UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LEONARDO PEREIRA SANTOS

# Cost-Effective Dynamic Repair for FPGAs in Real-Time Systems

Dissertation presented in partial fulfillment of the requirements for the degree of Master in Computer Science

Advisor: Prof. Dr. Luigi Carro
Co-advisor: Prof. Dr. Gabriel Luca Nazar

Porto Alegre, February 2016

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Santos, Leonardo Pereira

Cost-Effective Dynamic Repair for FPGAs in Real-Time Systems / Leonardo Pereira Santos. – 2016.

87 f.:il.

Orientador: Luigi Carro; Co-orientador: Gabriel Luca Nazar.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2016.

1. FPGA. 2. Detecção de erro 3. Probabilidade de reparo. I. Carro, Luigi. II. Nazar, Gabriel Luca. III. Título.

## AGRADECIMENTOS

Agradeço à minha esposa Rosane pelo apoio, paciência e compreensão nestes anos de trabalho. Agradeço também aos meus filhos, Antônio e Bianca, que por tantas vezes queriam brincar e o pai estava no computador fazendo o “tema de casa”, demorou um pouco, mas ficou pronto! Esta conquista só foi possível com o amor e força de vocês.

**Reparo Dinâmico de Baixo Custo para FPGAs em Sistemas Tempo-Real**

## RESUMO

*Field-Programmable Gate Arrays* (FPGAs) são largamente utilizadas em sistemas digitais por características como flexibilidade, baixo custo e alta densidade. Estas características advém do uso de células de SRAM na memória de configuração, o que torna estes dispositivos suscetíveis a erros induzidos por radiação, tais como SEUs. TMR é o método de mitigação mais utilizado, no entanto, possui um elevado custo tanto em área como em energia, restringindo seu uso em aplicações de baixo custo e/ou baixo consumo. Como alternativa a TMR, propõe-se utilizar DMR associado a um mecanismo de reparo da memória de configuração da FPGA chamado *scrubbing*. O reparo de FPGAs em sistemas em tempo real apresenta desafios específicos. Além da garantia da computação correta dos dados, esta computação deve se dar completamente dentro do tempo disponível (*time-slot*), devendo ser finalizada antes do tempo limite (*deadline*). A diferença entre o tempo de computação dos dados e a *deadline* é chamado de *slack* e é o tempo disponível para reparo do sistema.

Este trabalho faz uso de *scrubbing* deslocado dinâmico, que busca maximizar a probabilidade de reparo da memória de configuração de FPGAs dentro do *slack* disponível, baseado em um diagnóstico do erro. O *scrubbing* deslocado já foi utilizado com técnicas de diagnóstico de grão fino (NAZAR, 2015). Este trabalho propõe o uso de técnicas de diagnóstico de grão grosso para o scrubbing deslocado, evitando as penalidades de desempenho e custos em área associados a técnicas de grão fino.

Circuitos do conjunto MCNC foram protegidos com as técnicas propostas e submetidos a seções de injeção de erros (NAZAR; CARRO, 2012a). Os dados obtidos foram analisados e foram calculadas as melhores posição iniciais do *scrubbing* para cada um dos circuitos. Calculou-se a taxa de *Failure-in-Time* (FIT) para comparação entre as diferentes técnicas de diagnóstico propostas. Os resultados obtidos confirmaram a hipótese inicial deste trabalho que a redução do número de bits sensíveis e uma baixa degradação do período do ciclo de relógio permitiram reduzir a taxa de FIT quando comparadas com técnicas de grão fino. Por fim, uma comparação entre as três técnicas propostas é feita, analisando o desempenho e custos em área associados a cada uma.

**Palavras-chave**: *Field-Programmable Gate Array* (FPGA), diagnóstico de falhas, tolerância a falhas, reconfiguração parcial, tempo real.

**Cost-Effective Dynamic Repair for FPGAs in Real-Time Systems**

## ABSTRACT

*Field-Programmable Gate Arrays* (FPGAs) are widely used in digital systems due to characteristics such as flexibility, low cost and high density. These characteristics are due to the use of SRAM memory cells in the configuration memory, which make these devices susceptible to radiation-induced errors, such as SEUs. TMR is the most used mitigation technique, but it has an elevated cost both in area as well as in energy, restricting its use in low cost/low energy applications. As an alternative to TMR, we propose the use of DMR associated with a repair mechanism of the FPGA configuration memory called scrubbing. The repair of FPGA in real-time systems present a specific set of challenges. Besides guaranteeing the correct computation of data, this computation must be completely carried out within the available time (*time-slot*), being finalized before a time limit (*deadline*). The difference between the computation time and the deadline is called the *slack* and is the time available to repair the system.

This work uses a dynamic *shifted* scrubbing that aims to maximize the repair probability of the configuration memory of the FPGA within the available slack based on error diagnostic. The shifted scrubbing was already proposed with fine-grained diagnostic techniques (NAZAR, 2015). This work proposes the use of *coarse-grained* diagnostic technique as a way to avoid the performance penalties and area costs associated to fine-grained techniques.

Circuits of the MCNC suite were protected by the proposed techniques and subject to error-injection campaigns (NAZAR; CARRO, 2012a). The obtained data was analyzed and the best scrubbing starting positions for each circuit were calculated. The *Failure-in-Time* (FIT) rates were calculated to compare the different proposed diagnostic techniques. The obtained results validated the initial hypothesis of this work that the reduction of the number of sensitive bits and a low degradation of the clock cycle allowed a reduced FIT rate when compared with fine-grained diagnostic techniques. Finally, a comparison is made between the proposed techniques, considering performance and area costs associated to each one.

**Keywords**: Field-programmable gate arrays (FPGA), scrubbing, fault diagnosis, fault tolerance, real-time.

# LIST OF FIGURES

# LIST OF TABLES

## LIST OF ABBREVIATONS AND ACRONYMS

| | |
|---|---|
| ALU | *Arithmetic Logic Unit* |
| ASIC | *Application-Specific Integrated Circuits* |
| ASIP | *Application-specific instruction set processors* |
| AUT | *Area Under Test* |
| CED | *Concurrent Error Detection* |
| CG-DMR | *Coarse-grained Double-Module Redundancy* |
| CG-TMR | *Coarse-grained Triple-Module Redundancy* |
| CLB | *Configurable Logic Block* |
| CLB | *Configurable Logic blocks* |
| CRC | *Cyclic Redundancy Check* |
| CUT | *Circuit Under Test* |
| DMR | *Double-Module Redundancy* |
| DSP | *Digital Signal Processing* |
| ECC | *Error Correcting Code* |
| FG-DMR | *Fine-grained Double-Module Redundancy* |
| FG-TMR | *Fine-grained Triple-Module Redundancy* |
| FIT | *Failure-in-Time* |
| FPGA | *Field-Programmable Gate Arrays* |
| FSM | *Finite State Machine* |
| GP-CPU | *General Purpose Central Processing Unit* |
| HDL | *Hardware Description Language* |
| I/O | *Input/Output* |
| IC | *Integrated Circuit* |
| LUT | *Look-Up Table* |
| MAC | *Multiplier/Accumulator* |
| MTTR | *Mean Time To Repair* |
| PAR | *Place And Route* |
| PO | *Primary Output* |
| PR | *Partial Reconfiguration* |
| PST | *Perfect Signature Translator* |
| RHST | *Real-time Heuristic Signature Translator* |

| | |
|---|---|
| SET | *Single-Event Transient* |
| SEU | *Single-Event Upset* |
| SG-DMR | *Selective-grain Double-Module Redundancy* |
| SRAM | *Static Random Access Memory* |
| ST | *Signature Translator* |
| TMR | *Triple-Module Redundancy* |
| ALU | *Arithmetic Logic Unit* |
| ASIC | *Application-Specific Integrated Circuits* |
| ASIP | *Application-specific instruction set processors* |
| CED | *Concurrent Error Detection* |
| CG-DMR | *Coarse-grained Double-Module Redundancy* |
| CG-TMR | *Coarse-grained Triple-Module Redundancy* |
| CLB | *Configurable Logic Block* |
| CLB | *Configurable Logic blocks* |
| CRC | *Cyclic Redundancy Check* |
| CUT | *Circuit Under Test* |
| DMR | *Double-Module Redundancy* |
| DSP | *Digital Signal Processing* |
| ECC | *Error Correcting Code* |
| FG-DMR | *Fine-grained Double-Module Redundancy* |
| FG-TMR | *Fine-grained Triple-Module Redundancy* |
| FIT | *Failure-in-Time* |
| FPGA | *Field-Programmable Gate Arrays* |
| FSM | *Finite State Machine* |
| GP-CPU | *General Purpose Central Processing Unit* |
| I/O | *Input/Output* |
| IC | *Integrated Circuit* |
| LFSR | *Linear Feedback Shift Register* |
| LUT | *Look-Up Table* |
| MAC | *Multiplier/Accumulator* |
| MTTR | *Mean Time To Repair* |
| PAR | *Place And Route* |
| PO | *Primary Output* |

| | |
|---|---|
| PR | *Partial Reconfiguration* |
| PST | *Perfect Signature Translator* |
| RAM | *Random Access Memory* |
| RHST | *Real-time Heuristic Signature Translator* |
| SET | *Single-Event Transient* |
| SEU | *Single-Event Upset* |
| SG-DMR | *Selective-grain Double-Module Redundancy* |
| SRAM | *Static Random Access Memory* |
| ST | *Signature Translator* |
| TMR | *Triple-Module Redundancy* |

## TABLE OF CONTENTS

# 1 INTRODUCTION

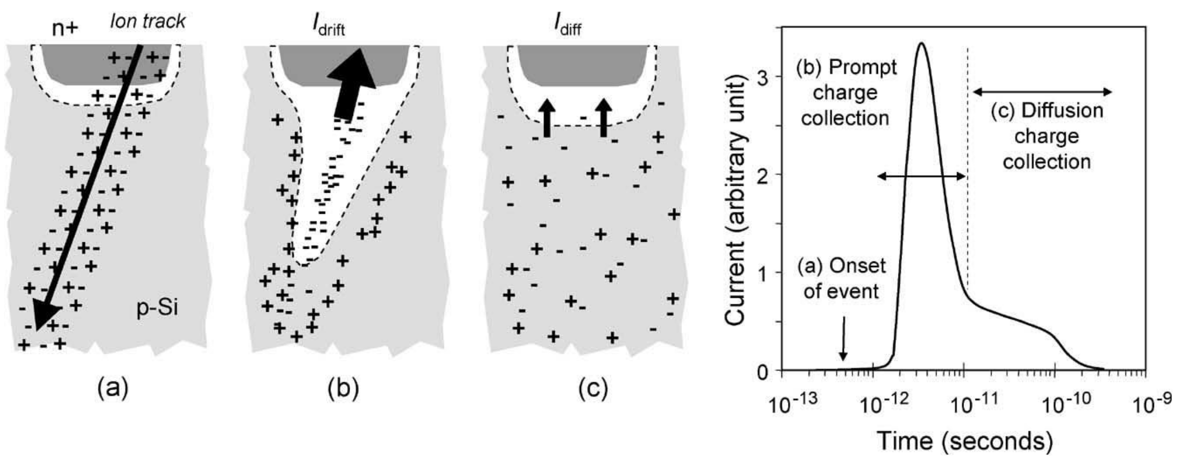
## 1.1 Radiation Effects on Semiconductors

Radiation effects pose a major reliability risk for digital circuits. Two main sources of radiation are of importance due to their effects on electronics, neutrons and alpha particles. Neutrons originate from the constant bombardment of the atmosphere of the Earth by high-energy particles from stellar and galactic origin. These high-energy particles collide with the atoms of the atmosphere and this collision creates neutrons, in particle shower effect (ALTERA CORPORATION, 2013). The highest altitude these collisions occur, the more energetic the produced neutrons are (NORMAND; BAKER, 1993; TABER; NORMAND, 1993). Another source of radiation that affects semiconductors are alpha particles from the packaging of the devices (MAY; WOODS, 1978), by trace contamination by uranium and thorium (BAUMANN, 2005).

The collision of neutrons on semiconductors has primary and secondary effects. A primary effect is the displacement of silicon atoms from the crystalline lattice, causing displacement damage. The displaced silicon atoms in turn might cause other displacements, until the energy of the colliding particles is less than a minimum of 25 eV (SROUR, 1982). This displacement causes damage and ionization. The secondary effects is the interaction of colliding neutrons on dopant elements, such as boron (BAUMANN, 2005). This causes the formation of other elements and alpha particles.

Alpha particles and other ionized ions have the primary effect of creating a funnel-shaped ionization path. The ionized particles will then drift and be diffused (BAUMANN, 2005). If the struck point is a reverse-biased junction, this strike might switch on a transistor. This effect is illustrated in Figure 1.1:

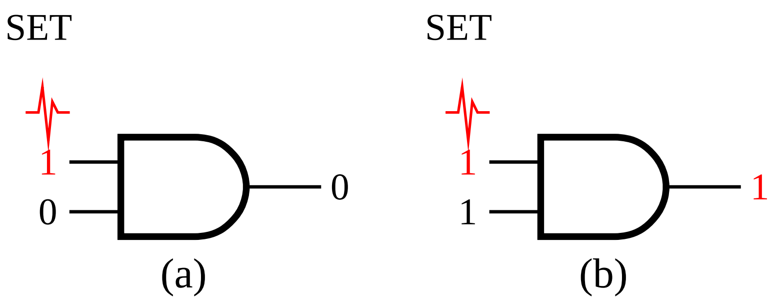
Figure 1.1 - Charge generation and collection phases in a reverse-biased junction and the resultant current pulse caused by the passage of a high-energy ion.



Source: Baumann (2005, p. 307)

This ionization effect might cause a current surge with enough intensity to invert the logic level of a digital port. If this happens, it is called a Single-Event Transient (SET) (GAILLARD, 2011). This SET might be masked by the circuit’s logic or not. Figure 1.2 shows two SETs, in (a) the SET was masked by the AND port, while in (b) it was not and propagated to the output of the AND port:

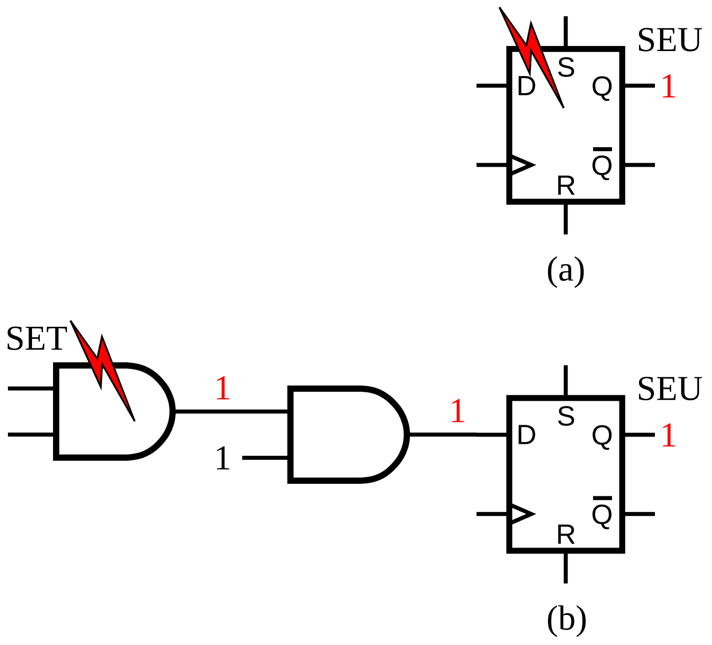
Figure 1.2 - (a) masked SET and (b) non-masked SET.



Source: author

A Single-Event Upset (SEU) occurs when a memory element, such as a flip-flop, is upset by radiation or when a SET is capture by a flip-flop, as illustrated in Figure 1.3:

Figure 1.3 - (a) SEU from memory upset, (b) SEU from captured SET



Source: author

If the strike does not cause permanent damage, it is a non-permanent fault and simply resetting the system will correct it. This type of error is called a *soft-error*. Soft-errors affect equally General Purpose CPUs (GP-CPUs), Application-Specific Integrated Circuits (ASICs) or Application-Specific Instruction set Processors (ASIPs).

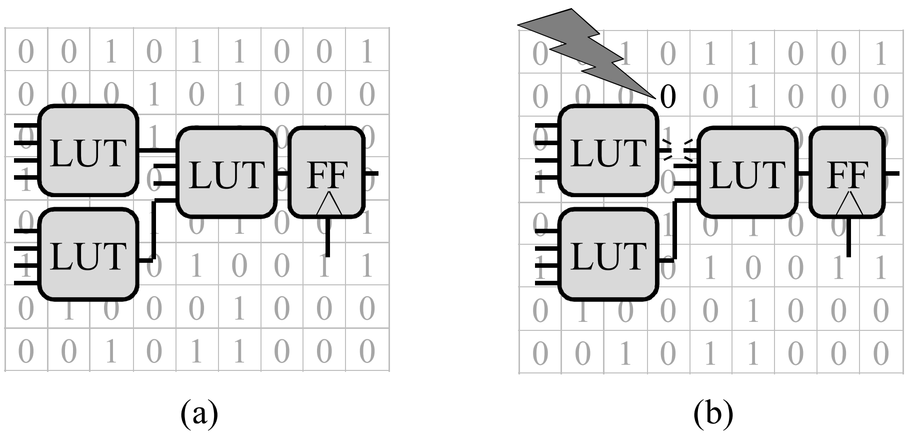
Earth’s atmosphere has a filter effect to radiation that helps protect circuits at ground level or low altitudes (NORMAND; BAKER, 1993; TABER; NORMAND, 1993). Because of this, radiation-induced effects in terrestrial applications were not a cause of concern. This scenario is fast changing, as the shrinkage of the components in Integrated Circuits (ICs) makes these circuits more susceptible to radiation-induced errors (BAUMANN, 2005), offsetting the filtering effect of the atmosphere. It is important even to terrestrial applications to consider radiation effects.

## 1.2 Radiation Effects on FPGAs

FPGAs have found use in critical systems (ALTERA CORPORATION, 2015; XILINX INC., 2015a), many of them real-time (BATLLE, 2002; KARIMI et al., 2008; UZUN; AMIRA; BOURIDANE, 2005), thus it is very important to create fault tolerance solutions for FPGAs. Some FPGA devices are built to be immune or resistant to radiation effects. These devices typically use anti-fuse or flash memory (ATMEL CORPORATION, 2015a; MICROSEMI CORPORATION, 2015a) to store the configuration data. While these devices allow for grater dependability, Random Access Memory (RAM) is inherently denser than anti-fuse or flash memory. While a ATF280, a hardened FPGA manufactured by Atmel, has 14,440 LUTs (ATMEL CORPORATION, 2015b) and a RT4G150, a hardened FPGA manufactured by Microsemi, has 151,824 LUTs (MICROSEMI CORPORATION, 2015b); a Xilinx Virtex VU13P has 1,635,840 LUTs (XILINX INC., 2015b), over 10x the density in logic blocks than hardened technologies, making the use of SRAM (Static RAM) devices very attractive to system designers.

SRAM FPGAs have a very distinct failure model from GP-CPUs, ASICs or ASIPs. In the case of FPGAs, all effects that affect general ICs or GP-CPUs also affect FPGAs, but if the FPGA is SRAM-based, then an SEU might occur in the FPGA’s configuration memory (Figure 1.4). Due to the FPGA’s reconfigurable substrate, this fault might cause a permanent change in the device’s functionality, thus making this error permanent until corrected. Any dependability solution involving FPGAs must mitigate permanent faults (SEXTON, 2003), radiation accumulated effects (BARNABY, 2006), transitory faults and configuration memory faults (CARMICHAEL; CAFFREY; SALAZAR, 2000), (LIMA et al., 2001), (REORDA; STERPONE; ULLAH, 2013), (NAZAR; SANTOS; CARRO, 2015).

Figure 1.4 - Fault-free circuit and its associated configuration bits (a) and faulty circuit due to a configuration upset (b)



Source: Nazar (2013, p. 12)

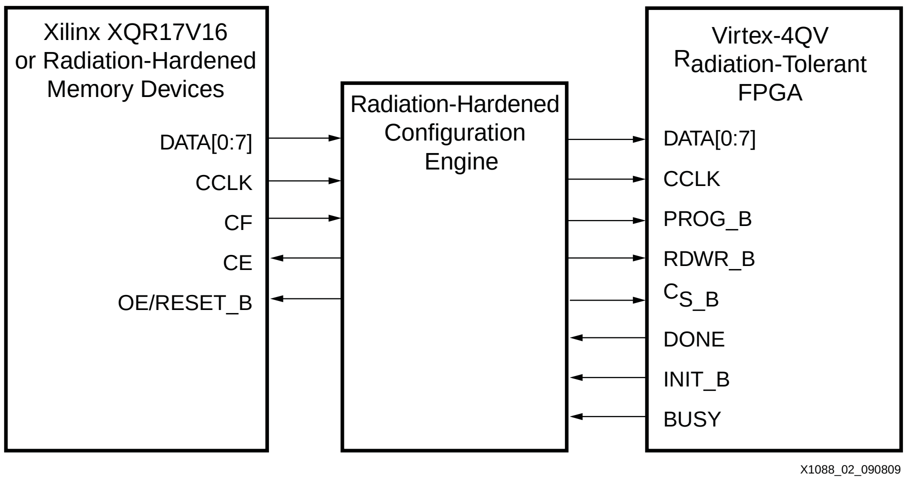
Real-time systems present a distinct failure model, in which not only computations must remain correct, but also deadlines must be met. This means that a fault-tolerant real-time system must be prepared to meet deadlines even in the presence of failures. If the system is implemented using SRAM-FPGAs, the failure model of the system has the particularities from the failure model of FPGAs and the particularities from the failure model of real-time systems.

## 1.3 Repair of Soft-Errors on SRAM FPGAs

As explained in the previous section, SEUs can cause errors in the configuration memory of the FPGA. As these errors cause the change of the contents of SRAM cells, they are not permanent, being denominated *soft*-errors, as opposed to *hard*-errors which are caused by permanent damage to the device. In an SRAM FPGA, most RAM cells are used to store the configuration of the device, so most SEU-induced errors will occur on the configuration memory. Errors in the configuration memory can change the functionality of the device or cause routing errors, opening or shorting connections. Soft-errors in the user design can be mitigated by classic techniques, such as simply resetting the system or using a rollback mechanism. Soft-errors on the configuration memory require other mitigation techniques, such as re-writing the configuration memory in a process called *scrubbing*.

Different mechanisms can be used to detect errors in the configuration memory, such as configuration readback or storing a CRC table for the configuration frames. Once detected, an error is corrected by re-writing the faulty configuration frame using partial reconfiguration. A much simpler approach is to simply re-write each configuration frame. This approach is called *scrubbing*. Scrubbing uses the mechanism of partial reconfiguration to periodically re-write the configuration memory of the FPGA while the device is operating, removing accumulated errors caused by SEUs (CARMICHAEL; CAFFREY; SALAZAR, 2000). The original bitstream and scrubbing controller can be implemented in radiation-hardened devices, as shown in Figure 1.5, in which a Virtex-4QV is scrubbed by radiation-hardened devices:

Figure 1.5 - Overview of an External Device Hosting Configuration Manager for a Virtex-4QV Device



Source: Carmichael; Tseng (2009, p. 12)

## 1.4 Main Goals and Contributions

The objective of this dissertation is to improve on works in the literature (NAZAR, 2015; SARI; PSARAKIS; GIZOPOULOS, 2013), in that the repair of real-time systems is addressed from a hardware perspective. Specifically, this work will focus on the repair of *soft-errors* in the FPGA configuration memory when these devices are deployed in real-time systems. As will be explained in section 3.1.1, implementing a fault-tolerant real-time system can degrade the performance of the system, so in this work the focus was on non-intrusive diagnostic architectures as a way to avoid such degradation. Different diagnostic architectures are evaluated and compared regarding diagnostic precision and overall efficiency in the repair of FPGAs.

Scrubbing can be used to repair soft-errors on the configuration memory of the FPGA; in this dissertation the technique of *shifted* scrubbing will be used to achieve a better repair performance than standard scrubbing. A double module redundancy (DMR) architecture is used to detect errors and initiate the shifted scrubbing process, instead of risking that the error lingers on the system until the next scrubbing round. *Shifted* scrubbing also differs from *standard* scrubbing as in the latter the process begins with the first frame of the device or partition being repaired, whereas with *shifted* scrubbing the first frame is dynamically chosen according to the error that was detected and according with how much time is available to repair the device or partition.

As the diagnostic architectures studied in this dissertation are considered coarse-grained, the fault coverage is limited to faults that propagate to the POs. These coarse-grained architectures are not able to avoid that an incorrect internal signal be captured by a flip-flop. This is not considered a limitation of the coarse-grained diagnostic techniques, as the capture of incorrect signals can be avoided by using a granularity that the duplicated modules are placed just before the temporal barriers. In this dissertation all benchmarks are combinational circuits and the granularity used in chapters 5 and 6 is the whole benchmark circuit. As this work does not deal with sequential circuits, there is not state to be saved and rolled back in the case of a failure.

## 1.5 Outline

This dissertation is structured as follows. Chapter 2 describes related works in fault tolerance for FPGAs, partial reconfiguration techniques used in fault tolerance, works that use the concept of shifted scrubbing and works that deal with fault tolerance for real-time systems. Core concepts as the challenges when protecting real-time systems, the proposed architectures for diagnostic and circuit repair, the concept of error signature, the signature translator block and a compression heuristic for signatures are presented in chapter 3. Chapter 4 explains the reasoning behind the experimental results, describes the benchmark circuits, the procedure for experimentation and result analysis, and the measurements chosen to verify the proposed work in this dissertation. The different diagnostic topologies are presented in separate chapters, chapter 7 describes the Selective-Grained Double-Module Redundancy (SG-DMR) architecture; chapter 6 describes the Coarse-Grained Double-Module Redundancy (CG-DMR) with delta placement architecture and chapter 5 describes the Coarse-Grained Double-Module Redundancy with a free placement architecture. The experimental results for each architecture are presented in sections 5.2, 6.2 and 7.2, respectively. A critical analysis and comparison of the diagnostics architectures is carried out in chapter 8. In the same chapter, the obtained results are compared to results from other works (NAZAR, 2015). Conclusions drawn from the work carried out in this dissertation are presented in chapter 9, along with future research opportunities envisioned.

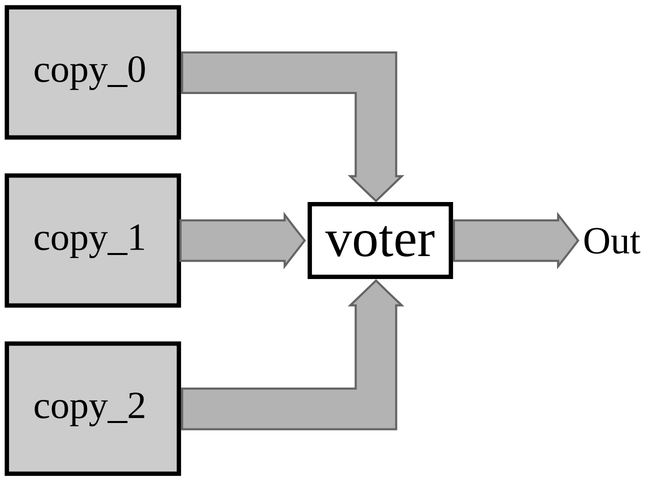
# 2 RELATED WORK

## 2.1 Fault Tolerance on FPGAs

The FPGA's reconfigurable substrate might be a cause of dependability issues in a radiated environment (KASTENSMIDT et al., 2004), but it also affords the creation of several fault mitigation techniques that would not be possible in ASICs or ASIPs.

Triple-Module Redundancy (TMR) (VON NEUMANN, 1956) is a general fault tolerance technique that can be used in FPGAs (ALTERA CORPORATION, 2013; XILINX INC., 2015c) to mask a single fault. It consists of three copies of the circuit to be protected with a majority voted output, illustrated by Figure 2.1:

Figure 2.1 - TMR-protected circuit



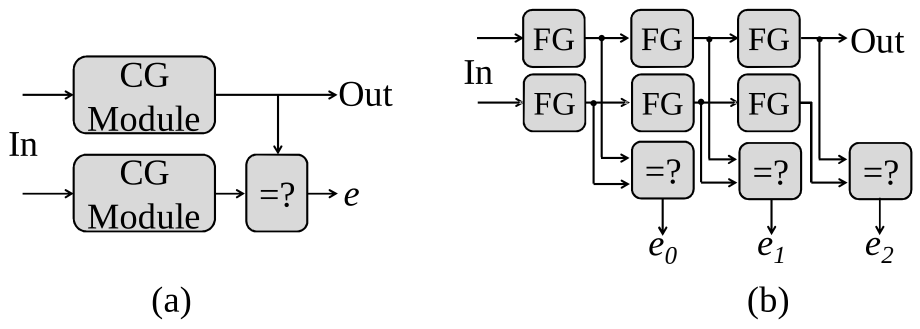
Source: author.

The work in Lima et al. (2001) analyzes different strategies to implement TMR in Xilinx Virtex, according to the nature of the structures used in the circuit, such as throughput logic, Finite State Machines (FSM), Input/Output (I/O) logic and proprietary features such as block RAMs. The authors discuss how SEUs might cause errors that are undetectable by configuration readback. In Virtex devices, logic constants are implemented through the I/O circuitry of unused pins. In the event of a SEU causing a momentary upset on the routing of such signals, the configuration will not be affected, but the error might be captured by sequential logic. Such an error does not manifest itself on the configuration memory, showing the importance of redundant circuits and the comparison of actual circuit elements as diagnosis tools. The work also considers the procedure of configuration scrubbing as a repair mechanism to SEU-induced errors in the configuration memory of the FPGA. Configuration scrubbing will be explained in greater detail in section 2.2. The paper then moves to fault tolerance analysis of the technique proposed earlier, first with 32-bit counters as benchmarks and a fault-injection tool to simulate SEUs, and later with an 8051 softcore processor as benchmark and radiation testing. The first analysis showed that TMR was not capable of protecting the benchmark for single-error faults in some cases, which is a counter-intuitive result. Thus is due to the SRAM FPGA's failure model, where an error caused an undesirable connection between a bit in one of the redundant 32-bit counters and a signal of a comparator, changing the voting result. The suggested solution is a structured floorplanning to avoid such routing errors. The radiation injection results show that TMR allied with scrubbing was able to reduce failure rates when compared with a circuit without scrubbing, at a high resource cost: 360 % in FFs, 300 % in block RAMs, 600 % in I/O pins and 367 % in LUTs. The paper does not compare the clock overhead incurred by the use of floorplanning.

Other work (KASTENSMIDT; KINZEL FILHO; CARRO, 2006) builds on the results obtained before (LIMA et al., 2001) to analyze in detail the causes or routing errors and propose redundant routing as a mitigation technique for such errors. The work analyzes different effects of routing errors, shortcuts and open connections. One example of how TMR is affected by routing errors is the occurrence of a shortcut connection between adjacent signals that belong to different copies of the TMR design. This might cause two modules to present the same error, which in turn will generate a failure on the output of the voters. As a mitigation technique, the article proposes the use of redundant routing in the general routing matrix. Redundant routing solutions for single and hex lines and for both open and shortcut faults are proposed and an automatic router tool was developed by the authors. The proposed solution is evaluated by a fault-injection campaign on the routing bits of a 16-bit multiplier. The presented results show that the proposed solution was effective in preventing failures in a TMR design. The paper does not evaluate clock performance penalties due to the proposed technique.

TMR is very effective as a fault tolerance technique, but has an overhead of over 300 % in area and power compared against the unhardened circuit. In some applications, these costs might not be tolerable or the reliability requirements might not be so strict as to demand triple redundancy. Dual-Module Redundancy (DMR) uses two copies instead of three. As it does not have an odd number of outputs, it is not possible to vote the correct output and thus DMR does not offers error masking/tolerance, it is able only to detect a fault when the outputs of the two copies differ. For the same reason, it is not possible to know which of the two copies is defective when an error is detected. DMR can also be implemented with different granularity levels, as shown in Figure 2.2:

Figure 2.2 - Coarse-grained (a) and fine-grained (b) DMR



Source: Nazar (2013, p. 21)

Coarse-grained granularity compares the outputs of each copy, thus lacking internal diagnostics, while fine-grained granularity compares internal elements of the circuit. In both granularity levels, there can be a variation in what the grain is. For FPGAs, LUT-level FG-DMR is the finest granularity possible, while if CG-DMR is implemented with the POs of the unhardened circuits, it is the coarsest granularity possible. But FG-DMR and CG-DMR can be implemented with intermediate granularity if internal blocks or elements are compared. As a rule of thumb, coarse-grained techniques cannot identify where in the circuit the error occurred as they do not have internal diagnostic information, and thus are not able to avoid a SET be captured by a memory element.

DMR is associated with Concurrent Error Detection (CED) in Kastensmidt (2004) to detect and identify errors. If DMR detects an error, CED uses temporal redundancy to determine which copy is faulty, thus providing error detection. An automatic tool was developed to apply the proposed technique. The authors use as fault coverage benchmarks a 8-bit multiplier, a 9-bit multiplier and a 9-tap FIR filter. The automatic tool generated the protected circuits and a fault-injection framework. The fault coverage results show a 99.95 % coverage for the 8-bit multiplier and 100 % fault coverage for the other circuits. Area, delay and power measurements were made using a 16-bit multiplier, compared against TMR and the unprotected circuit. The presented results show that the proposed technique has a clock cycle overhead of 11 %, an area overhead similar to TMR and no power overhead against the unprotected circuit. The authors propose the use of configuration scrubbing to repair errors on the configuration memory.

The work in Bolchini; Miele; Sandionigi (2011) proposes a design flow in which reliability constraints for different components are used to automatically explore the design space and reduce resource overheads. The authors propose the use of different types of redundancy with the possibility of different granularities for each type of redundancy. The design flow has three phases: circuit analysis, where information of the circuit is gathered; design space exploration; where different solutions will be tested and presented to the designer; and the solution specification, where the selected solutions will be synthesized. Three different specifications for protection are supported: fault tolerance, fault detection and fault ignore. In the design space exploration phase, a tool tries to find the best balance between the requirements and costs. The implemented design is divided according to the reliability requirements, and these hardened components are statically mapped to FPGA areas using floorplanning. Besides hardening, the paper proposes the use of on-demand configuration scrubbing of the faulty component (for those components that have fault tolerance of fault detection requirements). The framework was then tested on three circuits: an H.264 encoder, an edge detector and a JPEG encoder. The experimental results show a modest area reduction when compared to classical CG-TMR or Xilinx TMR Tool (XILINX INC., 2015c) and a reduction of over 86 % on the scrubbing time. The reduction on the scrubbing time is due to the static flooplanning used. The paper does not explain the reason for the fault tolerance requirements used in the different components of the benchmark circuits and does not present results for clock overhead due floorplanning and does not verify the clock overhead and costs of floorplanning.

The analyzed works present different shortcomings. TMR has an elevated cost in terms of area and power that is not acceptable in low-cost or low-power applications. Readback as an error detecting technique is simple to implement, but introduces a long delay in detection and does not discriminates between errors that generate faults and benign errors, thus wasting time and power. Works that propose the use of internal partitions do not verify if the hardened circuit still meets the performance requirements.

## 2.2 Fault tolerance and Partial Reconfiguration

Partial Reconfiguration (PR) allows for an FPGA to be partially reprogrammed while functioning, without loss of functionality (ALTERA CORPORATION, 2010; XILINX INC., 2012a). This enables several interesting uses for FPGA, as it makes it even more flexible. With PR, it is possible to have a static nucleus in the FPGA, with a dynamic region that houses a different version of the same circuit according performance, power or timing requirements (KELLERMANN; TAM, 2010).

One technique to clear SEU errors in the configuration memory is to reprogram the entire device periodically, this is called configuration scrubbing, henceforth called scrubbing (CARMICHAEL; CAFFREY; SALAZAR, 2000). Periodic scrubbing, while simple to implement, is wasteful in terms of energy and leaves the system in a vulnerable state for a full scrubbing cycle.

In Gokhale et al. (2004), PR is used with configuration readback to correct SEU-induced errors. A system with a radiation hardened RAD6000 CPU, three SRAM-FPGAs and a hardened Actel FPGA is used in the proposed architecture. The Actel FPGA uses the SelectMAP interface, as in Peattie (2009), to read the current FPGA configuration every 180 ms. It then calculates the CRC of the read frame and compares this with a CRC codebook stored in a flash memory module. If a fault is found, an interrupt is generated to the CPU. The CPU then reprograms the faulty frame though PR and resets the system. The authors note the difficulty in reading the configuration memory if Look-Up Tables (LUTs) used as RAM or shift registers are being written by the user circuit, as the contents of distributed memory elements probably have been altered by the user application and simply re-writing these elements will lead to inconsistencies in the user design. One solution cited is to have a finer PR granularity in that the re-written bitstream would only change the necessary bits, excluding the distributed elements contents. The possibility of using checkpoints is not discussed.

While scrubbing can correct faults in the configuration memory and TMR/DMR provide for diagnostic, scrubbing is not capable to repair or work around permanent faults, so other mechanisms must be found. As partial reconfigurations allows for different versions of the same block, this is used in the work of Psarakis; Apostolakis (2012) to provide fault tolerance against permanent faults. An area to be protected within the device is partitioned. Each partition houses a module that uses an area smaller than the partition. Different versions of each module map the unallocated area in different ways, thus creating versions of the same module, and these are not directly implemented in the FPGA, but stored in an external memory. This scheme leaves unused copies of the circuits inside the FPGA, but allocates a larger than necessary area for each module and has a resource overhead to maintain the common interface with the different versions of the modules. DMR is proposed to provide error detection. When an error is detected, the affected modules are scrubbed to remove transient errors. If the fault was not corrected, then each module that indicated an error is switched to a new version, repeating for all versions of that module. If the fault was not corrected, the process is repeated for the second DMR copy. If still the fault was not corrected, the faulty module is isolated by the use of a blank configuration and only one DMR copy is used to provide the module's functionality. The benchmark circuits used were the Arithmetic Logic Unit (ALU), the Multiplier/Accumulator (MAC) and the instruction fetch stage of the pipeline of the OpenRISC processor. The presented results show average area and delay overhead of less than 10 % over non-reconfigurable modules. The Mean Time To Repair (MTTR) was 455.58 µs. It is important to notice that this approach is limited by the hardwired resources, such as DSP slices, that are needed to implement the reconfigurable module, a permanent error on one such module would defeat the proposed technique or severely limit the use of such resources.

The authors of Reorda, Sterpone, Ullah (REORDA; STERPONE; ULLAH, 2013) divide the FPGA in two regions, a static and a dynamic region. A soft-core CPU and other resources are mapped in the static region and this region is assumed to be protected by TMR. Error detection in this region is provided by a fine-grain mechanism that uses the embedded carry chain present in the Configurable Logic blocks (CLBs) (NAZAR; CARRO, 2012b). Errors detected in the static region are repaired by scrubbing. The CPU in the static region also controls the reconfiguration in the dynamic region. The dynamic region is protected by DMR with different error detection granularity levels, CG-DMR and FG-DMR. FG-DMR is implemented by DMR on LUT level and the embedded carry chain is used to compare the LUTs' outputs. Several error detection signals are chained to create two error signals per configuration column, allowing for the detection of single-bit errors. It is possible to detect multi-bit errors, in that case the embedded carry chain is not used but a XOR gate present in the CLB, providing an error flag for each LUT pair in DMR. CG-DMR is implemented by using DMR on a module level and then comparing the outputs with LUTs. Circuits are placed in this region if they use the embedded carry chain, making it unavailable for error detection. An automatic tool was created to apply the proposed design flow. The experimental setup uses a Microblaze soft-core CPU to program the dynamic region with faulty bitstreams and then drive the Circuit Under Test's (CUT) inputs with test patterns. When an error was detected, it was repaired by reprogramming the FPGA with the correct bitstream. Ten benchmark circuits were used as case studies, each benchmark subjected to 10.000 bit flips. The presented results show a very variable error coverage percentage (percentage of induced errors that triggered a configuration repair), with values ranging from 5.75 % to 47.62 %, a mean error detection capability of 98 % and an average repair time of 76.26 µs for single-bit faults. Area results are compared between the unhardened circuits, TMR, DMR and the proposed method. Delay overhead results are not shown.

## 2.3 Shifted Scrubbing

The concept of configuration scrubbing consists in using partial reconfiguration to re-write the configuration memory from a golden copy stored in a hardened medium. The scrubbing process follows a first-to-last order of configuration frames. On the other hand, the Place And Route (PAR) tool is free to place the circuit inside the whole device or inside an area bound by placement constrains. This means that not necessarily the circuit begins in the first frame, the circuit could be placed on the end, frame-wise, of the device. Another important realization is that not all frames are equally important in terms of faults. Some frames are critical to the circuit, while others suffer more noticeable masking effects and are not as important to SEU-induced effects. The concept of shifted scrubbing builds in that the scrubbing process should begin on the most important frame, error-wise, and then proceed to scrub the whole device or area. The works in Nazar; Santos; Carro  (2013, 2015) and Santos; Nazar; Carro (2013) explore this concept with the aim of reducing the Mean Time To Repair (MTTR) of a circuit.

In Nazar; Santos; Carro (2013) the technique of *shifted* scrubbing is proposed. A circuit is first protected by CG-DMR and is subject to a fault injection campaign (NAZAR; CARRO, 2012a). This allows the mapping of critical configuration frames. With this mapping, a heuristic chooses the best starting position for the scrubbing process to minimize the MTTR for each circuit. Twenty one benchmark circuits are used as case studies, and results for area overhead and MTTR reduction are shown. The proposed technique was able to achieve a mean MTTR reduction of 33 % over regular scrubbing.

In Nazar; Santos; Carro (2015) the concept of shifted scrubbing is further refined with the introduction of error signatures. Previous work (NAZAR; SANTOS; CARRO, 2013) has only one possible starting position for the scrubbing position for each circuit. This is interesting because the scrubbing controller is very simple; as soon as an error is detected it will start scrubbing from a predetermined position. On the other hand, MTTR could be further reduced if it was known where in the protected circuit the error happened. Instead of CG-DMR, this work uses FG-DMR on LUT level to obtain improved diagnosis information. It calls the concatenation of each comparator as the error signature. With the information of each frame and error signature, using the same heuristic as before (NAZAR; SANTOS; CARRO, 2013), a best starting scrubbing position is chosen for each error signature. The scrubbing controller now needs to receive not an error detection signal, but the whole error signature and to translate this to a configuration frame number. With FG-DMR the error signatures can be very long, from 11 bits to 1080 bits in the benchmarked circuits; the scrubbing controller can become prohibitively expensive. To reduce the scrubbing controller to manageable sizes, a heuristic is proposed to compress the signature. The presented results show an average MTTR reduction of 62.32 % over standard scrubbing, for a signature size of seven or less. Area and delay results are also shown. The fault injection campaign used pseudo-random input vectors to stimulate the benchmarked circuits. To analyze the effect of non-random input vectors, the case of a 32-bit ALU executing two algorithms is studied. The presented MTTR results are consistent with those obtained with random input vectors for the same circuit, illustrating the generality of the proposed technique.

The concept of shifted scrubbing is further explained in section 3.4, as well as its application in this dissertation.

## 2.4 Fault Tolerance on Real-Time Systems

Fault tolerance for real-time systems presents a challenge to designers, as they have additional modes of failure. The common goal of fault tolerance is to detect and repair faults, striving to keep computations correct. Real-time systems have the additional requirement of respecting deadlines, so a real-time system can fail if a deadline is not respected, even if the computation is correct. If the real-time system is a critical system, the consequences could be catastrophic (LEVESON; TURNER, 1993; NEUMANN, 1995).

A combination of checkpointing and on-demand scrubbing for real-time systems with softcore processors is demonstrated in the work of Sari, Psarakis, Gizopoulos (2013). The fault detection method is configuration readback, using an embedded Error Correcting Code (ECC) available in Virtex devices for error detection in individual frames and a Cyclic Redundancy Check (CRC) verification of the entire configuration memory. After each scan iteration, a checkpoint is recorded. In the case of an error, the faulty configuration frame is restored by partial reconfiguration and the CPU state is restored from the last known correct checkpoint. This checkpoint is not the last checkpoint, but the recorded checkpoint before the last correct configuration scan. The authors comment on the balance that must be achieved between scan frequency (which they call scrubbing frequency) and checkpoint frequency. In terms of fault recovery, the more frequent the checkpoints are, the less computation time is lost in the rollback. On the other hand, having checkpoints more frequent than the scan process is problematic, as it does not guarantee that the recorded checkpoints are correct. Thus the checkpointing frequency is tied to the configuration scanning frequency. The proposed technique is applied to a Leon-3 SoC softcore processor in a XC5VLX50T device. A MicroBlaze processor was used as the scrubbing controller. Different scan strategies were tested, a *full scan*, in which the entire FPGA is read; a *partial* scan, in which only the sensitive frames are read back, and a *constrained scan*, in which follows a constrained placement and a *selective scan*, in which only the active processor blocks for a given task are scanned. The authors present results for task response time and number of checkpoint allowed.

Readback allows for a cheap, area-wise, fault detection technique if the detection time lag can be tolerated. If that is not possible, then a more responsive fault detection method is needed. Readback is also too strict, in the sense that if a bit is in error, it will trigger a repair and rollback, even if the effect of this bit was masked.

The work of Nazar (2015) aims to maximize the probability of repairing a fault within the slack time of the tasks in a real-time system. This is a new proposition that looks the repair time the opposite way it was looked when aiming to reduce MTTR. Given a fixed time, how to maximize the repair probability? The work in this dissertation uses some techniques developed in this work that will be explained in greater detail in section 3, so now only a brief explanation will be given. The work uses FG-DMR (LUT level) and a fault injection platform (NAZAR; CARRO, 2012a) to map error signatures, as in Nazar (2015). Using a heuristic, for all given signatures a best start scrubbing position is chosen. This position is dependent on the repair time available. This repair time, called slack, is the time difference between the deadline of a task and the computation time of said task. The signatures are compressed as per Nazar (2015). The proposed technique is evaluated with combinational and sequential benchmark circuits. As the results are dependent on the slack available, the results are evaluated for slacks from 10 µs to 600 µs. Results for repair probability and FIT rates are shown. The presented results show an improvement in repair probability and FIT over the regular scrubbing process. The presented results show an elevated cost for clock cycle overhead, in some cases (*apex2*, *misex3* and *seq* circuits) the degradation was so severe that the hardened circuit did not meet the minimum performance requirements even in the absence of faults.

# 3 REAL-TIME DIAGNOSTICS AND REPAIR

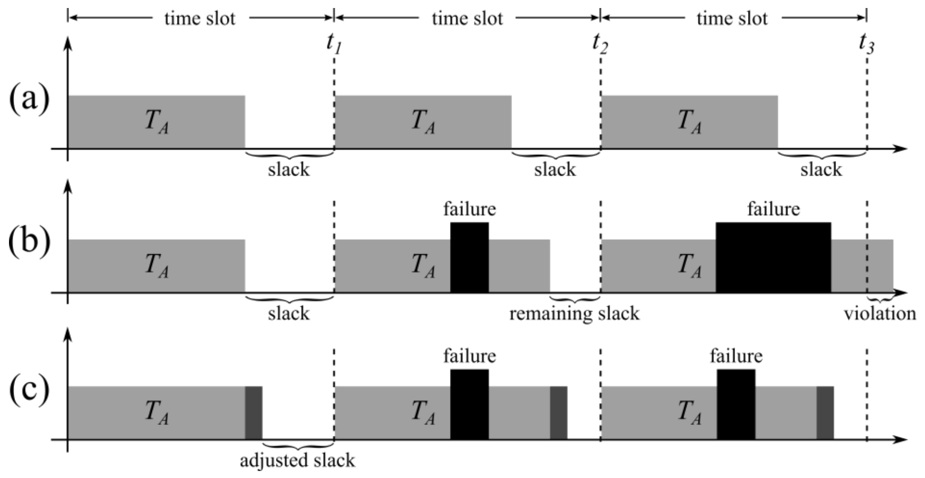
To improve the real-time repair over previous works, different diagnostic architectures were tested. These architectures are called Selective-grained DMR, Coarse-grained DMR with delta placement and Coarse-grained DMR with free placement. In SG-DMR, explained in detail in section 7, selected LUT pairs in a DMR circuit are chosen to be compared to detect errors. In CG-DMR with delta placement, explained in section 6, each bit of the POs of the circuit are compared, but each LUT in each copy is placed under constraints to have the same X slice coordinate (XILINX INC., 2012b). In CG-DMR with free placement, explained in section 5, each bit of the POs of the circuit is compared, but the Place And Route (PAR) tool is free to place the circuits in the area under test. The repair heuristic maximizes the probability of repairing an error within a bounded time, in the context of this dissertation, the real-time slack. The remainder of this section will explain the common blocks that compose the work, used with all diagnostic techniques.

## 3.1 Challenges

### 3.1.1 Failures in Real-Time Systems

To understand the challenges of fault tolerance in real-time systems, we must first consider the standard definition of a real-time system. One or more computational tasks must be executed within a bounded time, called time-slot. A correct system will be able to execute all designed tasks during their respective time slots. An execution timeline for a given task $T_A$ is shown in Figure 3.1(a), where $T_A$ is always finished before the deadlines, represented by $t_1$, $t_2$ and $t_3$. The remaining time between the end of $T_A$ and the deadline is called slack. Figure 3.1(b) shows an execution where two errors occurred. For the first error, the repair process was able to recover the system fast enough so $T_A$ could finish before the deadline for $t_2$. For the second error the repair process could not recover fast enough, so the failure caused a deadline violation for $t_3$. In hindsight, a designer might want to minimize the MTTR, as to leave the most remaining slack possible. However, just as a real-time system is normally more concerned with worst-case execution time than with average execution time, the mechanism herein proposed aims at maximizing the repair probability within a given timeframe, rather than minimizing the MTTR. As in (NAZAR, 2015), the expression “target repair time” will be used meaning the remaining slack time, which is the upper bound for the time interval the repair mechanism has to repair the circuit and not create a deadline violation.

Figure 3.1 - Correct execution (a), execution with errors (b), execution with repair (c)



Source: author

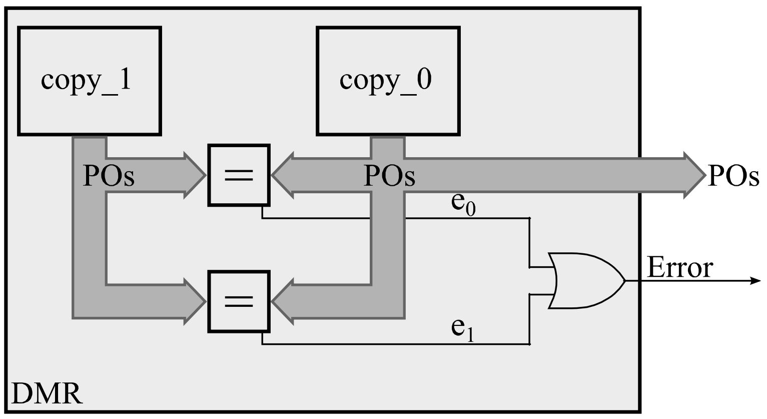
Besides repairing the circuit, the diagnostic and repair technique might cause a longer delay in the critical path, thus requiring a lower clock frequency, shown in Figure 3.1(c) as the darker area to the right of the task’s execution, reducing the available slack. If the available slack is reduced, so does the repair probability, which is contrary to the objective of this work of effective diagnostics and repair. So it is very important that the proposed diagnostic technique tries to preserve the clock cycle length as much as possible. Coarse-grained techniques, as opposed to fine-grained ones, do not place some many comparators in the clock critical path, leading to less degradation of the clock cycle and thus a leaving a longer slack available for repair procedures. The use of coarse-grained techniques has the added advantage of reducing the number of sensitive bits in the circuit, which is important when calculating Failure-In-Time (FIT) rates.

As explained in section 1.4, the use of only combinational benchmarks means there is not current state to worry about. It also means that no rollback mechanism is needed and thus there is no need to account for a recomputation time in the repair time needed.

### 3.1.2 System Architecture

To understand the proposed architecture for the protected system as a whole, it is important to explain how the proposed diagnostic technique differs from the regular DMR found in the literature. Regular DMR creates a single error bit to indicate a fault, two if the comparison is dual rail. Figure 3.2 shows how regular DMR with dual-rail comparators creates the error detection signal that would trigger repair or mitigating actions:

Figure 3.2 - Regular DMR

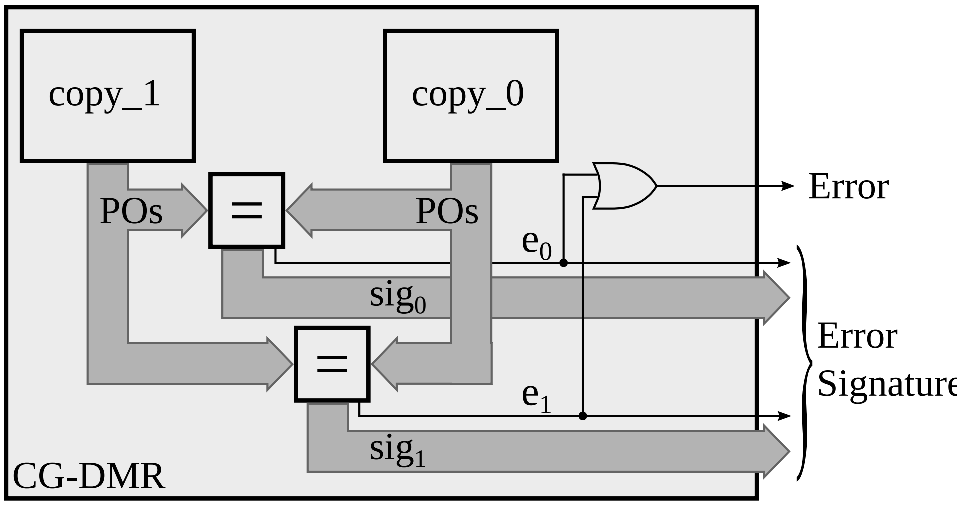


Source: author

It can be seen that a single error signal (named *Error*) is created by logic OR of two single error bits (*$e_0$* and *$e_1$*) that are generated from the comparison of the POs of each copy. This is an example of a coarse-grained diagnostic. It also can be noticed that there is no precision in the diagnostic regarding in which copy is at fault, let alone where inside the circuits the error occurred.

The use of better diagnostic architectures than regular DMR is proposed in this work. CG-DMR will be used to illustrate how the added diagnostic information helps the scrubbing process, but as will be explained in the following chapters, other diagnostic techniques can be used as well. Figure 3.3 shows the CG-DMR proposed in this work, with dual-rail comparison, in that several error bits are generated based on the comparison of PO signals from the copies of the circuit:

Figure 3.3 - Coarse-grained diagnostic

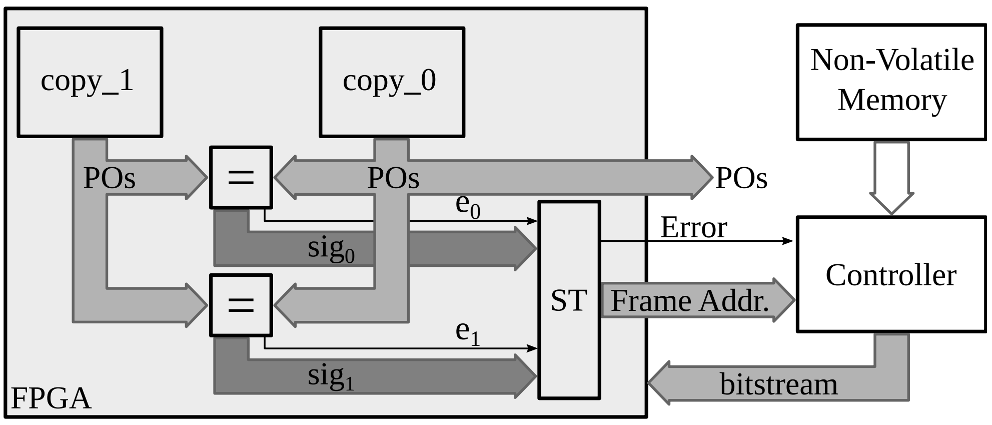


Source: author

Any error correcting technique using partial reconfiguration faces the challenge of mapping a detected error to where in the FPGA the circuit that generated the error is implemented. Ideally, one would be able to pinpoint a single configuration memory position (i.e., a frame) as the fault's location. However, this is unfeasible, since multiple errors in multiple frames can lead to faults with identical behavior on the user circuit. In this work we make use of an error injection platform (NAZAR; CARRO, 2012a) to simulate SEUs and track the relation between configuration memory errors and their effects on the user circuit. We extended the standard DMR to also make available the output of each comparison, i.e., one error-indicating signal will be generated for each bit in the POs of the circuit. We call the concatenation of these bits as the error signature ($sig_0$ and $sig_1$ in Figure 3.3). It can be seen from Figure 3.3 that dual rail comparison is used to generate the error signatures. As the comparators are subject to SEU errors themselves, this allows us to detect SEUs in the comparators. In Figure 3.3, copy_0 and copy_1 indicate the original unhardened circuit that was duplicated. It could also indicate different reconfigurable partitions within a larger design.

When using CG-DMR or SG-DMR, this information will be used to repair the system with added precision over regular DMR. The repair process uses an intelligent scrubbing process in which the starting scrubbing position depends on the error signature. Figure 3.4 shows an example of CG-DMR that feeds the repair circuit:

Figure 3.4 - System Architecture with CG-DMR



Source: author

The non-volatile memory and low-complexity scrubbing controller are implemented externally, using a hardened technology such as anti-fuse or flash-based FPGAs, as assumed in other works (GOKHALE et al., 2004), (BOLCHINI; MIELE; SANDIONIGI, 2011). By stablishing the relation between error signatures and configuration frames in error, it is possible to create a signature histogram, thus identifying the most critical frames for a given signature. In this work this relation is discovered by means of fault-injection campaign, as explained in section 4.

## 3.2 Signature Translator

A block called signature translator (ST in Figure 3.3) receives as input the error signature and outputs an error flag and the chosen frame address to start scrubbing in order to maximize the probability that the error is repaired given the slack available. The translation between an error signature and a single configuration frame is not straightforward. A single error might lead to several signatures, depending on the circuit's input stimuli and masking affects. The same signature could also manifest itself from different injected errors. This can be seen in Figure 3.3, if either the same bit in the POs of copy_0 or copy_1 are wrong, that bit in the signature will be a logic 1. Others factors such as routing play an important role, as demonstrated in Kastensmidt; Kinzel Filho; Carro (2006).

To map the relation between configuration frames and error signatures, a histogram $h_s[k]$ for signature *S* can be built, based on the number of times *S* was generated when errors were injected in configuration frame *k*. This histogram is then built for all configuration frames. The probability of repairing the system when *S* is generated by scrubbing frame *k* is the number of times *S* happens for all frames divided by the number of time *S* happens for frame *k*. In other words, this probability is the proportion of errors in frame *k* responsible for generating *S* among all *S*-generating errors. The probability of repairing an error for a given signature *S* when starting scrubbing by frame *f* and scrubbing at most *K* frames is the sum of the repair probability of all frames that were scrubbed:

$$P_S(f) = \sum_{k=f}^{f+K-1} \frac{h_s[k \bmod N]}{O_s} \qquad (1)$$

*Os* is the sum of all occurrences of *S* for all frames and *N* is the number of frames of the partition's configuration. The modulo division is present because if the scrubbing process reaches the end of the configuration space for that partition before scrubbing *K* frames, it should wrap around and start scrubbing at the beginning of the configuration space.

In order to repair a frame, a number of bits must be written in the FPGA. If *FS* is the configuration frame size in bits and *BR* is the configuration memory port bit rate, the time to scrub *k* frames is given by C + k·FS/BR, where *C* is the overhead associated with interfacing with the Internal Configuration Access Port (ICAP) interface, which is negligible for most devices, including the one used as a case study in this work. Given a real-time task slack of *SL*, in this time at most *K* frames can be scrubbed:

$$K = \left\lfloor \frac{(SL - C) \cdot BR}{FS} \right\rfloor \tag{2}$$

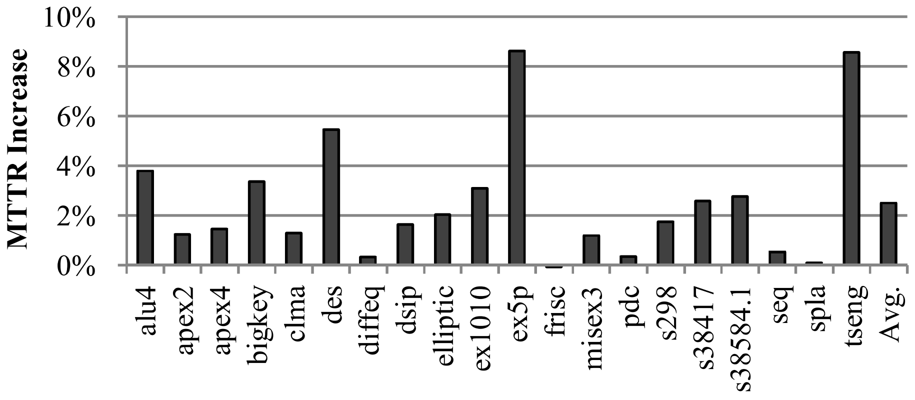
The heuristic then must try to maximize $P_S(f)$ for a given *K*. This is done iteratively, by calculating $P_S(f)$ for all *N* frames of the partition's configuration. When the optimum answer is found, it is recorded. When all optimum starting frames for all signatures have been found, it is possible to build the signature translator circuit, shown as ST in Figure 3.4. It has as inputs the error signatures and the error flags and as outputs an error signal and the frame address. We call the translator that generates the optimum starting frame for signatures with their full width as the Perfect Signature Translator (PST). As the PST could be too costly to implement for circuits with long signatures (or even impossible in some fine-grained scenarios), a heuristic to compact the signatures to manageable sizes is needed. In this work we will use the heuristic presented in Nazar (2015) to create a Real-time Heuristic Signature Translator (RHST).

### 3.2.1 Failures in the Signature Translator

The ST block is to be implemented in the SRAM FPGA (Figure 3.4). This raises the question on whether the ST is protected against SEU-induced errors and what effects errors in the ST have in the FIT rate. To address such question, it is important to remember what function the ST performs: it translates an error signature to a starting frame and through the *Error* signal (Figure 3.4) and starts the scrubbing process by way of the *Error* signal. So errors in the ST can affect one or both functions. If an error in the ST causes the starting frame selected to scrub to be an incorrect one, the scrubbing will probably achieve a poor repair probability. It will not, however, leave the system in an incorrect state, as shifted scrubbing will eventually repair the whole device. Another consequence of errors in the ST is that the *Error* signal is activated even when no error is detected, generating a false positive error detection. In this case, a scrubbing round will be executed, which will have no effect on the circuit other than wasting energy. As the failure model in this dissertation is that only one error happens at a time, as the repair time is reduced, a false negative is not possible, as it would require that an error happens on the user circuit, triggering a scrubbing round, and another error happens in the ST, deasserting the *Error* signal.

The work of Nazar; Santos; Carro (2015) evaluated the effect of faults on the ST block regarding the MTTR. The obtained results showed an increase in the MTTR, with more pronounced effect on benchmark circuits that the ST represented a higher area overhead. Figure 3.5 shows the increase in MTTR:

Figure 3.5 - MTTR increase owing to faults affecting the translation table



Source: Nazar; Santos; Carro (2015, p. 902)

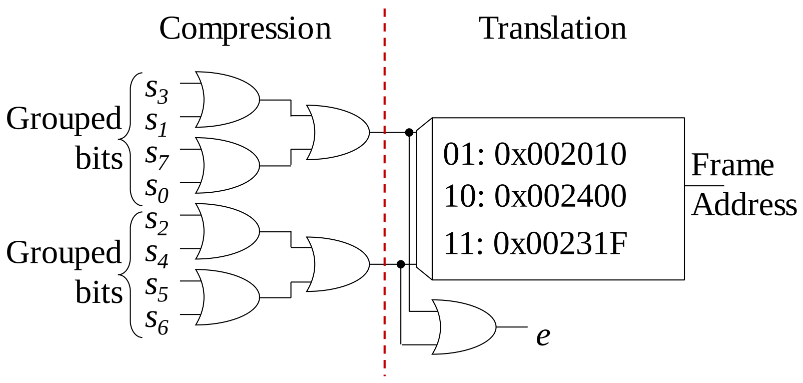
It can be seen that there is a small increase in the MTTR. The same effect in the repair probability is expected in the benchmarks used in this dissertation. Due to the small effect noticed from previous works, an experimental analysis of the effects of errors in the ST will not be carried out in this work.

## 3.3 Real-time Heuristic Signature Translator

The RHST is built on a compressed signature table. The compression algorithm must preserve as much as possible a precise mapping between signatures and configuration frames. The following algorithm tries to maintain this balance. This is not the only possible algorithm and future work could be carried to evaluate the effectiveness of the presented algorithm versus other alternatives. It is presented as one possible way to achieve manageable area costs for the RHST.

First a real-time slack must be defined, based on the available time slot and expected computation time. In this work we are considering a fixed slack, resulting from modules with constant workload and deterministic performance. Dynamically dealing with variable slacks is considered a promising future work, as it can be both challenging and relevant. It could be done, for example, with different translating tables, i.e., several RHSTs optimized to deal with different slacks, chosen at runtime according to current system parameters. Nonetheless, in this work we use a range of different slacks to evaluate our technique. After the slack has been defined, the algorithm starts by building a table with all signatures and their histograms, called *sigTable*, and another table with the optimum mapping between the signatures and the frame addresses, called *addrTable* (in the first iteration this table is the translation table for PST) using (1). Then iteratively it groups every two bits, using a criteria that will be explained shortly. This proceeds for all signatures in *sigTable*. The algorithm then checks for collisions in the new *sigTable* created with the compressed signatures, merging the histograms from the old *sigTable*. Then a new *addrTable* is built using (1), considering the new compressed signatures. This process continues while the signature length in bits is greater than a parameter called *maxSize*. After the round finishes, the last calculated *addrTable* is used to build the RHST circuit. An example of a 8-bit signature being compressed into a 2-bit signature is shown in Figure 3.6:

Figure 3.6 - Schematic of a ST circuit

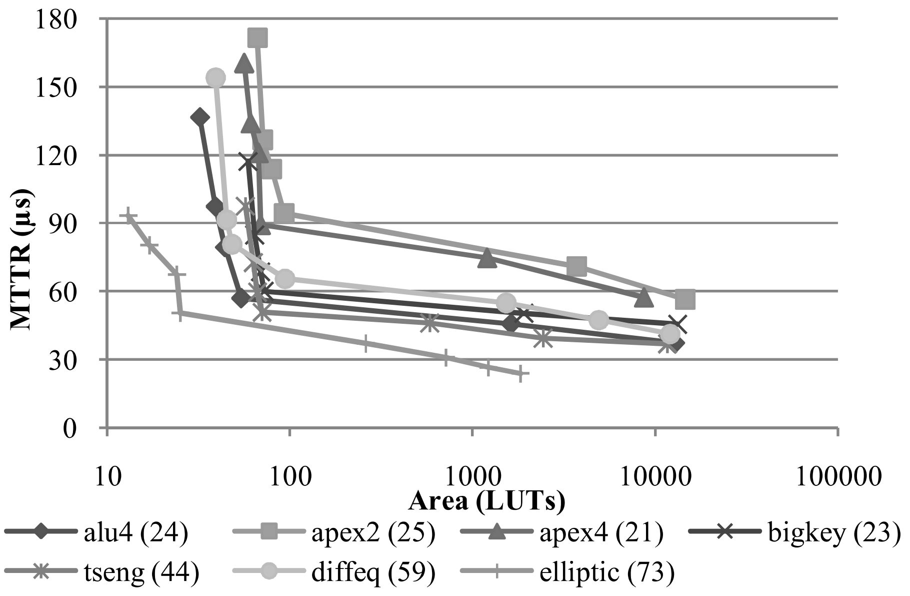


Source: Nazar; Santos; Carro (2015, p. 1113)

The RHST is built with a compression heuristic that compresses the potentially large signatures until a pre-specified *maxSize* parameter is reached. The compression heuristic groups signature bits by applying the OR function onto them, thereby reducing the signature size to half after each iteration. The choice of which bits must be grouped is extremely sensitive to the overall quality of the final solution. We build a complete graph in which each vertex is a bit (or group of bits) and edge is weighted according to the frequency with which those bits are activated by errors in nearby regions. Then, the maximum weighted matching, implemented in Dezso; Jüttner; Kovács (2011), is computed on this graph. The vertices linked by the chosen edges are then contracted, becoming a single vertex in the new graph, to be used by the next iteration. Once *maxSize* is reached, the final compressed signature is used to build a table of much reduced dimensions, when compared to the PST. For more details on the compression heuristic, please refer to Nazar; Santos; Carro (2015) and Nazar (2015).

The *maxSize* parameter has a great influence in the area occupied by the ST tables, as demonstrated by the work of Nazar; Santos; Carro (2015). There is an optimum *maxSize* value of 7, which has the best relation of diagnostic precision, in that case reducing MTTR, and occupied area. Figure 3.7 shows this relation:

Figure 3.7 - Area and MTTR for different maxSize values



Source: Nazar; Santos; Carro (2015, p. 9)

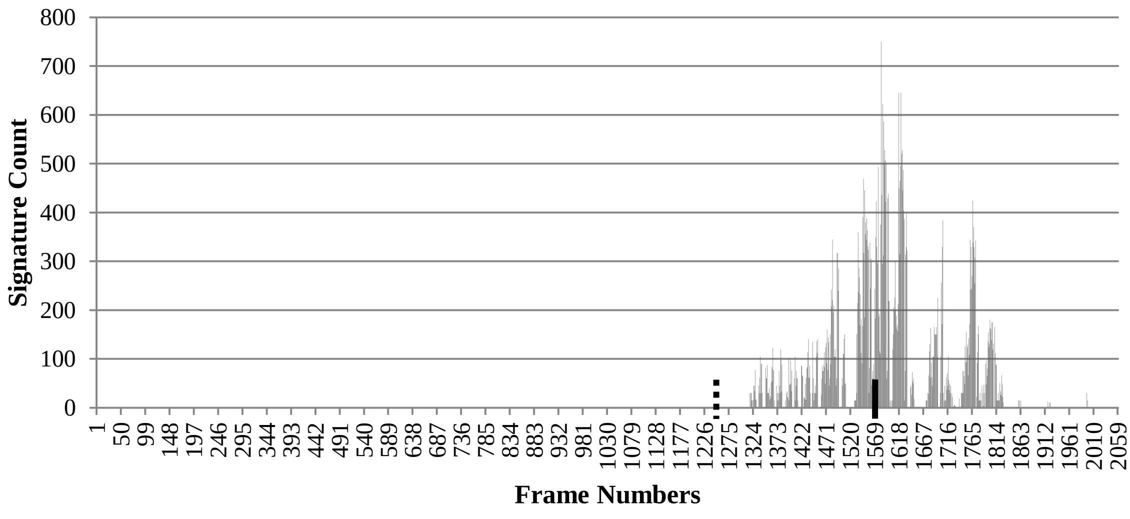
The same value of 7 is used for all tables in this dissertation. The explanation for the value of 7 is that the logic slices used in Virtex V and later Xilinx devices have 6 inputs (XILINX INC., 2012b) and a mux present in the slice (MUX7F) can be used to form logic functions with 7 inputs by concatenating two 6-input LUTs.

## 3.4 Shifted Scrubbing

The concept of *shifted* scrubbing is based on the realization that the actual time to repair any fault depends on how many frames we have to write until reaching the faulty frame. The standard approach is to start at beginning of the configuration addressing space (or the beginning of a reconfigurable partition). In this case, we are oblivious as to where the error most likely occurred and the repair time will depend on whether the error is located near the beginning or the end of the area being scrubbed. The basic idea behind shifted scrubbing is that, by starting the scrubbing operation at an appropriately chosen frame, the repair time will be smaller than that of the standard technique, as presented in the work of Nazar; Santos; Carro (2013). This dissertation uses a different criteria on the choice of first frame, as to maximize the repair probability of a faulty frame within a bounded time, the real-time slack

(NAZAR, 2015). The approach herein proposed attempts to dynamically choose an improved starting frame without the need for costly fine-grained checkers (NAZAR; SANTOS; CARRO, 2015), (REORDA; STERPONE; ULLAH, 2013), (NAZAR, 2015). To illustrate the concept of shifted scrubbing, the following example histogram for an arbitrary signature of a benchmark circuit is shown as Figure 3.8. It is important to remember that due to the way the configuration ports operate (SelectMAP and ICAP), the scrubbing direction goes from the left of the histogram to the right.

Figure 3.8 - Signature histogram and best configuration scrubbing starting frames

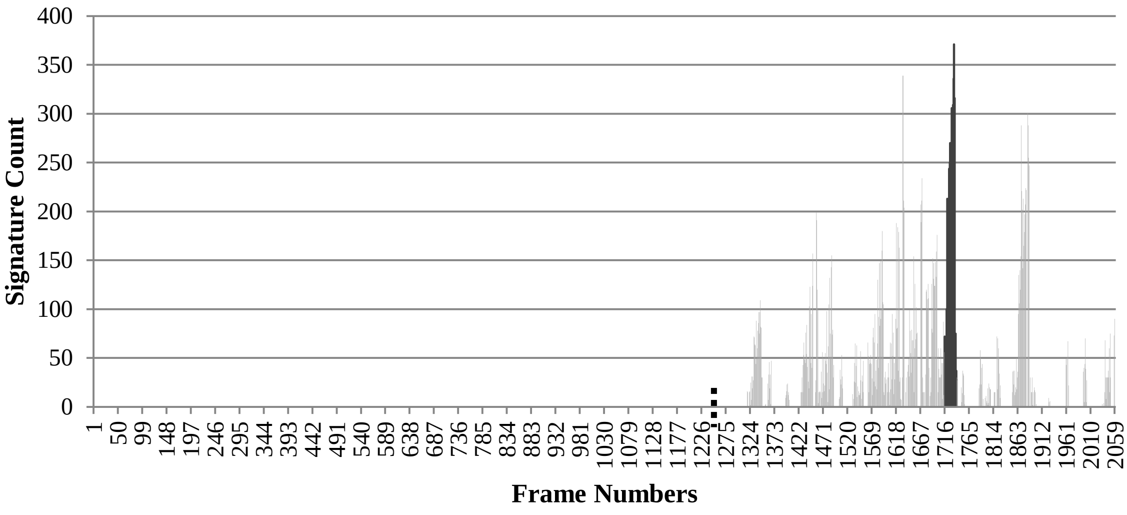


Source: author

Figure 3.8 shows that as the errors only occur on later frames, the standard scrubbing will take a long time to repair a frame that is actually able to generate errors. The two vertical marks on the X axis indicate the best frame to start the intelligent scrubbing for different slacks, the solid line for a slack of 10 µs and the dotted line for a slack of 600 µs. Because of the very small time available, the position for the 10 µs slack is very near the highest peak in the histogram; while the position for the 600 µs slack is farther to the right, as 600 µs is enough to repair all configuration frames.

To better illustrate the idea behind repairing a circuit within a bounded time, another histogram is shown as for the same circuit but for a different signature is shown below as Figure 3.9. The darker data bars indicate the frames that will repaired given a 10 µs repair time (10 µs slack):

Figure 3.9 - Signature histogram and best configuration scrubbing starting frames for 10 µs repair time

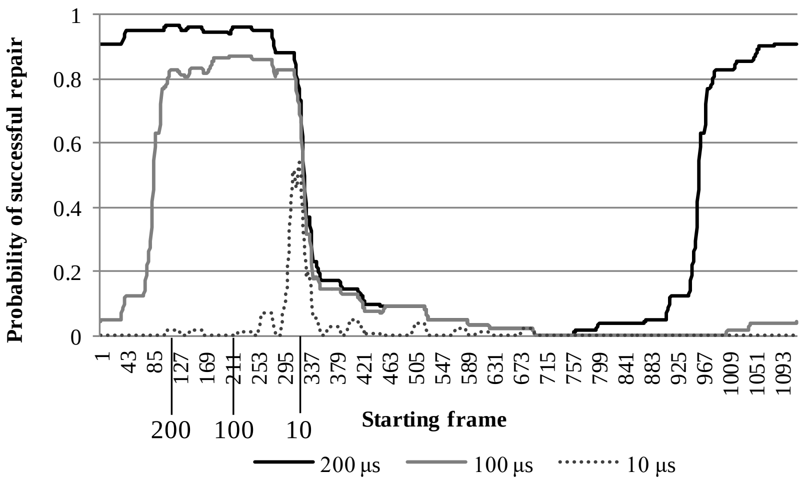


Source: author

As few frames can be repaired in such a short period, the heuristic choose the starting position for this slack just before the densest region of the histogram. If given 600 µs to repair the circuit, the heuristic will choose a starting position just before the histogram (shown as the dotted line in Figure 3.9) and will repair 1463 frames, meaning it will repair the whole histogram.

Figure 3.10 shows the repair probabilities for repair times (slacks) of 10 µs, 100 µs and 200 µs, again clearly indicating that the best starting frame address depends on the available repair time:

Figure 3.10 - Probabilities of successful repair for three different target repair times and for each starting frame



Source: Nazar (2015, p. 1113)

It is important to notice that shifted scrubbing will always repair the whole device or partition; it differs from standard scrubbing regarding the first frame to start the process. The slack is used to measure how many errors can be scrubbed before the system causes a deadline violation.

## 3.5 Best Static Starting Frame

The concept of shifted scrubbing relies on starting the scrubbing process at a specific frame, chosen according to some criteria; in this dissertation this criteria is the maximization of the repair probability given a bounded repair time. The starting frame is also dynamically chosen according the detected error signature. To limit the amount of resources needed to the ST implementation, the signature is compressed, as explained in section 3.3. As the compression rounds are executed, the signature width is halved, and the resulting histograms are calculated based on the collision of the histograms of the previous rounds. If this process is iterated until de error signature is only one bit wide, there is only one resulting choice of starting frame. This only choice is denominated the Best Static starting frame. The Best Static starting frame is the frame that, for a given slack, maximizes the repair probability considering all signatures.

It is important to notice that the Best Static starting frame is not dynamic, as it does not change according to the error signature (a 1 bit wide error signature). Starting the scrubbing process at the Best Static starting frame address instead of the first frame, as in the regular scrubbing process, presented gains of around 30 % in MTTR reduction, as demonstrated in the work of Nazar; Santos; Carro (2013).

The one-bit wide signature resulting from the HRST compression is actually an error detection signal. This means that for the Best Static starting frame there is not an ST block, as the choice of starting address is not dynamic. This means that the architecture shown in Figure 3.3 (CG-DMR) can be reduced to the architecture shown in Figure 3.2 (regular DMR), with the *Error* signal used as the one-bit wide signature. This has an important consequence as regular DMR is the baseline to measure the clock overhead, leaving the most slack possible to repair. This can be considered as an extrapolation of the trade-off of this dissertation, that is to use coarser diagnostic architectures with the hope that a less precise diagnostic is compensated by a large repair time; the Best Static starting frame can be considered the coarsest diagnostic possible.

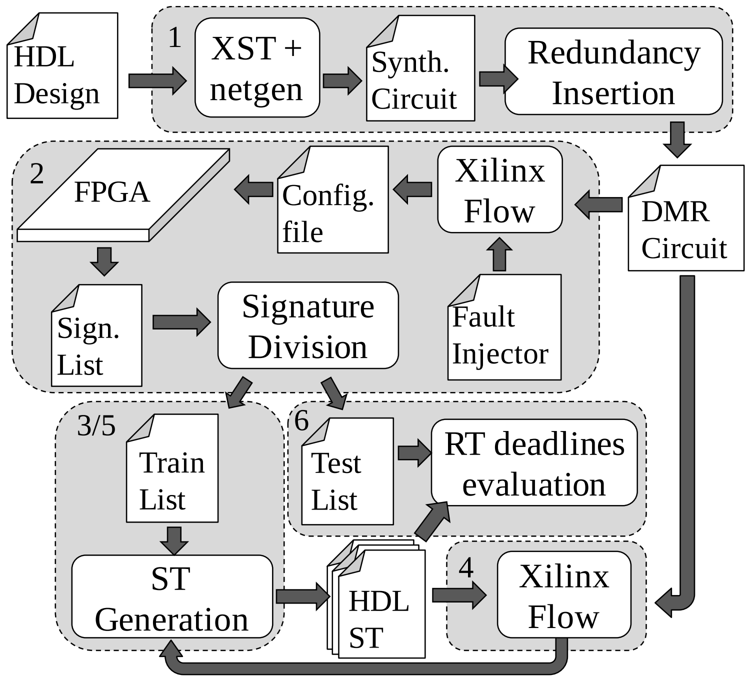
# 4 EXPERIMENTAL SETUP

The case studies used in this dissertation were taken from the MCNC benchmark suite found in (MINKOVICH, 2010) and modified ALU circuits compatible with a MIPS processor (alu 32b and alu 64b). All the benchmarked circuits are combinational, as the comparison of POs does not allow for detecting an error and halting a sequential circuit before the error is captured by a flip-flop. If the proposed technique is applied to sequential circuits, it must then be used in combinational blocks before the inputs of flip-flops. In this case, the error signals could be used with a rollback mechanism (SARI; PSARAKIS; GIZOPOULOS, 2013).

## 4.1 Experimental Design Flow

The experimental design flow used in this work is shown in Figure 4.1:

Figure 4.1 - Experimental setup design flow



Source: Nazar (2015, p. 1115)

In the first step, indicated "1" in Figure 4.1, a post-synthesis model of the unhardened benchmark circuits was generated using the netgen tool from Xilinx ISE 13.4. A C++ application analyzes the circuit and builds the hardened, fault-tolerant VHDL model with the architectures explored in sections 5, 6 and 7. In the second step, the benchmark circuits in their hardened versions are synthesized with Xilinx tools. The resulting bitstream for each test circuit is programmed in a Xilinx Virtex 5 XC5VLX110T FPGA contained in a Xilinx XUPV5-LX110T board. Errors were injected in the CUT, using the fault injection platform developed in Nazar (2012a) and modified in Nazar; Santos; Carro (2013), generating error signatures that are recorded in a PC. The captured are analyzed in the third step, which begins with the captured signatures being split in two groups, a training group and a testing group. The training group is used to build the RHST tables and the verification group is used to test the effectiveness of the generated RHST. In the work of Nazar (2015), the third step is the processing of the training signature group to generate the RHST tables. This is omitted in the work in this dissertation.

The fourth step is to process the unhardened circuits and the hardened circuits with the regular Xilinx design tools to evaluate area costs and delay values. With the increased clock delay values of the hardened circuits, the fifth step is the processing of the training signatures group in a C++ application that implements (1) and (2), to create tables for different metrics the RHSTs, for different time-slot occupation scenarios and for an arbitrary *maxSize* parameter of 7. This value is considered according to (NAZAR; SANTOS; CARRO, 2015) due to technological and implementation characteristics of the Virtex 5 devices. This step already considers that the available repair slack is reduced in the hardened circuit, obtained in the previous step. The effectiveness of the generated RHST tables is verified in the sixth and last step, in which the testing signature group is processed by another C++ tool that uses the RHST tables generated in the fifth step to calculate the obtained repair probability, ensuring the generality of the solution as the testing signature group was not used to create the RHST tables.

This design flux can be used generally to any combinational circuit described in a HDL language. Currently the software tools developed for this work are integrated with scripts. A future work could be to create an automatic design-space exploration tool that would evaluate different diagnostic architectures and/or parameters automatically.

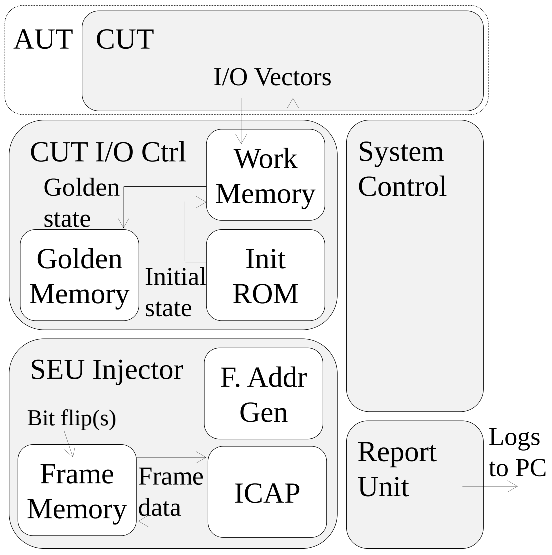
### 4.1.1 Fault Injection Platform

The concept of shifted scrubbing relies on knowing the relation of error signatures and faulty configuration frames. To obtain this relation, as explained in section, 3.1.2, a fault-injection tool is used to force errors on the benchmark circuits and then stimulate the circuit. In this dissertation it is used an internal fault-injection framework developed by Nazar; Carro (2012a).

This fault injection framework is synthesized along with the benchmarked circuits with the selected redundancy, as shown in Figure 4.1 (step 2). During synthesis, an area in the device is defined as the Area Under Test (AUT) through a placement constraint, with the

CUT wholly implemented in this area. Another placement constraint is also used to place the fault-injection platform itself on another area inside the device, not affected by the injection campaign. A system overview of the fault-injection system is shown in Figure 4.2:

Figure 4.2 - Fault injection system overview



Source: Nazar; Carro (2012a, p. 154)

The fault injection platform will read a configuration frame from the area under test using the ICAP interface, invert a bit and write the frame with the bit error back in the device. It will then stimulate the circuit for 50,000 cycles using a Linear Feedback Shift Register (LFSR). The POs of each copy are compared against each other and against a golden copy not placed in the area under test. If any of the POs bits are not equal (generating a non-zero error signature), then the stimulation process is halted and the error signature is transmitted to the PC collecting the data. As soon as the transmission is completed, the stimulation process is resumed. When the stimulation process is completed, the inverted bit is restored to its original state and the next bit is inverted, starting over the process. After a whole frame has been injected, the original frame is restored and the next frame is read back and so on, until the whole area under test has been tested.

### 4.1.2 Area Under Test Occupation

The area under test is the region on the FPGA that the CUT is placed by the use of placement constraints. It is also the portion of the device that is subject to the fault-injection campaign. Both copies of the CUT (copy_0 and copy_1 in Figure 3.3), the PO’s comparators and the signature translator are placed in the area under test (shaded area indicated as “FPGA” in Figure 3.4). In this dissertation the area under test for each experiment is calculated according to the number of LUTs used by each benchmark to achieve an occupation of the area under test of 85 % (DEHON, 1999). The reasons for having this high occupation are twofold. The first reason considers the real-world occupation of deployed devices. The second reason considers the comparison between regular scrubbing and shifted scrubbing.

In real-world applications, the device occupation tends to be high due to simple economic reasoning; larger devices are more expensive, so designers want to use the smallest device possible. So to emulate the real-world usage of FPGA devices, experiments should be carried out in a way that simulates this high occupation.

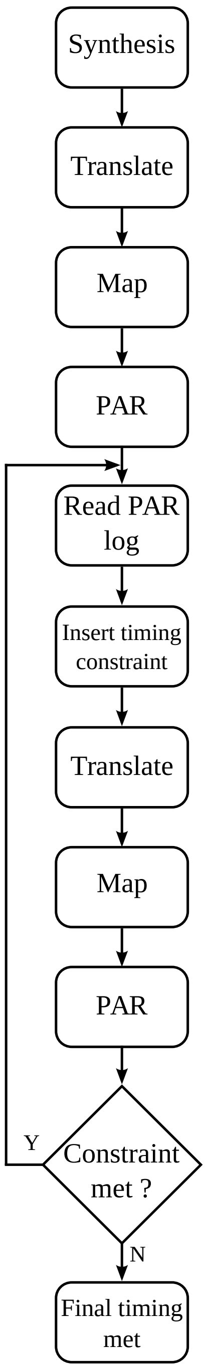
As can be seen in Figure 3.8 and Figure 3.9, the placement of the circuits within the area under test can leave large unused regions. These unused regions will be scrubbed by standard scrubbing, leading to wasted scrubbing time. As shifted scrubbing will start scrubbing on the beginning of the occupied region (see the 600 µs starting position in Figure 3.8 and Figure 3.9), these initial empty regions will have a much greater affect in standard scrubbing than of shifted scrubbing. By forcing the placement tools to use a smaller region, there is a tendency to avoid such empty regions, giving a fairer comparison against the standard scrubbing approaches, as they are benefitted by partitions that are as small as possible.

## 4.2 Area and Delay Overheads

The area overhead is important to evaluate the cost of a given technique, even so considering this work that is evaluating *low-cost* diagnostic techniques. This needs to translate to a low area overhead when compared to the standard DMR. To evaluate area costs, the benchmark circuits were synthesized according to the different diagnostic architectures evaluated in this dissertation and the number of LUTs used in each case is compared with the same benchmark circuits protected by regular DMR (Figure 3.2). This area evaluation synthesis was independent from the fault-injection circuits, as each benchmark circuit was described in VHDL, along with accessory blocks as comparators. These VHDL files were synthesized with scripted Xilinx (ISE 13.4) tools and the LUT number was read from the MAP tool log file.

Delay overhead translates to how much the clock cycle is lengthened by the proposed techniques. This is a critical point, in that low-cost also translates to low degradation of the regular DMR clock cycle. The hardened clock cycle is also used when evaluating the repair probability and FIT rates, in which FIR rates will be directly proportional to the clock cycle overhead. To evaluate the minimum clock cycle, a VHDL description of the hardened circuits with registered inputs and outputs, including the error signature and error signals, was synthesized to the PAR phase. The PAR tool log file was then read and the achieved clock cycle length was placed as a design constrain in a UCF file, then the circuit was re-synthesized. This process continued until the PAR was not able to meet the timing constraint. This method was used as the PAR tool will try to meet the timing constraints with a little room to spare. So when the PAR was not able to meet the constraint, that constraint is the minimum clock cycle length for that circuit. This process was automated by scripting the necessary Xilinx tools, and is shown below as Figure 4.3:

Figure 4.3 - Timing analysis flowchart



Source: author

## 4.3 Repair Probability

The repair probability, along with the clock overhead and number of sensitive bits, will determine the FIT rates for each circuit. This work proposes the use of less precise diagnostic techniques then FG-DMR, thus lowering the repair probability, but in a way that also lowers the number of sensitive bits and also has a lower clock cycle overhead, in a way that in the overall case, lower FIT rates will be achieved.

In order to evaluate just the diagnostic quality of each technique, the repair probability results will be shown, compared and analyzed between themselves and FG-DMR.

## 4.4 Failure-in-Time

We calculated the expected FIT values for each circuit. As the FIT value takes into account the number of sensitive bits in a circuit, it consists in an appropriate metric to compare circuits with different sizes implemented on the same technology. The FIT value for each circuit was calculated according to the formula from Nazar (2015):

$$\text{FIT} = F \cdot \sigma \cdot SB \cdot (1 - P_s) \cdot 10^9 \tag{3}$$

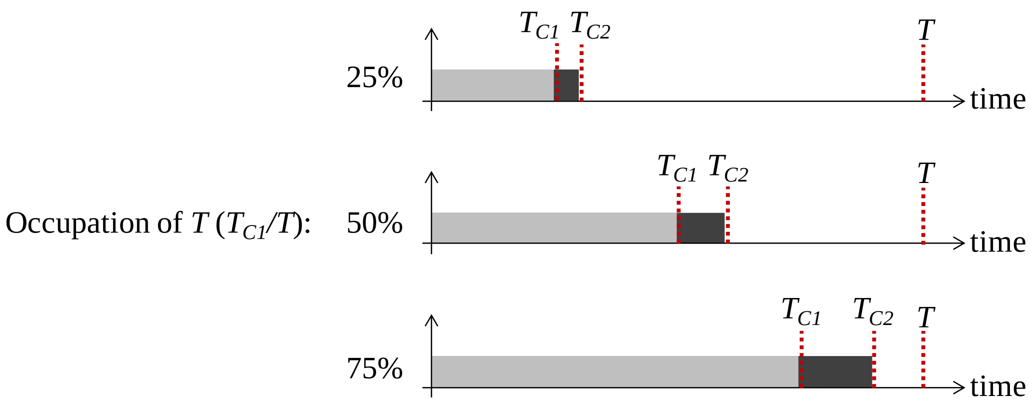
$F$ is the neutron flux. At sea level a value of 13 n/cm$^2$·h was used as typical for neutrons with energy above 10 MeV (JEDEC, 2006). $\sigma$ is the cross section per bit, as reported in (XILINX INC., 2015d). $SB$ is the number of sensitive bits for each circuit and was measured in the fault injection experiments. Finally, $P_S$ is the repair probability, obtained with (1) and considering the frame chosen for each signature by the synthesized ST circuit.

The neutron flux gives the number of neutrons that reach sea-level elevations each hour. The bit cross section is the probability that a neutron causes an SEU in the circuit. The product of both is the rate that SEUs happen for a single bit at sea level in an hour. This rate multiplied by the number of sensitive bits in the whole circuit means the rate of SEUs for the whole circuit at sea level in an hour. One minus the probability of repair means the probability of **not** repairing the circuit, i.e. leaving the circuit in an unrepaired state. This probability multiplied to the rate of SEUs for the whole circuit at sea level in an hour means the adjusted probability of leaving the circuit operating with SEU-induced errors at sea level for an hour. As this number is very small, it is more convenient to have FIT rates nearer to one, thus the multiplication of the adjusted probability of leaving the circuit operating with SEU-induced errors at sea level for an hour by 1x10$^9$, meaning the number of failures caused by SEUs for the whole circuit at sea level for 10$^9$ hours of operation.

FITs are calculated through (3), therefore, take into account any costs the introduced techniques may have both in terms of area (by means of an increased $SB$) and delay (which reduce the available slack time to conclude repair). Benefits observed in FIT come from an increased $P_S$, obtained through the described low-cost diagnostics and repair mechanism.

As from (2) and (3), the FIT rate depends on the available time to repair, it was evaluated for several slacks, from 10 μs to 100 μs in steps of 10 μs and from 100 μs to 600 μs in steps of 100 μs, for a total of 15 different slacks. We also considered three scenarios in which the task computing time occupies 25 %, 50 % and 75 % of the time slot, adjusted for the reduced slack due to the delay overhead (Figure 3.1 (c)). This is necessary as the increased clock cycle has a more pronounced effect for higher occupation scenarios than for lower ones. Figure 4.4 illustrates how the impact of the clock overhead is relative to the time slot occupation:

Figure 4.4 - Impact of delay increase on slack time for different occupation scenarios



Source: Nazar (2015, p. 1117)

Let *T* in Figure 4.4 be the deadline of a real-time computing task and $T_{C1}$ be time needed to compute said task in the original unprotected circuit. As the protected circuits have a clock cycle penalty, they will not operate at the same frequency as the original circuits, thus taking longer to complete the same computation, so let $T_{C2}$ be time needed to compute the same task as $T_{C1}$, but now in the protected circuit. If $T_{C1}$ originally represented 25 %, 50 % or 75 % of *T*, it can be seen from Figure 4.4 that $T_{C2}$ will represent a larger percentage of *T*. The available repair time in order not to have a deadline violation is the difference between *T* and $T_{C1}$ in the original circuit, called *slack*:

$$\mathrm{SL}_1 = \mathrm{T} - \mathrm{T}_{C1} \tag{4}$$

Considering that $T_{C1}$ is a proportion of T, then, considering as $\mathrm{O}_{\mathrm{CC}}$ as the occupation, equation (4) can we written as:

$$\mathrm{SL}_1 = \mathrm{T}(1 - O_{CC}) \tag{5}$$

$T_{C2}$ is a proportion of $T_{C1}$, then it is also a proportion of *T*. So let $O_H$ be the overhead of $T_{C2}$ over $T_{C1}$:

$$\mathrm{O}_H = \frac{\mathrm{T}_{C2}}{\mathrm{T}_{C1}} \tag{6}$$

Equation 5 and 6 can be used to define the adjusted slack $SL_2$ as:

$$\mathrm{SL}_2 = \mathrm{T}(1 - O_H . O_{CC}) \qquad (7)$$

As an example, let us assume that $T_{C2}$ is 20 % larger than $T_{C1}$ ($O_H$=1.2), so the available slack in the 25 % occupation scenario ($O_{CC}$=0.25) that originally was 75 % is now 70 %, a reduction of only 5 % of *T*. In the case of the 75 % occupation scenario ($O_{CC}$=0.25), the same 20 % overhead will lead to an adjusted slack of only 10 %, a reduction of 15 % of *T*. Equation (7) also can be used to find the maximum clock overhead for a given occupation, for 75 % occupation the maximum overhead is 33,33 %.

# 5 COARSE-GRAINED DOUBLE MODULE REDUNDANCY WITH FREE PLACEMENT

CG-DMR with free placement is the simplest coarse-grained technique and it is used in this work in its most coarse granularity, duplicating the entire benchmark circuits. The aim of this is to preserve the original clock cycle length as much as possible, to achieve the largest repair time possible. With a larger slack, we hope to compensate for the less precise diagnostic.

## 5.1 Proposed Architecture

The overall architecture of CG-DMR was already presented in Figure 3.3. The difference from CG-DMR with delta placement is that in CG-DMR with free placement there were no placement constraints for individual components (Figure 6.1(a)). As stated before, the benchmarked circuits were synthesized with placement constraints to simulate a real-world device occupation scenario of 85 %. The achieved occupations are shown in Table 5.1:

Table 5.1 - CG-DMR with free placement benchmark occupation

| | **Area Occupation** | | | **Area Occupation** | |
|---|---|---|---|---|---|
| **Benchmark** | ***DMR*** | ***RHST*** | **Benchmark** | ***DMR*** | ***RHST*** |
| *alu 4b* | 84,58 % | 87,50 % | *ex1010* | 87,86 % | 90,71 % |
| *alu 32b* | 89,00 % | 82,81 % | *ex5p* | 96,25 % | 92,29 % |
| *alu 64b* | 85,06 % | 85,26 % | *misex3* | 88,13 % | 82,44 % |
| *apex2* | 83,23 % | 84,43 % | *pdc* | 83,55 % | 86,45 % |
| *apex4* | 82,75 % | 85,94 % | *seq* | 82,79 % | 86,73 % |
| *des* | 80,88 % | 85,10 % | *spla* | 75,31 % | 90,63 % |

Source: author

## 5.2 Experimental Results

### 5.2.1 Area and Delay Costs

Table 5.2 shows the results for area and delay obtained with CG-DMR with free placement:

Table 5.2 - Area and delay results for CG-DMR with free placement

| **Benchmark** | **# POs** | **Area (LUTs)** | | | **Delay (ns)** | | |
|---|---|---|---|---|---|---|---|
| | | ***DMR*** | ***CG-DMR*** | ***Increase*** | ***DMR*** | ***CG-DMR*** | ***Increase*** |
| *alu 4b* | 9 | 812 | 839 | 3.33 % | 6.49 | 6.58 | 1.34 % |
| *alu 32b* | 34 | 712 | 794 | 11.52 % | 8.14 | 8.16 | 0.22 % |
| *alu 64b* | 66 | 1497 | 1637 | 9.35 % | 9.62 | 10.64 | 10.56 % |
| *apex2* | 4 | 1598 | 1620 | 1.38 % | 7.28 | 7.57 | 4.08 % |
| *apex4* | 19 | 1324 | 1375 | 3.85 % | 7.34 | 7.85 | 7.00 % |
| *des* | 246 | 1292 | 1768 | 36.84 % | 6.91 | 7.86 | 13.82 % |
| *ex1010* | 11 | 984 | 1015 | 3.15 % | 6.44 | 6.45 | 0.23 % |
| *ex5p* | 64 | 308 | 443 | 43.83 % | 5.39 | 5.30 | -1.61 % |
| *misex3* | 15 | 1410 | 1451 | 2.91 % | 7.01 | 7.31 | 4.16 % |
| *pdc* | 41 | 2540 | 2628 | 3.46 % | 8.71 | 9.05 | 3.99 % |
| *seq* | 36 | 1722 | 1804 | 4.76 % | 7.24 | 7.60 | 4.89 % |
| *spla* | 47 | 482 | 579 | 20.12 % | 6.18 | 6.47 | 4.69 % |

Source: author

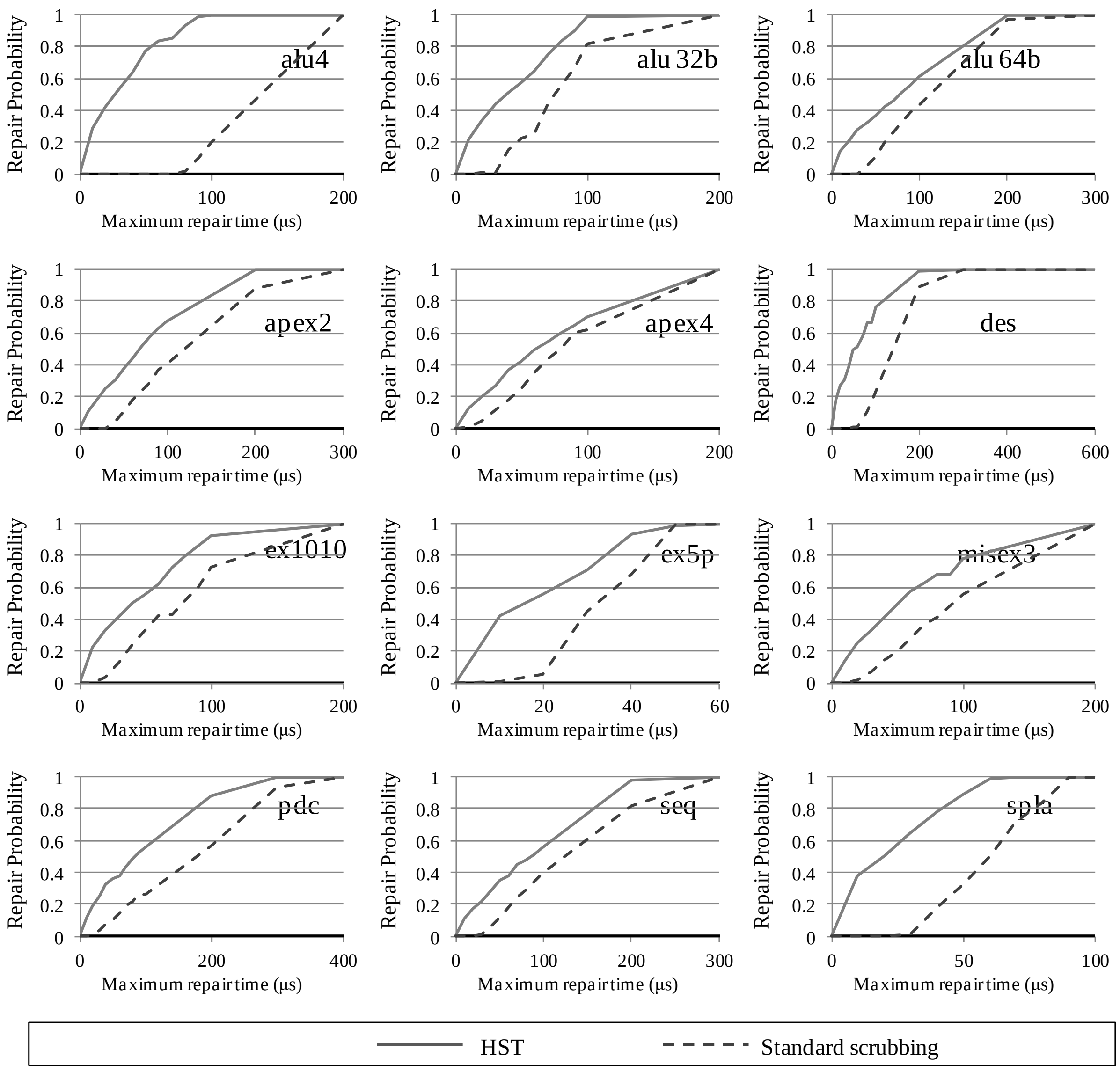
The area overheads shown in Table 5.2 are similar to those presented in Table 6.2, which is natural as both share the same architecture. As happened with the delta placement, the benchmarks with greater area overhead are the *des* and *ex5p* benchmarks, due the relation between the number of POs and the area of the regular DMR. In GC-DMR with free placement, we again choose to use one LUT for every compared bit on the RHST circuits. The average area overhead for all circuits is 12.04 %. The area costs for the RHST circuits already include the cost for the ST tables (as in Figure 3.4). We choose to use the worst case for each circuit, among all synthesized tables.

The average delay overhead is 4.45 %, which shows that our expectation was justified in that CG-DMR does not introduce a large clock cycle overhead, being smaller than SG-DMR and CG-DMR with delta placement (7.21 % and 21.89 % respectively). For the *ex5p* and *pdc* circuits the RHST circuit was actually faster than the regular DMR, which is likely due to the random optimizations of the implementation heuristics. The low delay overhead is in stark contrast with the overhead obtained with delta placement, indicating that diagnostic techniques should strive to leave room to the MAP and PAR tools to optimize the hardened circuits. It is important to also compare the delay results with those obtained in Nazar (2015), in which the average clock cycle overhead over regular DMR was 23.6 %.

## 5.2.2 Repair Probability

The repair probability curves show how good a diagnostic technique is, the faster the curve reaches 1, the better the diagnostic.

Figure 5.1 - Repair probabilities for CG-DMR with free placement



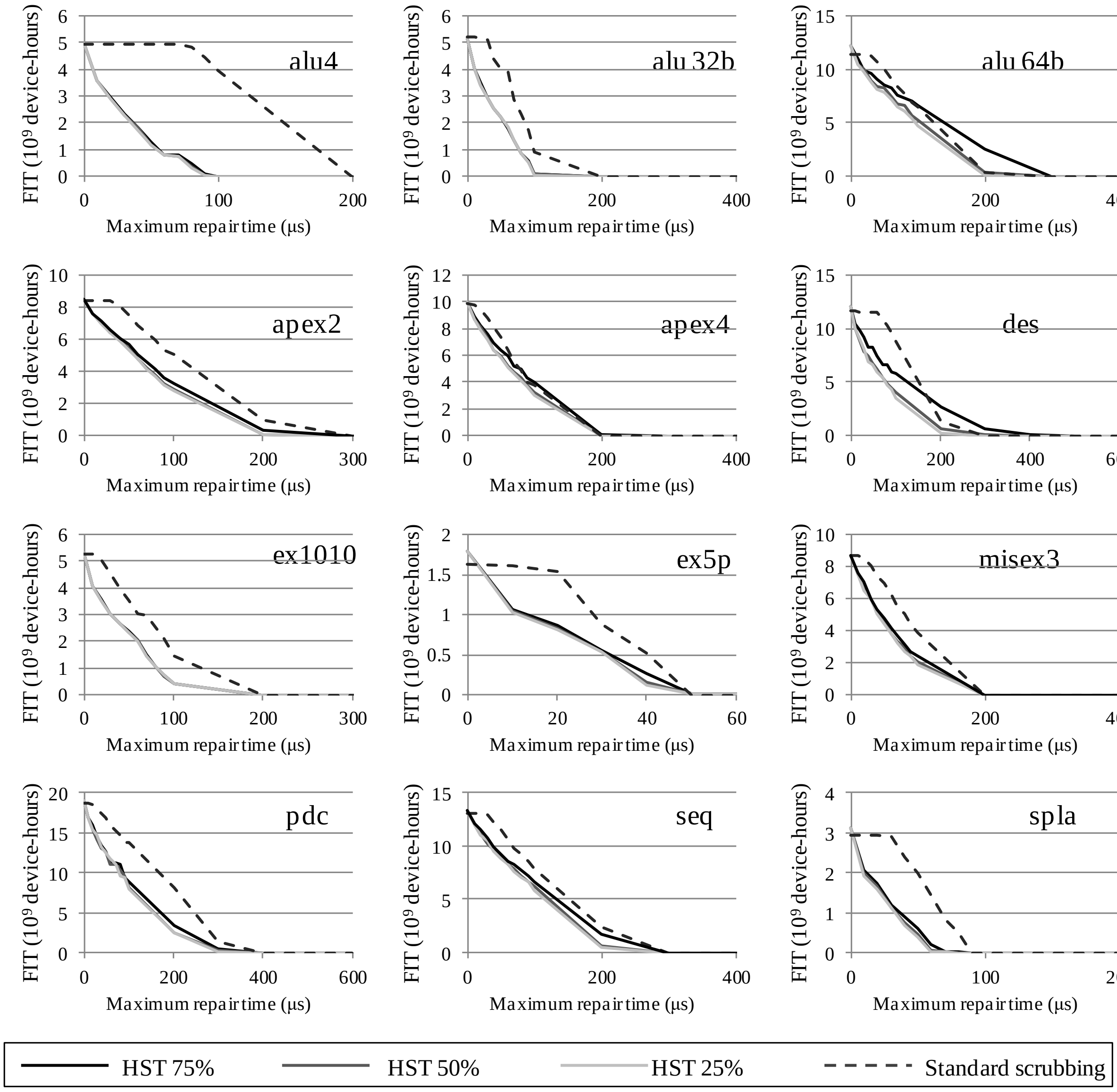
Source: author

CG-DMR with free placement provides a better repair probability than standard scrubbing for all benchmark circuits, for all slacks; this result is interesting, as CG-DMR does not relies on any internal information of the hardened circuit.

### 5.2.3 Failure-in-Time Results

FIT results take into account not only the repair probability, but the number of sensitive bits. The FIT results were calculated with (3), for the same slacks (10 µs to 600 µs) and timeslot occupation scenarios, and are shown in Figure 5.2:

Figure 5.2 - FIT results for CG-DMR with free placement



Source: author

# 6 COARSE-GRAINED DOUBLE MODULE REDUNDANCY WITH DELTA PLACEMENT

The use of a constrained placement is an attempt to improve on the results obtained on the previous section. The reasoning behind the delta placement is to position both copies of the same LUT in the same configuration frame. When an error occurs on either copy, it will map to the same frame, helping the signature-frame decision of the signature translator (ST).

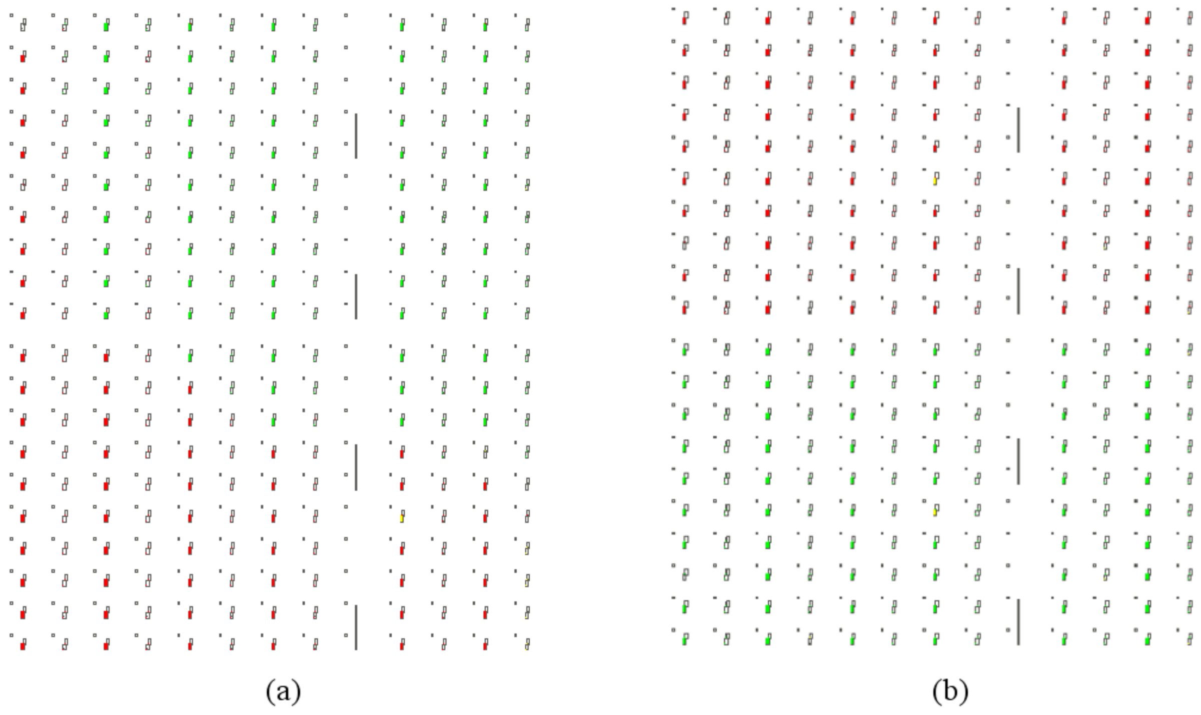
## 6.1 Proposed Architecture

CG-DMR with delta placement uses the same architecture as shown in Figure 3.3, in which the error signature is created from comparing only the POs of the circuits. The difference between CG-DMR with free placement and with delta placement is that in the delta placement, all LUTs in copy_1 are placed at the same X slice coordinate as the corresponding LUT in copy_0, with a Y slice coordinate 10 slices over the corresponding LUT in copy_0, the value of 10 being the half the height of a configuration frame (XILINX INC., 2012b). An example of four placement constraints for the *pdc* circuit as shown below:

```
INST "cut/cpy0/outputVector_39_9882" LOC=SLICE_X60Y140;

INST "cut/cpy1/outputVector_39_9882" LOC=SLICE_X60Y150;

INST "cut/cpy0/outputVector_39_9881" LOC=SLICE_X60Y140;

INST "cut/cpy1/outputVector_39_9881" LOC=SLICE_X60Y150;
```

The difference in placement for CG-DMR with free placement and CG-DMR delta can be seen in Figure 6.1:

Figure 6.1 - FPGA editor screenshot of the *apex2* circuit



Source: author

Figure 6.1(a) shows the area under test for the *apex2* circuit for CG-DMR with free placement. Copy_0 (colored in green) and copy_1 (colored in red) are roughly grouped together but there are elements of both in the whole area under test. Figure 6.1(b) shows the area under test for the *apex2* circuit for CG-DMR with delta placement. Copy_0 (colored in green) and copy_1 (colored in red) occupy non-overlapping areas and it can be seen that there is a copy_1 resource in the exact same X location as there is a copy_0 resource.

The design flow for CG-DMR with delta placement differs from the one shown in Figure 4.1, in that instead of beginning with the unconstrained unhardened post-synthesis model of the benchmark circuit; the delta placement starts with the creation of a post-mapping model with a constrained placement in that the unhardened post-synthesis circuit occupies only the bottom half configuration frame slices. For example, the unhardened *pdc* circuit was placed under the constraints:

```
INST "comb_benches_blif_pdc" AREA_GROUP = "cut_group";

AREA_GROUP "cut_group" RANGE = SLICE_X62Y140:SLICE_X99Y149;
```

The coordinates from Y140 to Y149 will be used by the fault tolerance C++ tool (step 1 in Figure 4.1) to place copy_0 and copy_1 will be placed at Y150 to Y159.

## 6.2 Experimental Results

The benchmarked circuits were synthesized with placement constraints to simulate a real-world device occupation scenario of 85 %. The achieved occupations are shown in Table 6.1. The column designated "DMR" shows the results for standard DMR and the column "RHST" shows the results for the architecture in the FPGA shown in Figure 3.3:

Table 6.1 - CG-DMR with delta placement benchmark occupation

| | **Occupation** | | | **Occupation** | |
|---|---|---|---|---|---|
| **Benchmark** | ***DMR*** | ***RHST*** | **Benchmark** | ***DMR*** | ***RHST*** |
| *alu 4b* | 84.58 % | 85.83 % | *ex1010* | 87.86 % | 89.11 % |
| *alu 32b* | 89.00 % | 68.75 % | *ex5p* | 96.25 % | 84.17 % |
| *alu 64b* | 85.06 % | 76.73 % | *misex3* | 88.13 % | 81.36 % |
| *apex2* | 83.23 % | 83.54 % | *pdc* | 83.55 % | 87.37 % |
| *apex4* | 82.75 % | 84.63 % | *seq* | 82.79 % | 86.63 % |
| *des* | 80.75 % | 80.38 % | *spla* | 75.31 % | 68.75 % |

Source: author

### 6.2.1 Area and Delay Costs

The delay and clock cycle overheads for CG-DMR with delta placement over regular DMR are presented in Table 6.2:

Table 6.2 - Area and delay results for CG-DMR with delta placement

| Benchmark | # POs | Area (LUTs) | | | Delay (ns) | | |
|---|---|---|---|---|---|---|---|
| | | ***DMR*** | ***Delta*** | ***Inc.*** | ***DMR*** | ***Delta*** | ***Inc.*** |
| *alu 4b* | 9 | 812 | 839 | 3.33 % | 6.49 | 7.30 | 12.38 % |
| *alu 32b* | 34 | 712 | 796 | 11.80 % | 8.14 | 9.63 | 18.35 % |
| *alu 64b* | 66 | 1497 | 1637 | 9.35 % | 9.62 | 11.21 | 16.57 % |
| *apex2* | 4 | 1598 | 1620 | 1.38 % | 7.28 | 9.32 | 28.12 % |
| *apex4* | 19 | 1324 | 1375 | 3.85 % | 7.34 | 8.57 | 16.80 % |
| *des* | 246 | 1292 | 1769 | 36.92 % | 6.91 | 11.04 | 59.77 % |
| *ex1010* | 11 | 984 | 1015 | 3.15 % | 6.44 | 6.94 | 7.78 % |
| *ex5p* | 64 | 308 | 443 | 43.83 % | 5.39 | 5.41 | 0.37 % |
| *misex3* | 15 | 1410 | 1451 | 2.91 % | 7.01 | 7.91 | 12.80 % |
| *pdc* | 41 | 2540 | 2684 | 5.67 % | 8.71 | 12.28 | 41.04 % |
| *seq* | 36 | 1722 | 1828 | 6.16 % | 7.24 | 9.02 | 24.54 % |
| *spla* | 47 | 482 | 579 | 20.12 % | 6.18 | 7.67 | 24.16 % |

Source: author

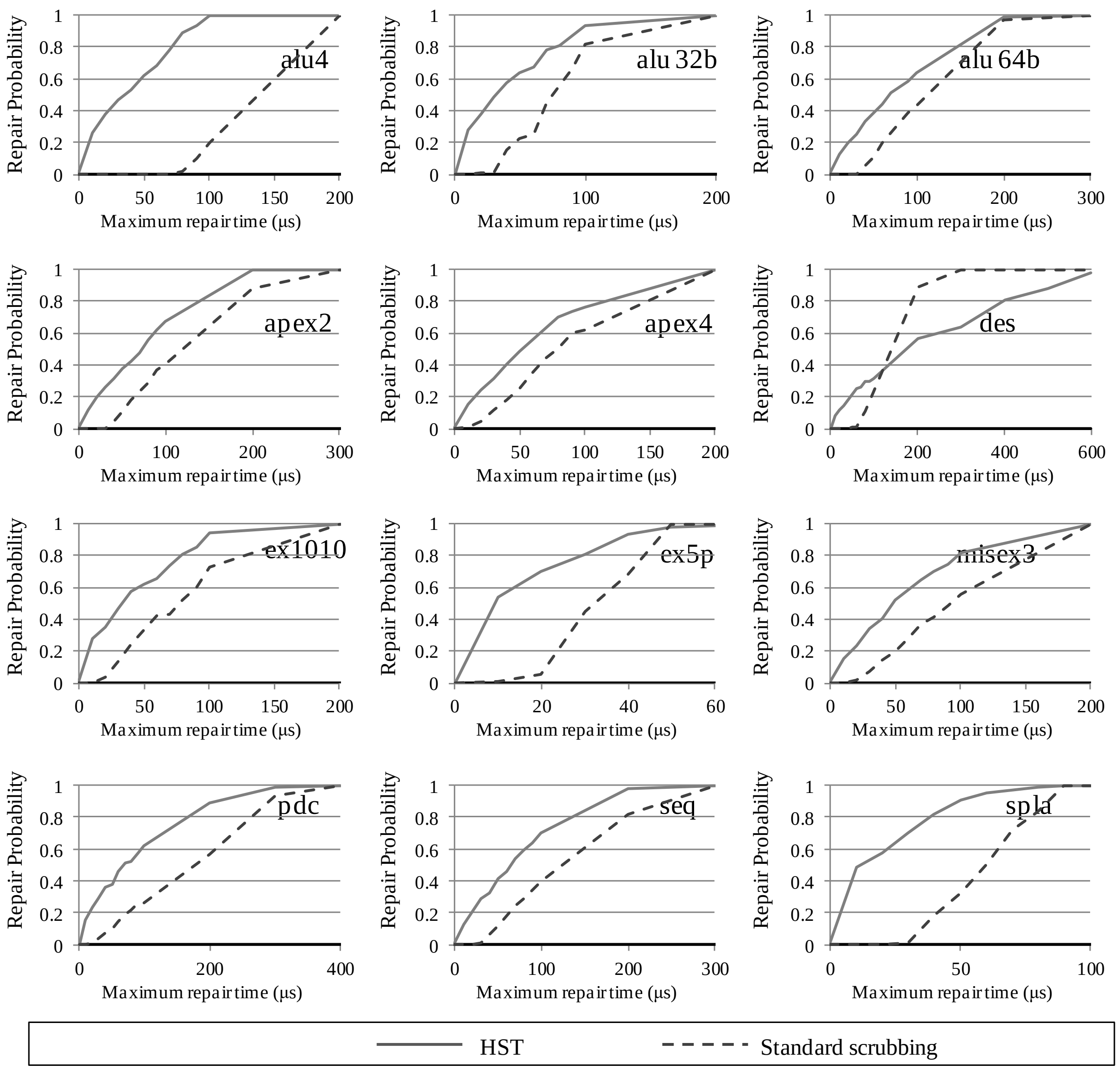
The average area overhead is 12.37 %. The presented results for area overhead show a large overhead for circuits with small area and many POs, such as *des* and *ex5p*. In GC-DMR, we choose to use one LUT for every compared bit on the RHST circuits and one LUT for every three compared bits for regular DMR. The opposite effect is true to circuits with large original areas and a few POs, such as *apex2*, *apex4*, *ex1010* and *misex3*, all of them with an area overhead smaller than 5 %. The overhead for the other circuits is closer to 10 %, suggesting acceptable costs. The area costs for the RHST circuits already include the cost for the ST tables (as in Figure 3.4). We choose to use the worst case for each circuit, among all synthesized tables (with different target slacks).

The average delay overhead is 21.89 %; this shows that the MAP and PAR tools had difficulty in achieving a good delay overhead. This can be attributed to the static placement of the LUTs; Figure 6.1 shows that when the MAP and PAR tools have the freedom to place all the elements of both copies, the resulting placement is very different from the delta placement, resulting in poor timing performance.

### 6.2.2 Repair Probability

The repair probability results are presented in Figure 6.2:

Figure 6.2 - Repair probabilities for CG-DMR with delta placement



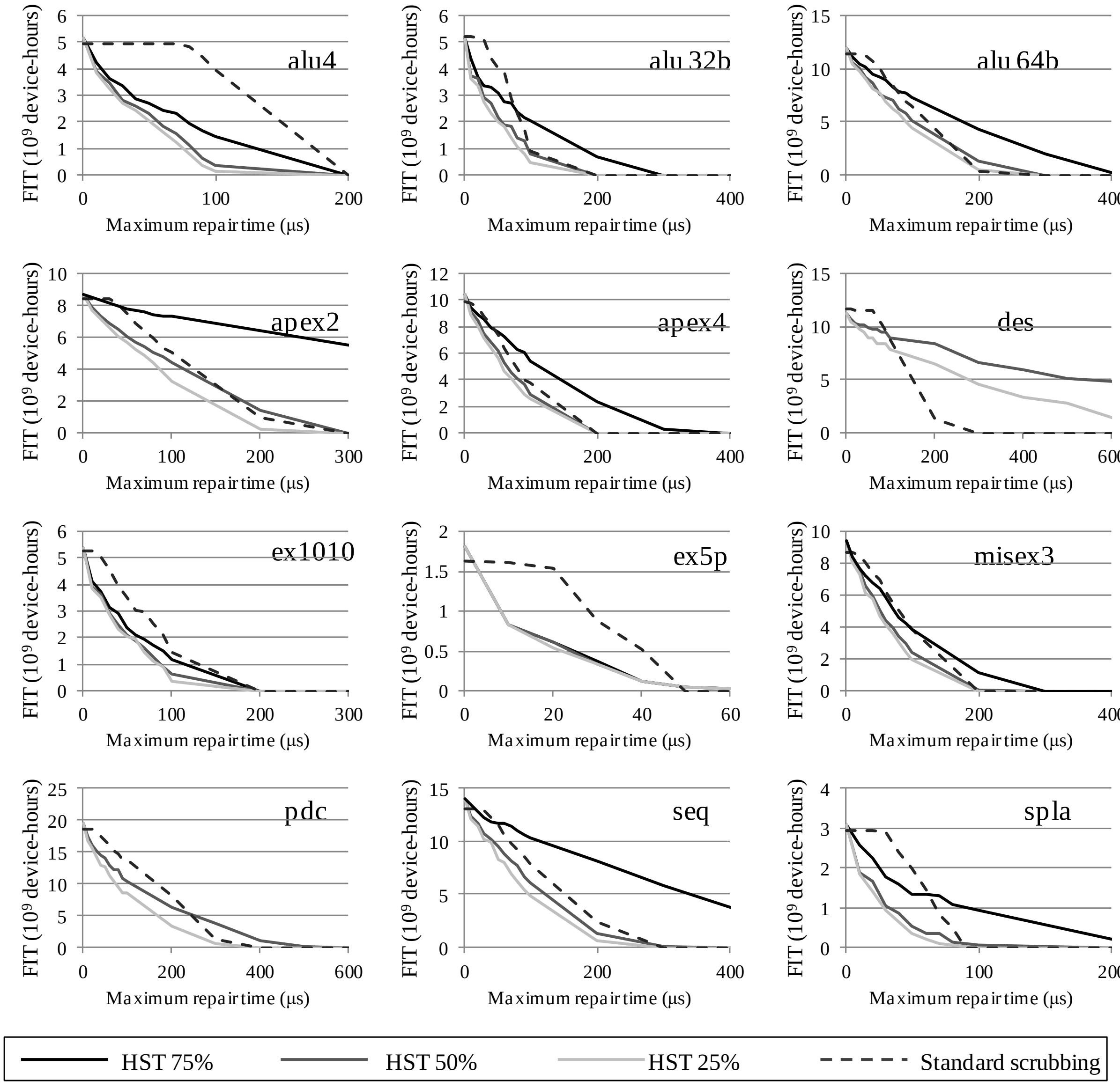
Source: author

The results presented show that with the exception of the *des* benchmark, CG-DMR with delta placement has an improved repair probability for all slacks. The poor performance for the *des* benchmark is due to the histograms being more distributed.

### 6.2.3 Failure-in-Time Results

FIT results for CG-DMR with delta placement are presented in Figure 6.3:

Figure 6.3 - FIT results for CG-DMR with delta placement



Source: author

The presented results show that FIT rates are higher than SG-DMR, due to the much larger clock overhead. In the case of the *pdc* and *pdc* circuits, the curve for the 75 % occupation scenario is not presented due to the circuits not respecting the real-time deadlines. The curves for the *des* benchmark are especially poor due to the poor repair probability curve shown in Figure 6.2. The curves for the *apex2*, *seq* and *spla* benchmarks show clearly the effect of the proportional degradation of the clock cycle according to time-slot occupation, explained in section 4.4, as the clock cycle overhead for these circuits was around 25 %, which is near the limit (33 %) for the 75 % occupation scenario. For the other benchmarks the FIT rates for the 75 % occupation scenario were much poorer than for the other two occupation scenarios. The FIT rates for the 25 % and 50 % occupation scenarios were lower than standard scrubbing for all slacks, with the exception of the *des* benchmark. It is clear that the high clock overhead took a heavy toll on the experimental results for FIT rates.

As already mentioned, the clock overhead results show that the MAP and PAR tools had severe difficulty achieving a good timing solution for the delta placement. The need of using placement constraints arises from the fact that the MAP and PAR tools are not aware of the placement taking into account reliability issues such as the delta placement.

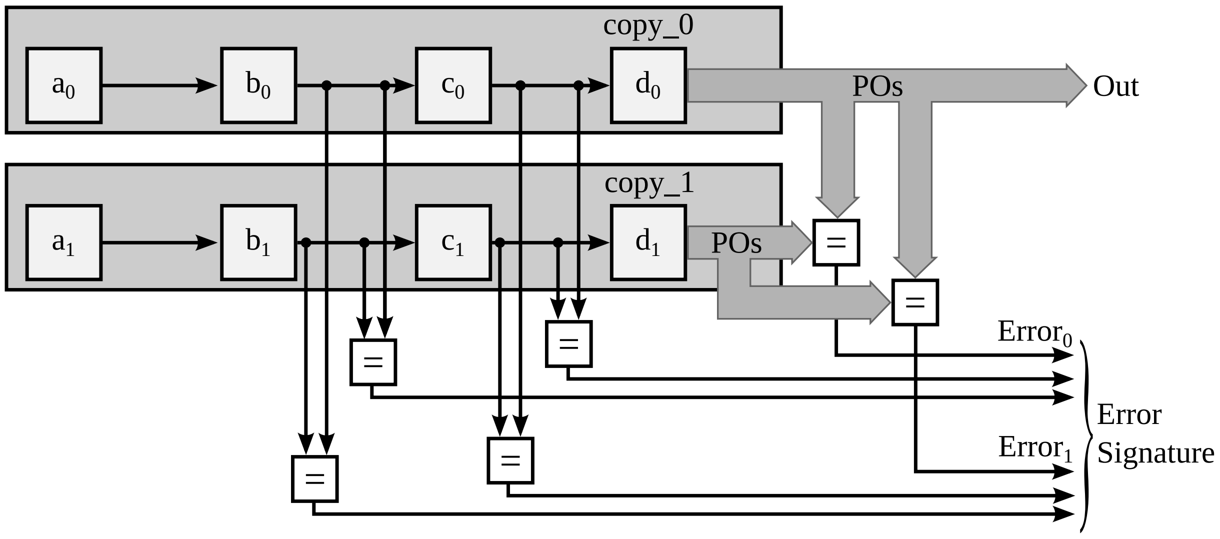
# 7 SELECTIVE-GRAINED DOUBLE MODULE REDUNDANCY

As can be seen from Figure 2.2(b), a fine-grained technique compares many signals, thus placing a burden on routing, this leads to a significant increase of the clock cycle length, as can be seen from the results presented in the work of Nazar (2015, p. 1116). The result is a reduced slack and thus lower repair probability, affecting negatively on FIT rates. Through the RHST the full-width signatures will be compressed and thus some information will be lost; some error signature bits carry more information than others, so it was wondered that if a large signature width is really necessary to begin with. Would it be possible to use just a few bits for comparison and still obtain good diagnostic information that could be used in the repair of real-time systems as presented in section 3?

## 7.1 Proposed Architecture

SG-DMR can be seen as a variation of FG-DMR in which not all, but some LUT pairs (functionally the same LUT in copy_0 and copy_1 of Figure 7.1) are selected for comparison. The diagnostic architecture is illustrated in Figure 7.1:

Figure 7.1 - Selective-grained diagnostic architecture



Source: author

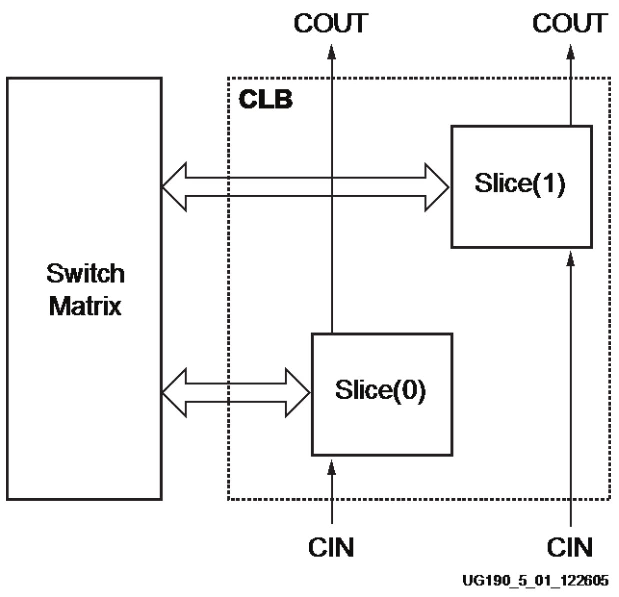
By using dual-rail comparison it is possible to detect errors in the comparators themselves. *Error*$_0$ and *Error*$_1$ are the coarse-grained error bits present in the regular DMR (i.e., comparison of primary outputs) that are also used in SG-DMR, they are needed in case none of the selective-grained comparators indicates an error. The error signature is composed of the concatenation of the bits of both copies of the selective-grained comparators and the coarse-grained error bits from the POs comparison.

The design flow to generate the SG-DMR circuit is similar to the one presented in Figure 4.1, but instead of directly creating the post-synthesis model from the unhardened circuit, the unhardened post-synthesis model is used to build a regular DMR circuit (Figure 3.2). This DMR model is used to create a post-mapping model, which in turn is analyzed by a C++ tool which generates the architecture shown in Figure 7.1. The rest of the workflow is the same already explained in section 4. To explain how the LUT pairs are selected, it is first necessary to present how a Virtex 5 FPGA is organized.

### 7.1.1 Virtex 5 Internal Organization

Virtex 5 FPGAs resources are organized in Configurable Logic Blocks (CLBs) (XILINX INC., 2012b). CLBs contain logic and sequential resources, organized in two *slices*, the first is called slice(0) and the second is called slice(1):

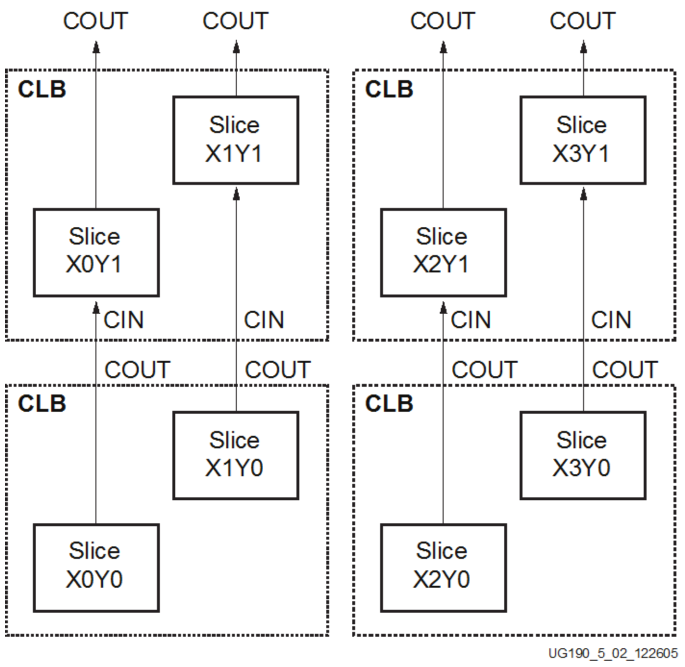
Figure 7.2 - Arrangement of Slices within the CLB



Source: Xilinx (2012b, p. 173)

Figure 7.2 shows that each slice has a carry-chain input (CIN) and output (COUT) and is connected to the routing matrix of the FPGA. Slices are uniquely identified within the device by an X coordinate, denoting a column number, and a Y coordinate, denoting a row number. As each CLB is composed of two slices, the first CLB has slices X0Y0 and X1Y0, the second X2Y0 and X3Y0, and so on, as shown in Figure 7.3. Virtex 5 configuration frames begin each 20 (twenty) rows.

Figure 7.3 - Row and Column Relationship between CLBs and Slices



Source: Xilinx (2012b, p. 174)

### 7.1.2 LUT Pair Selection

SG-DMR depends on having a "good" selection of LUT pairs to compare between copies, "good" being a criteria that yields repair probability curves that reach 100 % repair probability in a shorter period than regular DMR and hopefully the other techniques. On the other hand, this work is not about investigating the best possible LUT selection criteria for SG-DMR, so the criteria presented in this section is one of many possibilities, it was chosen as a logic criteria that would create concentrated histograms, thus the choice of selecting the nets with the *least* standard deviation, as this would lead to the selection of compact logic regions within the device.

To select the LUT pairs, the C++ tool first calculates for all nets in copy_0 and copy_1 the standard deviation of the X slice coordinate of the component that drives the net and the X slice coordinate of the components driven by the net. If *net[k]* is a net implemented in the copy_0 circuit, *cin0* is the component that drives *net[k]* in circuit copy_0, *cout0[n]* is one of the *N* components driven by *net[k]* in circuit copy_0. If *cin1* is the equivalent component of *cin0* but in circuit copy_1 and *cout1[n]* are the equivalent components of *cout0[n]* but in circuit copy_1, with $cin0_X$ being the X slice coordinate of *cin0*, $cout0_X[n]$ the X slice coordinate of *cout0[n]*, $cin1_X$ being the X slice coordinate of *cin1*, $cout1_X[n]$ the X slice coordinate of *cout1[n]*, then the mean X coordinate of *net[k]* ($M_x(net[k])$) is given by:

$$M_X(net[k]) = \frac{\sum_{i=0}^{N}(cout0_X[i]+cout1_X[i])+cin0_X+cin1_X}{2\times(N+1)} \tag{8}$$

The standard deviation σ of the X coordinate of *net[k]* ($\sigma_x$*(net[k])*) is given by:

$$\sigma_X(net[k]) = \sqrt{\frac{\sum_{i=0}^{N}[(M_X(net[k])-cout0_X[i])^2+(M_X(net[k])-cout1_X[i])^2]+(M_X(net[k])-cin0_X)^2+(M_X(net[k])-cin1_X)^2}{2\times(N+1)}} \tag{9}$$

It was calculated for each net:

$$\sigma'_X(net[k]) = \frac{\sigma_X(net[k])}{2\times(N+1)} \tag{10}$$

The C++ tool then creates a list of the 6 nets with the *smallest* $\sigma'_X$ from all $K$ nets in copy_0. These nets are the nets selected for comparison.

The number of selected LUTs for all circuits is 6. This number was chosen to yield a *maxSize* that is a multiple of 7, the seventh signal being the coarse-grained error signal ($\text{Error}_0$ and $\text{Error}_1$ in Figure 7.1). As the comparison is double-rail, all the error signatures for SG-DMR have a width of 14 bits. When processed by the RHST C++ tool (step 6 in Figure 4.1) with a *maxSize* parameter of 7, there will be only one round of compression and all resulting RHST signatures will have a width of 7 bits.

## 7.2 Experimental Results

As explained in section 4, the post-synthesis model was processed by a C++ tool that creates a new VHDL file with part of the architecture shown in Figure 7.1. The placement constraints were created in a way that the CUT would be placed in the beginning of a configuration row (in the case of a Virtex 5 device, the Y placement coordinate is a multiple of 20) and in an integer number of CLB columns (X placement coordinate is a multiple of 2). The attained occupations are shown in Table 7.1. The column designated “DMR” shows the results for standard DMR and the column “RHST” shows the results for the architecture in the FPGA shown in Figure 7.1:

Table 7.1 - SG-DMR benchmark occupation

| | Occupation | | | Occupation | |
|---|---|---|---|---|---|
| **Benchmark** | ***DMR*** | ***RHST*** | **Benchmark** | ***DMR*** | ***RHST*** |
| *alu 4b* | 84.58 % | 86.15 % | *ex1010* | 87.86 % | 75.97 % |
| *alu 32b* | 89.00 % | 90.50 % | *ex5p* | 96.25 % | 66.67 % |
| *alu 64b* | 85.06 % | 85.74 % | *misex3* | 88.13 % | 88.88 % |
| *apex2* | 83.23 % | 83.85 % | *pdc* | 83.55 % | 83.95 % |
| *apex4* | 82.75 % | 83.50 % | *seq* | 82.79 % | 83.37 % |
| *des* | 80.75 % | 81.50 % | *spla* | 75.31 % | 77.19 % |

Source: author

As the SG-DMR circuit was generated from the post-synthesis model of the DMR circuit by selecting some nets to compare, it is important that the component placement is equivalent between the post-synthesis model and the SG-DMR circuits. To achieve this, all LUTs in the hardened SG-DMR circuits were placed at the same locations as they were placed in the post-synthesis model by the use of placement constrains. These constraints are automatically generated by the C++ tool that generates the SG-DMR circuits (step 1 in Figure 4.1).

### 7.2.1 Area and Delay Costs

The delay and clock cycle overheads for SG-DMR over regular DMR are presented in Table 7.2:

Table 7.2 - Area and delay results for SG-DMR

| Benchmark | # POs | Area (LUTs) | | | Delay (ns) | | |
|---|---|---|---|---|---|---|---|
| | | *DMR* | *SG-DMR* | *Inc.* | *DMR* | *SG-DMR* | *Inc.* |
| *alu 4b* | 9 | 812 | 865 | 6.53 % | 6.49 | 7.22 | 11.24 % |
| *alu 32b* | 34 | 712 | 767 | 7.72 % | 8.14 | 8.36 | 2.78 % |
| *alu 64b* | 66 | 1497 | 1562 | 4.34 % | 9.62 | 10.03 | 4.22 % |
| *apex2* | 4 | 1598 | 1651 | 3.32 % | 7.28 | 7.61 | 4.60 % |
| *apex4* | 19 | 1324 | 1377 | 4.00 % | 7.34 | 8.09 | 10.23 % |
| *des* | 246 | 1292 | 1356 | 4.95 % | 6.91 | 7.47 | 8.14 % |
| *ex1010* | 11 | 984 | 1151 | 16.97 % | 6.44 | 7.22 | 12.08 % |
| *ex5p* | 64 | 308 | 359 | 20.45% | 5.39 | 5.40 | 0.24 % |
| *misex3* | 15 | 1410 | 1470 | 4.26 % | 7.01 | 7.43 | 5.86 % |
| *pdc* | 41 | 2540 | 2589 | 1.93 % | 8.71 | 9.69 | 11.28 % |
| *seq* | 36 | 1722 | 1788 | 3.83 % | 7.24 | 8.12 | 12.09 % |
| *spla* | 47 | 482 | 539 | 11.83 % | 6.18 | 6.41 | 3.75 % |

Source: author

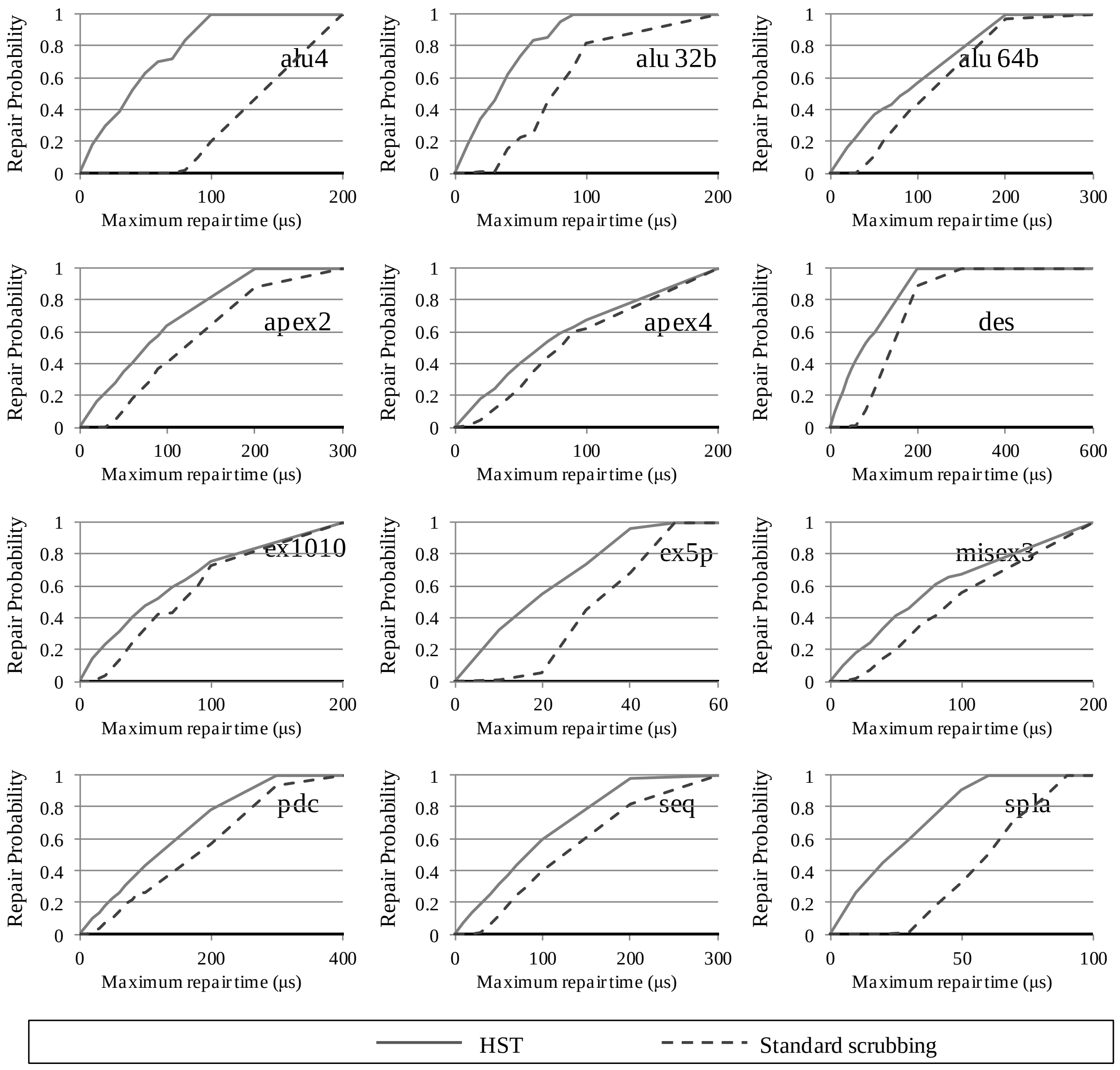
The area overheads already consider the cost of the Signature Translation (ST) tables needed to implement the architecture shown in Figure 3.4. The average area overhead is 7.51 % and the average delay overhead is 7.21 %. The results were obtained using the same placements constraints as the fault injection circuits, this caused some circuits to be larger than expected. To create the internal comparators shown in Figure 7.1, a single LUT was used, so as 6 signals were selected to comparison, SG-DMR was expected to use 12 LUTs over regular DMR. This was true to most circuits, with the exception of *alu4* (15 LUTs over DMR), *ex1010* (110 LUTs over DMR) and *exp5* (24 LUTs over DMR). We attribute these variations to optimizations done by the MAP tool when faced with many placement constrains. With the exception of the *ex1010*, *ex5p* and to a lesser degree the *spla* circuits, the area overhead was below 10 %. In the case of the *ex5p* and *spla* circuits, the ST table overhead is more significant as the these circuits are smaller, the opposite can be said for the *pdc* circuit, as being the largest the ST tables overhead is less significant.

SG-DMR presented an average delay overhead of 7.21 %, with higher values for the *alu4*, *apex4*, *ex1010*, *pdc* and *seq* benchmarks. It is interesting to notice that the smallest circuit (*ex5p*) has the smallest overhead, the *spla* benchmark is the second smallest and has the second smallest overhead and so on; it can be seen that there is a rough tendency of smaller circuits to have smaller overheads. This is attributed to the use of placement constraints for all LUTs in order to replicate the placement of the original regular DMR circuit in that larger benchmarks this lead to poor timing.

### 7.2.2 Repair Probability

As we are comparing different diagnostic techniques, it is important to have some unit to compare purely the diagnostic precision offered in each case. The results for repair probability can be used for this, as they do not take into account the delay overhead or the number of sensitive bits as do FIT rates. The repair probability for each benchmark circuit is shown in Figure 7.4:

Figure 7.4 - Repair probabilities for SG-DMR



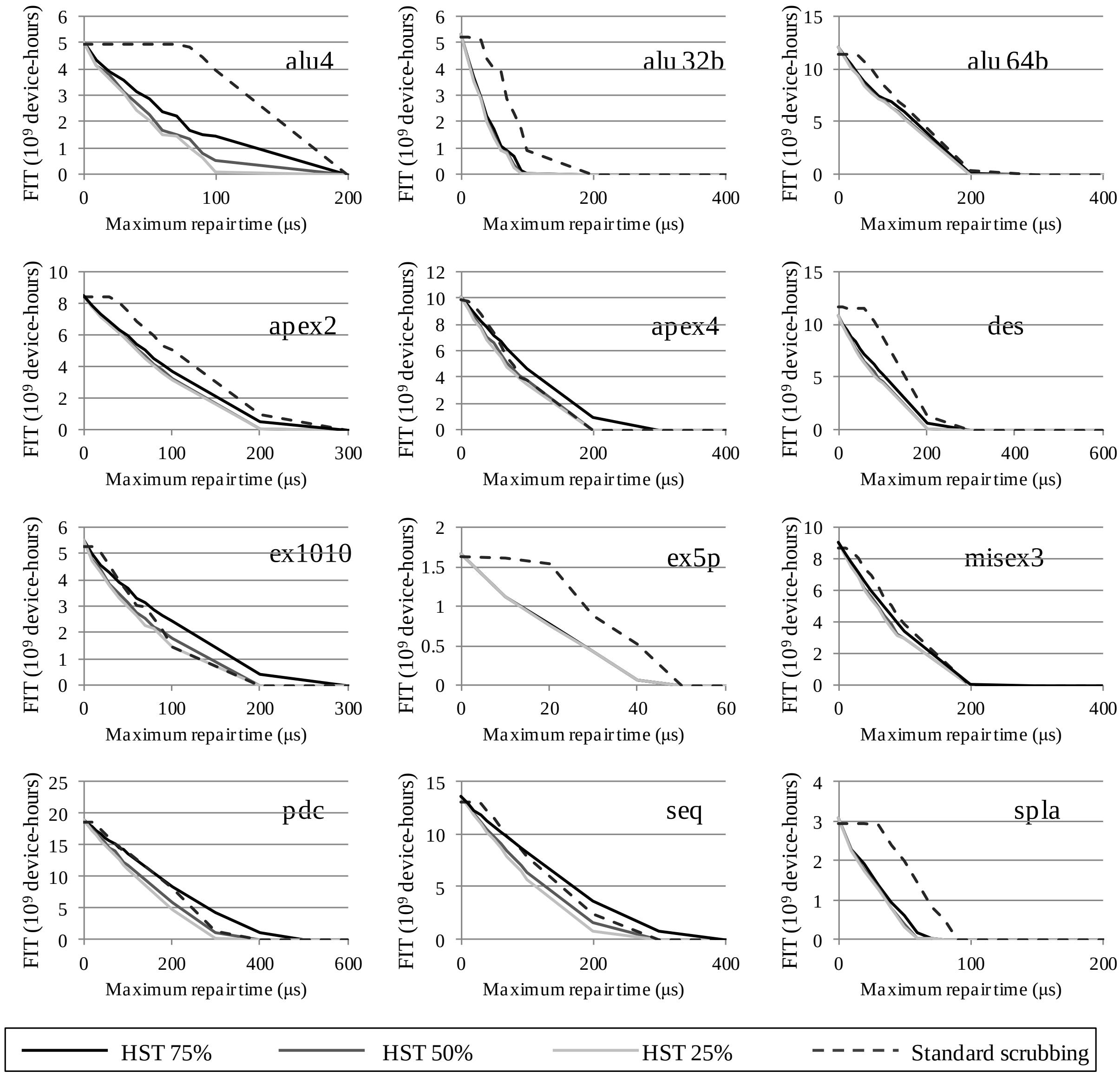
Source: author

The results show that SG-DMR was able to offer a better repair probability over standard scrubbing for all circuits. The curves for the *alu4*, *alu32*, *ex5p* and *spla* circuits show better diagnostic for these circuits, this is probably due to these circuits having the smallest area, as the number of compared signals was 6 for all circuits, for smaller circuits this number is more significant than for larger ones. Following this line of though, the *ex5p* circuit should have the best diagnostic, indicating that other factors play important roles and could be exploited in future works.

### 7.2.3 Failure-in-Time Results

As stated in section 4.4, FIT rates are a good indicator as the effectiveness of different architectures, techniques and technologies; because they take into account not only the diagnostic precision, but the delay introduced and the number of sensitive bits. The FIT results for SG-DMR are shown in Figure 7.5:

Figure 7.5 - FIT results for SG-DMR



Source: author

It is important to notice from the presented results that SG-DMR was able to respect all the deadlines of the real-time system, indicating a small clock overhead. For the circuits in which SG-DMR presented good diagnostic precision (*alu4*, *alu32*, *ex5p*, *spla*), the obtained FIT rates are considerably smaller than the ones for standard scrubbing across all slacks and for all occupation scenarios. For the other circuits, SG-DMR presented only modest gains over standard scrubbing.

# 8 CRITICAL ANALYSIS OF THE EXPERIMENTAL RESULTS

This dissertation presented results for three different diagnostic architectures to be used in a real-time intelligent scrubbing architecture. This chapter will compare the results presented in this dissertation between themselves and with those presented in the work of Nazar (2015), which used a fine-grained DMR diagnostic tool in the same manner used in this work. The results of both works will be compared in terms of area and delay costs, diagnostic accuracy by means of the repair probability results and FIT rates. Lastly, a statistical analysis is carried out to identify which factor between diagnostic precision, clock overhead and critical bits is more significant in determining FIT rates.

An important factor that must be kept in mind when comparing the results from this dissertation with those of Nazar (2015) is that the results obtained in this work strived to use a 85 % area occupation for the benchmark circuits, while the work of Nazar (2015) used a fixed area under test that yielded a lower density. The importance of a greater density can be seen from the results for the Best Static starting frame address, presented in section 8.2. To directly compare FG-DMR with the other diagnostic techniques the results for the former should be generated again using an average density of 85 % occupation of the area under test. Nevertheless, they are presented here and compared as an alternative diagnostic method and for the sake of completeness.

## 8.1 Comparison Between Diagnostic Architectures

### 8.1.1 Area and Delay Costs Results Comparison

The area costs of each diagnostic architecture compared to standard DMR are shown in Table 8.1, with the smallest overhead for each circuit in bold. The column “DMR” is the area, in LUTs, for regular DMR. The other columns, SG-DMR, CG_DMR with delta placement and CG-DMR with free placement have their values relative to the cost of regular DMR; the lowest values for each benchmark shown in bold:

Table 8.1 - Area values over regular DMR

| | **Area (LUTs)** | **Area (Over DMR)** | | | |
|---|---|---|---|---|---|
| **Benchmark** | ***DMR*** | ***SG-DMR*** | ***Delta*** | ***CG-DMR*** | ***FG-DMR*** |
| alu 4b | 812 | 106.5% | **103.3%** | **103.3%** | 117.7% |
| *alu 32b* | 712 | **107.7%** | 111.8% | 111.5% | 112.9% |
| *alu 64b* | 1497 | **104.3%** | 109.4% | 109.4% | 110.8% |
| *apex2* | 1598 | 103.3% | **101.4%** | **101.4%** | 110.8% |
| *apex4* | 1324 | 104.0% | **103.9%** | **103.9%** | 109.4% |
| *des* | 1292 | 105.0% | 136.9% | 136.8% | **102.2%** |
| *ex1010* | 984 | 117.0% | **103.2%** | **103.2%** | 117.4% |
| *ex5p* | 308 | 120.5% | 143.8% | 143.8% | **106.2%** |
| *misex3* | 1410 | 104.3% | **102.9%** | **102.9%** | 109.9% |
| *pdc* | 2540 | **101.9%** | 105.7% | 103.5% | 111.1% |
| *seq* | 1722 | **103.8%** | 106.2% | 104.8% | 109.7% |
| *spla* | 482 | 111.8% | 120.1% | 120.1% | **110.8%** |

Source: author

SG-DMR has the lowest area overheads for the *alu32*, *alu64*, *pdc* and *seq* benchmarks; CG-DMR with delta placement and free placement have the lowest area overheads for the *alu4*, *apex2*, *apex4*, *ex1010* and *misex3* benchmarks; FG-DMR has the lowest area overheads for the *des*, *ex5p* and *spla* benchmarks. It is important to notice that SG-DMR and CG-DMR (both) have the lowest area overhead for 9 of the 12 benchmarks, with CG-DMR with free placement having the lowest area overhead for 5 of the 12 benchmarks and an overhead slightly larger than SG-DMR and FG-DMR in other 5 circuits. Only on the *des* and *ex5p* CG-DMR has a much larger area overhead, due to the poor relation between circuit area and number of POs. SG-DMR was not able to provide a low overhead for the *alu4*, *ex1010* and *ex5p* benchmarks due to additional resources allocated by the MAP tool.

Delay overhead values compared with regular DMR are shown in Table 8.2; the lowest values for each benchmark shown in bold:

Table 8.2 - Delay values over regular DMR

| | **Delay (ns)** | **Delay (Over DMR)** | | | |
|---|---|---|---|---|---|
| **Benchmark** | ***DMR*** | ***SG-DMR*** | ***Delta*** | ***CG-DMR*** | ***FG-DMR*** |
| alu 4b | 6.49 | 111.2 % | 112.4 % | **101.3 %** | 115.8 % |
| *alu 32b* | 8.138 | 102.8 % | 118.3 % | **100.2 %** | 120.8 % |
| *alu 64b* | 9.62 | **104.2 %** | 116.6 % | 110.6 % | 124.5 % |
| *apex2* | 7.275 | 104.6 % | 128.1 % | **104.1 %** | 149.2 % |
| *apex4* | 7.338 | 110.2 % | 116.8 % | **107.0 %** | 133.3 % |
| *des* | 6.908 | **108.1 %** | 159.8 % | 113.8 % | 122.5 % |
| *ex1010* | 6.438 | 112.1 % | 107.8 % | **100.2 %** | 117.9 % |
| *ex5p* | 5.389 | 100.2 % | 100.4 % | **98.4 %** | 103.7 % |
| *misex3* | 7.014 | 105.9 % | 112.8 % | **104.2 %** | 143.1 % |
| *pdc* | 8.706 | 111.3 % | 141.0 % | **104.0 %** | 122.6 % |
| *seq* | 7.244 | 112.1 % | 124.5 % | **104.9 %** | 138.6 % |
| *spla* | 6.179 | **103.8 %** | 124.2 % | 104.7 % | 112.2 % |

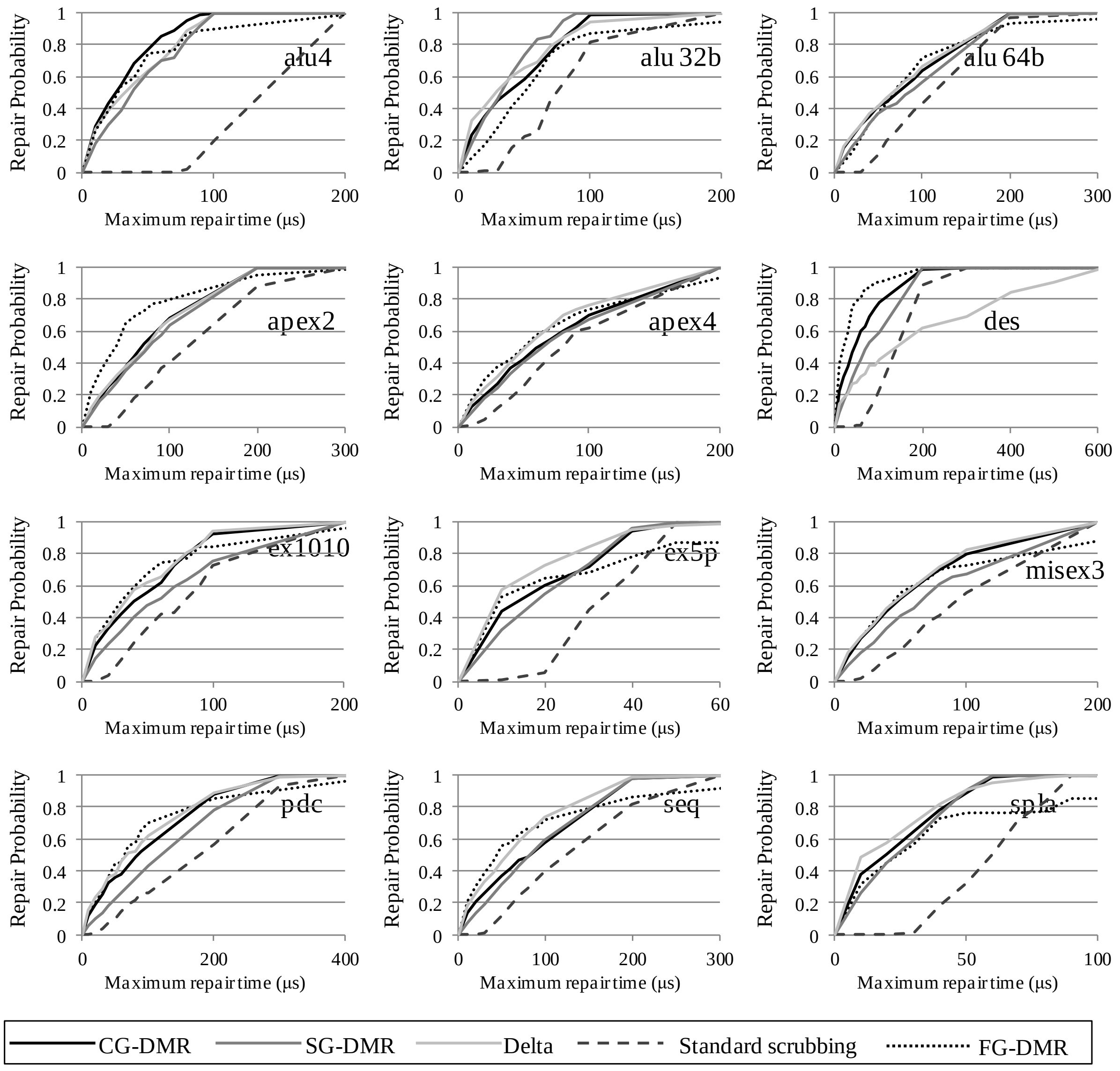
Source: author

For most circuits, CG-DMR with free placement had the lowest clock overhead, except for the *alu 64b*, *des* and *spla* benchmarks, in which SG-DMR had the lowest overhead. CG-DMR with delta placement suffers from poor placement, in which the MAP and PAR tools could not achieve a low-delay solution, even if the circuits are architecturally the same as CG-DMR with free placement, being the solution with the largest average overhead. FG-DMR has a much larger number of compared signals and thus a denser routing, so the MAP and PAR tools are not able to achieve a low overhead solution. These results confirm the initial premise of CG-DMR with free placement, that the MAP and PAR tools would be able to achieve good timing performance if given freedom to place the components. This is important that not only a low clock overhead solution preserves the performance of the original circuit, but also achieves the largest slack possible.

### 8.1.2 Repair Probability Results Comparison

The results for repair probabilities for all diagnostic techniques are presented and compared in order to verify if a particular diagnostic is clearly superior to the others. Figure 8.1 presents the repair probabilities curves:

Figure 8.1 - Repair probabilities for all techniques



Source: author

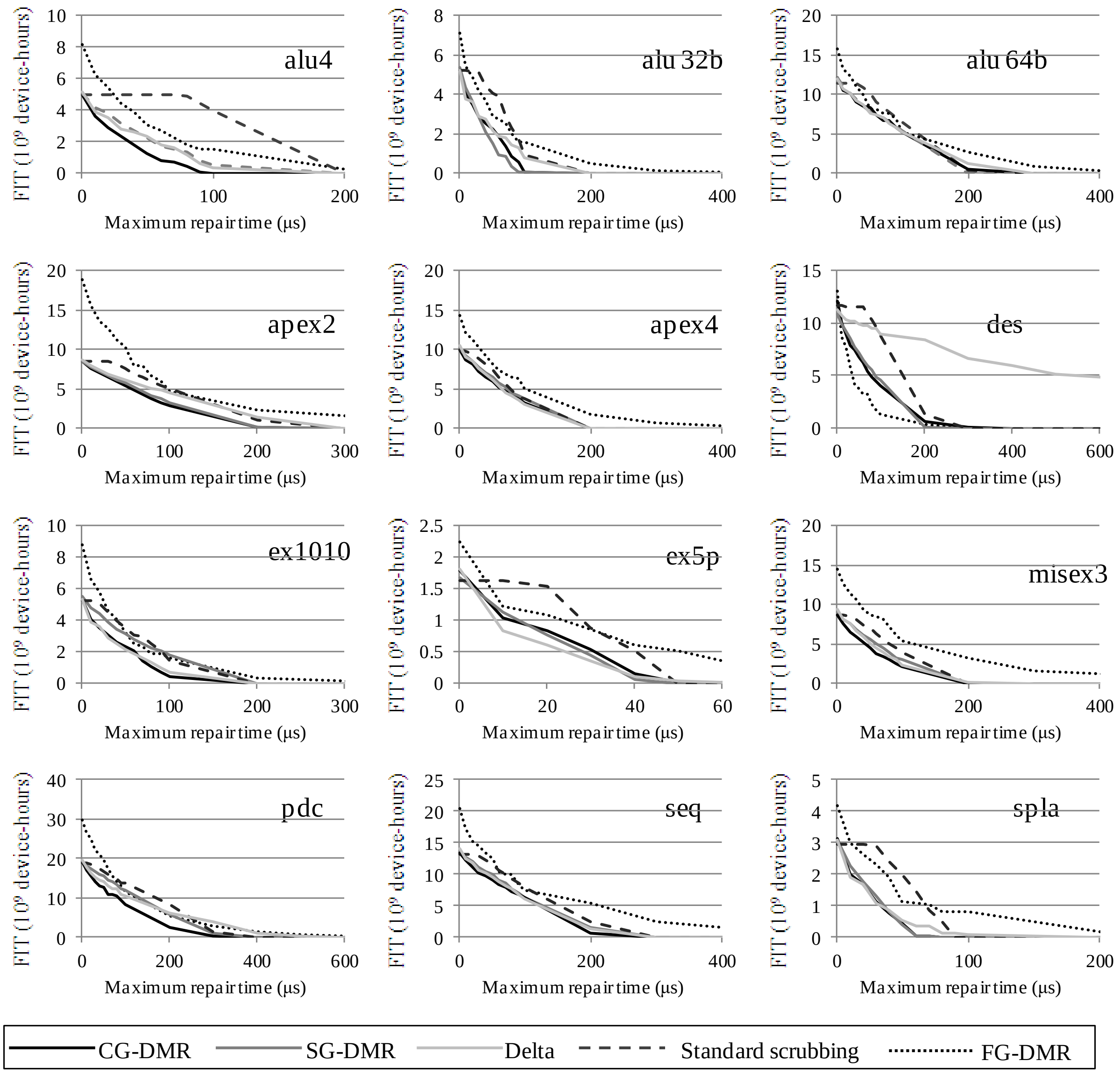
It can be seen that FG-DMR, by using all available internal information of the circuit, is able to achieve the highest repair probabilities for most benchmarks. Another point is that the curves for FG-DMR are near the curves for the other diagnostic techniques for most benchmarks, indicating that is no clear winner when it comes to diagnostic precision; on the other hand, FG-DMR results were obtained with lower occupation densities, so an interesting prospect is to obtain new results for FG-DMR with a high occupation area under test.

It is interesting to notice that the best diagnostic for the *alu4* comes from CG-DMR with free placement, which does not uses internal information of the circuit. SG-DMR does not perform badly, and a future work could be to explore other pair selection criteria.

## 8.1.3 FIT Rates Results Comparison

The FIT rate results are the most complete, as they take into account not only diagnostic precision, but clock overhead and number of sensitive bits for each technique.

Figure 8.2 – FIT rates for all circuits



Source: author

Figure 8.2 shows that FG-DMR has larger FIT rates than the coarse diagnostic techniques, this is due to FG-DMR having many more sensitive bits, indicated by the FIT rates for 0 µs.

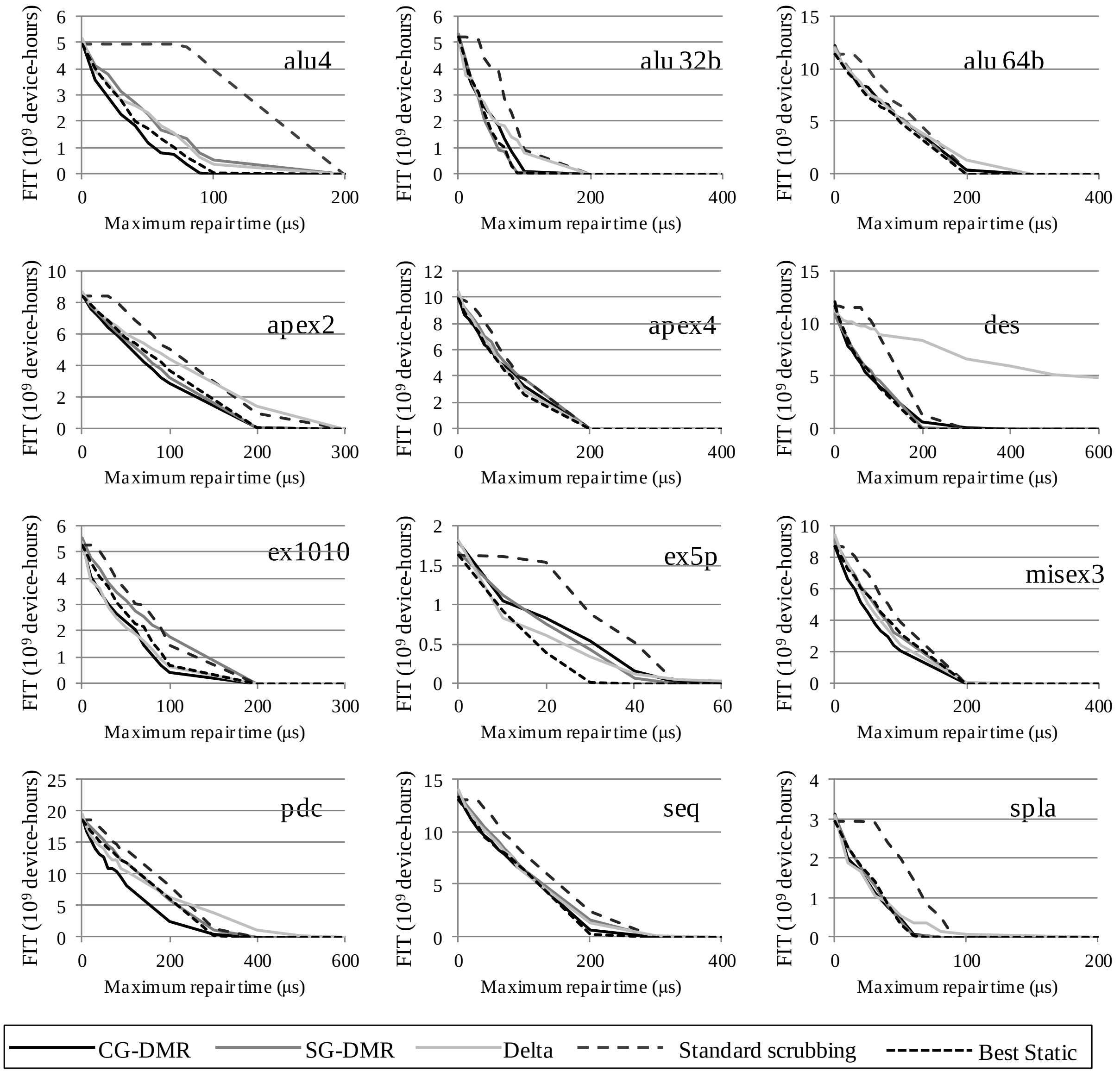
The FIT rates presented in Figure 8.2 show that CG-DMR with free placement has the lowest rates for almost all benchmarks; CG-DMR with delta placement presented the lowest rates for the *ex5p* benchmark, SG-DMR presented the lowest rates for the *alu 32b* benchmark and FG-DMR presented the lowest rates for the *des* benchmark. This confirms the initial hypothesis of this dissertation that diagnostic precision is one factor and poorer diagnostic could be compensated by low overhead solutions found in coarse-grained diagnostic.

It is of particular importance to compare the curves for standard scrubbing in Figure 8.2 with those presented in Nazar (2015, fig. 14) to understand how important is the effect of a high occupation. If we use the *ex5p* benchmark as an example, the standard scrubbing curve in Nazar (2015, fig. 14) reaches 0 for a slack of 200 µs, while in Figure 8.2 the same happens at 50 µs, indicating a much more compact circuit. The same happens for all other benchmarks, *ex5p* being the most striking case, indicating that gains can be achieved even with simple techniques if a high occupation is maintained.

## 8.2 Best Static Starting Frame

Figure 8.3 presents the FIT rates for CG-DMR, SG-DMR, regular DMR and Best Static:

Figure 8.3 - FIT rates with Best Static



Source: author

It can be seen that Best Static starting frame is able to achieve FIT rates comparable to the other diagnostic techniques. In the case of the *ex5p* benchmark, it achieves the lowest FIT rates among all techniques. This is an important result, as it indicates the importance of high occupation of the device in concentrating the signature histograms and thus providing a clearer choice to the repair mechanism.

## 8.3 FIT Results Analysis

The efficiency of the diagnostic techniques is usually performed by comparing the FIT rate plots. The FIT rate plots are a useful tool to visualize the different FIT rates according to fixed deadlines, but to evaluate the overall performance across all deadlines and circuits by visually comparing the FIT rate curves is difficult and imprecise. To numerically represent the effectiveness of a diagnostic technique the area under the FIT rate curves was chosen. The area under the FIT rates curves was calculated using Simpson's Rule implemented in the SciPy package (MILLMAN; AIVAZIS, 2011). For CG-DMR (both), SG-DMR and FG-DMR it were used the 50 % occupation curves as an average point of comparison. The results are presented in Table 8.3, with the *smallest* value for each benchmark indicated in bold:

Table 8.3 - Area under FIT rates curves

| **Benchmark** | **Area Under FIT Curves** | | | | | |
|---|---|---|---|---|---|---|
| | ***Standard DMR*** | ***SG-DMR*** | ***Delta*** | ***CG-DMR*** | ***FG-DMR*** | ***Best Static*** |
| alu 4b | 641.51 | 241.16 | 225.70 | **157.17** | 463.76 | 172.78 |
| *alu 32b* | 363.49 | 191.70 | 245.29 | **190.97** | 475.22 | 197.77 |
| *alu 64b* | 1276.04 | 1053.11 | 1149.55 | 1083.52 | 1462.86 | **957.08** |
| *apex2* | 1045.18 | 694.50 | 958.81 | **660.52** | 1670.14 | 739.70 |
| *apex4* | 884.09 | 815.61 | 717.52 | 731.79 | 1293.89 | **673.27** |
| *des* | 1610.91 | 937.42 | 4238.78 | 926.84 | **547.67** | 871.72 |
| *ex1010* | 386.55 | 393.98 | 248.46 | **233.65** | 491.66 | 283.18 |
| *ex5p* | 53.74 | 31.67 | 28.20 | 34.24 | 72.32 | **20.83** |
| *misex3* | 817.89 | 700.03 | 617.58 | **543.60** | 1772.17 | 685.22 |
| *pdc* | 3267.59 | 2744.81 | 3010.05 | **1836.29** | 3438.66 | 2618.20 |
| *seq* | 1694.35 | 1408.04 | 1352.74 | 1268.43 | 2657.82 | **1226.92** |
| *spla* | 173.55 | 80.21 | 83.86 | **74.79** | 232.57 | 80.94 |

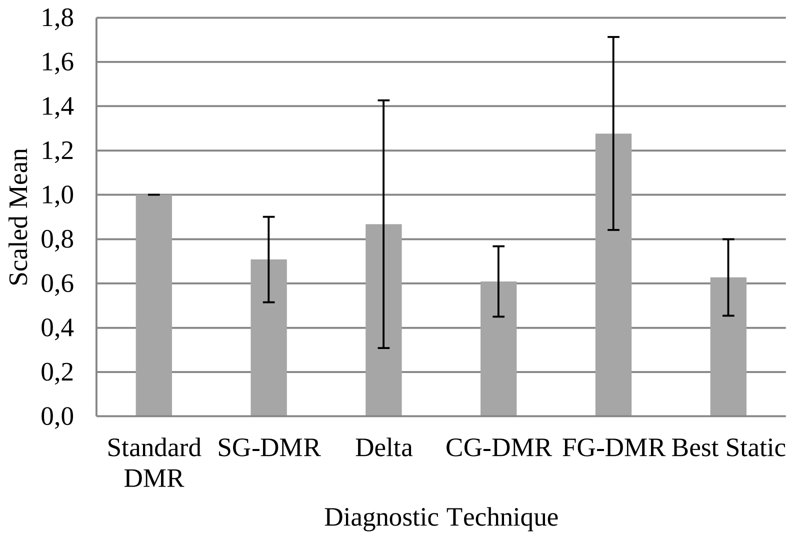
Source: author

CG-DMR with free placement achieved the lowest FIT rates for the *alu4*, *alu 32b*, *apex2*, *ex1010*, *misex3*, *pdc* and *spla* benchmarks. The Best Static approach achieved the lowest FIT rates for the *alu 64b*, *apex4*, *des*, *ex5p* and *seq* benchmarks, while FG-DMR achieved the lowest value for the *des* benchmark.

To compare the different techniques and, maybe, extrapolate the behavior of each technique to other circuits in general, a statistic analysis is required. The idea is to discover if the data obtained by the experiments with this set of benchmarks allows to indicate a clear choice of a technique to be applied to an unknown circuit, different from the benchmarks already used in this work.

It can be seen from the data presented in Table 8.3 that the values vary greatly across different benchmarks, i.e. the FIT values for the *ex5p* benchmark vary from 20 to 72, while the same values for the *pdc* benchmark vary from 1836 to 3438, so it is not possible to directly use the raw values to calculate the mean or variance of the data. To address this, all values from Table 8.3 were divided by the values in the standard DMR column. It was then possible to use this scaled data to calculate the mean and standard deviation, shown in Figure 8.4, with the mean for each technique represented by the solid grey bar and the standard deviation represented by the solid black line.

Figure 8.4 - Mean and standard deviation per technique



Source: author

Standard DMR has a standard deviation of 0 since it is a unitary column; CG-DMR with Delta placement showed the larger standard deviation value. CG-DMR with free placement and Best Static have the lowest mean and lowest standard deviation of all techniques; meaning that, for most circuits, they are the best choice, but the high standard deviation of CG-DMR with Delta placement indicates it can not be ruled out as the best choice for some circuits. Figure 8.4 also supports the conclusion that FG-DMR is the worst diagnostic technique in terms of FIT rates and thus circuit repair.

The data was subjected to analysis of variance (MONTGOMERY, 2001). The results showed that the benchmark had no significant effect on FIT rates values ($p$ = 0.243), however the different techniques presented statistically different values for the results ($p$ < 0.0001). So, the means of the FIT rates values from the different techniques were compared by Tukey’s Test, showing that CG-DMR and Best Static starting addresses presented markedly lower FIT rates values than the other techniques (respectively $p$ = 0.053 and $p$ = 0.074 compared to standard DMR), but did not show significant difference between them. Since the results proven to be independent of the benchmark, the efficiency of each diagnostic technique can be extrapolated beyond the benchmarks used in this dissertation. Details to the statistical analysis are presented in Appendix A.

Based on results, it is not possible to point to a clear winner between GC-DMR or Best Static as which one is the best technique in general. For some benchmarks, GC-DMR had the lowest FIT rates at a moderate cost in terms of area and clock overhead. It proved to be ill suited to small circuits with many POs, as it will have a large area overhead; for these cases Best Static is a better choice. It is important to notice the Best Static results can be calculated based on the fault injection results for the other techniques, so a designer could use CG-DMR to generate the results and then use a software tool (step 6 of Figure 4.1) to calculate the FIT rates for Best Static and then compared these with CG-DMR for the specific circuit to be protected.

# 9 CONCLUSIONS AND FUTURE PERSPECTIVES

The work carried out in this dissertation analyzed different diagnostic techniques applied to the problem of maximizing the repair probability of FPGAs used in real-time systems. SG-DMR, CG-DMR with delta placement and CG-DMR with free placement were applied to a set of combinational benchmark circuits. Area and clock cycle overheads were measured and compared against regular DMR, while FIT rates were compared against standard scrubbing. The same results were again compared against FG-DMR and Best Static starting frame. A statistical analysis was carried out to extrapolate the FIT results to more circuits and determine which diagnostic technique is the most promising.

The work in this dissertation can conclude that if a circuit implement in SRAM-based FPGA achieves a occupation of approximately 85 %, the best diagnostic techniques are CG-DMR with free placement and Best Static, with CG-DMR with free placement offering the smallest FIT rates for 7 of the 12 benchmarks; Best Static was able to offer the smallest FIT rates for 4 benchmarks, with the added advantage of having negligible cost over regular DMR. Both techniques do not differ statistically, thus if a diagnostic technique presented in this dissertation is to be chosen blindly to be applied to a new circuit, if an device occupation of around 85 % can be maintained, then Best Static starting address would be the one selected by its performance/cost relation.

This work indicated several promising research opportunities.  The study of critical-path delay violations by errors in the routing matrix of the FPGA is important because if true, some errors would leave the functionality of a circuit intact, but change the delay on one or more signal lines, this can cause some signals failing to be registered by flip-flops. The improvement of the RHST compression algorithm is also interesting, as would provide improved diagnostics for the same costs. This work used fixed slacks, in that variable slacks would have to be considered in their worst case, so different strategies could be studies to tackle variable slacks; one possibility being the use of PR to have different ST tables stored in a low-cost mass memory and loading the most appropriate table on-demand. The investigation of different criteria for the selection of LUT pairs in SG-DMR might indicate new possibilities of obtaining low-cost diagnostic. The study of the techniques presented in this applied to a soft-core CPU is a promising line of work, that might reveal both shortcomings and possibilities of improved diagnostics and repair.

# REFERENCES

ALTERA CORPORATION. **Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs**. San Jose, Jul. 2010. p.1-9.

ALTERA CORPORATION. **Introduction to Single-Event Upsets**. San Jose, Sep. 2013.

ALTERA CORPORATION. **Industry Solutions**. Disponível em: <https://www.altera.com/solutions/industry.html>. Acesso em: 30 ago. 2015.

ATMEL CORPORATION. **Rad Hard FPGAs**. San Jose, 2015a. Disponível em: <http://www.atmel.com/products/rad-hard/rad-hard-fpgas/default.aspx>. Acesso em: 24 nov. 2015.

ATMEL CORPORATION. **Atmel Aerospace**, San Jose, 2015b. Disponível em: <http://www.atmel.com/images/4015k-integratedcircuits-spacerad-hard_e_us_051215_web.pdf> Acesso em: 24 de nov 2015.

BARNABY, H. J. Total-ionizing-dose effects in modern CMOS technologies. **IEEE Transactions on Nuclear Science**, New York, v. 53, n. 6, p. 3103–3121, 2006.

BATLLE, J. A New FPGA/DSP-Based Parallel Architecture for Real-Time Image Processing. **Real-Time Imaging**, London, v. 8, n. 5, p. 345–356, Oct. 2002.

BAUMANN, R. C. Radiation-induced soft errors in advanced semiconductor technologies. **IEEE Transactions on Device and Materials Reliability**, New York, v. 5, n. 3, p. 305–315, 2005.

BOLCHINI, C.; MIELE, A.; SANDIONIGI, C. A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs. **IEEE Transactions on Computers**, New York, v. 60, n. 12, p. 1744–1758, 2011.

CARMICHAEL, C.; CAFFREY, M.; SALAZAR, A. **Correcting Single-Event Upsets Through Virtex Partial Configuration**. San Jose: XILINX, Jun. 2000.

CARMICHAEL, C.; TSENG, C. W. **Correcting single-event upsets in Virtex-4 FPGA configuration memory**. San Jose: XILINX, Oct. 2009.

DEHON, A. Balancing interconnect and computation in a reconfigurable computing array (or, why you don’t really want 100% LUT utilization). In: SYMPOSIUM ON FIELD PROGRAMMABLE GATE ARRAYS, 7th, 1999, Monterey. **Proceedings**. New York: ACM, 1999. p. 69-78.

DEZSO, B.; JÜTTNER, A.; KOVÁCS, P. LEMON - An open source C++ graph template library. **Electronic Notes in Theoretical Computer Science**, Amsterdam, v. 264, n. 5, p. 23–45, 2011.

GAILLARD, R. Single Event Effects: Mechanisms and Classification. In: NICOLAIDIS, M. (Ed.). **Soft Errors in Modern Electronic Systems**. Boston: Springer US, 2011. p. 27–54. Frontiers in Electronic Testing, v.41.

GOKHALE, M. et al. Dynamic reconfiguration for management of radiation-induced faults in FPGAs. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 18th, 2004, Santa Fe. **Proceedings**. New York, 2004.

JEDEC. **JESD89A:** Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Error in Semiconductor Devices. Arlington, Aug. 2001. Disponível em: <http://www.jedec.org/standards-documents/docs/jesd-89a>. Acesso em: 24 de nov 2015.

KARIMI, S. et al. FPGA-Based Real-Time Power Converter Failure Diagnosis for Wind Energy Conversion Systems. **IEEE Transactions on Industrial Electronics**, New York, v. 55, n. 12, p. 4299–4308, Dec. 2008.

KASTENSMIDT, F. G. DE L. et al. Designing fault-tolerant techniques for SRAM-based FPGAs. **IEEE Design and Test of Computers**, New York, v. 21, n. 6, p. 552–562, Nov. 2004.

KASTENSMIDT, F. L.; KINZEL FILHO, C.; CARRO, L. Improving reliability of SRAM-based FPGAs by inserting redundant routing. **IEEE Transactions on Nuclear Science**, New York, v. 53, n. 4, p. 2060–2068, 2006.

KELLERMANN, M.; TAM, S. **XAPP883:** Fast Configuration of PCI Express. San Jose, 2010.

LEVESON, N. G.; TURNER, C. S. An Investigation of the Therac-25 Accidents. **Computer**, New York, v. 26, n. 7, p. 18–41, Jul. 1993.

LIMA, F. et al. A fault injection analysis of Virtex FPGA TMR design methodology. In: EUROPEAN CONFERENCE ON RADIATION AND ITS EFFECTS ON COMPONENTS AND SYSTEMS, 6th, 2001, Grenoble. **Proceedings.** New York: IEEE, 2001. p. 275-282.

MAY, T. C.; WOODS, M. H. A New Physical Mechanism for Soft Errors in Dynamic Memories. In: INTERNATIONAL RELIABILITY PHYSICS SYMPOSIUM, 16th, San Diego, 1978. **Proceeding**. New York: IEEE, Apr. 1978.p. 33-40.

MICROSEMI CORPORATION. **Rad-tolerant FPGAs**. Disponível em: <http://www.microsemi.com/products/fpga-soc/rad-tolerant-fpgas>. Acesso em: 10 abr. 2015a.

MICROSEMI CORPORATION. **RTG4 FPGAs**. Disponível em: <http://www.atmel.com/products/rad-hard/rad-hard-fpgas/default.aspx>. Acesso em: 10 dez. 2015b.

MILLMAN, K. J.; AIVAZIS, M. Python for Scientists and Engineers. **Computing in Science & Engineering**, New York, v. 13, n. 2, p. 9–12, Mar. 2011.

MINKOVICH, K. **Kirill Minkovich's Home Page**. Disponível em: <http://cadlab.cs.ucla.edu/~kirill/>. Acesso em: 3 maio. 2015.

MONTGOMERY, D. C. **Design and analysis of experiments**. 5th. ed. New York: John Wiley, 2001.

NAZAR, G. L. **Fine-grained error detection techniques for fast repair of FPGAs**. 2013.

125 f. Tese (Doutorado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2013.

NAZAR, G. L. Improving FPGA repair under real-time constraints. **Microelectronics Reliability**, v. 55, n. 7, p. 1109–1119, Jun. 2015.

NAZAR, G. L.; CARRO, L. Fast single-FPGA fault injection platform. In: DEFECT AND FAULT TOLERANCE IN VLSI AND NANOTECHNOLOGY SYSTEMS (DFT), 2012 IEEE INTERNATIONAL SYMPOSIUM ON, Austin, 2012. **Proceedings.** New York: IEEE, 2012a. p. 152-157.

NAZAR, G. L.; CARRO, L. Exploiting Modified Placement and Hardwired Resources to Provide High Reliability in FPGAs. In: FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES (FCCM), 2012 IEEE 20TH ANNUAL INTERNATIONAL SYMPOSIUM ON, Toronto, 2012. **Proceedings**. New York: IEEE, 2012b. p. 149-152.

NAZAR, G. L.; SANTOS, L. P.; CARRO, L. Accelerated FPGA repair through shifted scrubbing. In: INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS, 23rd, Porto, 2013. **Proceedings.** New York: IEEE, 2013. p. 1-6.

NAZAR, G. L.; SANTOS, L. P.; CARRO, L. Fine-Grained Fast Field-Programmable Gate Array Scrubbing. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, New York, v. 23, n. 5, p. 893–904, May 2015.

NEUMANN, P. G. Reliability and Safety Problems. In: Idem. **Computer Related Risks**. New York: ACM Press/Addison-Wesley, 1995. Chap. 2

NORMAND, E.; BAKER, T. J. Altitude and latitude variations in avionics SEU and atmospheric neutron flux. **IEEE Transactions on Nuclear Science**, New York, v. 40, n.6, pt 1, p. 1484–1490, 1993.

PEATTIE, A. M. **Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode**. San Jose: XILINX, Jun. 2009. 17 p.

PSARAKIS, M.; APOSTOLAKIS, A. Fault tolerant FPGA processor based on runtime reconfigurable modules. In: IEEE EUROPEAN TEST SYMPOSIUM, 17th, Annecy, 2012. **Proceedings.** New York: IEEE, 2012.

REORDA, M. S.; STERPONE, L.; ULLAH, A. An error-detection and self-repairing method for dynamically and partially reconfigurable systems. In: IEEE EUROPEAN TEST SYMPOSIUM, 12th, Avignon, 2013. **Proceedings**. New York: IEEE, 2013.

SANTOS, L. P.; NAZAR, G. L.; CARRO, L. Dynamically Shifted Scrubbing for Fast FPGA Repair. In: Workshop on Self-Awareness in Reconfigurable Computing Systems (SRCS'13), 2nd, Porto, 2013. **Proceedings**. New York: IEEE, 2013.

SARI, A.; PSARAKIS, M.; GIZOPOULOS, D. Combining checkpointing and scrubbing in FPGA-based real-time systems. In: VLSI TEST SYMPOSIUM, 31st, Berkeley, 2013. **Proceedings.** New York: IEEE, 2013.

SEXTON, F. W. Destructive single-event effects in semiconductor devices and ICs. **IEEE Transactions on Nuclear Science**, New York, v. 50, n. 3, p. 603–621, Jun. 2003.

SROUR, J. R. **Basic mechanisms of radiation effects on electronic matrials, devices and integrated circuits**. Palos Verdes Peninsula: Northrop Corporation, 1982.

TABER, A.; NORMAND, E. Single event upset in avionics. **IEEE Transactions on Nuclear Science**, New York, v. 40, n. 2, p. 120–126, 1993.

UZUN, I. S.; AMIRA, A.; BOURIDANE, A. FPGA implementations of fast Fourier transforms for real-time signal and image processing. **IEE Proceedings - Vision, Image, and Signal Processing**, New York, v. 152, n. 3, 2005. p. 283-296.

VON NEUMANN, J. Probabilistic logics and synthesis of reliable organisms from unreliable components. **Automata Studies**, Princeton, 1956. p. 43-98.

XILINX INC. **UG702:** Partial Reconfiguration User Guide. San Jose, 2012a. 124p.

XILINX INC. **UG190:** Virtex-5 FPGA User Guide. San Jose, 2012b. 385p.

XILINX INC. **Aerospace and Defence**. Disponível em: <http://www.xilinx.com/applications/aerospace-and-defense.html>. Acesso em: 3 abr. 2015a.

XILINX INC. **DS890:** UltraScale Architecture and Product Overview. San Jose, 2015b. 34 p.

XILINX INC. **TMRTool**. Disponível em: <http://www.xilinx.com/ise/optional_prod/tmrtool.htm>. Acesso em: 1 set. 2015c.

XILINX INC. **UG116:** Device Reliability Report. San Jose, 2015d. 101 p.

# APPENDIX A : STATISTICAL ANALYSIS OF FIT DATA

As explained in section 8.3, the raw data presented in table Table 8.3 is not appropriate to statistical analysis, as the scale of values varies with the benchmark. To put the date on the same magnitude order, the results of each technique were divided by the results of the standard DMR column. Statistically this is not a normalization of values, so this term will not be used. The scaled data is shown in Table A.1.

Table A.1 - Scaled values for the area under the FIT curve data

| **Benchmark** | **Area Under FIT Curves** | | | | | |
|---|---|---|---|---|---|---|
| | ***Standard DMR*** | ***SG-DMR*** | ***Delta*** | ***CG-DMR*** | ***FG-DMR*** | ***Best Static*** |
| alu 4b | 1.00 | 0.38 | 0.35 | 0.25 | 0.72 | 0.27 |
| *alu 32b* | 1.00 | 0.53 | 0.67 | 0.53 | 1.31 | 0.54 |
| *alu 64b* | 1.00 | 0.83 | 0.90 | 0.85 | 1.15 | 0.75 |
| *apex2* | 1.00 | 0.66 | 0.92 | 0.63 | 1.60 | 0.71 |
| *apex4* | 1.00 | 0.92 | 0.81 | 0.83 | 1.46 | 0.76 |
| *des* | 1.00 | 0.58 | 2.63 | 0.58 | 0.34 | 0.54 |
| *ex1010* | 1.00 | 1.02 | 0.64 | 0.60 | 1.27 | 0.73 |
| *ex5p* | 1.00 | 0.59 | 0.52 | 0.64 | 1.35 | 0.39 |
| *misex3* | 1.00 | 0.86 | 0.76 | 0.66 | 2.17 | 0.84 |
| *pdc* | 1.00 | 0.84 | 0.92 | 0.56 | 1.05 | 0.80 |
| *seq* | 1.00 | 0.83 | 0.80 | 0.75 | 1.57 | 0.72 |
| *spla* | 1.00 | 0.46 | 0.48 | 0.43 | 1.34 | 0.47 |

Source: author

The next step is to verify that the data is independent from the benchmark circuit. This is important as a dependent data would mean that a specific technique is very well suited to a particular circuit. This step then consists of a double-variable analysis of variance, done by the Statistica 12.0 software package from Statsoft (Table A.2).

Table A.2 - Analysis of variance

| Effect | Sum of Squares | Degrees of Freedom | Mean Square | *F* test | *p*-value |
|---|---|---|---|---|---|
| Benchmark | 1.48532 | 11 | 0.13503 | 1.31 | 0.24261 |
| Technique | 3.99998 | 5 | 0.80000 | 7.78 | $1.37 \times 10^{-5}$ |
| Error | 5.65872 | 55 | 0.10289 | | |

Source: author

The *p*-value for the benchmark variable confirms that the results do not depend on the benchmark and thus the diagnostic technique is the only variable to be analyzed further. The means of the results for each diagnostic technique were compared by Tukey's Test (comparison of means) (Table A.3).

Table A.3 - Tukey's Test for diagnostic techniques

| | Technique | | | | | |
|---|---|---|---|---|---|---|
| Technique | Standard DMR | SG-DMR | Delta | CG-DMR | FG-DMR | Best Static |
| mean | 1.000 | 0.708 | 0.868 | 0.609 | 1.277 | 0.627 |
| Standard DMR | - | 0.264 | 0.921 | 0.053 | 0.320 | 0.074 |
| SG-DMR | 0.264 | - | 0.840 | 0.976 | 0.001 | 0.991 |
| Delta | 0.921 | 0.840 | - | 0.394 | 0.037 | 0.477 |
| CG-DMR | 0.053 | 0.976 | 0.394 | - | 0.000 | 1.000 |
| FG-DMR | 0.320 | 0.001 | 0.037 | 0.000 | - | 0.000 |
| Best Static | 0.074 | 0.991 | 0.477 | 1.000 | 0.000 | - |

Source: author