CPE-322 Homework #4

Kevin Barresi, Nishant Panchal, Giancarlo Rico, Bryan Bonnet

March 4, 2014

Abstract

The issue of digital circuit degradation is certainly an important one; as devices become more complex and expensive, the cost of circuit failure is very high. A single stuck-on fault can render an entire complex system inoperable. To date, existing methods focus on detecting rather than repairing these faults. This document proposes a method and framework of fault identification and repair in consumer-oriented markets using a hardware evolution approach. This system results in highly robust and reliable digital circuitry. The targeted platform for initial proof of concept (PoC) demonstrations is Field Programmable Gate Array (FPGA) devices.

1 Division of Work

Table 1 below shows the groups contribution to this project. Each member incorporated an even amount of effort to this homework.

Name	Per. Contribution
Kevin Barresi	25%
Nishant Panchal	25%
Giancarlo Rico	25%
Bryan Bonnet	25%

Table 1: Group contribution distribution

Kevin Barresi completed the algorithms section.
Nishant Panchal completed the introduction and abstract.
Giancarlo Rico completed the prior research section.
Bryan Bonnet completed the system overview and component details.

2 Summary of New Information Found

2.1 Introduction

Previous research investigating evolutionary self healing circuitry has fallen short on several fronts. One approach demonstrated that an evolutionary approach was feasible, but the methods used are impractical in field use. Their method was limited to the processing element (PE) level rather than the Configurable Logic Block (CLB) level and the self repair was limited to offline use. That is, during reconfiguration, the device was inoperable. Other work displayed promise, but was only demonstrated through simulation, and no furthering of work has been made.

The group presents an idea of circuit repair that takes advantage of all the issues mentioned above. The idea the group proposes is one where the analysis of the circuitry is performed online meaning that the device is completely operable during the reconfiguration phase. The groups method will ensure that no interruption or degradation of service is detectable in any manner. Our method will utilize intrinsic hardware evolution, where physical tests are run on the hardware block being reconfigured. The proposed method operates well for Single Event Upset (SEU) errors through partial FPGA scrubbing, along with permanent fault.

The remainder of this section is organized as follows. In subsection 2.2, we discuss prior research obtained from a multitude of industry sources. In subsection 2.3, we discuss the system configuration as a whole. In subsection 2.4, we detail each of the components the make up the system design. In subsection 2.5, we discuss potential algorithms for achieving our goals.

2.2 Prior Research

There are currently various methods of achieving fault resistance. Of these, notable solutions are: (a) locating and masking faults by circuit redundancy, (b) chipwise synthesis, (c) and triple-modular redundancy (TMR).

Fault Masking via Circuit Redundancy

In essence, redundancy involved building many circuit blocks of the same functionality onto the same FPGA, with the intent of only having one such circuit block operating at a time. If a fault is detected in one of these blocks, then one of the redundant blocks effectively takes over operation, with the failing block being completely bypassed by the rest of the device. Usually, redundant parts are built onto the device in columns and rows, though some redundancy architectures exist in which redundant routing resources are evenly distributed in the FPGA. This method of fault tolerance is transparent to FPGA users, and synthesis is very easy and can be used for all chips using the same FPGA implementation.

However, while this is perhaps the most basic solution to circuit faults, it is not without its disadvantages. Because of the large degree of redundancy, there is a high physical area overhead for the FPGA. Also, there is extra latency within the circuit due to the need to bypass any failing parts.

Chipwise Synthesis

Chipwise synthesis has been applied to circuits with high fault rates, especially with regards to implementations using nano-technology. In this methodm, each fault is located, and then placement and routing is customized for each chip so that faults are effectively worked around.

This solution, however, does not have any appropriate degree of scalability; it would not be suitable for production en masse of a single FPGA application. Also, testing costs are exceedingly high when the number of faults is large.

Triple-Modular Redundancy

Triple-modular redundancy is a specific form of circuit redundancy, in which three systems perform a process in parallel and the result is processed by a majority-voting system. This system produces a single output based on the output of each individual system. If any one of the three systems fails, then the other two systems can correct the fault and mask it. This voting system is implemented as a full adder, with its carry output acting as the vote output.

There are still weaknesses to this method. Foremost is the fact that this implementation relies on no more than one of the three systems failing at once; if two or more of them happen to fail, then TMR will produce an incorrect result. In addition, the majority-voting system itself could (with a low probability) be at fault. This can be addressed by implementing a TMR system for the first TMR system, however, the problem does still arise should the secondary system also fail.

2.3 System Overview

Various system designs and approaches were researched to effectively implement versatile, self-repairing digital circuits. This self-repairing device would attempt to find any damage present via an algorithm that calculates alternate configurations around the damage. The new configuration would be applied to the existing circuit alleviating the effect of the damage on circuit performance. This process of alleviating the damaged circuit should maintain the same level of performance as the undamaged device.

The system would be composed of two components: Operation Modules (OM) and a Configuration Module (CM). The Operation Module embodies the required action of the device and provide the desired function. Meanwhile, the Configuration Module acts as a controller for one or more Operation Modules. The operation of both

components complement each other in a systematic fashion to generate the emergent properties of a self-repairing digital circuit.

Physically, each component can be implemented either on the same or multiple FPGA devices. Components must be connected to each other via communication channels.
Further details of the each component are discussed in the next subsection.

2.4 Component Details

Operation Modules (OM)

Operation modules implement the desired function of the device.

Multiple Operation Modules can be implemented on separate FPGA devices or on the same FPGA device.

Configuration Module (CM)

The CM would act as the "brain" of the architecture, orchestrating error detection, new configuration design, and new configuration application to the OMs.

Configuration Module can be implemented either as a hard microprocessor or as a soft microprocessor on an FPGA device.

The configuration module operates as follows:

Fault Detection:
The Configuration Manager detects faults in one or more Operation Modules through constant application of test vectors. Predicted output with the actual output of the applied inputs to the Operation Modules are compared by the CM. Discrepancies in the predicted versus actual output means a fault in the Operation Module exists. Immediately after fault detection, device input is re-routed to one or more of the remaining redundant Operation Modules, ensuring no loss of operation. This feature ensures that the invention described by the present disclosure maintains operation through online hardware evolution.

Fault Localization:
The fault is localized to a specific area on the faulty Operation Module. By identifying the type and position of fault to a high degree, the Configuration Module can better devise a new configuration for the faulty Operation Module. The precise method of fault localization is further discussed in the *Algorithms* section

Configuration Generation:
The Configuration Module then works towards developing a new configuration for the faulty Operation Module that implements the same functionality of the original, working Operation Module. The Configuration Module takes into account the localized fault, as well as previously encountered faults in the same Operation Module. In this way, the Configuration Module is able to account for cumulative faults in the new design.

Reconfiguration:
Once a suitable replacement configuration is found, it is applied to the faulty Operation Module. Device input is re-routed back to the previously faulty Operation Module, and the Operation Module used in the interim is once again deactivated.

During reconfiguration, the Operation Module will continue using the secondary Operation Module to continue evolving potential solutions for the working Operation Module, with the goal of increasing circuit speed, decreasing circuit power consumption, and decreasing circuit CLB usage. As a result, the entire system as a whole is constantly evolving towards more and more efficient methods of achieving the desired functionality.

2.5 Algorithms

A vital part of the proposed system is an effective algorithm base for achieving the desired functionality. The system can be divided into three areas:

Configuration Module Behavior

The Configuration Module runs in a basic loop. The pseudo-code for operation is showed in Figure 1. Essentially, the Configuration Module acts as a watchdog for hardware faults. Once detected, it quickly switches the load to working Operation Modules in order to preserve device operation. It then derives a new circuit configuration, and applies it to the faulty Operation Module, switching back to it. Once a fault has been discovered, the Configuration Module continues testing potential configurations on inactive Operation Modules, with the hope of discovering more effective configurations than what currently exists on chip.

Figure 1: Basic operation of the Configuration Module

```
1: bool previousFault = false;
2: bool failed = false;
3: repeat
4:     failed = runTestVector();
5:     if failed then
6:         switchActiveOM();
7:         deriveNewConfig();
8:         applyConfig();
9:         previousFault = true;
10:        failed = false;
11:        switchActiveOM();
12:    end if
13:    if previousFault then
14:        runPopulationOnInactiveOM();
15:    end if
16: until forever
```

Fault Detection and Localization

In order to be given an accurate idea of where faults lay, it is important to be able to detect and localize faults with a high degree of certainty.

In order to detect faults, the Configuration Module runs test vectors. These vectors consist of already known inputs, and the related expected outputs. By applying these to the inputs of the Operation Modules and measuring the output, the Configuration Manager is able to determine whether or not the Operation Module is operating correctly. Naturally, high quality test vectors are vital to detecting faults.

Once a fault it detected, the localization process begins. Several effective methods for localizing these faults exist, including interconnect testing[6][7], LUT/RAM testing[8], and logic block testing[9]. A large portion of these methods utilize the Built-in Self Test (BIST) architecture, allowing devices to test themselves. However, the proposed architecture has the added benefit of having the Configuration Module to perform testing, resulting in better localization and detection than BIST-based testing.

New Circuit Generation

The ability to generate new circuits when faults are detected is critical to proper operation of the proposed architecture. We use an evolutionary approach, where a population consisting of potential circuit solutions are allowed to evolve over time into highly efficient solutions.

Genetic algorithms (GA) are a type of search heuristic that model themselves after the process of natural selection. GAs generate a population set of individuals, each representing a possible optimization solution. Though iterations of slight modification and recombination, the population converges on a true global optimization value. These iterations, known as generations, occur in three stages: selection, reproduction, and mutation.

The first step, selection, focuses on ranking individuals as judged by a fitness function. Least fit individuals are removed, while those remaining move onto the reproduction stage. During the reproduction stage, parents are selected and produce offspring by combining characteristic through a process called recombination. The number of

characteristics exchanged during recombination is controlled by the probability of crossing over, P_c. The population then undergoes mutation, a critical stage that introduces random modification to individuals. The rate of mutation is characterized by the probability P_m. Adaptive Genetic Algorithms (AGA) modify P_c and P_m as the population matures.

Hardware evolution is a subtype of GAs that aim to produce highly effective hardware design. In our architecture, we proposed using Cartesian Genetic Programming (CGP) as a technique for efficiently producing hardware configurations. Benefits include simple implementation, and fast, efficient population evolution[10].

References

[1] R. Salvador, A. Otero, J. Mora, E. de la Torre, L. Sekanina, and T. Riesgo. Fault tolerance analysis and self-healing strategy of autonomous, evolvable hardware systems. *In Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, pages 164-169, 2011.

[2] J. Lohn, G. Larchev, and R. Demara. Evolutionary fault recovery in a virtex fpga using a representation that incorporates routing. *In Parallel and Distributed Processing Symposium*, 2003. Proceedings. International, pages 8 pp.-, 2003.

[3] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb. Fpga partial reconfiguration via configuration scrubbing. *In Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 99-104, 2009.

[4] S. Harding, J.F. Miller, and W. Banzhaf. Self modifying cartesian genetic programming: Parity. *In Evolutionary Computation, 2009. CEC 09. IEEE Congress on*, pages 285-292, 2009.

[5] L. L. Goh. Novel fault localization approach for atpg / scan- fault failures in complex sub-nano fpga/ asic debugging. *In Physical and Failure Analysis of Integrated Circuits (IPFA), 2010 17th IEEE International Symposium on the*, pages 1-4, July 2010.

[6] Renovell, M.; Portal, J.M.; Figueras, J.; Zorian, Y., Testing the interconnect of RAM-based FPGAs, *Design and Test of Computers*, IEEE , vol.15, no.1, pp.45,50, Jan-Mar 1998.

[7] L. Zhao, D. M. H. Walker, and F. Lombardi, Bridging Fault Detection in FPGA interconnects using IDDQ, *International Symp. on Field Programmable Gate Arrays*, 1998, pp. 95-104.

[8] M. Renovell, J. M. Portal, J. Figueras, and Y. Zorian, SRAM-based FPGAs: Testing the LUT/RAM Modules, *Proc. International Test Conference*, 1998, pp. 1102-1111.

[9] M. Abramovici, C. E. Stroud, BIST-Based Test and Diagnosis of FPGA Logic Blocks, *IEEE Trans. on VLSI Systems*, 2001, pp. 159-172

[10] Miller, J.F.; Smith, S.L., Redundancy and computational efficiency in Cartesian genetic programming, *Evolutionary Computation, IEEE Transactions on*, vol.10, no.2, pp.167,174, April 2006