purposes will reduce the test costs [8] [9] [10] [11] [12].

FPGAs have revolutionized the digital systems business after their introduction, almost 20 years ago. In its simplest form an FPGA consists of regular array of configurable logic blocks (CLBs) with a programmable interconnect structure. A CLB is a memory unit, Read Access Memory (RAM) and can be programmed by the user to act as a small logic module for a wide range of functions. The programmable interconnect between the CLBs is then configured to achieve the desired functionality. Today, state-of-the-art FPGAs allow accommodating digital systems with more than 10 million equivalent gates in its

reconfiguration fabric alone. FPGAs have traditionally been used for hardware prototyping or as glue logic, but today's gates are clocked at rates that make the prototype nearly as fast as the end product and much more flexible. As the logic density of FPGAs is outpacing Moore's law, reconfigurable computing platforms potentially offer enhancement of reliability and recovery from catastrophic failures through partial and dynamic reconfigurations; and eliminate the need for redundant hardware resources typically used by existing fault tolerant systems. Today's reconfigurable computing platforms have the capability, especially aggregated on specially designed printed circuit boards, to become self-contained, high-end computers. Moreover, their flexibility raises the possibility of meta-architecture; "morphing" hardware configurations with software as needed to improve efficiency, robustness, security and capability on-the-fly. Standalone computational nodes of the future will integrate FPGA cores along with several accelerator units and microprocessor core(s) on a single chip (SoC). In such systems, FPGAs will be used for both application acceleration and self-healing purposes.

Based on this projection, this paper introduces a novel reconfigurable architecture based on FPGA integrated SoC platform and presents implementation details of the self-healing methodology for the next generation services. We propose the development of a dynamically recoverable, reconfigurable FPGA based self-healing architecture that can recover from faults during and after deployment.

An innovative solution to dynamically reconfigure an FPGA is described below and best illustrated, at a top-level, in Figure 1, where the code stream is monitored by a Prognostics and Detection logic module that detects faults and initiates backup and recovery of program operation, on a relatively seamless basis. Our strategy defines a Built-In Self-Test (BIST) for systems to identify and isolate faults. The rest of the paper is organized as follows: a discussion on existing self-healing techniques that use reconfigurable logic, we then describe the proposed self-healing methodology followed by experimentation, results, an application and conclusion.

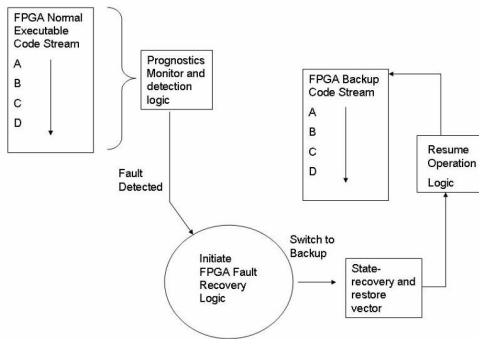


Figure 1: FPGA Fault-Tolerant Strategy.

## Related work

The current trend with FPGA based healing solutions is based on introducing module level redundant hardware units. Triple Modular Redundancy (TMR) [22] is a well-known healing technique that requires three copies of same logic or module with same inputs and outputs. It uses a voting mechanism to find out the "correct" output of the logic, which is based on the 2 out of 3 votes. TMR based approaches suffer from area overhead and complex control mechanisms associated with management of the redundant units. [17] uses dynamic and partial reconfiguration methods for testing, fault detection and healing with a decentralized controller unit. This approach suffers from several drawbacks. In a decentralized approach, failure of the controller unit may result with an unreliable system. Testing uses TMR with large area overhead and also interferes with the system operation, degrading the performance. Healing is achieved after multiple testing stages where the FPGA is configured for each test. This results with complex and lengthy reconfigurations. [19] and [20] propose a self-healing mechanism based on run time reconfiguration purely meant for the systems on FPGA. This approach enables the soft core interact with internal configuration port (ICAP) of the reconfigurable device (Xilinx Virtex2 family). A partial configuration mechanism manipulates configuration bit streams for self-healing. However, this approach employs "Triple TMR" that results with large area overhead, power consumption and performance degradation. Another approach is embedding reconfigurable logic at gate level of abstraction along with very fine grained circuit [18]. This approach not only introduces large area overhead but also makes the control system very complicated due to very fine level system healing. More over in case of no fault, reconfigurable logic is dispersed and is difficult to use for any application, hence logic utilization is reduced. Fine grain healing isolates the fault with minimal hardware loss however it is practically impossible for system level healing. Time to reconfigure such a complex system offsets the advantages gained with such granularity.

SoC platforms of the near future will integrate hundreds of components. There is a need for methodologies to find a minimum cost test vector in order to reduce the testing time. Additionally hardware overhead associated with the healing services should be minimized by isolating the faults at subcomponent level. Thirdly, advances in FPGA integrated systems offer potential solutions for testing and then re-using the same reconfigurable core for adapting to changing standards over time. Moreover, through partial configuration, embedded FPGA resources are available for healing the system.

## Self-testing and self-healing methodology

Our methodology involves the following steps: Partitioning (Design for testing), Testing, Detection,

Isolation, Recovery and Verification. Figure 2 shows a SoC model with components A, B, C, D, an FPGA, a controller and communication channels. The FPGA is used for configuring the controller that manages the healing process and replacing the subcomponent(s) (shown as A1, A2, A3, B1, B2...) of the faulty component(s). Each component of the system is first partitioned into subcomponents and additional control circuitry is introduced within these subcomponents to be used during failure isolation and recovery steps. Initial testing is applied to detect a failure at component level and then a second round of testing is applied to subcomponents of the failed module(s) to pin point the error location. After the detection stage, functionality of the failed subcomponent is replaced by configurable logic on the FPGA. The control circuitry introduced during the partitioning phase is used to bypass the failed subcomponent and connect the FPGA to the system completing the healing process. While partitioning is done at the design level before fabrication, the rest of the steps are done at post fabrication stages. The following sections give details of our self-testing and self-healing methodology.

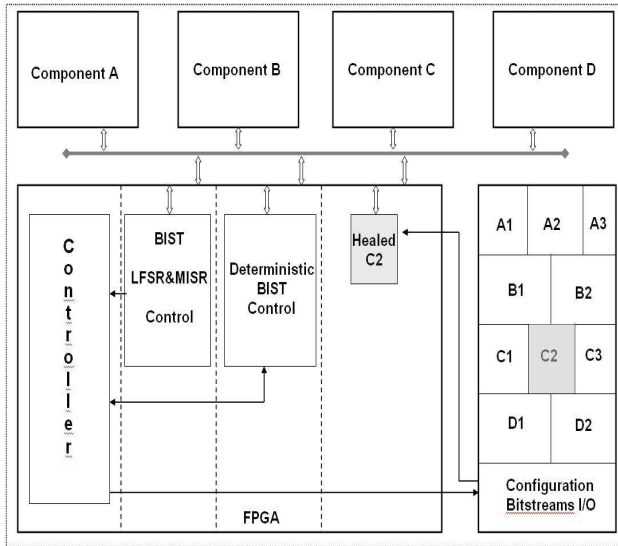


Figure 2. Self-healing Architecture

## Partitioning

Detecting the exact location of the fault is a very difficult task. Increased costs associated with Automatic Test Equipment (ATE) for testing a complex VLSI chip has made the "Design for Test" an attractive approach as it embeds testing logic at design level. We introduce a novel technique by which the circuit under test is partitioned into smaller subcomponents and the controller is aware of the functionality of the subparts (Figure 3).

The partition methodology considers the number of inputs of each partition, critical path elongation with the testing features, possible state errors and timing constraints. The partition parameters may vary with applications depending

on the criticality of each parameter in its own context. The partitioning problem is an iterative process, which ends when there is a feasible set of parameters for the partition. A feasible set of parameters does not affect the design constraints. For example, the critical path delay is one of the design constraints and testing logic should not affect this constraint. During partitioning, the testing logic should be added to the combinational logic path with maximum slack. After adding testing logic, the partition methodology should ensure that the combinational logic path does not affect the critical path delay. If this partition affects the critical path delay constraint, another partition is chosen.

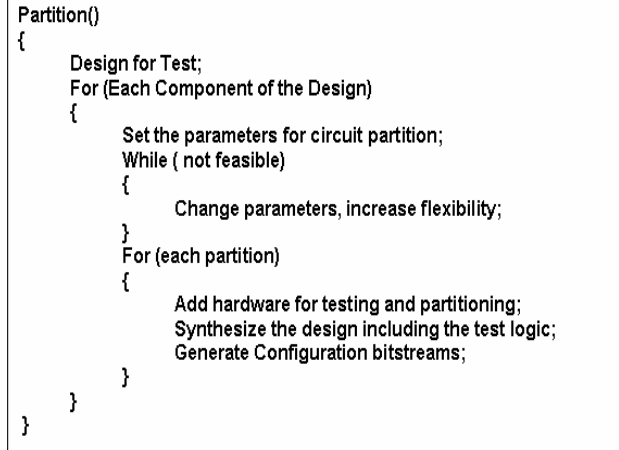


Figure 3. Partitioning approach

The partition methodology starts with low cost and low area parameters. If this set of parameters does not meet the timing constraints, another set of parameters are chosen with an increase in cost and area. The tradeoff is between cost, area and timing. Finding a feasible partition is followed by introducing routing resources for isolation; and the control logic for diverting Input/Output (I/O) signals during the healing process. This iterative process is repeated for each partition of all the components. Finally the design for testing is completed by synthesizing the system with the above testing features incorporated. An example sub-circuit under test (adder) with extra healing hardware (control circuitry) is shown in Figure 4.

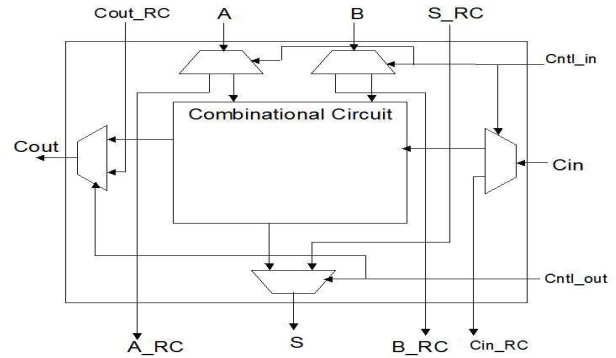


Figure 4. Example of a Partition with self-healing routing circuitry

If there is fault within the component, the original inputs A, B and Cin are redirected to the healed part (or reconfigurable fabric) through a demux with the “Cntl\_in” control signal, while output S and Cout are derived from the healed part through multiplexers with the “Cntl\_out” control signal. The signals A\_RC, B\_RC, Cin\_RC are the input signals given to the corrected part while S\_RC, Cout\_RC are the outputs from the corrected parts. The signals cntl\_in and cntl\_out determine whether to redirect the input and output signals or to continue the normal operation. This partition is only done when all the partition parameters are met and the controller is aware of the nature of the subcomponent. The functionality of each subcomponent is stored as a configuration bitstream in memory to be loaded onto the FPGA in case of a failure. In this study the circuit under test is kept simple just to avoid any deviation from the actual goal, which is the self-testing and self-healing than to worry about the complexities involved in partitioning the circuit. Since the proposed methodology is independent of the nature of circuit under test, once the healing methodology is validated for any (simple) system, it can be applied to any complex system without any concern.

### Self-testing methodology

Testing is applied hierarchically at two levels (Figure 5) similar to the approach taken by [21]. Component level testing is implemented using a pseudo random pattern generator to detect the faulty component(s) of the system. Then deterministic testing is applied to the partitions of each faulty component in order to localize the failure. That way full fault coverage is ensured.

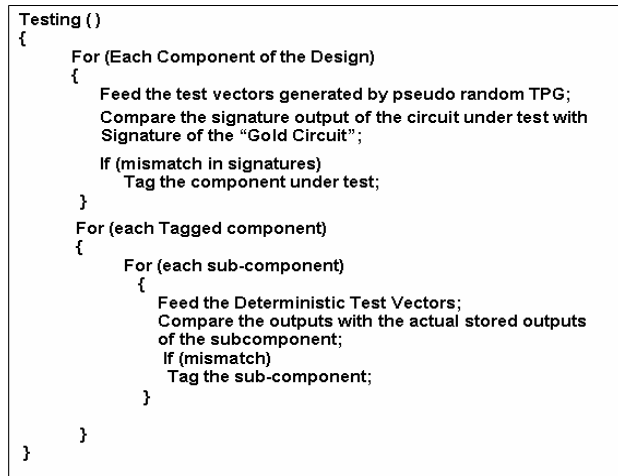


Figure 5. Two level Self-testing methodology

**Level 1: Pseudo Random Testing.** We consider Linear Feedback Shift Register (LFSR) and Multiple Input Shift Register (MISR) Testing [13] as it allows concurrency and efficiency in fault coverage. As shown in Figure 6, the LFSR generates pseudo random patterns and feeds them to the component under test. Through a compaction process MISR generates a signature of the output test vectors. This

is necessary to reduce the complexity of comparison operation. The signature generated by the MISR is then compared with the expected signature stored in memory. If there is a difference between the two signatures, a ‘1’ MISR tag is generated else a ‘0’ MISR tag is generated. This first level of testing “tags” the faulty component. All the tagged components then undergo second level of testing to localize the fault.

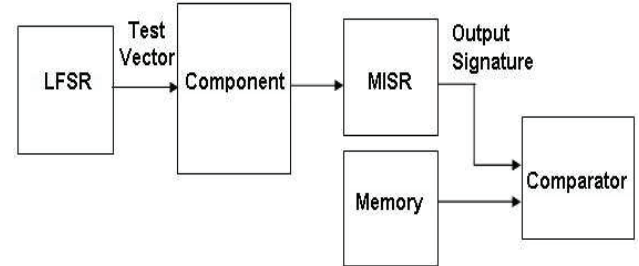


Figure 6. LFSR & MISR testing concept

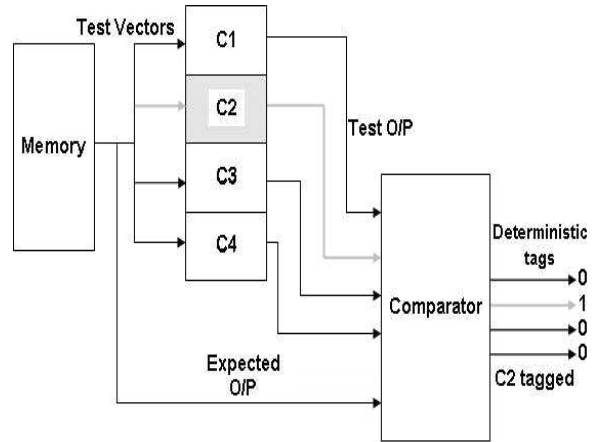


Figure 7. Deterministic testing concept

**Level 2: Deterministic Testing.** In this phase deterministic test patterns particular to each subcomponent of a “tagged” (faulty) component are applied to locate the fault. Figure 7 illustrates the flow of this deterministic testing phase. Once the faulty subcomponent is found, it is tagged appropriately. The same procedure is followed for all the faulty components. This second level of testing is straightforward as the test vectors for any particular component are fed from the memory. The outputs are then compared with the expected outputs stored in memory. If there is any discrepancy between two outputs, a deterministic tag is generated for that part of the component. Any mismatch initiates healing procedure for that subcomponent.

### The Controller

The controller unit manages and controls all the testing and healing processes. The controller unit oversees the pattern supply and initiates the test response analyzer to evaluate the output of the circuit under test. It is a state machine (Figure 8) based on 3 bit input vectors to determine the

output mode. An input vector consists of the following bits: [2] *Testing signal*, [1] *Deterministic tag* and [0] *MISR tag*. All bits are reset to 0 initially. The controller starts with LFSR and MISR testing phase, *Mode: '00'*. After testing, an MISR tag is generated. If there is no error, a '0' *MISR tag* is generated by the testing module along with a '0' for the *Testing signal*. Since the resulting input vector is "000", the controller goes to normal state S2, *Mode: '11'*. If there is any error in the circuit, the testing module generates a '1' *MISR tag*. This generates a '1' *Testing signal*. The input vector becomes "101" and from this input the controller goes to deterministic testing state S1, *Mode: '01'*. In state S1, deterministic testing is applied to the subcomponents of the fault circuit and the testing module generates the *deterministic tag*. This tag identifies the subcomponent. The input vector then becomes '111' and, the controller goes to the healing mode S3, *Mode: '10'*. In state S3, controller initiates the healing process. Following that, the *Testing signal* is reset to '0' and the controller goes to Normal state S2, *Mode: '11'*.

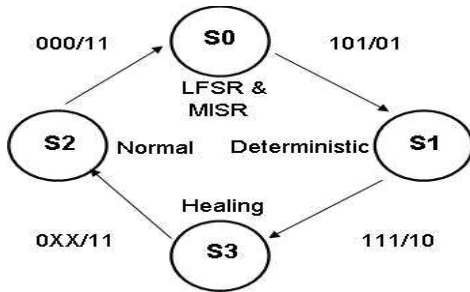


Figure 8. State diagram of the controller

### Self-healing methodology

When the controller unit receives deterministic tags (Figure 9), it initiates the partial configuration of the faulty subcomponent on the reconfigurable core. The healing signal indexes the configuration memory and loads the configuration bitstream onto the FPGA through its internal configuration access port (ICAP). Connections between the subcomponents are then diverted based on the data dependencies using the control circuitry introduced earlier during partitioning stage. The embedded routing and control logic helps isolation of the fault and healing of subcomponent tasks precede smoothly. The controller then evaluates the timing constraints and other system constraints. If they are in coherence with the original system, the controller proceeds to next faulty subcomponent. Otherwise some of the synthesis parameters are tightened to meet the constraints imposed by the SoC. This is an iterative process and is continuously done until a feasible solution is found (Figure 10).

### Experimentation, assumptions, limitations

The Xilinx Virtex II Pro XC2VP30 FPGA is used for simulating the above model. The partial reconfiguration

method introduced by [15] is used to configure the FPGA. The proposed self-testing and healing methodology is implemented on a Xilinx ISE 8.1 platform. Figure 11 summarizes the integrated healing environment developed for this study. Simple combinational circuits have been chosen to avoid the complexities involved in partitioning and to focus on self-testing and self-healing mechanisms. Secondly, in this study we considered only combinational logic. The above methodology is feasible to be implemented on a sequential circuit as it uses BILBO architecture. The sequential circuit can be taken to any state using BILBO architecture. During the experimentations, a Pseudo Random Pattern Test generator is employed with aid of appropriate polynomial [13] for feeding test patterns to component under test. The test pattern chosen is long enough to avoid any "aliasing" [14] effect. During the self-testing step, patterns are generated with a primitive polynomial for input size N=4. We have generated a test vector of 16 bits in size and fed the test vector to the circuit under test. The test pattern generator is prototyped on a Reconfigurable Logic with all the necessary control to interact with healing system. After allowing sufficient circuit delay we observed the outputs from the circuit under test and generated a "signature" for the set of inputs. This signature is compared with the already stored "Gold Circuit" Signature for any mismatch. Mismatches are recorded by tagging the faulty components.

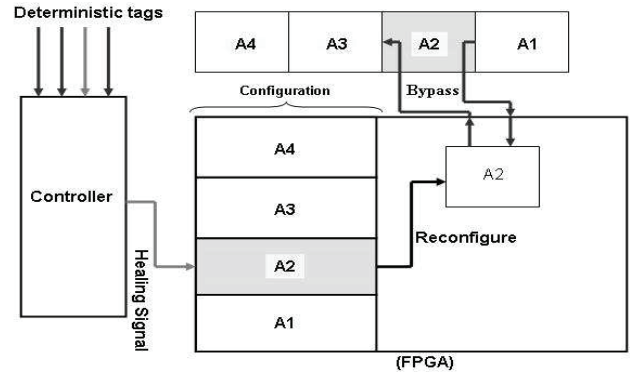


Figure 9. Self -Healing System

```

Healing ()
{
    For (Each Faulty Sub-Component of the design)
    {
        Obtain configuration bitstreams;
        While (timing constraints not met)
        {
            Replicate on Reconfigurable Logic and Transfer I/Os;
            Perform Timing analysis;
            If (timing constraints not met)
                Change the synthesis parameters;
        }
    }
}

```



Figure 10. Self-Healing methodology

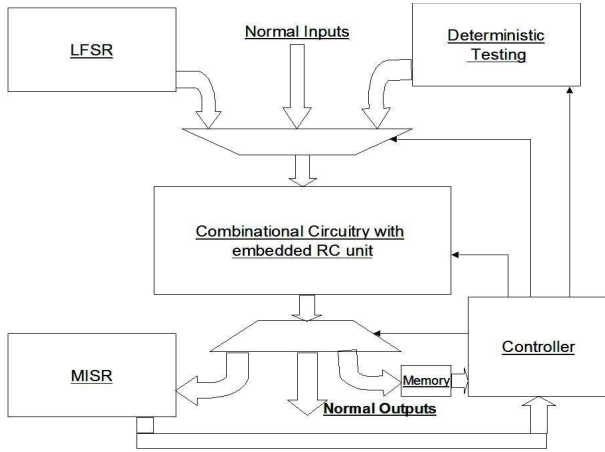


Figure 11. Healing Platform

## Results

Figure 12 shows LFSR and MISR step for fault free circuit (4-bit adder). MISR\_tag is the MISR tag generated when there is a fault in the component. Signature\_out is the signature generated by the MISR. The test uses a 4 bit LFSR which generates 15 random patterns. Therefore the signature at 15th clock cycle (the underlined signature), is checked and it is 10. As this is the correct signature, the MISR\_tag does not go high.

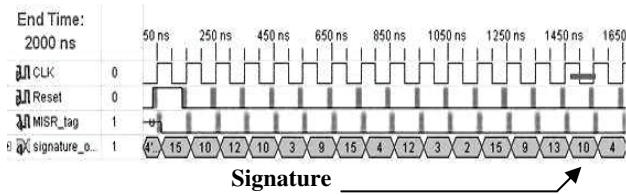


Figure 12. Signature and MISR tag for correct component

The system is then re-synthesized with an induced fault. As shown in Figure 13, there is a discrepancy in the signature generated at 15<sup>th</sup> clock (underlined signature) with the signature generated in Figure 12. Therefore, the MISR\_tag signal goes high at 16<sup>th</sup> clock cycle. But this only detects that the component is faulty. To find the particular faulty subcomponent, deterministic testing is then applied.

The component in this case is partitioned into four subcomponents and then tested deterministically. As shown in Figure 14, components of the 4-bit adder are represented with a four bit deterministic tag (f3, f2, f1, f0). Deterministic testing starts with component 0 and progresses sequentially. As shown in the waveform, and f1 (2<sup>nd</sup>. subcomponent) is detected to be faulty. Once the

deterministic tag is generated, it is given to the controller and the self-healing process is initiated. The reconfiguration of the subcomponent is performed and the system goes to normal state. Then, to validate the healing process, we again feed the test vectors to find out whether the faulty component is replaced or not. In Figure 15, X and Y are the inputs to the component (4bit adder) under test, Cin and Cout are the carry in and carry out signals of the circuit, S is the output of the healed component. Heal\_sig identifies the subcomponent that is healed (or configured on FPGA). As shown in Figure 15, the 4 bit adder functionality is resumed after the system healing part is completed.

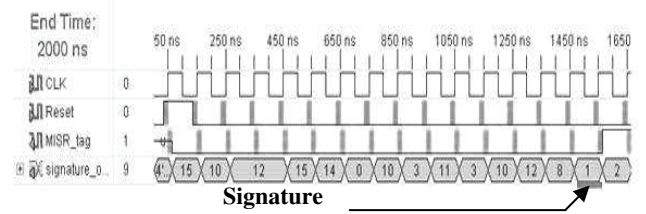


Figure 13. Signature and MISR tag for faulty component

### Faulty subcomponent detected

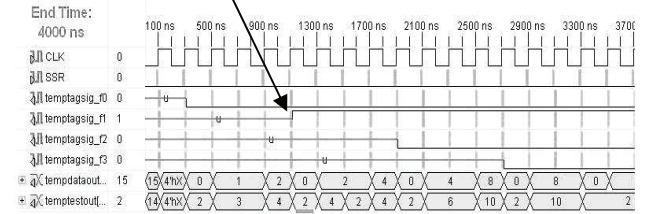


Figure 14. Deterministic testing on subcomponents of the faulty circuit

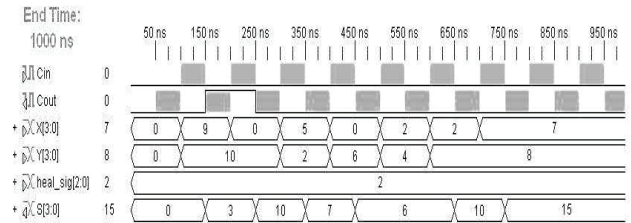


Figure 15. Output of the self healed component

## Integrated Vehicle Health Management (IVHM)

Our proposed FPGA-based self-healing architecture can be used to recover from faults during and after deployment in space. FPGAs are used in more than 80% of new digital designs; yet suffer from perturbations producing faults, like the intermittencies in the Ball Grid Array (BGA) package from shock and environmental stresses. Today, state-of-the-art FPGAs accommodate systems with more than 10 million equivalent gates in their reconfiguration fabric

alone. FPGAs are an increasingly attractive solution for many applications. Unfortunately, FPGAs are also susceptible to single-event upsets (SEU) [23].

The capacity for partial and dynamic reconfiguration eliminates the need for the on-board redundancies found in existing fault-tolerant systems. Existing strategies offer reliability, but are expensive in terms of resource utilization and power consumption. Fault-tolerant, recoverable FPGAs offer to NASA's IVHM project the following benefits:

- Seamless recovery of operational capability after fault occurrences and SEUs in FPGAs
- Support for autonomous operation of electronic systems in space
- Self-awareness and prognosis; anomaly detection and identification; and in-flight damage, degradation, and failure mitigation

FPGAs with fault-tolerant performance offer extended service life to space systems. The extension on service life will depend on the fault coverage achieved by reconfiguring the application for self-healing.

This fault isolation and healing methodology addresses IVHM across a wide range of digital applications. Because the solution can be implemented at the hardware description (Verilog or VHDL) level and subsequently realized at the physical level, it is extensible to any application where digital logic is synthesized, including ASIC. Certain parallel architectures, such as those used in Digital Signal Processing (DSP) and in some image processing, contain repetitive structures and are prime candidates for using these fault isolation techniques and self-healing. An ASIC specifically designed with this capability can therefore combine the advantages of high speed and performance with the fault tolerance required for high reliability missions in a harsh environment such as space. This capability can also be provided without the need for excess redundancy or external intervention. In contrast to scrubbing and TMR, which rely on reprogramming of the gate array, fault isolation and self-healing through direct rerouting, offers the flexibility to extend high reliability to other SoC applications.

## Conclusions and Future Work

This paper presents a methodology to design a Reconfigurable Core-Built in Self-healing system for a System on Chip (SoC). A two phase hierarchical self-testing approach is introduced to search for fault at both coarse and fine granularity levels. This approach reduces the complexity of test pattern generation. Locating the fault at subcomponent level reduces the amount of logic to be healed through the reconfigurable fabric hence leads to better resource utilization. Even for a simple component, partitioning of a circuit and inserting control circuitry for testing preparation is a labor-intensive process. For complex designs this task will require significant effort.

One should understand the details and functionality of the target component before attempting to partition the circuit and inserting control circuitry for healing. We believe that automating this process will be an essential part of this methodology in order to explore complex systems and help designers in self-healing domain make informative choices. In the future, we plan to extend our studies to explore various test pattern generations such as Weighted Pseudo Random Pattern test generator [30] and consider complex systems with sequential circuits.

This effort would set the basis of novel architectures for reusable space missions. The distributed, reconfigurable architecture will increase efficiency of development and life-time. Transparency to applications will increase productivity in new mission development/reuse and shorten development time, reducing costs. Overall adaptive capability makes for increase science return from optimized signal processing and control, longer lifetime from adaptive power management, increase reliability from optimal health monitoring including fault prevention schemes.

The work will result in a revolutionary advance in reconfigurable architectures for reusable missions. Optimal self-configuration will offer highly efficient implementations and enhance the science capabilities offered by computational platform. Reconfigurable computing platforms offer enhanced reliability and recovery from catastrophic failures through partial and dynamic reconfigurations; and reduce the need for high-block (less efficient) redundant hardware resources. Our hardware level self-healing mechanisms will result in evolutionary for autonomous systems by employing a hierarchical testing and healing approach. Hardware redundancy will be greatly reduced leaving more area for advancing the computation power.

## References

- [1] J. O. Kephart, D. M. Chess, "The Vision of Autonomic Computing", *Computer*, v.36 n.1, January 2003, pp. 41-50.
- [2] D. Kewley, J. Bouchard, "DARPA information assurance program dynamic defense experiment summary", *IEEE Transactions on Systems, Man, and Cybernetics – Part A*, 2001, vol. 31, no. 4, pp. 331-336.
- [3] Xie, M., Dai, Y.S., Poh, K. L. *Computing Systems Reliability: Models and Analysis*, Kluwer Academic Publishers: New York, NY, U.S.A, 2004.
- [4] A.T. Haytes, A. Martinoli, R. M. Goodman, "Swarm robotic odor localization", In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robot and Systems*, 2001, pp. 1073-1078.
- [5] R. Kube, C. R. Bonabeau, "Cooperative transport by ants and robots, *Robotics and Autonomous Systems*", 2000, vol.30, pp. 85-101.

- [6] L. Paulson, "Computer system, heal thyself", *Computer*, vol. 35, no. 8, 2002, pp. 20-22
- [7] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff, "NASA's swarm missions: the challenge of building autonomous software", *IT Professional*, vol. 6, no. 5, 2004, pp. 47-52.
- [8] P.S. Zuchowski, C.B. Reynolds, R.J. Grupp, S.G. Davis, B. Cremen, and B. Troxel, "A hybrid ASIC and FPGA architecture," *Proc. Int. Conf. on Computer Aided Design*, Nov. 2002, pp.187-194
- [9] S. Wilton and R. Saleh, "Programmable logic IP cores in SoC design: Opportunities and challenges," *IEEE Custom Integrated Circuit Conference*, May 2001
- [10] DATE 2001 Roundtable, "Adding reconfigurable logic to SOC design," *IEEE Des. Test Comput.*, vol.18, no.4, July-Aug. 2001, pp.65-71
- [11] M. Abramovici, C. Stroud, and M. Emmert, Using embedded FPGAs for SOC yield improvement," *Proc. Design Automation Conf.*, June 2002, pp.713-724
- [12] G. Zeng and H. Ito, "Efficient test data decompression for System-on-a-chip using an embedded FPGA core," *Proc. Int. Symposium on Defect and Fault Tolerance in VLSI Systems*, 2003, pp.503-510
- [13] Agrawal, V. D., Kime, C. R., and Saluja, K. "A Tutorial on Built-in Self-Test. I. Principles." *IEEE Des. Test* 10, 1, Jan. 1993, 73-82
- [14] Charles E. Stroud, "A Designers Guide to Built-In Self-Test," Kluwer Academic Pub
- [15] Manuel G. Gericota, Gustavo R. Alves, Miguel L. Silva, José M. Ferreira, "Run-Time Management of Logic Resources on Reconfigurable Systems," *Proc. of the Design, Automation and Test in Europe 2003 Conference and Exhibition (DATE'2003)*, Munich, Germany, March 2003, pp.974-979
- [16] Peter Messmer, Ralph Bodenner, "Accelerating Scientific Applications using FPGAs", *Xcell journal*, Second Quarter, 2006, pp.70-73
- [17] Katarina Paulsson, Michael Hübner, and Jürgen Becker, "Strategies to On-Line Failure Recovery in Self-Adaptive Systems based on Dynamic and Partial Reconfiguration," *Proc. of the First NASA/ESA Conference on Adaptive Hardware and Systems*, 2006
- [18] Vinu Vijay Kumar, and John Lach, "Fine-Grained Self-Healing Hardware for Large-Scale Autonomous Systems", *Proc. of the 14th International Workshop on Database and Expert Systems Applications (DEXA'03)*, 2003
- [19] Manuel G. Gericota, Gustavo R. Alves, and Jose M. Ferreira, "A Self-Healing Real-Time System Based on Run-Time Self-Reconfiguration," *IEEE International Conference on Emerging Technologies and Factory Automation*, Catania, Italy, Sept. 2005
- [20] Manuel G. Gericota, Gustavo R. Alves, and Jose M. Ferreira, Robust, "Configurable System Design with Built-In Self-Healing", *XX Conference on Design of Circuits and Integrated Systems*, Lisbon, Portugal, November, 2005
- [21] Gang Zeng† and Hideo Ito, "Hybrid BIST for System-on-a-Chip Using an Embedded FPGA Core", *Proc. of the 22nd IEEE VLSI Test Symposium*, 2004
- [22] Sandi Habinc, "Functional Triple Modular Redundancy (FTMR) Design and Assessment Report", Gaisler research, Dec. 2002
- [23] Pratt, B, et al. "Improving FPGA Design Robustness with Partial TMR", *Reliability Physics Symposium Proceedings*, 2006. 44th Annual., IEEE International, March 2006.