# Catastrophic Fault Recovery with Self-Reconfigurable Chips

**Will Hua Zheng, Neville I Marzwell, Savio N Chau**
Division 34 Autonomous Systems
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr, Pasadena, California 91109, United States of America
Phone: 1-818-354-2974, Fax: 1-818-393-4494, E-mail: Hua.Zheng@jpl.nasa.gov

**Abstract – Mission critical systems typically employ multi-string redundancy to cope with possible hardware failure. Such systems are only as fault tolerant as there are many redundant strings. Once a particular critical component exhausts its redundant spares, the multi-string architecture cannot tolerate any further hardware failure. This paper aims at addressing such catastrophic faults through the use of "Self-Reconfigurable Chips" as a last resort effort to "repair" a faulty critical component.**

**Keywords:** Reconfiguration, FPGA, fault-recovery, fault-tolerance, Self-reconfiguration

## I. INTRODUCTION

Future space missions will take us to destinations that are so far away that even the fastest spacecrafts would have to take decades to reach. Now imagine one such spacecraft that has traveled for twenty years and is near its final destination. During the twenty-year journey, the spacecraft's avionics have suffered multiple faults, and the spacecraft has used up its redundant resources. And it is at this critical moment comes the final blow – a catastrophic failure that is to disable the spacecraft. But almost miraculously the spacecraft recovers despite the lack of resources and continues on, thanks to a last-resort effort built into the critical components of the spacecraft avionics – chips that can reconfigure themselves, using a technique we call "Chip Salvaging".

The technologies needed to realize the good ending of the story above are not far from maturation. This paper will focus on the technology that saves the spacecraft at the end – Self-Reconfigurable Chips (SRC).

## II. BACKGROUND

Space is an environment with high levels of ionizing radiation, which is particularly harmful to the microelectronics commonly found in modern spacecraft avionics. Space-certified microelectronics with high levels of tolerance of radiation are typically of lower performance and cost much more than Commercial Off-The Shelf (COTS) counterparts. [Thi, 03] Even space-certified microelectronics is not completely immune from ionizing radiation. Hardware faults may occur as a result of Single Event Effects (SEEs) induced by ionizing radiation. These faults could be memory failure, microprocessor computation failure, etc. These faults, temporary or permanent, could render the spacecraft difficult or impossible to control.

System level redundancy is usually applied to mitigate the faults in the critical components of the avionics. Spacecrafts with long missions or going to places with high amount of ionizing radiation typically carry one or more "copies" of critical components of its avionics. These redundant copies can carry on certain predefined functions should its primary copy fail during the mission. However, such redundancy usually leave large amount of redundant resources unutilized, thus waste valuable space and weight on the spacecraft.

The SRC described in the beginning of the paper approaches the problem from a different level. The SRC carries certain amount of redundancy in itself. Therefore it can cope with SEEs better. When fault is detected through traditional means of cross-strapping, the faulty chip could then repair itself while other chips continue to function in the system. When repair is finished, normal operation can resume.

An SRC is a logic device with the ability to reconfigure its own logic at run-time. Currently, the best candidate for this job is the Field Programmable Gate Array (FPGA) chips, which first appeared in the 1980s as an evolution of Complex Programmable Logical Devices (CPLDs). These chips have come a long way in terms of size and speed and are slowly making their way into spacecraft avionics (Example: JPL's Multi-Mission System Architecture Platform).

While modern FPGAs typically fall into different categories: SRAM, fuse, antifuse, EPROM, EEPROM, and flash, we are most interested in dealing with the SRAM type because of its true field reprogrammability. SRAM-based FPGAs are organized as matrices of Configurable Logic Blocks (CLBs) with reconfigurable interconnects (routing resources). CLBs can be configured as any kind of logic device including adders and multipliers, other blocks may include reconfigurable I/O banks. Because the CLBs are physically identical prior to run-time, logic implemented in certain CLBs could be relocated to others.

## III. SYSTEM-ON-CHIP AND SRC

The current trend in electronics design is that entire systems are built into FPGAs that incorporate everything from microprocessors, memory, high speed I/O, to bus controllers. Some designs even incorporate multiple full-fledged microprocessors in a single FPGA chip. There is a special property that is associated with such System-on-a-Chips (SoCs) implemented in FPGAs – that most aspects of the system are realized in reconfigurable logic, including internal wiring. This special property enables the possibility of reusing the same chip for different purposes after deployment.

Reusing reconfigurable chips in a system of SoCs has many advantages and applications. One of the most obvious applications, as described in the beginning, is that SoCs implemented in FPGAs that have failed partially can be reconfigured by either the chip itself or an off-chip controller to avoid the damaged regions and continue original operations, or may sacrifice certain non-critical functions to continue operations at reduced performance. Second, a system of SoCs may have most parts of the system implemented with identical reconfigurable SoCs and therefore increase survivability through increased redundancy (each identical reconfigurable SoC could be reused to become redundant backup for others). [ZHE, 04] Additional applications include using reconfigurable FPGAs as a platform to support demand-paged hardware, allowing a small chip to perform large functions sequentially.

## IV. GAINING ACCESS TO FPGA CONFIGURATION

Traditional static FPGA designs are stored in an on-board storage and configured prior to run-time through a proprietary serial interface or a JTAG interface. On most board designs, these interfaces are typically routed outside of the FPGA to provide the configuration interfaces to the user.
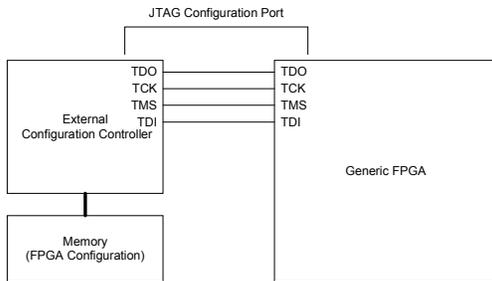
Fig 1. Traditional FPGA Configuration Access

In order for an SRC to reconfigure itself, the configuration interface must also be accessible from within the FPGA.

The Internal Configuration Controller is essentially an internal implementation of the external configuration controller, accessible by internal logic. In figure 2, the Internal Configuration Controller gives FPGA configuration access to a processor core, which also has access to the original FPGA configuration in memory. Such a processor would allow users to write software to dynamically reconfigure the FPGA during run-time.
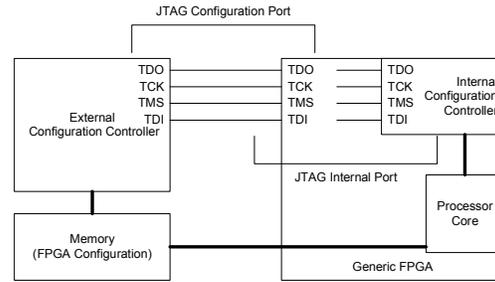
Fig 2. SRC FPGA Configuration Access

Recent FPGAs have simplified the matter, using proprietary configuration solutions to give internal access to FPGA configuration, such as the Internal Configuration Access Port implemented as a primitive in Xilinx Virtex II FPGAs [ECK, 04][BLO, 04]. An FPGA design with an ICAP interface is included in Xilinx Application Note 661 [HUA, 04], which could be used as a reference when designing embedded configuration interface.

## V. GENERATING RECONFIGURATION DATA USING TRADITIONAL VENDOR TOOLS

Gaining access to an FPGA's configuration interface is relatively simple compared to the process of generating reconfiguration data. Our previous paper, *In-System Partial Run-Time Reconfiguration for Fault Recovery Applications on Spacecrafts* [ZHE, 05] discussed how to generate such reconfiguration data using vendor tools. We conducted research based on the method proposed in the above paper and have met some partial success.

Using a technique [XIL, 04] that isolated reconfigurable logic from static logic, two copies of the same configuration are generated, each of them having the reconfigurable logic located in different regions in the FPGA (see fig. 3).

The figure shows two configuration implementations of the same logic in a Xilinx Virtex II Pro FPGA. The Xilinx vendor tool comes with a configuration generator that is able to read in two configurations and generate a new configuration file based on the difference between the two. Therefore, the difference configuration could be fed to the ICAP interface discussed earlier to effectively

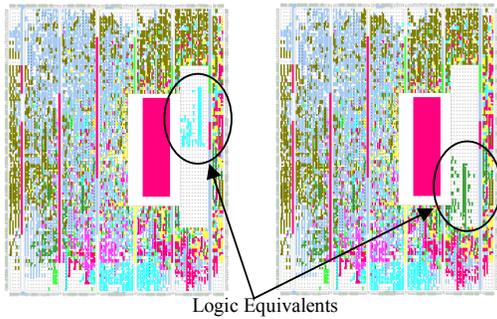reconfigure the FPGA during run-time.



Logic Equivalents

Fig 3. Same FPGA Logic Placed at Different Regions

The reconfigured logic is a stepping motor controller that is hooked up to a motor (via a TTL voltage shifter IC) to demonstrate that reconfigured logic can resume previous operation. (See fig 4.)
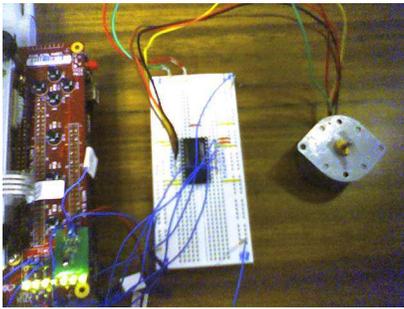


Fig 4. Stepping Motor and TTL Driver

The research team also wrote software to control the reconfiguration process and read the reconfiguration data from on-board memory to the ICAP interface. To show the effects of the reconfiguration, a graphical interface was implemented that shows the contents of the FPGA before and after the configuration. (See fig 5.)



Fig 5. Entire Setup Showing GUI

This reconfiguration process confirmed our hypothesis that the FPGA can be reconfigured during run-time safely. The FPGA continued operation despite the configuration change while it was running – we made sure that the reconfiguration process did not interfere with existing logic. Since the reconfiguration process involved re-routing of signal lines, the I/O pads used by the reconfigured logic were also re-routed to the newly configured logic.

However, this reconfiguration process has many limitations. The target FPGA can only be reconfigured once. The reconfiguration data must be generated at design time.

From the above research, we have come to realize that vendor tools are not yet well-adapted to generate FPGA reconfiguration data effectively. New methods must be developed to fulfill this task.

## VI. PROPOSED METHODOLOGY FOR GENERATING RECONFIGURATION DATA

Simply put, moving the configuration generating process to the FPGA will allow the FPGA chip to reconfigure itself. However, the configuration generating process typically take hours even on high-end workstations, it is unreasonable to embed it into the SRC as-is. We propose selectively embed parts of this process into the SRC.

The configuration generating process typically involves three major steps: (1) Synthesis, which takes human developed hardware description (logic) and produces an intermediate format called Netlist for further processing; (2) Mapping, which maps Netlist logic to primitive parts of an FPGA; (3) Place-and-routing, which takes mapped Netlists, and implement them as real hardware in FPGAs.

The synthesis and mapping processes do not directly deal with the layout of the FPGA configuration. Therefore these two processes can be left to be done during the design stage on a workstation. On the other hand, the place-and-routing process directly deals with how the configuration is laid out. It is therefore our prime candidate for embedding into the SRC. This process itself may also be done during design-stage to minimize the amount of time needed to reconfigure a chip. Only the portion of the chip that needs to be reconfigured must be re-place-and-routed during run-time, and this process can take as little as minutes.

## VII. THE CHALLENGE

Embedding the place-and-routing process into the SRC dictates that the processor in the SRC is able to run place-and-routing software of some sort. The main challenge is that such software is unavailable from FPGA vendors themselves.

Some alternatives exist today. The JHDL tool suite [BEL, 98] can be used to a certain extent to effectively reconfigure FPGAs during run-time. A tool developed using the JHDL tools suite to conduct run-time reconfiguration of FPGAs is well described in *Radiation Mitigation and Power Optimization Design Tools for Reconfigurable Hardware in Orbit* [FRE, 05]. The paper describes a software "RHinO" (Reconfigurable Hardware in Orbit), which uses Commercial-Off-The-Shelf (COTS) tools and JHDL to map hardware design logic into vendor specific configuration as well as JHDL-based configuration. The latter can be used to reconfigure the FPGA at run-time. One major disadvantage of JHDL is that it requires a Java run-time environment, which may not be available for all embedded processors.

The ideal method to generate reconfiguration data is still software written for embedded processors to run natively. However, the place-and-routing process is a complicated one and FPGA vendors do not supply the necessary chip layout databases needed for this process to the public. Some research efforts have gone into developing these databases in-house, by reverse-engineering Xilinx database files shipped with design software. [STE, 02] These efforts currently only cover a small selection of FPGAs and require extra efforts for every new FPGA. Therefore, such software efforts must involve the FPGA vendors themselves.

## VIII. BENEFITS TO SPACECRAFT DESIGN

Technologies of SRC can be used to increase the fault tolerance of a spacecraft. The last-resort scenario described in the beginning of this paper utilizes an SRC to recover itself by avoiding damaged regions. SRCs can also be reconfigured to replace other similar SRCs.

Consider the following simple spacecraft avionics architecture (fig. 6.). There are two controller boards used by the main processor board. All three boards implement the same generic SRC with different configuration to perform different functions. Apparently the architecture employs no redundancy. However, the nature of SRC gives this architecture some redundancy by design. The system can tolerate a fault in either the telemetry or the science instrument controller. If either controller fails, the other could reconfigure itself to fulfill the role of the other.

Take telemetry controller for example. If the telemetry controller fails, the science instrument controller could collect science data into memory, reconfigure itself as the telemetry controller, upload science data, and reconfigure as science instrument controller. The system could repeat the process as many times as needed to complete the mission.

In addition, the system can recover a fault in any of the

boards with only one redundant controller, achieving the same fault tolerance of a traditional dual-string redundant system with three redundant copies.
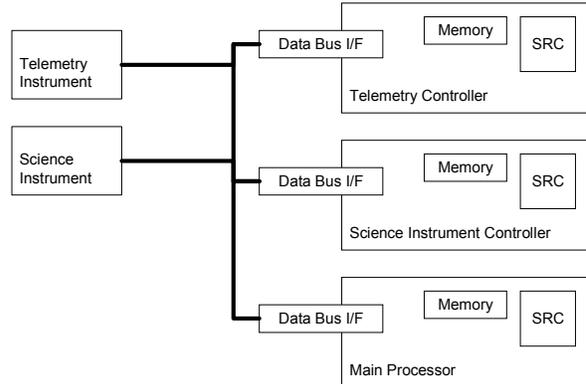


Fig 6. Example SRC-based Spacecraft Architecture

## IX. OUR VISION

The goal of this research had always been the development of a modular reconfigurable spacecraft avionics design with increased survivability, minimal redundancy and mass, and the ability to endure long missions. The development of SRC is a part of this vision. We envision an avionics system of identical reconfigurable subsystems. Each of such a subsystem can be configured to perform a unique function or act as a redundant backup for other subsystems.

This vision would require the maturation of the methodologies of reconfiguring an FPGA during run-time. There are two possibilities ahead of us. One is to work with FPGA vendors such Xilinx and Altera, or other research institutions to develop embedded place-and-routing software jointly and move at least part of the place-and-routing process onboard. A second possibility is to develop FPGAs that are built for reconfiguration in-house. This of course would require much more resources than the first proposal, as FPGA design and production requires much resource. At the present time, working with FPGA vendors to come up with the necessary tools and methodologies for run-time reconfiguration of FPGAs is still the most viable solution.

The I/O redirection problem of these reconfigurable subsystems is also being looked at Jet Propulsion Laboratory. Project MRHE (Modular Reconfigurable High Energy) calls for a spacecraft with wireless connectivity among its avionics subsystems. Follow-on studies on wireless avionics I/O interfaces after the cancellation of MRHE are in progress at the time of writing.

## X. CONCLUSION

Self-Reconfigurable Chips give spacecraft better fault tolerance by design, without conflicting with the traditional multi-string redundancy approach. In fact, if implemented correctly, SRCs can multiply the effectiveness of multi-string redundancy. One weakness of SRCs is that they do not address the possible difference in component interfaces. However, this team has consistently associated the success of reconfigurable chips with the adoption of wireless technology for spacecraft. [ZHE, 04] With that said, SRCs will greatly improve the survivability of future spacecraft avionics.

## ACKNOWLEDGEMENTS

## REFERENCES

[Thi, 03] Chad Thibodeau, An Integrated Approach with COTS Creates Rad-Tolerant Single Board Computer for Space, 2003 [COTS Journal, December 2003]

[BEL, 98] Peter Bellows, Brad Hutchings, JHDL - An HDL for Reconfigurable Systems, 1998

[STE, 02] Neil Joseph Steiner, A Standalone Wire Database for Routing and Tracing in Xilinx Virtex, Virtex-E, and Virtex-II FPGAs

[BLO, 04] Brandon Blodget, Philip James-Roxby, Eric Keller, Scott McMillan, Prasanna Sundararajan, A self-reconfiguring platform, 2004, pp. 4.

[ECK, 04] Vince Eck, Punit Kalra, Rick LeBlanc, Jim McManus, In-Circuit Partial Reconfiguration of RocketIO Attributes, Xilinx Application Note 622, 2004

[HUA, 04] Dai Huang, Michael Matera, RocketIO Transceiver Bit-Error Rate Tester, Xilinx Application Note 621, 2004

[XIL, 04] Xilinx Corp., Two Flows for Partial Reconfiguration: Module Based or Difference Based, Xilinx Application Note 290, pp13-18

[ZHE, 04] Will Zheng, Savio Chau, Neville Marzwell, Run-Time Reconfigurable System over Wireless Network , 2004 [The Space Technology and Applications International Forum, 2005]

[FRE, 05] Matthew French, Paul Graham, Michael Wirthlin, Li Wang, Gregory Larchev, Radiation Mitigation and Power Optimization Design Tools for Reconfigurable Hardware in Orbit, 2005

[ZHE, 05] Will Zheng, Neville Marzwell, Savio Chau, In-System Partial Run-Time Reconfiguration for Fault Recovery Applications on Spacecrafts, 2005 [IEEE International Conference on Systems, Man, and Cybernetics, 2005]