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

MONICA MAGALHÃES PEREIRA

A Reliability Analysis Approach to Assist the Design of Aggressively Scaled Reconfigurable Architectures

Thesis presented in partial fulfillment of
the requirements for the degree of
Doctor of Computer Science

Prof. Dr. Luigi Carro
Advisor

Porto Alegre, February 2012

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Magalhães Pereira, Monica

A Reliability Analysis Approach to Assist the Design of Aggressively Scaled Reconfigurable Architectures / Monica Magalhães Pereira. -- 2012.
137 f.

Orientador: Luigi Carro.

Tese (Doutorado) -- Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2012.

1. Reliability Analysis. 2. Fault Tolerance. 3. Reconfigurable Architectures. 4. Scaling. I. Carro, Luigi, orient. II. Título.

TABLE OF CONTENTS

LIST OF ABREVIATIONS AND ACRONYMS

ALU	Arithmetic and Logic Unit
ARISE	Automatic Resources Investigation System based on Application Execution
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BIAST	Built-In Applicable Self-Test
BIST	Built-In Self Test
BLE	Basic Logic Element
CAEN	Chemically Assembled Electronic Nanotechnology
CG	Configuration Generator
CMOS	Complementary Metal-Oxide-Semiconductor
DDG	Data-Dependence Graph
DV	Dependence Verification
ECC	Error-Correcting Code
EX	Execution
FPGA	Field Programmable Gate Array
FU	Functional Unit
HD	Hard Drive
IC	Integrated Circuit
ID	Instruction Decode
ID	Instruction Decode
IF	Instruction Fetch

ITRS	International Technology Roadmap for Semiconductors
LAC	Low Area Cost
LOwER-FaT	Low Overhead without Extra Redundancy Fault-Tolerant
LUT	LookUp Table
MEM	Memory Access
MIN	Multistage Interconnection Network
MPSoC	Multiprocessor System-on-Chip
MTBF	Mean Time Between Failures
MTTF	Mean Time To Failure
NBTI	Negative Bias Temperature Instability
NGL	Next Generation Lithography
NoC	Network-on-Chip
PC	Program Counter
PE	Processing Element
PIP	Programmable Interconnect Points
RA	Reconfigurable Architecture
RA	Resource Allocation
RADC	Rome Air Data Centre in the USA
RAW	Read After Write
RISC	Reduced Instruction Set Computer
SSAF	Single Stuck-At Fault
STAR	Self-Testing Area
TDDB	Time-Dependent Dielectric Breakdown
TMR	Triple Modular Redundancy
TU	Table Update
VHDL	VHSIC hardware description language

VLIW	Very Long Instruction Word
WB	Write Back

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

As computer systems are built with aggressively scaled and unreliable technologies, some implementations rely on function specialization with reconfigurable computing to increase performance by exploiting parallelism, with possible energy gains. However, the use of reconfigurable devices in general purpose computing also brings extra reliability challenges at the system level. Solutions to cope with that are generally accompanied with the addition of excessive area, performance and power overheads to the overall system. These overheads could be reduced if a more extensive analysis was performed to evaluate the best fault tolerance strategy to balance the tradeoff between reliability and the mentioned aspects. In this context, this work present a comprehensive analysis of architectural design that includes the use of reliability modeling and takes into consideration aspects such as area, performance, and power. The analysis aims to assist the design of reliability-aware reconfigurable architectures by giving some indications about what kind of redundancy should be used in order to increase reliability. In the proposed analysis, we show that communication among functional units is critical to the overall reliability of reconfigurable architectures. Therefore, where most of the reliability investments should be made. Moreover, the analysis also demonstrate that there is a threshold in the amount of redundancy that can be added in order to increase reliability. This limit is determined by the fact that adding redundancy increases area overhead. This overhead influences reliability until overcomes the reliability gains. Therefore, even disregarding area cost, the gains in reliability will cease or even decrease. To provide a more extended evaluation, a fault tolerance approach was proposed to cope with permanent faults. The LOwER-FaT strategy is a mechanism embedded in a run-time reconfiguration mechanism that automatically selects the fault-free resources without adding extra time overhead to the configuration generation mechanism. The fault-tolerant strategy takes advantage of the on-line transparent configuration generation mechanism to transparently avoid faulty functional units and interconnects. Moreover, the strategy does not require the addition of spare resources. All the resources are used to accelerate execution, and only in case of fault, a resource is replaced by a working one, with a performance penalty caused by the reduction in the amount of resources. In spite of that, experimental results showed a mean performance degradation of 14% on overall performance under 20% fault rate. Moreover, reliability results indicated gains of around six orders of magnitude when the fault tolerance strategy was in place.

Keywords: Reconfigurable architectures, Fault tolerance, Reliability analysis, Scaling.

1 INTRODUCTION

Fault tolerance is defined as the ability of a system to continue its correct operation even after the occurrence of a fault (AVIZIENIS, LAPRIE, RANDELL, *et al.*, 2004). How to cope with faults and provide this ability to the system has been topic of studies since the 1950's. Back then, when the existence of personal computer was only speculation and computers were used to very specific activities, such as military and later on space missions, fault tolerance emerged with the purpose of working in hostile and remote environments, where any small damage could jeopardize the mission and manual repair was impractical. Nowadays, fault tolerance is still used with this same purpose. However, taking into account the fact that computers are now ubiquitous, many other areas require fault-tolerant systems capable of operating in spite of the occurrence of faults. Examples are avionics; car braking system; banks; etc. These can be considered now mission-critical applications that require a highly fault-tolerant system to avoid catastrophic consequences.

Furthermore, the continued scaling of the geometric dimensions of integrated circuits (ICs) has also increased, in an alarming speed, the fault rates of integrated circuits (CONSTANTINESCU, 2003), (DEHON and NAEIMI, 2005). The IC scaling is responsible for the high improvement in our personal computer's speed and the size reduction of our electronic devices. At the same time, scaling has increasing fault rates in such a high magnitude that a fault tolerance approach in all devices will be mandatory in the near future technologies.

In this document, the reasons for such increased concern with the effects of faults are exposed, some fault tolerance techniques are analyzed, and the research and development of a reliability analysis to assist the design of reliable reconfigurable architectures is proposed as the thesis work.

1.1 Technology

Shrinking feature sizes has been possible due to the evolution of the fabrication process techniques. Next Generation Lithography (NGL) and Chemically Assembled Electronic Nanotechnology (CAEN) techniques promise to reduce feature sizes to 20nm or less. However, the manipulation of such small devices brings many challenges to the fabrication process. Firstly, nanometer-scale transistors and wires are more fragile and for this reason, more susceptible to break during the fabrication process. Moreover, the fabrication process itself is statistical, with a probability of failing during the fabrication

of the device (MISHRA and GOLDSTEIN, 2003). All these challenges lead to a high defect density, significantly affecting the yield.

Shrinking feature sizes also increases the occurrence of permanent faults during circuit lifetime. Phenomena like time-dependent dielectric breakdown (TDDB); electromigration; negative bias temperature instability (NBTI); etc, happen more often in scaled technologies (ITRS, 2011b). The consequence of these faults is circuit malfunction and/or reduction in circuit lifetime. Since researches about the behavior of circuits in deep-submicron technologies are performed by semiconductor manufacturing companies, it is difficult to find more precise information about fault rate prediction. The Intel Corporation fellow, Borkar in 2004, revealed that in a future 100 billion transistor circuit, 20 billion will be unusable due to manufacturing defects and 10 billion will fail over time due to wear-out (BORKAR, 2004). This information was widely spread in academic community and even nowadays, it is still used as reference.

Solutions to cope with the aforementioned problems have been proposed in the past years (PRADHAN, 1996). The traditional solutions such as Triple Modular Redundancy (TMR) are very costly. TMR requires three times the area of the component plus the area for the voter. The same overhead is required for power. Other solutions found in the literature might present an efficient fault-tolerant strategy. However, in most cases, there is no comprehensive study about the most effective strategy considering the tradeoff between fault tolerance and area cost and/or in relation to other aspects, such as performance and power. For this reason, the solutions range between: 1) having an efficient fault-tolerant solution based on a strategy that requires a large amount of hardware redundancy and/or performance penalty. 2) having a limited fault-tolerant strategy that only covers part of the faults but presents a reduced overhead. Therefore, a solution that provides a balanced tradeoff between fault tolerance and area, performance and/or power overheads is required to attain the increase demand for reliable devices.

1.2 Reconfigurable Architectures

Reconfigurable architectures emerged to balance the tradeoff between the high performance and low flexibility of application specific integrated circuits (ASICs) and the flexibility of general-purpose processors given by their software programming (COMPTON and HAUCK, 2002).

The ability to adapt their behavior according to the application demand has proven to be an efficient solution, as one can see in several architectures found in the literature (COMPTON and HAUCK, 2002), (HARTENSTEIN, 2001), (VAHID, STITT and LYSECKY, 2008) and (BECK and CARRO, 2010). For this reason, in the past years, reconfigurable architectures, such as FPGAs (Field Programmable Gate Arrays), are becoming more popular with a large amount of commercial devices available (XILINX, 2012) and (ALTERA, 2012).

To provide flexibility and high performance, reconfigurable devices have a specific design that allows the rearrangement of the architecture to adapt itself according to application requirements. This design consists of several identical processing elements (PEs) and an interconnection model that provides communication among processing elements.

With this design, reconfigurable architectures are capable of adapting to the requirements of each application independently. For example, the same architecture can

be configured to perform a multimedia application, which is characterized for intensive computation of highly regular operations. On another moment, the architecture can be configured to perform a sort algorithm, which is characterized for having many control operations (comparisons).

The large amount of identical PEs is also a very powerful tool to exploit application's parallelism, more specifically instruction level parallelism (ILP). For applications with a large amount of ILP, the operations can be distributed along the PEs to be executed in parallel. This capability allows the reconfigurable architectures to achieve high performance when executing highly parallel applications, which is the main characteristic of embedded market applications. This is the case of multimedia, telecommunication, cryptography, and other similar application domains (BECK, RUTZIG, GAYDADJIEV, *et al.*, 2008b). Recently, Intel announced the first configurable Intel Atom-based processor. The processor E600C series contains an Intel Atom E600 processor and an Altera FPGA. According to Intel, the Atom E600C processor series aims to meet the needs of embedded device market (INTEL, 2010).

Although reconfigurable architectures may present high performance when targeted to computational intensive applications, they do not present well when other aspects are considered or other application domains are executed. Power consumption and configuration time overhead still are a challenge for reconfigurable architectures designers and users (BECK and CARRO, 2010).

More specifically to this work, reliability is also a concern that must be carefully studied. Providing an architecture with reconfiguration capability implies in having a large amount of processing elements and a complex interconnection model that allows an ample communication among them. Relying on this highly dense architecture might be a challenge. With several small elements becoming even smaller due to the scaling, the probability that a fault damages the processing elements and interconnects increases. Therefore, as devices shrink, fault tolerance solutions that encompass reconfigurable architectures become more necessary.

Therefore, in face of the ever-increasing demand for flexible and high-performance reconfigurable architectures, the requirement for solutions to cope with faults considering the aforementioned aspects of these architectures also increases. This requirement becomes even more essential when considering the high fault rates that future scaled technologies should introduce.

Based on this, the focus of this work is in fault-tolerant solutions targeted to increase reliability of reconfigurable architectures. Next section describes the main contributions of this thesis.

1.3 Main Contributions

This work presents three main contributions. The first contribution is finding that there is a threshold in the amount of hardware redundancy that can be added to reconfigurable architectures in order to increase reliability. Depending on where and how much of redundancy is applied to the system, it can negatively impact on reliability, and even reduce reliability. This threshold is related to the area overhead that is introduced when hardware redundancy is added. For this reason, a comprehensive analysis should be performed to find the best strategy to increase reliability.

The second main contribution of this work is related to identify in which part of the reconfigurable architecture one should invest in order to improve reliability. There is a consensus in the reconfigurable architecture research community that the interconnection model is critical to the system in many aspects, such as area, power and fault tolerance. In some aspects, this general consideration is based on critical analysis as a result of several practical studies and many years of research. This is the case of area and power aspects. However, in relation to fault tolerance, there is no a comprehensive study that takes into consideration all the characteristics of the architecture and quantifies exactly how much the interconnection model influences the reliability of the underlying architecture. In this sense, this work presents a detailed analysis of reconfigurable architectures, estimating the impact of the interconnection model on system reliability. In the analysis we show that the interconnects are in fact the critical elements to system reliability. We discuss about alternatives to overcome this issue and continue improving reliability.

The third contribution regards the means to achieve the first two contributions. To perform a comprehensive reliability, we propose the use of reliability modeling combined with the evaluation of other aspects, such as area, performance. To model the architecture reliability, we propose the use of a mathematical representation where each component of the architecture presents an individual reliability. Additionally, the total reliability of the system is a function of the reliabilities of all components that compose the system. The analysis is targeted to evaluate the effects of permanent faults in the architecture and find the best strategy to mitigate these effects in order to allow device usage in spite of the faults.

Therefore, the analysis aims to assist the design of reliable reconfigurable architectures by giving some indications about what kind of investments and where they should be made. Consequently finding the best strategy to design a reliable architecture and avoid high overheads of area, performance and/or power.

In the context of this work, we also propose a new fault-tolerant strategy implemented in reconfigurable architecture originally proposed in (BECK, RUTZIG, GAYDADJIEV, *et al.*, 2008). The fault-tolerant strategy dynamically generates the configuration selecting only fault-free resources to perform operations. The strategy exploits the regularity of the reconfigurable architecture, avoiding the inclusion of spare resources. Therefore, in this approach, all the functional units are used to accelerate execution and only when a fault affects a resource, this resource is eliminated.

1.4 Methodology

The methodology to perform the proposed analysis consists in four main steps:

1. **Modeling reliability:** uses mathematical model to represent the architecture.
2. **Analyzing reliability:** use the reliability model to assess the system reliability. If the reliability analysis considering aspects such as performance, area and power is not satisfactory, some modifications in the architecture represented by the mathematical model can be performed aiming to enhance reliability and/or reduce overheads (area, power and/or performance). In this case, it is necessary to return to step 1.
3. **Implementing design:** after a comprehensive analysis, the architecture can be implemented based on its mathematical representation.

4. **Injecting faults:** to obtain experimental results, some fault injection experiments are performed to validate the architecture.

Figure 1 depicts a flowchart describing the methodology.

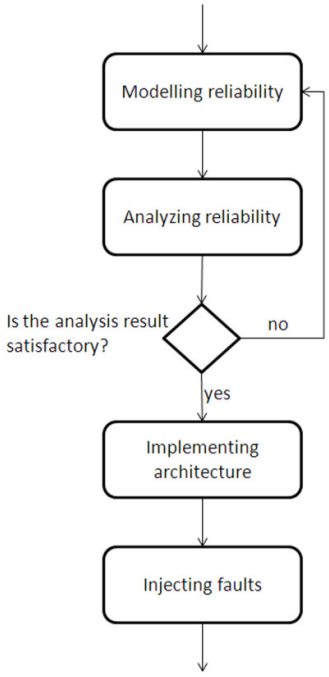
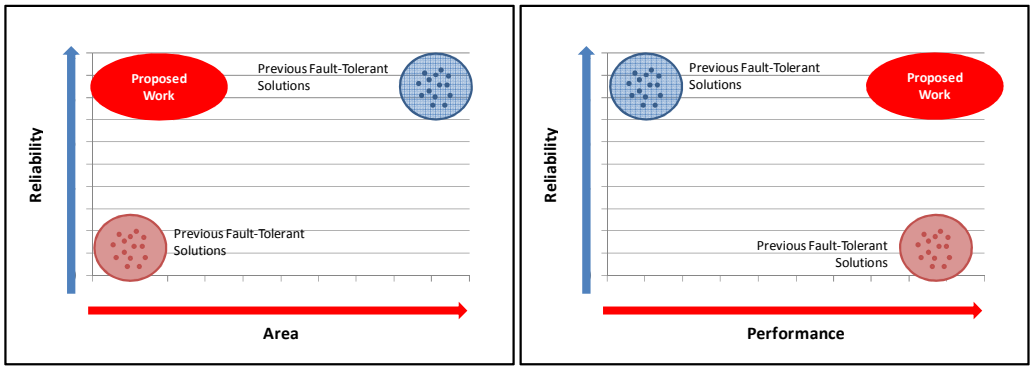


Figure 1. Analysis methodology

Considering all the aspects analyzed and the architectural modifications, at the end of the analysis, the goal is to have a highly reliable architecture with low overheads of area, performance and/or power.

To contextualize the proposed work, Figure 2 presents a graph where the reliability is a function of area (Figure 2.a) and performance (Figure 2.b) overheads. The main fault-tolerant solutions found in the literature either present a high reliability with a high area and/or performance overhead or a low area and/or performance overhead, consequently with low reliability. On the other hand, our proposed analysis aims to assist in the architectural designing process to allow the design of highly reliable-aware reconfigurable architectures with low overheads. It is important to highlight that Figure 2 demonstrates only two examples of unbalanced tradeoff. However, in the proposed analysis, other costs can also be considered, such as power.



a) To increase reliability, the previous solutions present a large area overhead

b) To increase reliability, the previous solutions present a high performance overhead

Figure 2. Contextualizing the proposed analysis

1.5 Thesis Outline

Chapter 2 presents the terminology and general concepts used in this work. Chapter 3 describes the proposed analysis with the reliability modeling strategy and specifies the parameters required to a proper reliability analysis, such as failure rate and reliability estimation. Chapter 4 presents a comprehensive analysis of the first case study. First, a detailed description of the reconfigurable architecture is presented, followed by its reliability model. The following sections present the reliability, performance and energy analyses. Chapter 5 presents the second case study, the reliability model description and analysis, as well as some comparisons between this case study and the former. Chapter 6 presents the related works, discussing the drawbacks of previously proposed fault-tolerant techniques. Chapter 7 concludes this thesis with final considerations and some discussions about the future research topics that can be investigate.

2 TERMINOLOGY AND GENERAL CONCEPTS

In this chapter, we introduce the main technical terms used in this work and discuss the reasons why fault tolerance has gained more attention as technology scales.

2.1 Fault

The design of fault tolerant systems consists in including some approach to prevent that a physical fault causes an error in the system leading to a system failure. Therefore, there is a cause-effect relationship among fault, error and failure as demonstrated in Figure 3. The definitions presented next follow the ones presented by Avizienis et al in (AVIZIENIS, LAPRIE, RANDELL, *et al.*, 2004), and will be adopted in this document:

Failure: it is the system malfunction. It is commonly expressed in terms of the delivered service of a system. The delivered service is considered correct when it is according to the system specification. Therefore, a failure occurs when the delivered service deviates from the correct service.

Error: when the system presents an error, it means that some part of it is not working as expected or specified. However, the system may or may not work correctly (i.e. delivered the correct service) due to the error.

Fault: it is the event that causes the error and may or may not lead to the system failure. It can be a physical defect or imperfection, or flaw that occurs within some hardware or software component.

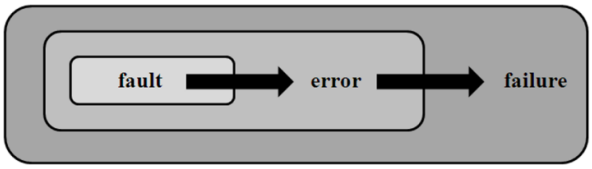
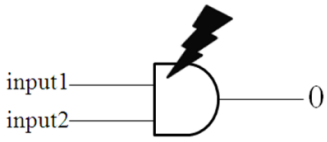


Figure 3. Fault, error and failure relationship

A fault may never lead to an error and consequently the system may never fail due to that fault. In some cases, to a fault manifest itself, it is necessary that some specific conditions be achieved. For example, a faulty AND gate, which the result is always 0. An error will be only observed when the two inputs are 1, in this case the correct result should be 1. However, since in all other cases the correct result is 0, it is not possible to know if the 0 in the output is the result of a correct computation or it is an error

manifested by a fault. Figure 4 illustrates the example of a fault that manifests only in a specific situation.



input1	input2	Correct output	Current ouput
0	0	0	0
0	1	0	0
1	0	0	0
1	1	**1**	**0**

Figure 4. Faulty AND gate with output stuck at 0

In Figure 4, based on the truth table of the AND gate, the error is manifested only when the inputs are 1. For this reason, to detect a faulty component, the testing techniques are based on using different combination of inputs to distinguish between the correct circuit operation and faulty circuit behavior (ABRAMOVICI, BREUER and FRIEDMAN, 1994).

There are three types of faults: permanent, transient and intermittent:

Permanent faults, as the name indicates, remain in the system during its whole lifetime. They can happen at any moment, since circuit fabrication until the end of the circuit usage. Usually, faults that happen during manufacturing phase are called defects.

Despite being permanent, faults that happen during circuit manufacture are caused by different physical phenomena than the ones that happen during usage. During manufacturing, the main causes of defects are contaminations on the silicon surface and surface roughness causing gate oxide breakdown; incorrect metallization causing short-circuits and open-circuits in the interconnects (SRINIVASAN, ADVE, PRADIP, *et al.*, 2005). According to DeHon and Naeimi (DEHON and NAEIMI, 2005), for nanoscale technologies, the defect rate is predicted to be 1% to 15% for wires and connections. On the other hand, the physical phenomena that can cause permanent faults during system usage (wear-out) are responsible for the aging effect. The aging effects appear over time and are caused by the device operation under specific conditions. Over time, the device becomes more prone to some physical effects that cause permanent faults. Some examples of these physical phenomena are: time-dependent dielectric breakdown (TDDB), which occurs when the gate dielectric breaks down and become electrically shorted with time. Electromigration, when metal ions migrate over time causing voids and deposits in the interconnects and creating open and short circuits. Negative bias temperature instability (NBTI) that occurs in *p*-channel MOS devices stressed with negative gate voltages at elevated temperature. Stress migration; thermal cycling; etc (ITRS, 2011b), (STOTT, SEDCOLE and CHEUNG, 2010).

The second type of fault is the **transient fault** and its effect is hazardous only during circuit usage. Transient faults occur when radiation particles strike the circuit and deposit charges in the silicon. The consequence of this effect is a switch in the logical state of the nodes or a flip in the bits of memory. Once the deposited charges dissipate, the effects usually disappear (HEIJMEN, 2002). For this reason, this type of fault is called *transient* and the effect that it causes is called *single event transients* (SETs). If the SET is latched by memory storage elements, the incorrect information is stored causing a *single event upset* (SEU). This incorrect information remains in memory until new information replaces it. The error caused by this type of fault is

called *soft error* and has this name because it will pass as soon as the information is replaced. The two main sources of radiation are neutrons and cosmic rays coming from outer space and alpha particles. These last ones are originated by the decay of radioactive elements present in the packaging materials of the device.

Intermittent faults appear and disappear, repeatedly. Intermittent faults are highly correlated with stressful operating conditions, such as power supply voltage noise and timing faults caused by insufficient cooling (WEAVER and AUSTIN, 2001).

To mitigate the effect of faults in circuits, many studies try to predict when the faults will happen. Although this is not an exact science, it is possible to estimate an approximate time when some types of faults may manifest. These estimations are basis for many other studies that attempt to extend this time and make the circuit operates correctly for a longer period.

One of the most widespread studies about these estimations is the failure rate during lifetime of an entire population of devices. The *bathtub curve*, depicted in Figure 5, is used to demonstrate the effects of fault distribution along the lifetime of the devices. The fault distribution in time is split in three main phases. The first one corresponds to the initial phase of the device's life, right after manufacturing. A high failure rate can be observed during this phase, called *infant mortality*. The second one is the stable useful life, where the failure rate remains constant. In the last phase, the failure rate increases again. This is the wear-out period of the devices. In a general analysis, for all types of electronic devices, the bathtub curve indicates that during the initial phase, many devices may be lost due to infant mortality. On the other hand, the failure rate decreases and remains constant during useful life, where few devices are lost. In the wear-out phase, the failure rate increases again and because of that, many devices have to be discarded.

Traditionally, the bathtub curve of failures for VLSI circuits is represented by curve A. The fault rate increase caused by feature size reduction is changing this scenario and charts more similar to curves B and C are now used to describe the failure rate during circuit's lifetime (WHITE and CHEN, 2008). The expected scenario in a short-term future is a reduction in the useful life period, with aging effects affecting the circuits earlier.

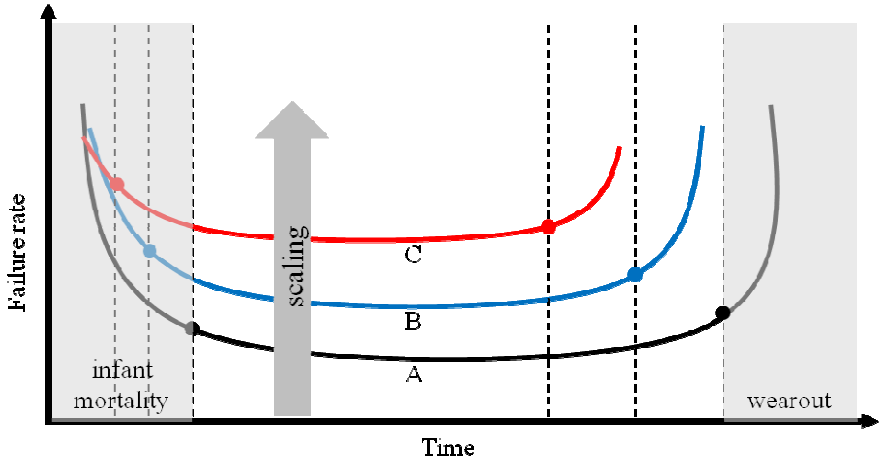


Figure 5. The Bathtub curve

The phenomena that cause the faults in reduced feature sizes are the same of larger technologies. The main difference is the fact that at nanoscale basis transistors have only few atoms, and consequently are more fragile and susceptible to effects. Moreover, the fabrication process is more prone to fail due to the statistical nature of the process (BORKAR, 2005). The effects that cause permanent faults during device lifetime are more intensified in deep-submicron technology (SRINIVASAN, ADVE, BOSE, *et al.*, 2004).

2.2 System Dependability

According to Laprie (1996), computer system dependability is the quality of the delivered service such that reliance can justifiably be placed on this service. The delivered service consists in the system behavior, as it is perceived by another system (or user) that interacts with the former. If the service is delivered correctly (as specified in the system function), the dependability is assured. However, a failure occurs when the delivered service deviates from the correct service.

Dependability comprises the following attributes (AVIZIENIS, LAPRIE, RANDELL, *et al.*, 2004).

Availability: consists in the probability, A(t), that a system is available to perform its function at an instant of time, t.

Reliability: is also a function of time, R(t), defined as the probability that the system is operating correctly during the interval of time, [t_0,t], given that it was operational at time t_0.

Safety: is the absence of catastrophic consequences on the user and the environment. In other words, it is the probability, S(t), that a system either will perform its functions correctly or will discontinue its functions without disrupting the operation of other systems or compromise safety of any user associated with it.

Maintainability: quantifies how easy a system can be repaired once it has failed. Therefore, maintainability is the probability, M(t), that a failed system will restore the correct operation within a period of time t.

Reliability is widely used to evaluate fault-tolerant techniques. Highly reliable systems present a high probability that the system is operating correctly during a long period of time.

To describe the reliability of a system, firstly it is necessary to model the system taking into consideration the relationship among the components, as well as the physical characteristics that determine the probability laws that govern the failures (KUO and ZUO, 2003).

Modeling and measuring system reliability is one of the key elements to the proposed approach presented in this work. Therefore, more detail explanation describing how to model a system and measure its reliability is presented in next chapter.

3 RELIABILITY MODELING

Reliability is the probability that a system works correctly during a period of time under specified conditions. In other words, reliability is a measure that indicates for how long the system can work correctly even when faults affect parts of this system.

Fault tolerance techniques implemented in computer systems directly impacts on reliability. A fault-tolerant system is capable of taking actions to prevent that faults lead to errors and consequently to system failures that affect the correct functioning of system. Therefore, the system can work properly for a longer time in spite of faults.

To describe the reliability of a system, it is necessary to model it taking into consideration some important aspects related to the system design and the physical characteristics that influence the probability of failures.

In this section, we describe the reliability model and present details about reliability analysis, as well as how to use this strategy to improve architecture design targeting reliability.

3.1 Reliability Modeling

To model the reliability of a computer system, two essential information must be included to ensure a correct description of the system. The first information consists in the description of the components and the relationship between them.

In this work, we describe the components and the relationships using the reliability block diagram representation (KUO and ZUO, 2003). As the name indicates, the components are represented by blocks and the relationship between blocks represents the way the components connect to each other into the system.

Besides that, it is also necessary to specify the probability law that governs the failures (KUO and ZUO, 2003). For electronic devices, the probability law is assumed to be an exponential failure distribution called *exponential failure law*, which is a function of the failure rate (λ) and time. The failure rate describes the amount of errors that will occur in time. It is a constant determined by the device model (which is a function of parameters that describe physical and operating conditions of the device), and the environmental conditions in which the device operates.

According to Vigrass (2012), the failure rate is defined as a function of the *total device hours* and an acceleration factor. The total device hours is calculated based on the number of units that compose the device and the hours that the device remained under stress (during the test of the device). Moreover, the acceleration factor is a

function of the temperature, the thermal activation energy and the Boltzmann's constant (k).

Equation (1) presents the reliability function for an electronic device,

$$p(t) = e^{-\lambda t}, \quad (1)$$

where t is the time in hours and λ is the failure rate defined by equation (2),

$$\lambda \propto \frac{1}{TDH\ X\ AF}, \quad (2)$$

where TDH in equation (2) is the total device hours (number of units x hours under stress) and AF is the acceleration factor given by equation (3),

$$AF = exp\left(\frac{E_a}{k}\left(\frac{1}{T_{use}} - \frac{1}{T_{stress}}\right)\right), \quad (3)$$

where E_a is the thermal activation energy, k is the Boltzmann's constant (8.63×10^{-5} eV/K), T_{use} is the used temperature and T_{stress} is the life test stress temperature.

In case of more tests are performed in the device, reliability is the result of the accumulation of all tests that have different failure mechanisms. Therefore, a comprehensive failure rate is desired. In this case, the failure rate is defined as a function of the total device hours, the acceleration factor and other parameters defined based on the number of tests and the different failure mechanisms adopted (VIGRASS, 2012).

An electronic system consists of several independent electronic devices, each one with a different reliability determined by equation (1). The total reliability of the system is a function of the reliabilities of all electronic devices that compose the system.

3.1.1 Block diagram representation

After defining the individual reliability of the components, it is necessary to determine their connection with each other. In the block diagram representation, each component is represented as a block and the connections among components are represented by edges.

In a system with no redundancy, all the components are essential to the proper functioning of the system. To represent this relationship, the components are connected by only one edge, forming a series connection. Systems composed of only series connections are called series system. Figure 6 illustrates a series system with two identical components.

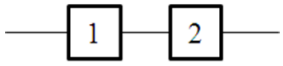


Figure 6. Series system block diagram

Based on probability properties presented in (KUO and ZUO, 2003), the probability that a series system works at time t is the probability that all the components that compose this system work. Thus, the series system reliability function is given by:

$$R_{series}(t) = \prod_{i=1}^{n} \left(p_i(t)\right), \tag{4}$$

where *n* is the number of components connected in series.

When hardware redundancy is added to increase system reliability, two main strategies can be used. In the first strategy, the extra components can work in parallel and in case one fails, the result of the other component is used. Another alternative of exploring redundancy is leaving the extra components in idle state until the original component presents a fault and cannot work properly anymore. In this case, some control unit (or the user) must replace the faulty component by the extra one. The extra components are called *spares*. In both cases, the redundant components are not essential to system operation. To represent this, the components are connected to each other in a parallel fashion. Figure 7 illustrates a parallel system with three identical components. The parallel system reliability function is defined in equation (5).

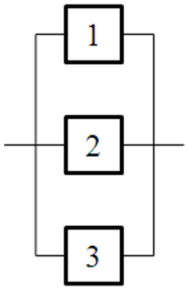


Figure 7. Parallel system block diagram

$$R_{parallel}(t) = 1 - \prod_{i=1}^{n} (1 - p_i(t)), \tag{5}$$

where *n* is the number of components connected in parallel.

A third representation also used in redundant systems describes a situation when from all the components, only few of them need to work so the system can work correctly. This is called *k-out-of-m* system, where *m* is the total number of components and *k* is the number of components required. For a completely parallel system *k=1* and for a completely series system *k=m*. The k-out-of-m reliability function is given by equation (6).

$$R_{k-out-of-m}(t) = \sum_{i=k}^{m} \binom{m}{i} p(t)^i (1 - p(t))^{m-i}. \tag{6}$$

Usually, the reliability function of complex systems is defined by the combination of several subsystems that can be parallel, series and k-out-of-m. There are other ways to describe the reliability of a computer system, such as the *Fault Tree Analysis* (ERICSON, 1999). However, because block diagram is a simple, widely used model,

we have adopted this representation to reliability modeling and analysis presented in this work.

3.2 Reliability Estimation

In this section, we discuss how to analyze the system`s reliability in order to consider it reliable. The main goal of this discussion is to provide a better understanding about how to interpret the reliability results used in the analyses presented in this work, and what is considered a high and low reliability.

We start this section firstly mentioning that in our analyses we always seek solutions to prolong the time the system reliability is 1, i.e. the system is 100% reliable. However, depending on the application, a lower reliability is also acceptable.

To define the acceptable reliability used in this work, we used another metric that is also employed in reliability measuring, the Mean Time Between Failures (MTBF).

MTBF is the average time between consecutive failures that happen in a system. This metric is used only for repairable systems, where after repairing the failure, the system returns to its correct operation. We use this metric because in our analysis we consider fault-tolerant system, with the capability of cope with faults in their components. For non-repairable systems, the metric used is the Mean Time to Failure (MTTF) (PRADHAN, 1996).

MTBF is often used by manufacturers to indicate the reliability of their electronic products. This metric is combined with the product useful (or service) life, and it is applied to the aggregate analysis of a large number of products. For example, the specification of the Intel SSD 320 Series hard drive (HD) indicates a MTBF of 1,200,000 hours (INTEL, 2012). Moreover, the minimum useful life is 5 years. This means that a product of this type is supposed to last for 5 years, and that a large group of HDs operating within this timeframe will accumulate, on average, 1,200,000 operational hours before the first failure affects any of the HDs.

The correlation between reliability R(t) and MTBF is described in equation (7),

$$R(t) = e^{-\left(\frac{t}{MTBF}\right)}. \quad (7)$$

Assuming the statistics used to Intel's HD MTBF, the reliability would be:

$$R(t) = e^{-\left(\frac{5*365*24}{1{,}200{,}000}\right)} = 0.9642.$$

This result indicates that after 5 years (43,800 hours) the reliability is approximately 0.96, i.e. if one of this hard drives is used 24 hours a day for 5 years, the probability of it surviving this time is about 96% (EPSMA, 2005). Based on the statistics presented above, in this work, we also assume a reliability of 96% in 5 years as acceptable in the analyses.

Therefore, the analyses presented in this work evaluate reliability in two different estimations: the instant in time where reliability decreases from 1 to 0.99999, and the instant in time where reliability is 0.96. In the first case, the goal is to evaluate for how long the architecture remained 100% reliable (or very close to this). The second case

helps to identify if, in spite of the architecture's reliability decreases, it still presents an acceptable reliability.

3.3 Fault Model and Failure Rate

The research presented in this work is concentrated in finding the best redundancy strategy to increase reliability. For this, we assume that a testing tool detects and diagnoses the faulty resources, by using classical testing techniques (LEE, BASOGLU and SULLIVAN, 2011). Therefore, proposing a testing mechanism is out of the scope of this work. In spite of that, some observations about the fault model and the failure rate used in this work are presented next.

3.3.1 Fault Model

The first observation is related to the fault model. In order to differentiate between a good resource and a faulty one, a fault model must be designated. The model typifies the representation of the physical faults in a higher level of abstraction, and it is used by the testing techniques to detect faults (ABRAMOVICI, BREUER and FRIEDMAN, 1994).

In this work, two types of permanent faults are considered, the manufacturing defects and the permanent faults that happen during useful lifetime of the circuit. For manufacturing defects, we consider the single stuck-at fault model (SSAF model), which is the most widely used model.

The SSAF model, first published in (GALEY, NORBY and ROTH, 1961), assumes that the effect of the fault is to tie individual lines to either V_{DD} (logical 1) or Gnd (logical 0). The model also assumes that there is only one faulty line in the circuit at a time. Additionally, the node can be gate or module level, with the most common solutions being at gate level.

The main advantages of the single stuck-at model are:

- Detects 2n faults, where n is the number of nodes (gates or modules)
- Requires low computational effort for automatic test pattern generation (ATPG)
- Covers many possible manufacturing defects in CMOS circuits, such as source-drain shorts, metallization shorts, diffusion contaminations, etc.

Additionally, for defects not covered by the SSAF model, (e.g. breaking of a line internal to a CMOS gate) other fault models can be mapped into sequences of single stuck-at faults. For the aforementioned example, the stuck-open fault model can be used (WADSACK, 1978).

For permanent faults caused by aging effect, several studies indicate that many of aging effects cause a performance degradation before the total breakdown of the device. Some examples are negative bias temperature instability and soaring leakage power (YAN, HAN and XIAOWEI, 2009), (PAUL, KUNHYUK, KUFLUOGLU, *et al.*, 2005). For these types of faults, some models have been proposed to detect the delay and predict the failure before it actually happens. According to (YAN, HAN and XIAOWEI, 2009), the aging delay can be modeled as a fault model to estimate the performance degradation. In addition, stuck-at fault model can also be applied to detect faults that cause the complete breakdown of the circuit. Furthermore, in opposite to manufacturing defects that can be tested and detected immediately after manufacturing,

for aging effects, the tests should run in periodical intervals, during the useful life of the device.

3.3.2 Failure Rate

The second observation is related to the failure rate (λ) described in equations (2) and (3). The equations demonstrate that the failure rate is calculated based on parameters defined during test. The total device hours and the life test stress temperature are examples of these parameters. In this work, we have adopted the failure rates presented in (SMITH, 2007). According to the authors, one can find many collections of failure rate data for different fields, such as defense, telecommunications, process industries, etc. However, these data were mainly collected during the 1980s, and after this date, the published sources stopped updating these data. For this reason, it became very difficult to find information about these failure rates in current technologies. In spite of that, the data published by the time are still of great value for research community and still used as reference data at the present time (SMITH, 2007).

For electronic devices, the most used failure rates are the ones published in *US Military Handbook 217* (MIL-217) (US DEPARTMENT OF DEFENSE, 1986) from RADC (Rome Air Data Centre in the USA). Although MIL-217 is widely used as reference document in failure rates, there is no detailed description about the tests and the conditions that the testes were performed. What it is known is that the tests took into consideration faults caused when the devices were exposed to high temperatures and voltages. The exposure time and the environment conditions were also considered (US DEPARTMENT OF DEFENSE, 1986).

4 RECONFIGURABLE ARCHITECTURE

The reconfigurable architecture and the configuration mechanism were firstly proposed by Beck et al, in 2008 (BECK, RUTZIG, GAYDADJIEV, *et al.*, 2008). From the initial architecture to the current one presented in this work, many design improvements were made, mainly to improve performance and reduce power consumption (LO, BECK, RUTZIG, *et al.*, 2010), (FAJARDO, RUTZIG, BECK, *et al.*, 2011), (RUTZIG, BECK and CARRO, 2011). In this work, we have modified the architecture to include a fault tolerance strategy targeted to tolerate manufacturing defects and permanent faults.

This section starts with a description of the architecture and its reliability model. Then, we present the fault-tolerant strategy that we have included in the architecture. The reliability model considering the fault tolerance approach is also described. Finally, we present the reliability analysis, followed by some possible modification on the architecture to improve overall reliability. To conclude the chapter, we present a performance degradation analysis and considerations about energy consumption based on simulations.

4.1.1 Architecture Description

The system consists of a coarse-grained reconfigurable array (RA) tightly coupled to a MIPS R3000 processor; a mechanism to generate the configuration; and the reconfiguration memory that stores the configuration. Figure 8 presents the block diagram of the complete system.

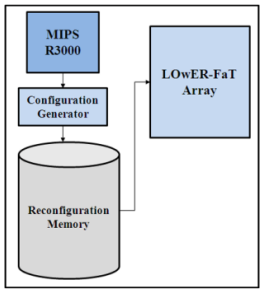


Figure 8. Coarse-grained reconfigurable system block diagram

Reconfigurable Array (RA)

The RA consists of a combinational circuit that comprises three groups of functional units: the arithmetic and logic units group (ALUs), the load/store units group

and the multiplier group. Figure 9 presents the RA block diagram and the parallel and sequential paths of the array.

Each group of functional units can have a different execution time, depending on the technology and implementation strategy. Based on this, in this work, the ALU group can perform up to three operations in one equivalent processor cycle and the other groups execute in one equivalent processor cycle. The equivalent processor cycle is called level. The different execution times presented by each group of functional units allow the execution of more than one operation per level. Therefore, the RA can perform up to three arithmetic and logic operations that present data dependence among each other in one equivalent processor cycle, consequently accelerating the sequential execution. Moreover, the execution time can be improved through modifications in functional units and with technology evolution, consequently increasing the acceleration of intrinsically sequential parts of a code. Even non-parallel code can have a better performance when executed in the architecture illustrated in Figure 9.

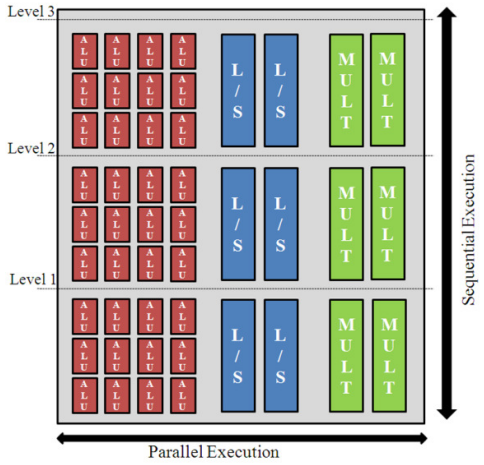


Figure 9. Reconfigurable architecture block diagram

Although the architecture depicted in Figure 9 is partitioned in levels, it is important to highlight that there is no sequential logic among the levels. All the architecture from the first row to the last row is combinational. The communication among the functional units is provided through buses and multiplexers. The buses are called context lines and receive data from the context registers, which store the data from the processor's register file. Figure 10 illustrates the interconnection model.

As depicted in Figure 10, the reconfigurable architecture design presents a bottom-up datapath. All data that is generated by the functional units flow from the bottom row to the top row through the interconnection model. Moreover, instructions placed in the same row are executed in parallel, since there is no data flowing from one unit to the other in the same row. Dependent instructions are executed in different rows because data among units flow from bottom to top of the architecture. Therefore, to exploit the instruction level parallelism (ILP) the instructions must be allocated in the same row.

There are two groups of multiplexers: the input group and the output group. The former selects data used by each functional unit, and the latter selects the context line

that receives the result from the operations. Moreover, each output multiplexer also has the context registers as input. This input is used when the multiplexer needs only to bypass the previous data without the need of any functional unit.

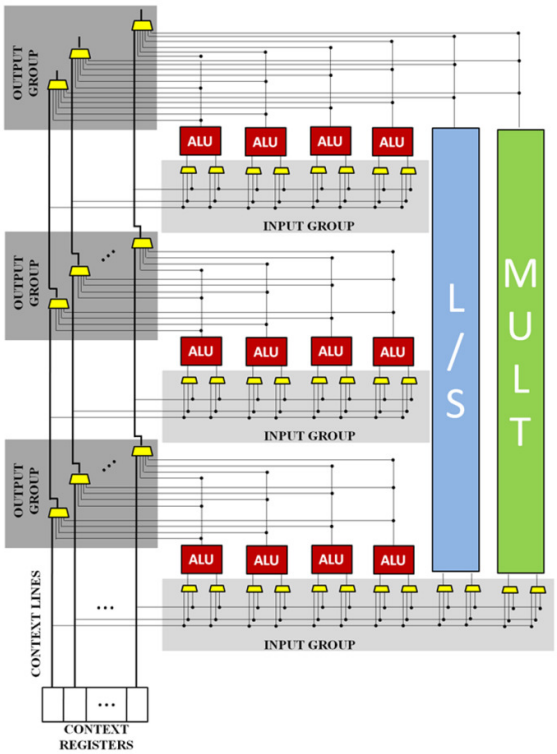


Figure 10. Reconfigurable architecture interconnection model

MIPS R3000 Processor

MIPS R3000 is a 32-bits RISC (Reduced Instruction Set Computer) processor developed by MIPS Technologies (former MIPS Computer Systems) (KANE and HEINRICH, 1992).

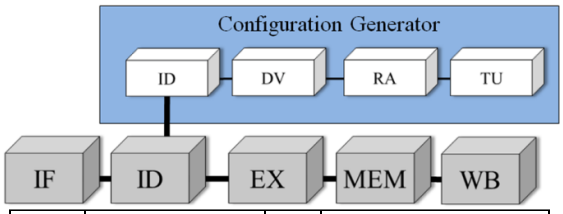
It consists in a 5-stages pipeline processor with an on-chip cache controller and memory management unit, and it supports up to 4Gb direct memory addressing and 512Kb cache. Figure 11 illustrates the five stages of MIPS R3000 processor.

Configuration Generator

The Configuration Generator (CG) implements a mechanism that dynamically transforms sequences of instructions to be executed in the array. The transformation process is transparent, with no need of instruction modification before execution, preserving software compatibility. Furthermore, the CG works in parallel with the processor's pipeline, presenting no extra overhead to the processor. Figure 11 illustrates the CG's steps attached to the MIPS R3000 pipeline.

To generate the configuration, the first stage of CG's pipeline (ID) selects the instructions that can be executed in the array, and breaks these instructions into operations. Next, the dependence verification stage checks data dependence among the

current instruction and the previous ones that were already analyzed. Based on data dependence, in the resource allocation stage, the CG searches for available functional units to perform resource allocation. Both data dependence and resource availability verification are performed through the management of tables that are updated in the last stage (TU). At the end of CG's pipeline, a configuration is generated and stored in the reconfiguration memory indexed by the program counter (PC) value of the first instruction from the sequence.



IF	Instruction Fetch	ID	Instruction Decode
ID	Instruction Decode	DV	Dependence Verification
EX	Execution	RA	Resource Allocation
MEM	Memory Access	TU	Table Update
WB	Write Back		

Figure 11. Configuration Generator's pipeline coupled to the processor's pipeline

The tables updated in the last pipeline stage are used to analyze data dependence, allocate instructions in the functional units and route the operands inside the array as described:

1. Write bitmap table: to find true data dependences (RAW - read after write), this table keeps track of which registers will be written by each functional unit of each row.
2. Resource map: each position of the table corresponds to a functional unit. When a FU is allocated, the position is set to 1, indicating that the functional unit is busy.
3. Read table: it informs which operands must be read by each functional unit. The operands are initially stored in the context registers.
4. Write table: it indicates which context line will receive the result of an operation performed by the functional unit. It is also possible to bypass data through the context line without send it to a functional unit. This is also indicated in the write table.
5. Context table: this table is used to manage the input in the context registers. It has two rows. The first row indicates the input context and it will be used during the configuration phase to fetch data. The second row keeps track of the context used during execution phase and it will be used to inform which data will be written in the end of the RA's execution (write back).

Although all the tables are used to generate configuration, the fault tolerance approach proposed in this work only requires modifications on the resource map. For this reason, we will present more details about the management of this table by the CG.

The resource map consists in a table where each position represents one functional unit of the array. The positions in the rows of the resource map (X-axis) represent the

parallel functional units and the positions in the columns (Y-axis) represent the sequential execution. Considering a faulty-free architecture, when CG starts a configuration all positions of the resource map are set as free. Every time a functional unit is allocated to perform an operation, its relative position in the resource map is set as busy. Therefore, to select a functional unit, after checking data dependence, CG needs to search in the resource map a free unit and generate the configuration bits for that unit and its multiplexers. If there are no more available units, then CG breaks the configuration and starts a new one with all units set as free. Figure 12 presents an activity diagram describing the configuration generation algorithm.

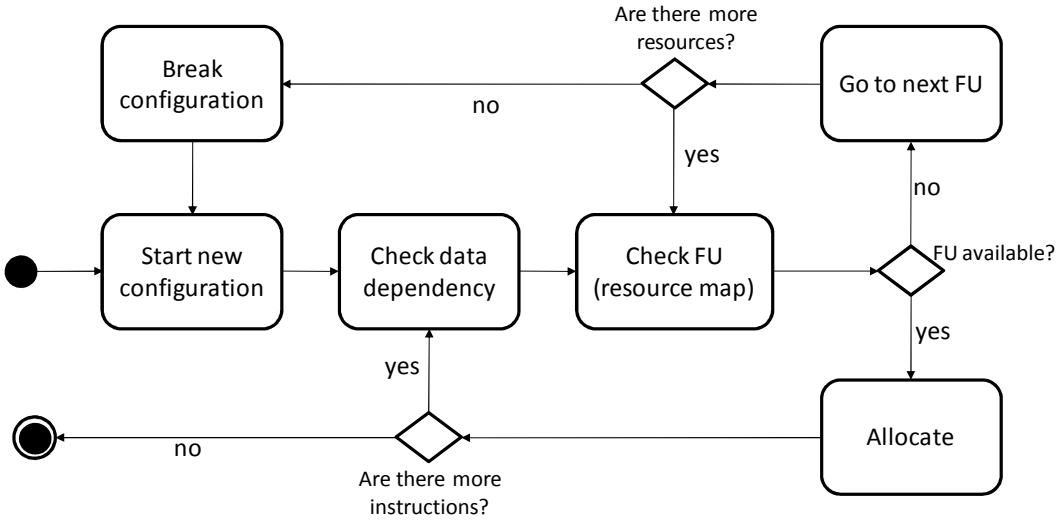
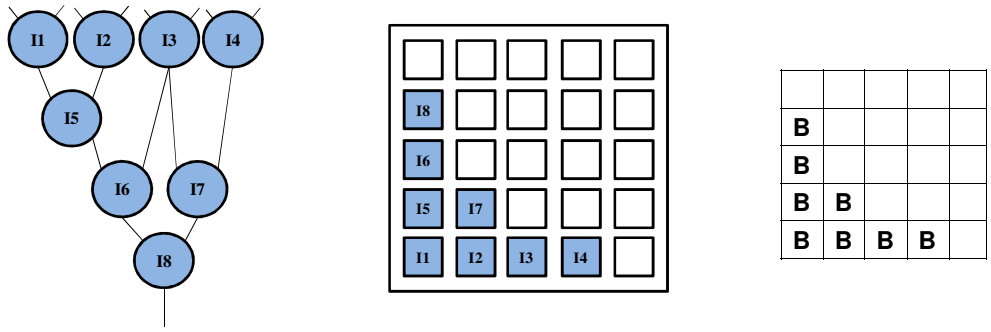


Figure 12. Configuration generation algorithm - Fault-free approach

According to the activity diagram presented in Figure 12, to allocate resources in the RA, the first step is to create a new configuration. This means that, after detecting that the instruction can be executed in the RA, the CG starts decoding the instruction. Next, the data dependencies are checked using the write bitmap table. Then, based on data dependence, the generator keeps searching in the resource map for an available resource. If the resource is found, it is set as busy in the resource map and the generator goes to the next instruction. Otherwise, if there are no resources available, the generator breaks the configuration and starts a new one with the current instruction as the first one.

Figure 13 presents an example to illustrate how the resources are allocated using the resource map. Figure 13.a presents the data dependence graph of the instruction sequence mapped to the array. Figure 13.b represents the array with 25 identical functional units, and Figure 13.c presents the resource map. For clarity's sake, in this example we are considering only arithmetic and logic units. As can be observed, the resource map has 25 positions, each one representing one resource of the array.

As described in last section, the parallel instructions are placed in the same row of the array and the dependent instructions, which must be executed sequentially, are placed in different rows. To select the functional units to execute the instructions, the mechanism starts by allocating the first available unit (bottom-left). The next instructions are placed according to data dependence and resource availability.



a) Data dependence graph b) Resource allocation c) Resource map

Figure 13. Fault-free resource allocation

Based on Figure 13.a the instructions 1; 2; 3; and 4 do not have data dependence among each other, hence they are all placed in the same row of Figure 13.b. On the other hand, instruction 5 is dependent on the result of instructions 1 and 2. Hence, instruction 5 is placed in the first available unit of the second row. Continuing the allocation, the inputs of instruction 6 are the outputs of instructions 3 and 5. Since instruction 5 is in second row, instruction 6 must be placed in third row. Instruction 7 depends on instructions 3 and 4 that are placed in the first row. Thus, it can be placed in second row. Moreover, since instruction 8 depends on instructions 6 and 7 that are in the third and second rows, respectively, instruction 8 must be placed in fourth row. Additionally, as one can observe in Figure 13.c, the positions of the resource map that correspond to the allocated functional units are set as busy. In this example, all the positions are already set as busy but, in real execution, the positions are filled one per time during run-time.

Reconfiguration Memory

Two storage components are part of the reconfigurable architecture: address cache and reconfiguration memory. The address cache holds the memory address of the first instruction of every configuration built by the configuration generator.

It is used to check the existence of a configuration in the reconfiguration memory: an address cache hit indicates that a configuration was found. The address cache is implemented as a 4-way set associative table containing 64 entries. The reconfiguration memory stores the routing bits and the necessary information to fire a configuration, such as the input and output contexts, the immediate values and the operation executed by each functional unit.

Reconfiguration and Execution

While CG generates and stores the configuration, the processor continues its execution. Next time a PC from a configuration is found the processor changes to a halt stage, the respective configuration is loaded from the reconfiguration memory and the RA's datapath is reconfigured. Moreover, all input data are fetched. Finally, the configuration is executed and the registers and memory positions are written back.

It is important to highlight that the overhead introduced by the array reconfiguration and data access are amortized by the acceleration achieved by ILP exploitation. Moreover, as mentioned before, the configuration generation does not impose any overhead.

4.1.2 Reliability Model

In this section, we present the reliability model of the reconfigurable architecture described above. This model will be used as reference to evaluate all modifications performed in the architecture aiming to increase reliability.

The reliability modeling starts by finding the *atomic* component of the system, i.e. the component that cannot be divided into smaller components. This component (or components) will have the reliability defined by equation (1). The next step consists in defining the connection between the components. The connected components form a subsystem. Hierarchically, each subsystem connects to each other forming the complete system.

Although the functional units can be divided into smaller components, since we are working in a coarse grain level, in this work we assume that the functional units are atomic elements, and therefore, the smallest components of the architecture.

Based on this, the reliability as a function of time of the ALU, Multiplier and Load/Store are:

$$R_{ALU}(t) = e^{-\lambda_{ALU}t}, \quad (8)$$

$$R_{Multiplier}(t) = e^{-\lambda_{Multiplier}t}, \quad (9)$$

$$R_{LoadStore}(t) = e^{-\lambda_{LoadStore}t}. \quad (10)$$

The reconfigurable architecture has two types of multiplexers. They differ between each other only in the number of inputs. However, each input is 32-bits long. To determine the reliability function of the interconnections, we decomposed the multiplexers as a subsystem composed of 2:1 1-bit multiplexers. Therefore, each 32-bits multiplexer form the first subsystem of the architecture. Assuming that the multiplexers have no redundancy targeted to fault tolerance, all the 2:1 1-bit multiplexers must work, so the multiplexer subsystem can work correctly. In this case, the multiplexer subsystem consists in a series system, whose reliability function is given by equation (12). Since the input and output multiplexers have different number of inputs, each one has a different reliability function, described in equations (13) and (14). The reliability of the 2:1 1-bit multiplexer is determined by equation (11). Figure 14 illustrates an 8:1 1-bit multiplexer subsystem.

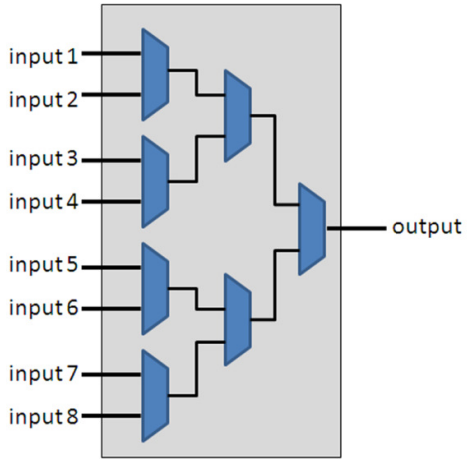


Figure 14. Chain of multiplexers 2:1 1-bit

$$R_{1bMux}(t) = e^{-\lambda_{Mux}t} \tag{11}$$

$$R_{32bMux}(t) = \prod_{i=1}^{32} \left(R_{1bMux[i]}(t)\right) \tag{12}$$

$$R_{inputMux}(t) = \prod_{i=1}^{ip} \left(R_{32bMux[i]}(t)\right) \tag{13}$$

$$R_{outMux}(t) = \prod_{i=1}^{op} \left(R_{32bMux[i]}(t)\right) \tag{14}$$

where *ip* is the number of 2:1 32-bits multiplexers that compose an input multiplexer, and *op* is the number of 2:1 1-bit multiplexers that compose an output multiplexer.

The next subsystem in the hierarchy is the functional units connected to the input multiplexers. As illustrated in Figure 10, each functional unit has two input multiplexers. The functional unit can only operate properly if the correct data is sent to its inputs through the input multiplexer. To represent this restriction in the model, the connection among the multiplexers and the functional unit must be in series. Therefore, the input multiplexers and the functional unit form a series subsystem with reliability function given by equations (15), (16) and (17) for ALU, multipliers and load/store, respectively.

$$R_{ALU+inputMux}(t) = R_{ALU}(t)\, x \prod_{i=1}^{2} \left(R_{inputMux[i]}(t)\right), \tag{15}$$

$$R_{Mult+inputMux}(t) = R_{Multiplier}(t) x \prod_{i=1}^{2} \left(R_{inputMux[i]}(t)\right), \tag{16}$$

$$R_{LoadStore+inputMux}(t) = R_{LoadStore}(t)\, x \prod_{i=1}^{2} \left(R_{inputMux[i]}(t)\right), \tag{17}$$

The next subsystem in the hierarchy is the functional units subsystem. In spite of the fact that the functional units in one row work in parallel and independently, the fact that there is no fault tolerance strategy included in the architecture implies that a fault does not interfere in the proper system functioning if and only if this fault damages a unit that is not being used. Given the fact that the architecture presents a large amount of identical units and the selection of the units that will be used is dynamically determined, there is no way to know *a priori* if the faulty unit will be used or not.

Based on this, two assumptions can be made: 1) all units are used and therefore, if one fails, it may lead to a system error. 2) only part of the units is used. In this case, there is a probability that the fault affects the unit that is not being used.

The total amount of units and their usage is dependent on the architecture design constraints and application execution. For this reason, we are going to discuss more about this in the reliability analysis section. Therefore, at this point, it is not possible determine which assumption will be used. However, as described in chapter 3, the k-

out-of-m system, with function given by equation (6), can be used in both cases. In the first assumption, all units are used, forming a series system with k=m. In the second assumption, k depends on the amount of units used.

Therefore, a row composed of parallel ALUs forms a k-out-of-m subsystem with function given by equation (18),

$$R_{ALU\_row}(t) = \sum_{i=k\_row}^{m\_row} \binom{m\_row}{i} \left(R_{ALU+inputMux}(t)\right)^{i} \left(1 - R_{ALU+inputMux}(t)\right)^{m\_row-i}, \quad (18)$$

where *m_row* is the total number of ALUs in a row and *k_row* is the amount of used ALUs.

As demonstrated in Figure 10, in each row there is a set of output multiplexers. Even assuming that only part of the ALUs in one row is necessary, all the multiplexers are required. This happens because the multiplexers not only select the output of the ALUs, but they also bypass the data from the context registers. In this case, without fault tolerance strategy, if a multiplexer fails, it is not possible to send data throughout the architecture. For this reason, the output multiplexers in one row form a series subsystem, with reliability function given by equation (19). Moreover, ALU and the output multiplexers subsystem are also connected in series, with function described in equation (20),

$$R_{OutMuxesRow}(t) = \prod_{i=1}^{or} \left(R_{OutMux[i]}(t)\right), \quad (19)$$

where *or* is the number of output multiplexers in one row and

$$R_{ALU_{row}+outMuxes}(t) = R_{ALU_{row}}(t) x R_{OutMuxesRow}(t), \quad (20)$$

Following the same reasoning applied to the ALUs, the multipliers and load/store units form k-out-of-m subsystems. However, in this case, it is considered the amount of units in one level. Equations (21) and (22) present the reliability function of the multipliers and load/store units, respectively.

$$R_{Mult\_level}(t) = \sum_{i=k\_mult}^{m\_mult} \binom{m\_mult}{i} \left(R_{Mult+inputMux}(t)\right)^{i} \left(1 - R_{Mult+inputMux}(t)\right)^{m\_mult-i}, \quad (21)$$

$$R_{LoadStore\_level}(t) = \sum_{i=k\_ls}^{m\_ls} \binom{m\_ls}{i} \left(R_{LoadStore+inputMux}(t)\right)^{i} \left(1 - R_{L/S+inputMux}(t)\right)^{m\_ls-i}, \quad (22)$$

where *m_mult* is the total number of multipliers and *k_mult* is the used amount, in equation (21), and *m_ls* is the total amount of load/store units and *k_ls* is the used amount, in equation (22).

Thus, the *ALU_row, Mult_level* and *LoadStore_level* subsystems and the *OutMux* subsystem form a series subsystem, with functions presented in equations (23) and (24). Because there are three rows of ALUs in each level, in equation (23), the *ALU_row* is replicated three times.

$$R_{PartialLevel}(t) = \prod_{i=1}^{3} \left(R_{ALU_{row}+outMuxes[i]}(t) \right) x\, R_{Mult\_level}(t) x\, R_{LoadStore\_level}(t), \quad (23)$$

$$R_{Level}(t) = R_{PartiaLevel}\, x\, R_{OutMuxesLevel}(t), \quad (24)$$

where $R_{OutMuxesLevel}$ is the reliability of the output multiplexers in one level, determined by equation (25).

$$R_{OutMuxesLevel}(t) = \prod_{i=1}^{ol} \left(R_{OutMux[i]}(t) \right), \quad (25)$$

To ensure the correct system operation, it is necessary that all levels of the reconfigurable architecture remain working. Therefore, the reliability of all levels is a series system with reliability function given by equation (26), where *l* is the number of levels,

$$R_{Levels}(t) = \prod_{i=1}^{l} \left(R_{Level[i]}(t) \right). \quad (26)$$

Finally, to complete the reliability model, all levels are connected in series to the context registers subsystem, which is also a series subsystem. The context register subsystem is described in equations (27) and (28). The reconfigurable architecture is described in equation (29),

$$R_{Register}(t) = e^{-\lambda_{Register} t}, \quad (27)$$

$$R_{ContextRegister}(t) = \prod_{i=1}^{ct} \left(R_{Register[i]}(t) \right), \quad (28)$$

$$R_{ReconfigurableArray}(t) = R_{Levels}(t) x R_{ContextRegister}(t). \quad (29)$$

In section 4.5, we present a quantitative analysis taking into consideration the total amount and the distribution of resources along the architecture. However, by just analyzing the equations described above, it is possible to observe that most of the subsystems are composed of series connections, which is a strong indicator that a low system reliability should be expected. Therefore, some fault tolerance strategy should be introduced as a mean to improve reliability.

In the next section, we present the model of the LOwER-FaT architecture, which presents fault tolerance strategy to cope with faults in functional units and multiplexers.

4.2 LOwER-FaT Array

The main characteristics of the fault tolerance strategy implemented in the reconfigurable architecture are:

- Tolerates manufacturing defects and permanent faults;
- Introduces low area, performance and power overheads;
- Does not require the addition of spare resources;
- Works at run-time.

In this work, we propose to exploit the intrinsic redundancy of the reconfigurable architecture by using the working functional units to replace the faulty ones. In this approach, it is not necessary to add spare resources, which would increase area. Therefore, in a fault-free situation, all the functional units are used to accelerate application execution and only in case of fault, a functional or interconnect unit is replaced. Furthermore, the reconfiguration capability is exploited to change the resources allocation based on the faulty and operational resources. In addition, dynamic reconfiguration can be used to avoid faulty resources and generate a new configuration at run-time. Thus, there is no performance penalty caused by the allocation process, nor extra steps in the manufacture are required to correct each circuit. Finally, the capability of adaptation according to the application can be exploited to amortize the performance degradation caused by the replacement of faulty resources by working ones, as it will be shown in section 4.5.3.

Since the proposed fault tolerance approach handles only manufacturing defects and permanent faults, in this work we assume that the information about the faulty resources is generated before the array starts its execution, by some classical testing techniques (LEE, BASOGLU and SULLIVAN, 2011).

To differentiate the fault-free array and the one with our proposed fault tolerance approach, we will refer to the fault-tolerant array as LOwER-FaT array (**L**ow **O**verhead **w**ithout **E**xtra **R**edundancy **Fa**ult-**T**olerant array). Next, we present the proposed solution to tolerate faulty functional units and interconnects.

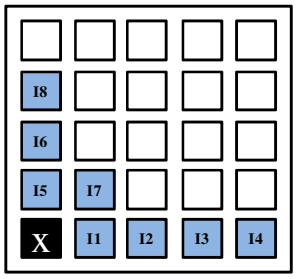
4.2.1 Architecture Description

4.2.1.1 Functional Units

To tolerate faulty functional units, the allocation algorithm is exactly like described in the example of Figure 13. The only difference is that to allocate only the resources that are effectively able to work, before the array starts and after the traditional testing steps, all the faulty units are set as permanently busy in the resource map, as if they had been previously allocated. With this approach, no modification in the configuration generator algorithm itself is necessary.

To demonstrate the approach to tolerate faulty functional units, Figure 15 presents two examples based on the example of Figure 13.

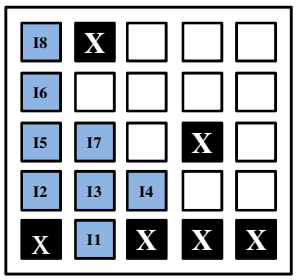
In Figure 15.a, the configuration mechanism placed the instruction in the first available unit, which, in this case, corresponds to the second functional unit of the first row. Since the first row still has available resources to place the four instructions, the LOwER-FaT array sustains its execution time. In this case, the presence of a faulty functional units does not affect the performance.



a) Not affecting performance

B				
B				
B	B			
B	B	B	B	B

b) Resource map – example I



c) Affecting performance

B	B			
B				
B	B		B	
B	B	B		
B	B	B	B	B

d) Resource map – example II

Figure 15. Resource allocation considering faulty functional units

Figure 15.c illustrates an example where faulty functional units affect the performance of the LOwER-FaT array. In this example, the first row has only one available functional unit. In this case, when there are not enough resources in one row, the instructions are placed in the next row, and all the data dependent instructions must be moved upwards. In Figure 15.c, instruction 5 is dependent on instruction 4. Hence, instruction 5 was placed in the next row, and the same happened with other instructions (6 to 8). In this example, because of the faulty units, it was necessary to use one more row of the LOwER-FaT array. Since the sequential path of the array flows from the bottom row to the top row, the use of one more row leads to the increase of execution time, consequently affecting performance. Figure 15.b and Figure 15.d illustrate the resource map of both Figure 15.a and Figure 15.c examples, respectively. As one can observe, the faulty units have their positions in the resource maps set as busy.

4.2.1.2 Multiplexers

To avoid faulty multiplexers from the interconnection model of Figure 10, the strategy can be different depending on which group of multiplexers is affected.

In case of a faulty input multiplexer, one of the inputs of the respective functional unit will have invalid data, consequently the result of the operation will be incorrect. Therefore, to keep the fault tolerance approach simple and avoid introducing overhead to the configuration mechanism, the strategy is to consider the multiplexer and its respective functional unit as faulty and place the instruction in the next available unit. Figure 16 illustrates the approach to tolerate faulty input multiplexers.

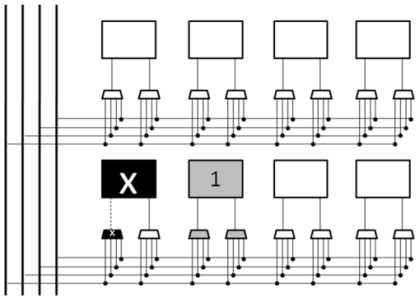


Figure 16. Faulty input multiplexer tolerance approach

As in the case of functional units, it is assumed that all faulty multiplexers are detected before CG starts. According to Figure 16, the faulty multiplexer invalidates the FU input (dashed line) and this invalidates the functional unit execution. Therefore, the approach is to avoid this FU and to place the instruction in the next available FU (grey resources).

On the other hand, a faulty output multiplexer invalidates part of the respective context line. Figure 17 illustrates a faulty output multiplexer. For clarity's sake, the input multiplexers were omitted. As can be observed in Figure 17, the data remains invalid from the faulty output multiplexer until the next valid output. The dashed line indicates that this part of the context line has invalid data. The context line will have valid data again when an instruction placed in any row, positioned after the faulty multiplexer, writes valid data in this context line.

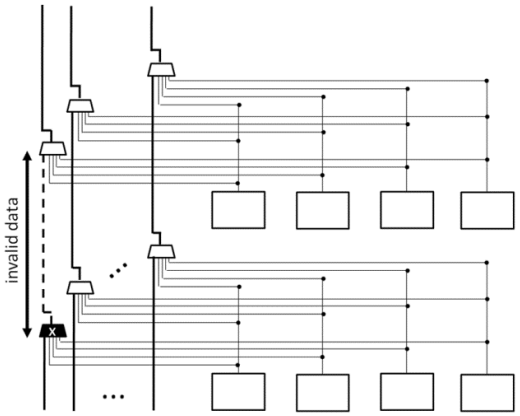


Figure 17. Example of a faulty output multiplexer

Therefore, if an output multiplexer is faulty, it is necessary to evaluate if the functional unit needs to read or write in this multiplexer.

When a functional unit needs to read data from a context line, and this same context line has a faulty output multiplexer in any previous row, the CG must check whether the context line is valid or not. Figure 18 presents an example to illustrate this situation, called reading case.

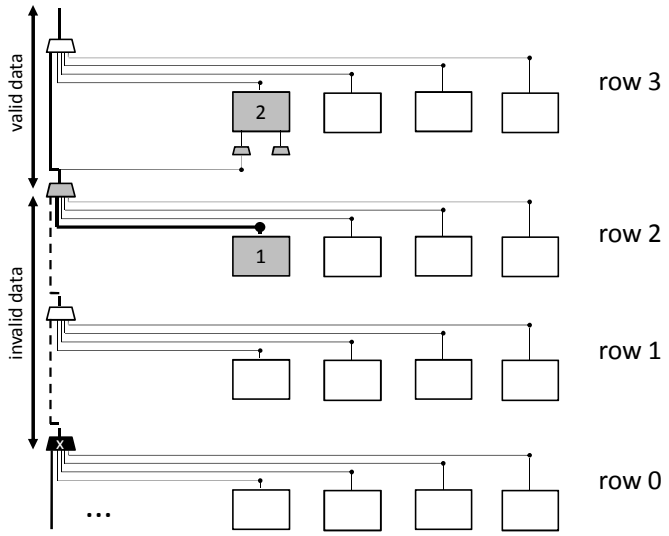


Figure 18. Faulty output multiplexer - reading case solution

In the example of Figure 18, instruction 2 that reads from the context line must be placed in row 3. However, the output multiplexer from row 0 is faulty. Before placing the instruction in row 3, the CG must check if any instruction placed in the rows between row 3 and row 0 wrote in this context line. From Figure 18, it can be observed that instruction 1, placed in row 2, has written in the context line making this line valid again. Thus, the instruction 2 can be placed in row 3 because it will use the output data from row 2.

In case there is no FU that writes in the context line, this context line remains invalid from the faulty output multiplexer until the last row of the array. Thus, the instruction cannot be placed in any functional unit and the configuration must be broken in two configurations. One configuration includes all the instructions that can be placed before the faulty multiplexer position and the other configuration will have the other instructions.

The second case is used when a functional unit needs to write in a context line and the output multiplexer is faulty. This case is called writing case and it is illustrated in Figure 19.

The solution to tolerate faulty output multiplexers in the writing case consists in placing the instruction in the next available FU. However, since each row has only one output multiplexer for each context line, the strategy in this case is to place the instruction in the next available FU of the next available row. Furthermore, it is also necessary to check if the output multiplexer from the next row is operational. Thus, the instruction can be placed in the next row (as demonstrated in Figure 19) or in any other row with fault-free multiplexer. For example, if the multiplexer from row 1 was also faulty, the instruction would be placed in the row 2 and all the instructions dependent on this instruction would be placed from row 3 upwards.

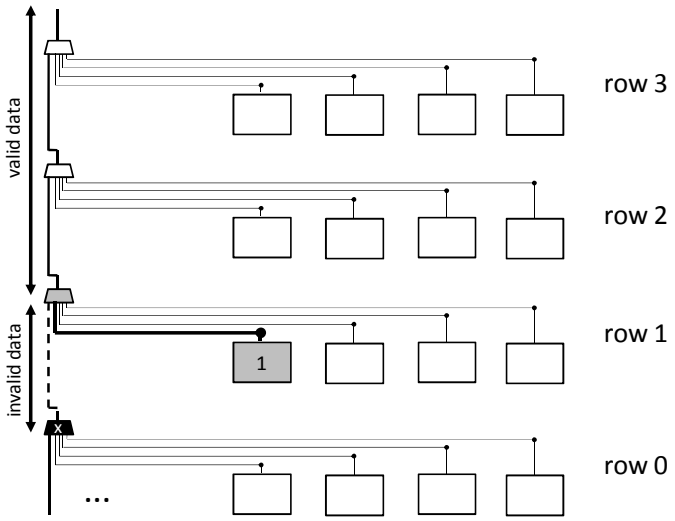


Figure 19. Faulty output multiplexer - writing case solution

Figure 20 presents the activity diagram summarizing the configuration generation algorithm including the faulty unit management. The algorithm is the same presented in Figure 12 with additional steps (grey boxes).

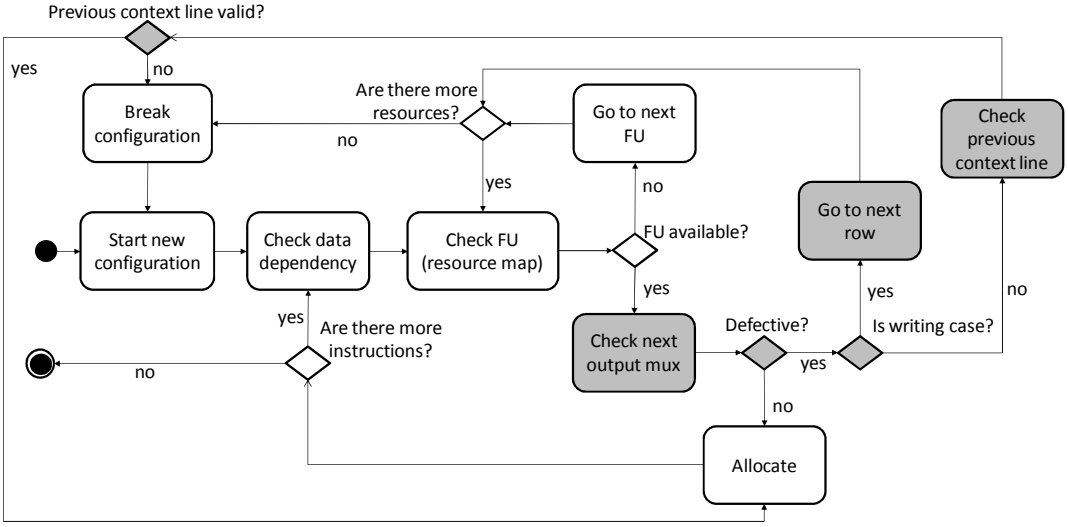


Figure 20. Configuration generation algorithm considering faulty units

As can be observed in the diagram, the additional steps are responsible for managing only the faulty output multiplexers. This is the only case that requires modifications in the algorithm. Moreover, the information about faulty functional units and input multiplexers is transparent to CG. This information is translated into the resource map that indicates which functional unit is available and which one is busy. Therefore, the FU that cannot be allocated due to faults is permanently set as busy.

Although some modifications are required to the faulty output multiplexer approach, the new steps included in the CG are part of the same algorithm already implemented and do not increase the number of cycles required to generate the

configuration. Therefore, the CG continues working in parallel with the processor's pipeline even with the fault tolerance approach. More important, since CG works in parallel to the processor's pipeline, no performance overhead is incurred while generating the configuration, since it takes fewer cycles than the processor's pipeline, even when fault tolerance is in place. Experimental results evaluating the performance degradation and fault tolerance efficiency were published in (PEREIRA and CARRO, 2009), (PEREIRA, LO and CARRO, 2009) and (PEREIRA and CARRO, 2011).

4.2.2 Reliability Model

Since this is the same architecture but now with fault tolerance strategy included, all the resources are the same. The difference is in the way the resources are connected to each other in the reliability model. For that reason, the functional units and multiplexers present the same reliability functions already described in equations (8) to (14).

The subsystem composed of functional unit and the two input multiplexers forms a series connection, which is the same type of connection described when the architecture has no fault tolerance. This happens because according to the fault tolerance strategy implemented, if an input multiplexer is faulty, the respective functional unit is considered as faulty (Figure 16). Thus, equations (15) to (17) are also used in the LOwER-FaT array model to represent the reliability function of the ALU, multiplier and load/store units with the input multiplexers.

Up to now, all the equations used to describe the reliability model of the reconfigurable architecture were also used to describe the LOwER-FaT array model. The difference in the modeling when fault tolerance strategy is included starts with the description of next subsystem, the row of ALUs.

In both systems, with and without fault tolerance, the arithmetic and logic units in one row work in parallel and independently. However, in the LOwER-FaT array, when an ALU is faulty, the fault tolerance strategy allows to move the operation that should be executed in the faulty ALU to a fault-free one. This is not possible in the architecture without fault tolerance, because the mechanism that generates the configuration does not know which are the faulty and fault-free units. In fact, the configuration generator does not even know about the existence of faulty units and that they should not be allocated. In this way, now that only the fault-free ALUs in one row are allocated, this is translated into a parallel subsystem. Since all the subsystems composed of ALU and input multiplexers are identical with reliability equal to $R_{ALU+inputMux}(t)$, from equation (5) we have:

$$R_{ALU\_row}(t) = 1 - \prod_{i=1}^{a}\left(1 - R_{ALU+inputMux[i]}(t)\right), \tag{30}$$

where a is the total number of ALUs in one row. The other functional units follow the same reasoning, but considering the level instead of the row. Therefore, the reliability functions of the multiplier subsystem and load/store subsystem are identical to equation (30). They are presented in equations (31) and (32), respectively,

$$R_{Mult\_level}(t) = 1 - \prod_{j=1}^{m}\left(1 - R_{Mult+inputMux[j]}(t)\right), \tag{31}$$

$$R_{LoadStore\_level}(t) = 1 - \prod_{k=1}^{l}\left(1 - R_{LoadStore+inputMux[k]}(t)\right), \quad (32)$$

where *m* is the number of multipliers in one level and, *l* is the number of load/store units in one level.

The result of an operation executed by the functional units is sent to the next functional units or to the registers. Since data can only flow in a bottom-up direction, the FUs' output is sent to the output multiplexers located on top of each row (Figure 10). Due to the fault tolerance approach, it is possible to avoid a faulty output multiplexer and assign the operation to a functional unit in a different row. This strategy is described in Figure 19. Although this strategy is possible only in one specific case (when the functional unit needs to write in the output multiplexer), it implies that not all the output multiplexers need to work to ensure that the system will work properly. This characterizes the k-out-of-m system. Therefore, equation (19) is replaced by equation (33), where *m_outMux* is the total amount of output multiplexers in one row, and *k_outMux* is the used amount,

$$R_{OutMuxesRow} =$$

$$\sum_{i=k\_outMuxAlu}^{m\_outMuxAlu} \binom{m\_outMuxAlu}{i} \left(R_{outMux}(t)\right)^{i}\left(1 - R_{outMux}(t)\right)^{m\_outMuxAlu-i}. \quad (33)$$

Another difference from the reference architecture model and the LOwER-FaT array one is to compose the level. Although all ALUs in one row are now connected in parallel, as described in equation (30), the output multiplexers in each row need to work, so the system can work properly. At the same time, the ALU rows are also connected in parallel. This is due to the fact that if one ALU in the level (no matter which row) works, the system works properly. Therefore, before connecting the output multiplexers to the ALUs, it is necessary to create a new equation to describe the connection between all ALU rows in one level. This equation is a new one introduced in this model, and does not replace any equation of the reference architecture model. The subsystem composed of ALU rows is called ALU block, and its reliability function is given by equation (34). Since there are three rows of ALUs in each level, equation (34) is a parallel subsystem with three rows,

$$R_{ALU\_block}(t) = 1 - \prod_{i=1}^{3}\left(1 - R_{ALU\_row[i]}(t)\right). \quad (34)$$

Moreover, with all the functional units working in parallel, the *PartialLevel* subsystem, described in equation (23), is replaced by equation (35) in the LOwER-FaT array model,

$$R_{PartialLevel}(t) =$$

$$1 - \left(\left(1 - R_{ALU\_block}(t)\right)x\left(1 - R_{Mult\_level}(t)\right)x\left(1 - R_{LoadStore\_level}(t)\right)\right). \quad (35)$$

At this point, all output multiplexers from all rows and from the end of the level are connected in series to the *PartialLevel* forming the *Level* subsystem. Therefore, in equation (36), the output multiplexer term is in fact a product of a sequence of output multiplexers from the rows. This equation replaces equation (24),

$$R_{Level}(t) = R_{PartialLevel}(t)x \prod_{i=1}^{out} R_{OutMuxesRow[i]}(t). \quad (36)$$

To ensure the correct system operation, it is necessary that all levels remain working. Therefore, the *Levels* subsystem is the same series subsystem described in equation (26) and reproduced in equation (37), where *l* is the number of levels,

$$R_{Levels}(t) = \prod_{i=1}^{l} R_{Level[i]}(t). \quad (37)$$

The last part of the reliability model is the connection with the context register subsystem. Although it is possible to avoid some of the faulty output multiplexers, all the context registers must work, so the architecture can receive the input context from the processor. Therefore, the context registers are the same series subsystem described in equations (27) and (28). Moreover, the connection between the levels and the context register subsystem is also in series. This means that the three last equations of the LOwER-FaT array model are the same as the reference architecture model. These equations, (27), (28) and (29), are reproduced in equations (38), (39) and (40), respectively,

$$R_{Register}(t) = e^{-\lambda_{Register}t}, \quad (38)$$

$$R_{ContextRegister}(t) = \prod_{i=1}^{ct} \left(R_{Register[i]}(t)\right), \quad (39)$$

$$R_{ReconfigurableArray}(t) = R_{Levels}(t)\ x\ R_{ContextRegister}(t). \quad (40)$$

4.3 Fault Detection

Since the focus of this work is in tolerating faults after they are detected, we do not specify any fault detection mechanism. Therefore, we assume that, when required, a fault detection mechanism is responsible for detecting and diagnosing the faulty resources, using some traditional testing technique, like the ones presented in (LEE, BASOGLU and SULLIVAN, 2011).

In case of aging effects, the fault detection mechanism should run in specific periods to detect resources that present faults along the device's lifetime. This is the detection mechanism required for the LOwER-FaT array. Although we do not specify the fault detection approach, we believe that a software level detection mechanism would be satisfactory to our requirements. Therefore, in specific intervals, the configuration generator would load a testing configuration that would test all the resources. Optimizations aiming at reducing the time spent in testing the resources could also be applied.

4.4 Fault Tolerance in MIPS R3000, Configuration Generator and Configuration Memory

The fault tolerance technique proposed in this chapter aims to tolerate faults in the reconfigurable architecture. For this reason, its main advantage is to explore the abundant amount of resources available.

For the other components presented in the reconfigurable system, such as processor, configuration generator and configuration memory (Figure 8), we propose the use of traditional techniques as a mean to cope with faults.

In case of the processor and the configuration generator, two solutions can be applied. The first solution is to fabricate both components in larger technologies and avoid the fault rate increase expected to future scaled technology. In this case, the reconfigurable system chip would have components in two different scales. One large scale to the processor and configuration generator, e.g. 180nm technology, and another reduced scale to the reconfigurable fabric, e.g. 32nm technology. Figure 21 illustrates the size ratio when the reconfigurable architecture in 180 nm technology is scaled to 32 nm. These dimensions were calculated based on the transistor density from (ITRS, 2011). In the figure, the MIPS R3000 is only one dot in 32nm technology.

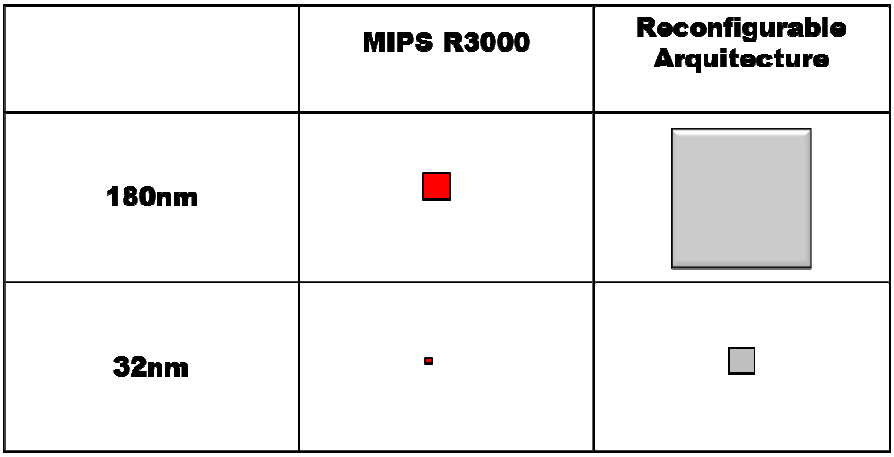


Figure 21. Area analysis considering different technologies

The other solution is to include some traditional redundancy technique, such as triple modular redundancy. Since the processor and the configuration generator present such a small area when scaled to smaller technologies, this could be an advantage when applying traditional hardware redundancy. At the same time, power consumption would still be a problem in this solution (PRADHAN, 1996).

Fault tolerance in memory devices has been studied for at least 35 years (SCHUSTER, 1978), (WEY and LOMBARDI, 1987). For hard errors caused by manufacturing defects, the solutions are mainly based on adding spare rows and columns to replace the defective row or column. Because memories are extremely redundant devices, applying the spare technique is a simple and efficient solution. Moreover, memories also make large use of error-correcting codes (ECC), which consists in the addition of redundant information to the data word. Before manipulating

data, the sequence is checked with the original generated sequence, and if it matches, no error was detected. Depending on the algorithm, it is possible not only to detect but also correct data (PETERSON and WELDON, 1980).

The selection or implementation of fault tolerance techniques in memory devices is out of the scope of this work. Nevertheless, to protect the configuration memory, we assume that some already consolidated techniques can be implemented, such as redundancy for manufacture defects and ECC to correct data errors during memory usage.

4.5 Experimental Results

4.5.1 Methodology

4.5.1.1 Area

All analyses considering the area of the architecture were based on the reference architecture area presented in Table 1.

The reference architecture has 17 levels of functional units with 9 ALUs in parallel in each row, 2 multipliers per level and 4 load/stores per level. Since there are two input multiplexers in each FU, the total number of input multiplexers is twice the number of FUs. Moreover, since there is one output multiplexer per context line in each row, the amount of output multiplexers is the number of context registers multiplied by the number of rows. The reference architecture has 16 context registers and 51 rows (3 rows per level), so there are 816 output multiplexers.

Table 1. Number of resources of LOwER-FaT array

Unit	Amount
ALU	459
Multiplier	34
Load/Store	68
Input Multiplexer 16:1	1122
Output Multiplexer 16:1	816

The area in number of gates was estimated based on the area of the functional units and multiplexers obtained by synthesizing the VHDL description of these elements in Mentor Graphics tool, *LeonardoSpectrum* (MENTOR GRAPHICS, 2012). Furthermore, for area estimation in different technologies, we used the transistor density predicted for the technologies from (ITRS, 2011). Table 2 presents the total amount of gates per resource and the area overhead of each resource in the overall total area.

Table 2.Total area

	Total #gates	Ratio (%)
Input Mux2:1 32bits	3,818,880	54.60
Output Mux2:1 32bits	2,350,080	33.60
ALU	555,849	7.95
Load/Store	38,760	0.55
Multiplier	228,820	3.27
Register	2,048	0.03
Total	**6,994,437**	

4.5.1.2 Failure rate

The failure rate (λ) of each resource was calculated based on the following assumption: considering different circuits fabricated in the same technology and exposed to the same conditions (temperature, hours under stress, etc), the only variable in the failure rate is the area of each component.

Since we have the total area of the 32-bits MIPS R3000 processor, we used this information to have an approximate failure rate of each element based on the number of gates. In this estimation, we have calculated the failure rate based on the transistor density from ITRS (ITRS, 2011) and the relative failure rate per transistor from (SMITH, 2007). Table 3 presents the failure rates for each component.

Table 3. Failure rates (λ) per 10^6

	90nm	**32nm**	**18nm**	**11nm**
Multiplexer 2:1	0.0000050956	0.0000093912	0.0000168620	0.0000261427
ALU	0.0041138863	0.0075818390	0.0136132238	0.0211058396
Load/Store	0.0007745385	0.0014274643	0.0025630182	0.0039736841
Multiplier	0.0228624731	0.0421352409	0.0756540017	0.1172933942
Register	0.0005435358	0.0010017294	0.0017986093	0.0027885502

4.5.1.3 Reliability

To evaluate the reliability, we have implemented the reliability models and calculated the reliability as a function of time using MATLAB version 7.12.0 (MATHWORKS, 2012). All reliability results are concentrated in appendix A.

4.5.1.4 Performance and Energy Results

The performance of the reconfigurable fabric was evaluated over three different architectures. A tool, called ARISE (*Automatic Resources Investigation System based on application Execution*), was used to generate the exact number of functional units required in each level of the array (RUTZIG, BECK and CARRO, 2008). The ARISE tool shapes a new reconfigurable fabric based on the data-dependence graph (DDG) of applications to be performed. This new shape is optimized in resources to these applications.

The first shape was generated only according to the amount of parallelism available in the applications. Thus, it has the ideal number of functional units, without any area constraints. However, since this shape was generated to achieve the highest possible performance, other two more realistic shapes were generated based on the available parallelism and area constraints. ARISE generated these shapes according to an allowed performance loss defined at design time. Therefore, shapes with lower area cost were defined to have 40% and 80% less performance than the ideal shape.

Table 4 describes the amount of functional units available in each shape. LAC indicates Low Area Cost, and LAC I and LAC II are the shapes generated with 40% and 80% of performance loss. Furthermore, LAC I corresponds to the LOwER-FaT array analyzed in the reliability, performance and energy result.

Table 4. Number of resources in three array shapes

	Ideal shape	LAC I	LAC II
ALU	786	459	68
Load/Store	248	68	49
Multiplier	31	34	1
Total	**1065**	**561**	**118**

In LAC II, since there is only one multiplier, in case of this unit presents fault, all the multiplications will be performed by the processor.

To obtain the energy results, VHDL description was synthesized in Synopsys Design and Power Compiler tools (SYNOPSYS, 2012), using a CMOS 90nm technology. Moreover, the power consumption of the reconfiguration memory and the address cache were obtained with the CACTI 6.5 tool (WILTON and JOUPPI, 1996). The results of power were used to calibrate the simulator to obtain the overall energy consumption.

4.5.1.5 Fault Injection Simulation

To evaluate the performance degradation when the number of available resources is reduced due to the faults (LOwER-FaT array strategy), we performed some fault injection simulations in the reconfigurable architectures described in Table 4.

To introduce faults in the resources during performance simulations, a tool was implemented to randomly select the faulty units in the architecture. The tool receives as input the amount of resources (functional units and multiplexers) and the fault rate, and randomly selects some of the resources as faulty.

It is important to highlight that the random selection of faulty resources was based on the area of the functional units. To consider the different areas per group we associated weights to each group based on the components ratio presented in Table 2. This approach correlates the probability of a fault occurrence according to the amount of area used by each group (circuits with larger area have higher probability of presenting faults).

This work evaluated the performance degradation based on five different fault rates (0.01%; 0.1%; 1%; 10%; and 20% of faults), and the reference design was the fault-free ideal shape. The highest fault rates (1%-20%) were chosen based on the deep-submicron device's fault rate (BORKAR, 2004) and the lowest fault rates (0.01%-0.1%) were chosen to cover technologies with larger feature sizes.

In order to have an appropriate evaluation on how performance was affected with the reduction of available resources, several simulations using the same shape and fault rates were performed. The only difference among the simulations was the location of the faults, since those were randomly selected. Thus, the performance results of each set of application, shape and fault rate are an average of the performance results of all the simulations that used the same set.

4.5.1.6 Benchmarks

To evaluate the proposed approach several simulations were performed to obtain the performance degradation and energy results. The applications used in the simulations were selected from the MiBench Benchmark Suite (GUTHAUS, RINGENBERG, ERNST, *et al.*, 2001). This benchmark was chosen because it is

composed by control flow and data flow applications, representing a heterogeneous behavior, which is commonly found nowadays in embedded systems like smartphones (RUTZIG, BECK and CARRO, 2011).

To demonstrate the heterogeneous behavior of the applications, Figure 22 presents the selected applications arranged according to the average number of instructions executed between two branches, a metric that correlates to the amount of instruction level parallelism available without speculation.

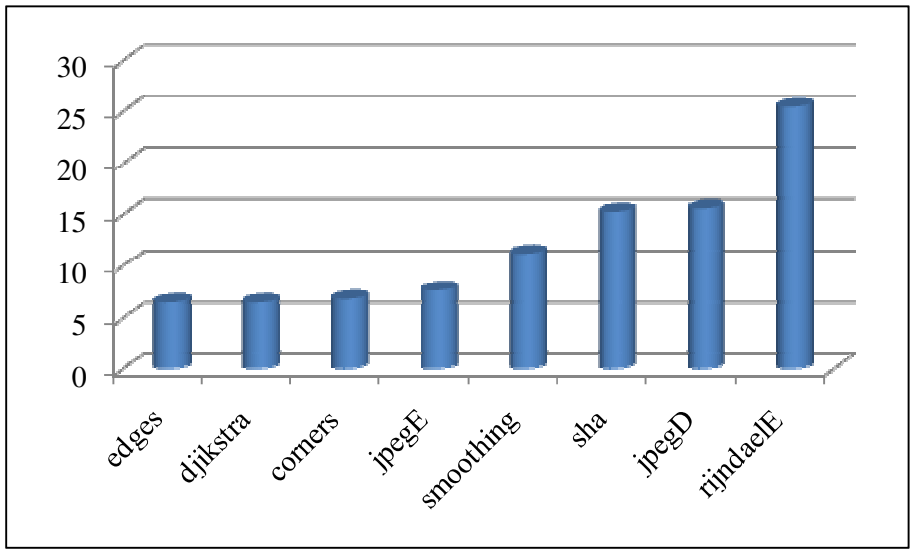


Figure 22. Average number of instructions executed per branch

4.5.2 Reliability Analysis

This section presents the reliability analysis of the reconfigurable architecture with and without the fault tolerance approach. The charts depict the reliability as a function of time for each case analyzed. For sake of clarity, the time scale (x-axis) differs in each chart. Moreover, the reliability results discussed in this work are presented in Appendix A.

4.5.2.1 Reference reconfigurable architecture

The chart in Figure 23 presents the reliability of the reconfigurable array used as reference, in 90nm technology. The first curve (*All Levels*), presents the system reliability when all the units are operating. In this case, a fault in any unit (functional or interconnect) prevents the system from working properly. As expected, a system where all the resources are required, and without any approach to tolerate the faulty elements would present a catastrophic reliability.

Although this is not an uncommon scenario for electronic circuits in general, in reconfigurable architectures, due to the large amount of redundancy, in many cases only part of the architecture is actually operating. According to our simulations on the underlying reconfigurable architecture, depending on the application, it is possible that less than half of the units are operating. When this happens, there is a probability that the faulty unit is in idle state and does not interfere in the correct functioning of the system. This situation is represented with the dotted curve (*60% Levels*). To provide this scenario, we used the performance simulation results described in section 4.6.3, which

indicates that, in most applications execution (using our benchmarks) on average, 60% of the architecture is in use and the other 40% is in idle state.

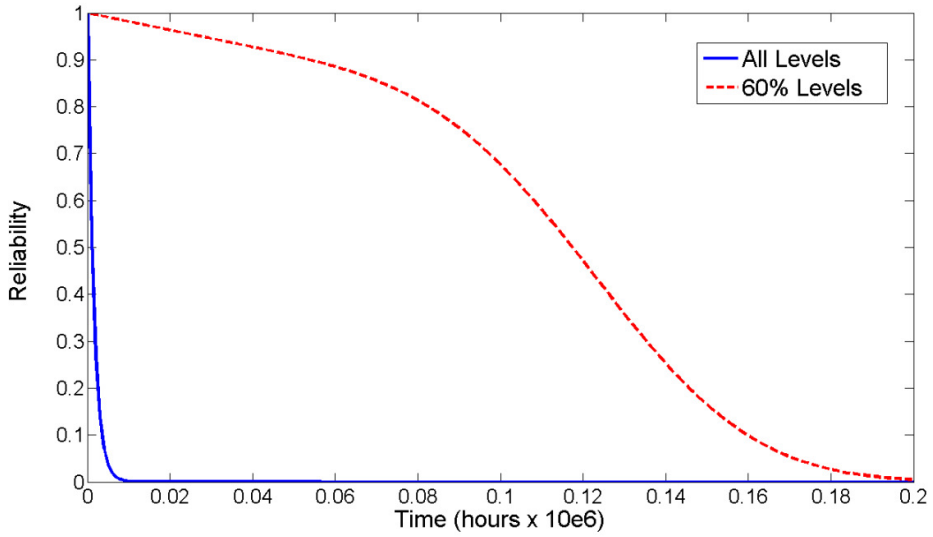


Figure 23. Reconfigurable array reliability - without fault tolerance

Because there is no fault tolerance mechanism to work around the faulty resources and prevent system from failure, the reliability is significantly low. Even when the system presents parallelism, if a resource that must be working is faulty, the entire system operation is compromised. This is the reason why the reliability curves depicted in Figure 23 present a severe decrease.

When all resources are in use, the reconfigurable architecture reliability is 0.99999 in 0.008 hour, i.e. after 29 seconds of operation, the probability that the system works correctly is less than 100%. For the architecture with 60% of resources in use, this reliability (0.99999) is observed after 2.7 hours.

Although there is an improvement in the overall reliability of approximately 334 times only considering the fact that some units are not used, this is yet a very low reliability. It indicates that after 2.7 hours of useful life, the system is not 100% reliable. Moreover, according to Appendix A, after 21,000 hours (around 2.4 years), the reliability is 0.96. In an effort to improve these results, next section presents the reliability analysis of the fault tolerance strategy described in section 4.2.

4.5.2.2 LOwER-FaT array

The main difference between this model and the reference architecture model are:

1. The reliability function of the functional units is a parallel system;

2. Only part of the multiplexers needs to work so the architecture works properly.

In Figure 24, we present the system reliability as a function of time for the architecture with all resources in use and when only 60% of resources are required.

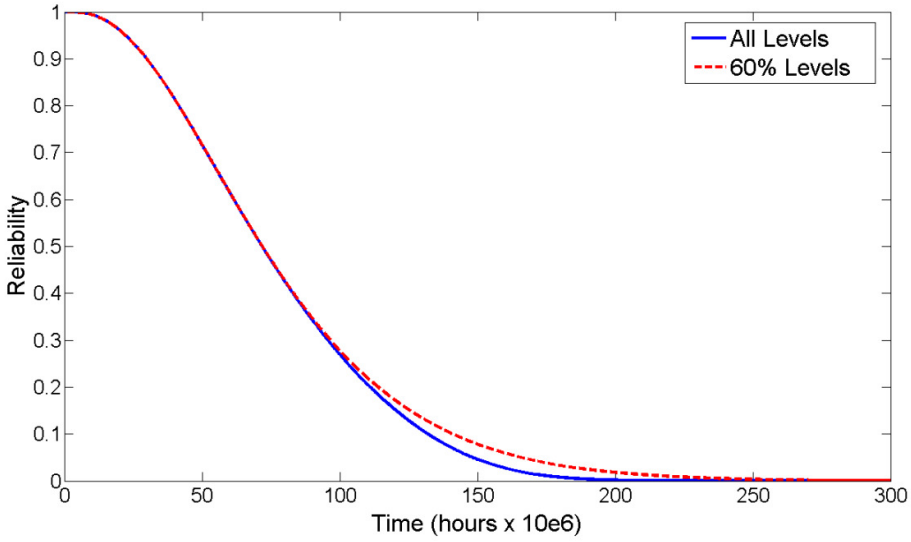


Figure 24. LOwER-FaT array reliability

Comparing the reliability presented in Figure 24 and the one presented in Figure 23, a significant improvement can be observed. In the reconfigurable architecture without fault tolerance, the reliability decreases to 0.99999 in 0.008 hour. On the other hand, the LOwER-FaT array reliability decreases to 0.99999 in 860,000 hours (around 98 years). At the same time, when the fault tolerance strategy is in place, the fact that only part of the architecture is in use does not influences the reliability as it does when the architecture have no fault tolerance. This happens because in this model, the functional units are already connected in parallel, which means that if one unit fails, it does not affect the rest of the architecture.

Based on the analysis presented above, a question arises: if the functional units already present the best connection possible in terms of reliability, which part of the architecture should we invest in fault tolerance to increase reliability even more?

To answer this question, we need to identify the components that most influence the overall system reliability. These are the critical components to reliability, since they are the dominant terms in the reliability function. If these components present low reliability, the overall reliability will be low. On the other hand, any improvement in the reliability of these components will affect positively the overall reliability.

Before continuing the reliability analysis, we have to take a step back from the reliability analysis curves and open a short parenthesis to present a mathematical analysis that helps to identify the critical elements.

Mathematical Analysis

We first start this analysis considering the two corner cases in our reliability model. The series connection, where all components must work so the system works properly, and the parallel connection, where only one component needs to work. The reliability function of both cases, already described in equations (4) and (5), are repeated next.

$$R_{series}(t) = \prod_{i=1}^{n}\left(R_i(t)\right), \quad (4)$$

$$R_{parallel}(t) = 1 - \prod_{i=1}^{n}(1 - R_i(t)). \quad (5)$$

Assuming that R(t) is defined by the exponential failure law, where $R(t) = e^{-\lambda t}$, and considering that there are different components with different failure rates (components A, B, C, … Z have failure rates λ_A, λ_B, λ_C, …, λ_z). Consider that we have two subsystems, one in series and one in parallel, each one with two components. Solving the equations, we have the following results:

Series system	Parallel system
$R_{series}(t) = \prod_{i=1}^{2}\left(e^{-\lambda_A t}\right)$	$R_{parallel}(t) = 1 - \prod_{i=1}^{2}\left(1 - e_i^{-\lambda_B t}\right)$
$R_{series}(t) = e^{-\lambda_A t} x\, e^{-\lambda_A t}$	$R_{parallel}(t) = 1 - \left(\left(1 - e^{-\lambda_B t}\right) x \left(1 - e^{-\lambda_B t}\right)\right)$
$R_{series}(t) = e^{-2\lambda_A t}$	$R_{parallel}(t) = 1 - \left(1 - e^{-\lambda_B t} - e^{-\lambda_B t} + e^{-2\lambda_B t}\right)$
	$R_{parallel}(t) = 2e^{-\lambda_B t} - e^{-2\lambda_B t}$

Calculating the ratio between each subsystem reliability and the reliability of one component, we can identify how much the reliability is affected when the individual components are connected.

Series system	Parallel system
$\frac{R_{series}(t)}{R(t)} = \frac{e^{-2\lambda_A t}}{e^{-\lambda_A t}}$	$\frac{R_{parallel}(t)}{R(t)} = \frac{2e^{-\lambda_B t} - e^{-2\lambda_B t}}{e^{-\lambda_B t}}$
$\frac{R_{series}(t)}{R(t)} = \frac{1}{e^{-\lambda_A t}} x \frac{1}{e^{2\lambda_A t}}$	$\frac{R_{parallel}(t)}{R(t)} = \frac{2e^{-\lambda_B t}}{e^{-\lambda_B t}} - \frac{e^{-2\lambda_B t}}{e^{-\lambda_B t}}$
$\frac{R_{series}(t)}{R(t)} = \frac{1}{e^{\lambda_A t}}$	$\frac{R_{parallel}(t)}{R(t)} = 2 - e^{-\lambda_B t}$
$\frac{R_{series}(t)}{R(t)} = e^{-\lambda_A t}$	

From the results presented above, one can observe that when the components are connected in series, the reliability decreases exponentially, when compared to the reliability of the individual components. On the other hand, when the individual components are connected in parallel, they present an increase in reliability. As the time passes, the second term of the function, ($e^{-\lambda_B t}$), approaches to zero and the reliability increase approaches to 2. Therefore, for a general case, we can conclude that the series connection decreases reliability in a much higher rate than the parallel connection increases it.

This conclusion helps to understand why changing the connections of the functional units from series to parallel did not result in a significant reliability increase.

In fact, there are other components that are contributing to reduce reliability in a much more significant amount than the functional units increase.

According to equation (36), the output multiplexers are connected in series to all the other subsystems, which may cause a considerable impact on reliability. Moreover, in spite of their reliability function be a k-out-of-m (equation 33), if *k* is too close to *m*, the reliability function approaches to a series function.

To evaluate how much the output multiplexers are affecting overall reliability, next we present some reliability analysis when redundancy to the output multiplexers is introduced.

Output Multiplexers Analysis

The strategy consisted in adding extra output multiplexers to be used as spare in case of faults affect the working multiplexers. In the LOwER-FaT array, this is done by adding more context registers, with their respective context lines, to the architecture. In terms of reliability model, this strategy does not make any change in the output multiplexers subsystem, which continues to be a k-out-of-m subsystem. The only change is in the number of required units (*k*).

Figure 25 presents the system reliability when this strategy is applied for different numbers of *k*. Table 5 presents the amount of redundancy added and the impact on overall area. We also evaluated the impact on performance when this strategy is implemented. These results are presented in section 4.5.3.

Table 5. Spare output multiplexer strategy - area overhead

Spare output multiplexers		Area overhead (%)
%	Amount	
30	21	25.2
50	24	33.6
80	29	47.6
100	32	56.0

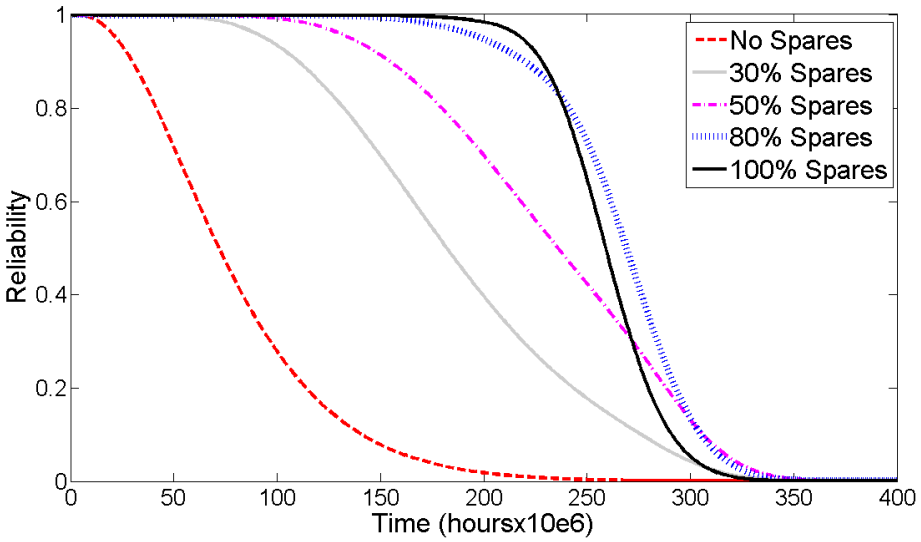


Figure 25. Spare output multiplexer strategy

In Figure 25, the curve with lowest reliability is the LOwER-FaT array reliability without redundancy strategy, already shown in Figure 24. The other curves are the reliabilities of the LOwER-FaT array with different amounts of spare multiplexers.

A comparison among all strategies shows significant improvements in overall reliability when the spare multiplexers are added to the architecture. According to Figure 25, by adding 50% more multiplexers, the reliability increases 48 times. Moreover, by adding 100% more multiplexers, the reliability increases 117 times. However, these reliability improvements come with 33.6% and 56% of overall area increase. At the same time, as can be visualized in Figure 25, the reliability gains are saturating as the amount of spares is increasing. In fact, if we neglect the area costs and extrapolate the number of spare multiplexers, we will see that the reliability starts to decrease, as shown in Figure 26.

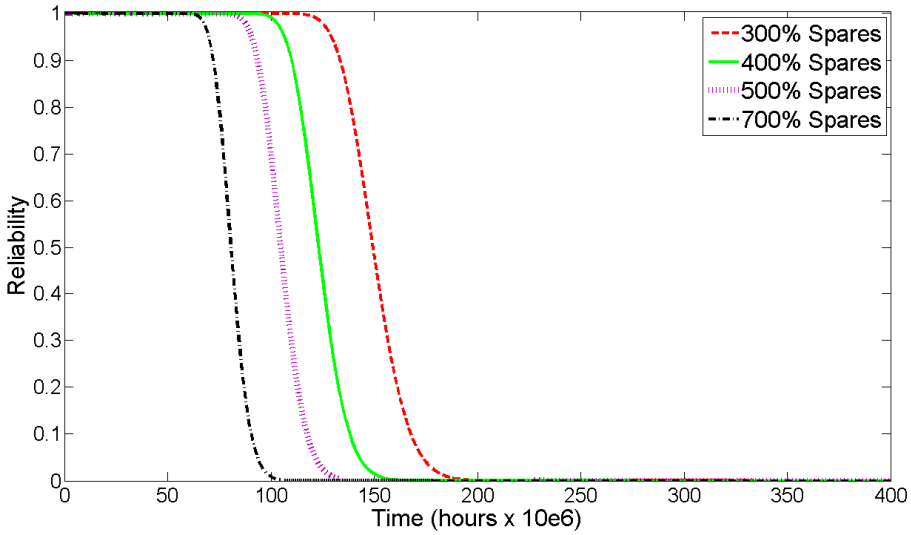


Figure 26. Spare output multiplexer strategy – reliability saturation

Based on the curves presented in Figure 26, the reliability increase is detected only up to 300% spares. After that, the reliability starts to decrease drastically. At a first moment, this result is contra-intuitive. However, after analyzing the reliability function, it becomes clear why this happen.

Firstly, the increase in the amount of spare output multiplexers does not change the reliability model, however, it changes the architecture design by increasing the number of inputs that all the input multiplexers will receive. As shown in Figure 10, in each row, the multiplexer's output is sent to the two input multiplexers of each functional unit in the next row. This means that, if the number of output multiplexers in each row increases, the size of the input multiplexers also increases. Since the reliability model is also a function of the input multiplexers (equations 15, 16 and 17), when the number of output multiplexers increases, it changes not only the reliability of the output multiplexers subsystem itself, but indirectly, it changes the reliability of the input multiplexers.

$$R_{ALU+inputMux}(t) = R_{ALU}(t)\ x \prod_{i=1}^{2} \left(R_{inputMux[i]}(t)\right) \quad (15)$$

$$R_{Mult+inputMux}(t) = R_{Multiplier}(t) x \prod_{i=1}^{2} \left(R_{inputMux[i]}(t)\right) \quad (16)$$

$$R_{LoadStore+inputMux}(t) = R_{LoadStore}(t)\ x \prod_{i=1}^{2} \left(R_{inputMux[i]}(t)\right) \quad (17)$$

At some point, this reliability becomes so low, that it starts to affect the overall system reliability. To illustrate that, Figure 27 shows the reliability of the output multiplexers in one row and the reliability of the two input multiplexers connected in series, considering 300% output multiplexers as spare. The same reliabilities are shown when the amount of spare output multiplexers increases to 700%. It can be observed that the reliability of the output multiplexers increases when the number of spares increases. However, the reliability of the input multiplexers decreases drastically due to their area increase.

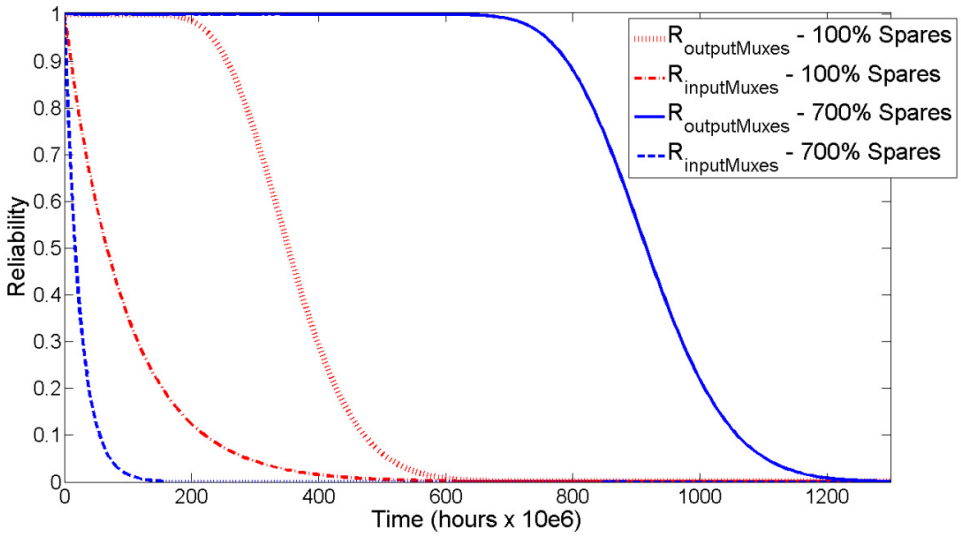


Figure 27. Spare output multiplexer strategy – justification

The area overhead due to the increase in the amount of inputs of the input multiplexers is shown in Table 6.

Table 6. Spare output multiplexer strategy – input multiplexer's area overhead

Spare output multiplexers		Area overhead (%)
%	Amount	
30	21	18.2
50	24	29.1
80	29	47.3
100	32	58.2

After finding the critical elements by investigating the reliability model and proposing a solution to improve reliability, we fall into the same type of problem but now in a different part of the architecture design. At this point, the input multiplexers are now the critical elements to the architecture.

In an attempt to increase reliability, in the next analysis we concentrated the efforts in the series connection between the input multiplexers and the functional unit (equations 15, 16 and 17).

Input Multiplexers Analysis

Again, the strategy here is to add spare multiplexers to the architecture. However, differently from the output multiplexer strategy, in this approach to replicate one multiplexer, it is necessary to add another one to select between the output of the original multiplexer and the replicated ones. Figure 28 illustrates this approach.

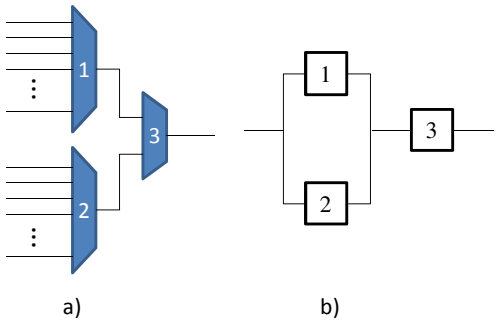


Figure 28. Replicating a multiplexer

In Figure 28.a, it is shown two multiplexers, the multiplexer with the number one is the original one, and the multiplexer with the number two is the spare. To select between the outputs of the two multiplexers, a third multiplexer must be included. The original and spare multiplexers have different amounts of inputs than the third one. While the multiplexers one and two have as input all the context lines, the third one has as input only the output of these two multiplexers. It is important to emphasize that the third multiplexer will have as many inputs as the total amount of spares plus the original one.

To evaluate the reliability improvement when input multiplexers are included in the architecture, we had to modify the reliability function of the subsystems composed of functional unit and input multiplexers. Thus, equations (15), (16) and (17) are replaced by equations (41), (42) and (43), respectively. In all three equations, the series connection between the two input multiplexers becomes a k-out-of-m connection. In addition, there is a third multiplexer connected in series to the first two, as represented in Figure 28.a.

$$R_{ALU+inputMux}(t) = R_{ALU}(t)\, x$$

$$x \sum_{i=k\_inputMux}^{m\_inputMux} \binom{m\_inputMux}{i} \left(R_{inputMux}(t)\right)^{i} \left(1 - R_{inputMux}(t)\right)^{m\_inputMux-i} x$$

$$x\, R_{thirdMux}(t), \tag{41}$$

$$R_{Mult+inputMux}(t) = R_{Mult}(t)\ x$$

$$x \sum_{i=k\_inputMux}^{m\_inputMux} \binom{m\_inputMux}{i} \left(R_{inputMux}(t)\right)^{i} \left(1 - R_{inputMux}(t)\right)^{m\_inputMux-i} x$$

$$x\ R_{thirdMux}(t), \tag{42}$$

$$R_{LoadStore+inputMux}(t) = R_{LoadStore}(t)x$$

$$x \sum_{i=k_{inputMux}}^{m_{inputMux}} \binom{m_{inputMux}}{i} \left(R_{inputMux}(t)\right)^{i} \left(1 - R_{inputMux}(t)\right)^{m_{inputMux}-i} x$$

$$x\ R_{thirdMux}(t). \tag{43}$$

Figure 29 presents the reliability when the input multiplexers are replicated. For these analyses, the output multiplexers are also replicated in the 300% spares strategy. The first curve is the system reliability without spare input multiplexers. In the other results, each input multiplexer is replicated 1, 3, 7, 15 and 31 times (2, 4, 8, 16 and 32 input multiplexers, respectively), plus the third multiplexer to select one of the outputs. Once again, we extrapolated the area to investigate the potential reliability that the architecture can achieve. As one can observe in Figure 29, the reliability starts to decrease when the number of input multiplexer spares becomes too numerous. This indicates that there is a limit in the amount of spares that can be included due to the third multiplexer area.

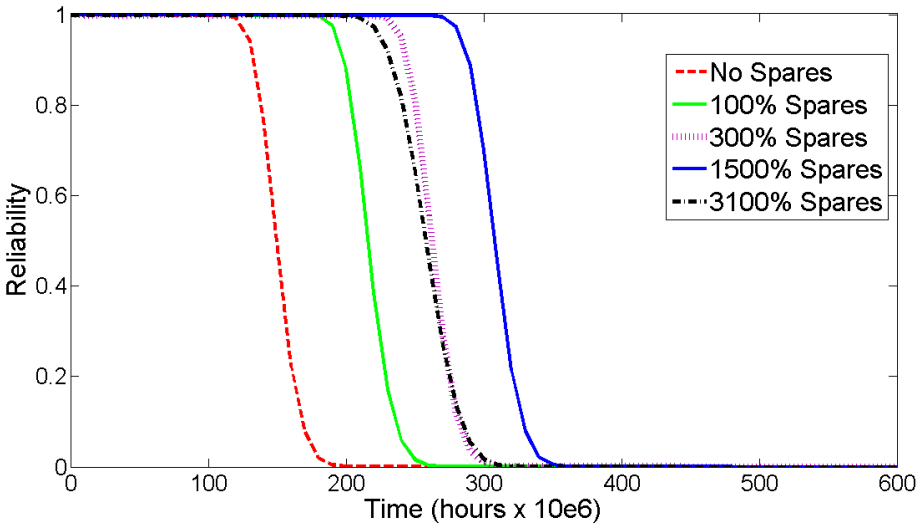


Figure 29. Spare input multiplexer strategy

When comparing the reliability of the LOwER-FaT array in Figure 24 and the reliability when the output and input multiplexer redundancy strategies are applied, two observations can be made:

1) The increase in reliability is significantly higher when the redundancy strategy is applied in the output multiplexers. One can improve the reliability by approximately nine orders of magnitude when comparing the time when reliability achieves 0.99999 in the reference architecture and the spare interconnection model architecture with 50% spares. Additionally, an improvement of 48 times is achieved when comparing the 50% spare output multiplexer strategy and the LOwER-FaT array without spares. This strategy also introduced an overall area overhead of 34% due to the inclusion of output multiplexers and 29% due to the increase of the input multiplexer area.

2) Even neglecting the area cost and considering unlimited area, the reliability increase saturates and at some point, the redundant resources compromise the system reliability. In case of the LOwER-FaT array, the ideal amount of redundancy was found when combining the addition of 300% output multiplexer spares and 1500% input multiplexer spares. The results showed an increase of 10 orders of magnitude when comparing the ideal redundant architecture to the architecture without fault tolerance. Moreover, an improvement of 2 orders of magnitude was detected when comparing with the LOwER-FaT array without any extra redundancy. Once again, this ideal redundancy does not consider area cost, which would be extremely high in this case.

As demonstrated by the reliability model, this is a consequence of the high dependence on the interconnect elements to send the correct data throughout the architecture. On the other hand, the functional units have no significant influence to increase reliability because they are already in parallel. Thus, if there is only one functional unit, it is still possible to execute all the operations in spite of the performance degradation.

It is important to mention that solutions to increase the amount of functional units are not considered in this analysis, since for each functional unit added, two input multiplexers must be included. Moreover, adding functional units would also increase the number of output multiplexers and/or the output multiplexer area that selects among the functional units output.

Technology Analysis

To conclude this analysis, we have also estimated the architecture reliability considering smaller feature size devices. Figure 30 illustrates the system reliability with 300% of redundant output multiplexers and 1500% of redundant input multiplexers in four technologies, 90nm, 32nm, 18nm and 11nm. As can be observed, the reliability decreases as the technology shrinks. In this estimation, we have calculated the failure rate based on the transistor density from ITRS (ITRS, 2011) and the relative failure rate per transistor from (SMITH, 2007).

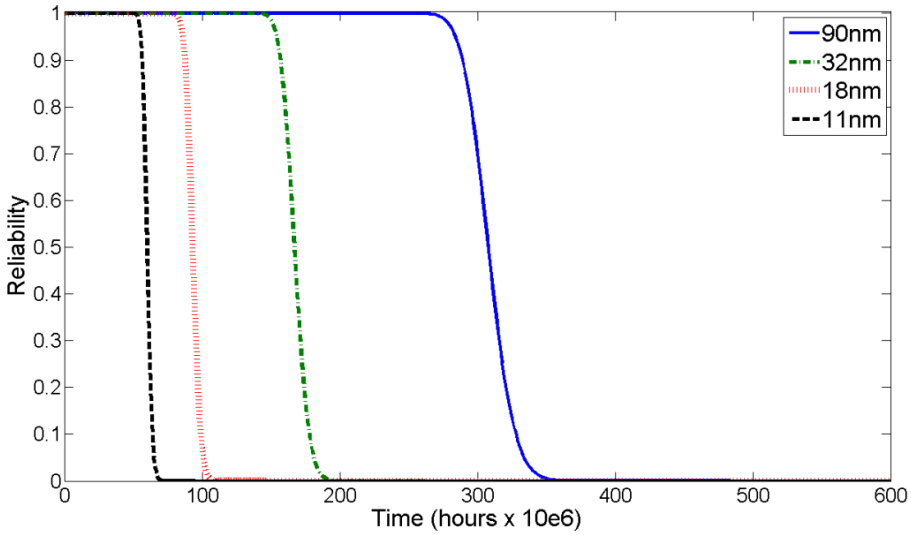


Figure 30. LOwER-FaT array reliability with spare multiplexers strategy – different technologies

From the analyses presented above, it was possible to demonstrate that the interconnection model is the critical element to system reliability. In the case study presented, the interconnection model consists of buses and multiplexers. Next section we present the reliability analysis of the reconfigurable architecture when a multistage interconnection model is used.

4.5.2.3 Multistage Interconnection Network as Interconnection Model

A multistage interconnection network (MIN) consists of a set of switch columns or stages, where each stage is connected to the previous and to the next one. Initially used to connect processor and memory modules (WU and FENG, 1980), as an efficient model to balance the tradeoff between cost and performance, MINs have become even more popular due to the high degree of fault tolerance. For this reason, MINs were (and still are) widely used as a reliable interconnection model.

Figure 31 illustrates a multistage interconnection network composed of switches and links connecting the inputs to the outputs. The links between the stages can have different routing according to the type of MIN. The one presented in Figure 31 consists in an Omega MIN with $\log_2 N$ stages, where N is the number of inputs.

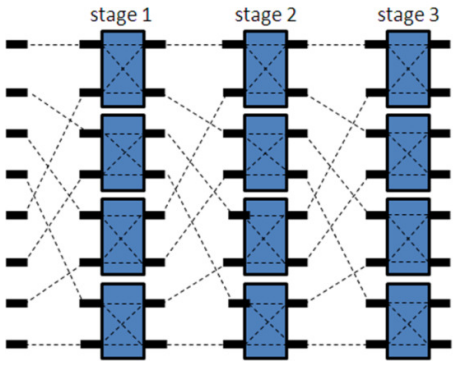


Figure 31. Omega multistage interconnection network

Depending on the routing strategy, a MIN can be non-blocking, rearrangeable or blocking. The non-blocking MINs can connect any input to any output, regardless the connections already established across the network. An example of this type of MIN is the Clos network (CLOS, 1953). On the other hand, if the MIN can perform any input/output connection but to do so, it is necessary to reprogram the internal switch, it is called rearrangeable MINs. The Benes network is an example of rearrangeable MIN (BENES, 1965). A MIN is blocking when at least one input/output path cannot be performed. Blocking MINs can be unique-path blocking, where there is only one path to connect the input to the output. They can also be multiple-path blocking, with more than one path to connect the input to the output. An example of the unique-path blocking MIN is the Omega network illustrated in Figure 31 (LAWRIE, 1975). On the other hand, an Omega MIN plus extra stages is an example of a multiple-path blocking MIN. Figure 32 depicts an example of an extra stages Omega network. The Omega network is a sub-set of the banyan networks (GOKE, 1973).

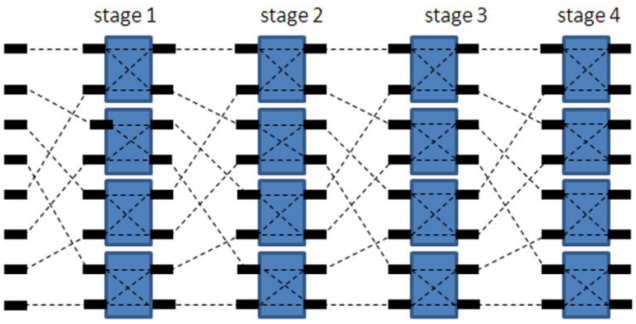


Figure 32. Omega MIN with extra stages

Due to the widespread use of MINs as fault-tolerant cost-effective interconnection solutions, many works start to use MINs to connect components inside processors, such as functional units and memory elements (registers, latches). This is the context where we use MINs in this work. To connect functional units to functional units and to registers, we decided to modify the interconnection model from the traditional bus and multiplexer to multistage interconnection networks.

More specifically, we used the multiple-path blocking network Omega with extra stages illustrated in Figure 32. We used a blocking MIN because, differently from previous solutions, where most works try to ensure that any input connects to any output, even in the presence of faults, we use redundancy to send the input to different outputs using alternate paths without affecting the correct operation of the system. This is done by connecting the inputs to identical functional units and performing a broadcast of the input. When the input reaches one of the outputs, i.e. when one of the functional units receives the inputs, the functional unit is dynamically reconfigured to perform the operation assigned to the inputs. With this strategy, we eliminated the need to use a more complex and expensive non-blocking network.

This work was firstly proposed by (FERREIRA, BUENO, LAURE, *et al.*, 2011). In an attempt to increase performance by increasing the number of paths, the authors proposed the use of parallel networks. Because the architecture was arranged as a set of

functional units in a 1-D array, resembling a traditional VLIW (Very Long Instruction Word) architecture, the authors called it *SuperVLIW*.

In (FERREIRA, VENDRAMINI, MUCIDA, *et al.*, 2011), the authors proposed the implementation of this coarse-grained architecture on top of an off-the-shelf FPGA. Since there is a lack of coarse-grained reconfigurable devices, as well as tools and compilers, in this work, the authors proposed as alternative implementing a coarse-grained architecture as a virtual device on top of the FPGA. This approach eliminates the need to handle directly with complex fine-grained FPGAs. To reduce reconfiguration time, the authors also propose a new algorithm that performs scheduling, placement and routing (SPR) in one-step.

Although it is a fact that multistage interconnection networks present a high fault tolerance, as shown in many studies in the past 30 years (ADAMS, AGRAWAL and SIEGEL, 1987), a system reliability is a combination of its logical elements and its interconnects. For this reason, we have evaluated the reliability of the architecture with MIN as interconnection model and compared with the original architecture based on buses and multiplexers (Figure 10).

Omega network reliability model

The reliability function of the network starts with the switch subsystem. As illustrated in Figure 32, each switch has two inputs and two outputs. To allow an input to be sent to any of the two outputs, it is necessary a crossbar-like connection, using two multiplexers as illustrated in Figure 33. Because the architecture is 32-bits, each multiplexer is also 32-bits.

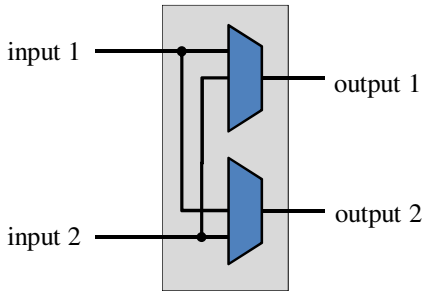


Figure 33. 32-bits switch

As well as the 32-bits multiplexers of the LOwER-FaT array, the multiplexers inside the switch are also composed of a chain of 2:1 1-bit multiplexers. Therefore, the first subsystem is the 2:1 32-bits multiplexer, with reliability function described in equations (11) and (12).

Since there is no fault tolerance strategy implemented in the switches, to allow the switch to work properly, both multiplexers need to work. Therefore, the switch subsystem is a series system with function given by equation (44),

$$R_{switch}(t) = \prod_{i=1}^{2} \left(R_{32bMux}(t)\right). \tag{44}$$

As mentioned earlier, the Omega network has $\log_2 N$ stages with N being the number of inputs. Considering the original Omega network without extra stages, there is only one path between an input/output pair. Hence, the switch used in the first stage to

send data must work so the data can pass to the next stage. This same requirement is demanded to the other stages. Therefore, the switch used in each stage must work, and if a switch fails, the data cannot be sent from the input to the output.

Figure 34 depicts an Omega network with 8 inputs and $\log_2 8 = 3$ stages. To send data from input A to output B, the only path available requires that the grey switches work properly. This characterizes a series system with the three switches connected, as illustrated in Figure 35. The reliability function for the Omega network is described in equation (45), where *s* is the number of switches.

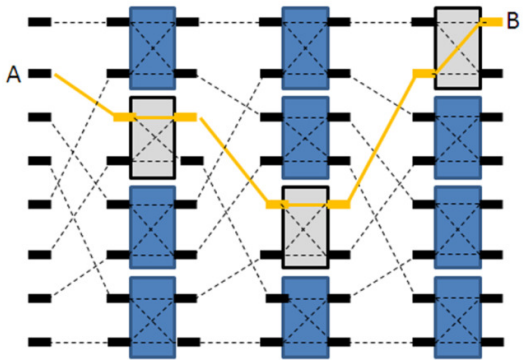


Figure 34. An example of Omega network with one path

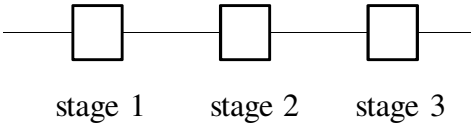


Figure 35. Series system representing one path of the Omega network

$$R_{MIN}(t) = \prod_{i=1}^{s} \left(R_{switch[i]}(t)\right). \quad (45)$$

To increase fault tolerance in MINs, the typically used strategy is increase the number of paths between an input/output pair by adding extra stages to the network. For each extra stage added to the network, we have 2^s paths, where *s* is the number of extra stages. Thus, adding 1 extra stage, 2 paths between an input/output pair are created. With 2 extra stages, 4 paths are created, with 4 stages 8 paths are created, and so on.

The amount of extra stages not only affects reliability, but also changes the reliability function by adding parallel subsystems into the architecture. This is due to the fact that more switches in each stage can be used to send data from an input to an output. The only exceptions are the first and last stages. In this case, there is only one switch that can be used, it does not matter how many extra stages were added. To demonstrate this, Figure 36 depicts an Omega network with one extra stage (two paths).

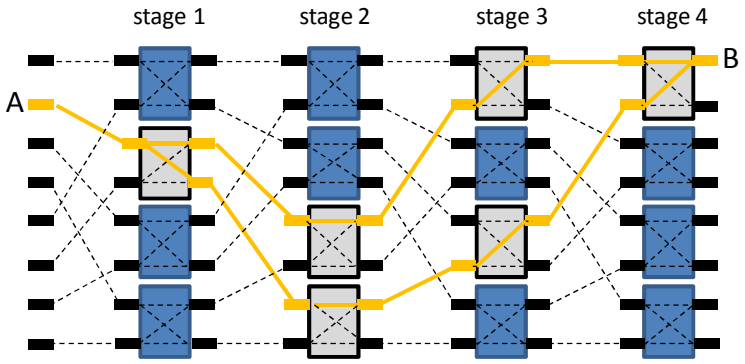


Figure 36. Omega network with one extra stage and two paths

Adding an extra stage to the Omega network implies in adding one more path between the inputs A and B. Consequently, in stages 2 and 3, two switches can be used to pass data throughout the stages. However, as mentioned before, stages 1 and 4 continue demanding that one switch works properly. Figure 37 illustrates the subsystem that connects input A to output B.

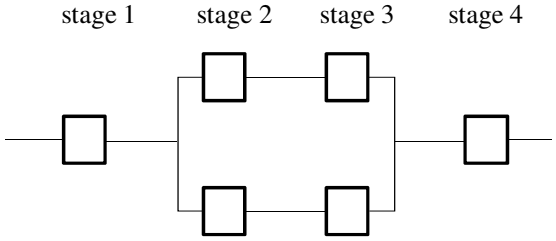


Figure 37. Omega network with one extra stage block diagram

Now, adding one more extra stage to the network, two more paths are created connecting A to B. In this case, with 5 stages, the network has four different paths to connect A to B. However, the paths share switches in the two first stages and two last stages. Figure 38 presents the block diagram representation of this subsystem.

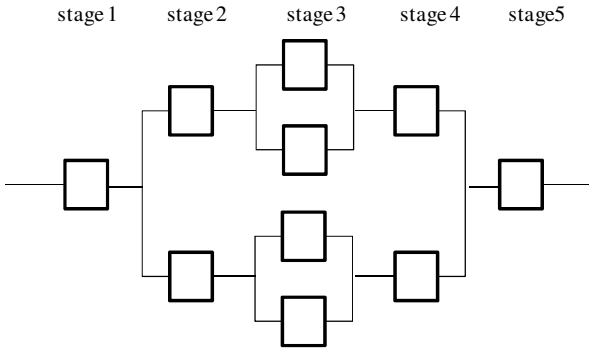


Figure 38. Omega network with two extra stages block diagram

The reliability model of the Omega network with two extra stages is described next, based on the block diagram depicted in Figure 38.

We start presenting the reliability model of one input/output pair, and then we extend the model to the other pairs. In one input/output pair, there are two possible paths, as one can see in Figure 36. Since the two paths present the same amount of switches and are connected in the same manner, we start defining only one reliability function. This function will be used in both paths until the connection to the switches in the first and last stages. Therefore, the first subsystem in hierarchy is composed by two subsystems of parallel switches in stage three. Equation (46) describes the reliability function,

$$R_{stage3}(t) = 1 - \left(\prod_{i=1}^{2} \left(1 - R_{Switch[i]}(t) \right) \right). \quad (46)$$

Following the hierarchy, the parallel switches in stage three are connected in series to one switch in stage 2 and another one in stage 4. The series connection is presented in equation (47),

$$R_{stages2and4}(t) = R_{Switch}(t) x\, R_{stage3}(t)\, x\, R_{Switch}(t). \quad (47)$$

At this point, only one path was described. The four paths compose a parallel subsystem with reliability function given by equation (48),

$$R_{FourPaths}(t) = 1 - \left(\prod_{i=1}^{2} \left(1 - R_{Stages2and4[i]}(t) \right) \right). \quad (48)$$

The first and last stages have only one switch, each one connected in series to the two paths subsystems. For this reason, the reliability model for one input of the Omega network is a series system with reliability function given by equation (49),

$$R_{OneInputOmega}(t) = R_{Switch}(t)\, x\, R_{TwoPaths}(t)\, x\, R_{Switch}(t). \quad (49)$$

To conclude the reliability model of the Omega network with extra stages, we have to extend the model for one input described above to all the other inputs. Figure 39 illustrates the Omega network with two extra stages. The figure illustrates the four possible paths between one input/output pair. As can be observed, in the third stage all the switches are used to provide the four paths. This means that, in spite of the fact that four paths per input have been created with the extra stages, there is not enough switches in each stage to provide *N**4 parallel paths, where *N=8* is the number of inputs. Therefore, the paths need to share switches along the stages.

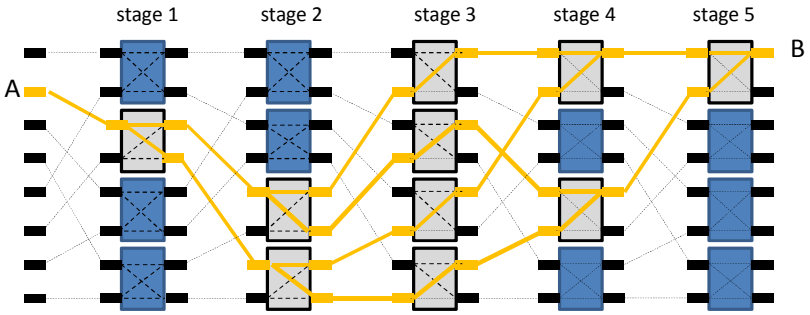


Figure 39. Omega network with two extra stages and two paths

To represent this condition in the reliability model, it is necessary to connect all the one-input subsystems, taking into consideration that, in some stages, it is possible to use different switches, e.g. stages 1, 2, 4 and 5 use only part of the switches, the other switches can be used by a different input.

All these particularities of the Omega network with extra stages were translated to the reliability model, but for sake of clarity, the rest of the equations were omitted, and from now one we assume that the reliability function of the Omega network with extra stages is defined by $R_{OmegaExtraStages}(t)$.

Comparing the interconnection models

To compare between the two models, firstly it is necessary to specify the two equivalent models. The main characteristic of the MIN is the fact that it can connect all *N* inputs to all *N* outputs. In this way, to provide the same interconnection with multiplexers, *N* multiplexers are required in a crossbar-like connection, as illustrated in Figure 40.

Moreover, for a 16-input Omega network with no extra stages, 4 stages are required ($\log_2 16 = 4$), 8 switches per stage and only one possible path between one input/output pair. The total amount of 2:1 32-bits multiplexers is twice the numbers of switches. Thus, the network has in total 32 switches (8 switches per stage * 4 stages), totalizing 64 2:1 multiplexers.

To connect 16 inputs to 16 outputs, the multiplexer model requires 16 16:1 multiplexers. Each 16:1 multiplexer has a chain of 15 2:1 multiplexers. Therefore, the model has in total 240 2:1 multiplexers. This is almost four times the area of the network. Figure 40 illustrates the multiplexer-based model to connect *N* inputs to the each one of the *N* outputs.

As demonstrated in equation (45), the network without extra stages is a series system of all switches. This is the same premise for the multiplexer-based model, without redundancy, all multiplexers are connected in series. In this sense, both reliabilities are very low and the multiplexer-based reliability is worse than the network, since the former has more multiplexers connected in series. For this reason, a comparison between the two models with fault tolerance seems more interesting, once we can evaluate if adding extra stages is actually an effective strategy to increase reliability.

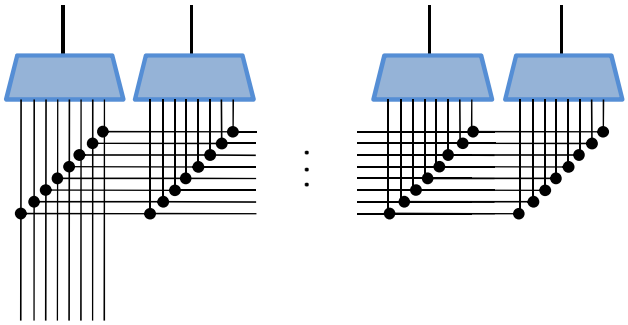


Figure 40. Multiplexer-based model

The chart in Figure 41 presents a comparison between the 16 input Omega network with three extra stages and the multiplexer-based model. The network has 56 switches in total, which is an area overhead of 75% in comparison to the original network. Additionally, four possible paths between an input/output pair are created. Adding this same amount of redundancy in the multiplexer-based model, we have 180 more multiplexers (75% of 240). This corresponds to a k-of-m model where m=420 (240+180) and k=240.

For this comparison, two spare multiplexer strategies were used. In the first one (Multiplexer-based model 1), adding spare multiplexers in the model implies in adding one more multiplexer to select between the original and the spare (this problem was discussed in section 4.5.2.2 - Figure 28). For these results, it was assumed that each spare multiplexer is added to a different multiplexer forming a pair of original and spare. Consequently, by adding 180 spares, the same amount of multiplexers must be added to select between the original and the spare. In the second multiplexer-based model, we assume that adding spare multiplexers consists in simply including more spares outputs, as it is done with the output multiplexers of the LOwER-FaT array (section 4.5.2.2).

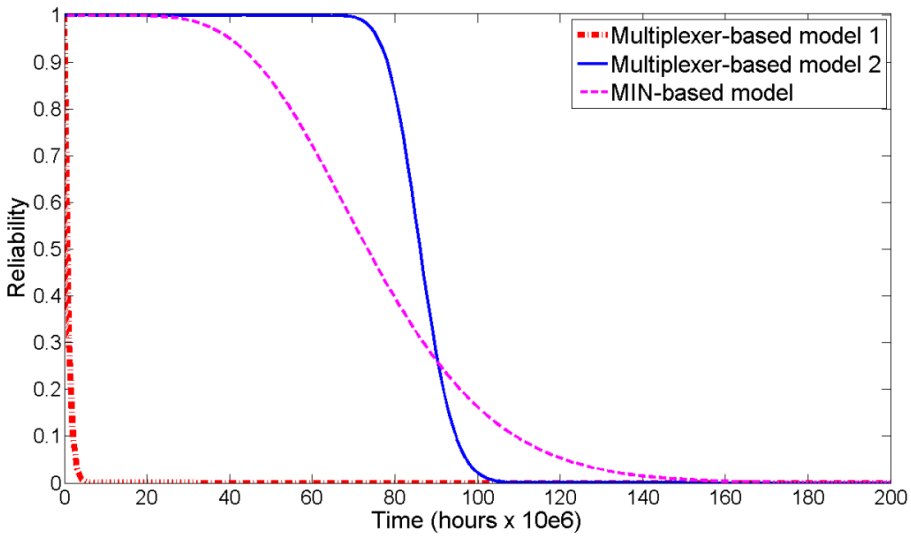


Figure 41. 16:1 Multiplexer-based model *versus* 16-input Omega network with 3 extra stages

From Figure 41, it can be observed that the MIN-based model presents significant higher reliability than the multiplexer-based model 1. The multiplexer-based model decreases to 0.99999 in 10 hours while the MIN-based model decreases to the same reliability at 11.01 x 10^6 hours. On the other hand, when comparing the alternate multiplexer-based model, the MIN-based one presents lower reliability. The results show a difference of 5.5 times difference, considering 0.99999 reliability, and 1.9 times at 0.96 reliability. This result indicates that in spite of the extra paths to connect an input/output pair, sharing switches among paths is not as efficient as having spare multiplexers using the strategy implemented in the output multiplexers model of the LOwER-FaT array.

In spite of the result presented above, we decided to compare the LOwER-FaT array with both models. This decision was based on the fact that to replace the models, not only the output multiplexers but also the input group has to be modified.

Reconfigurable architecture with different interconnection models

Since this is the same architecture already analyzed, only with a different interconnection model, the functional units and context registers present the same reliability functions described in equations (8), (9), (10) and (39).

Figure 42 illustrates one level of the reconfigurable architecture with multistage interconnection networks. As can be observed, there are four networks to send the input context (from the context registers) to all functional units. Because the networks have multiple outputs, each output is sent to one input of the functional units. Hence, in this approach, the multiplexers that select the inputs in each functional unit are not necessary. In addition, to preserve the capability of bypassing the input context, the networks also have the context from the previous network as output.

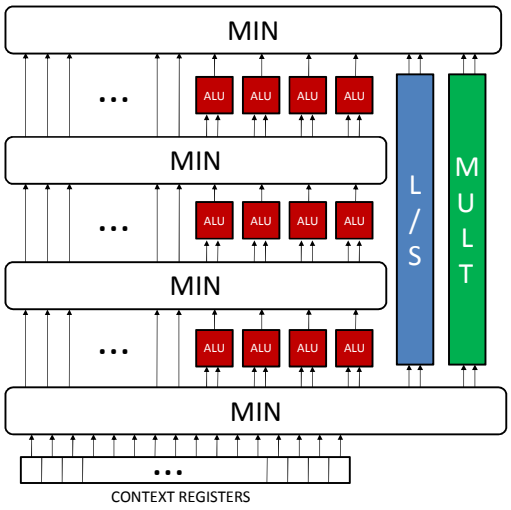


Figure 42. Array with multistage interconnection networks

It is important to highlight that this is only one design solution proposed to replace multiplexers by networks. There are different ways of placing the functional units and interconnects. However, more advanced architecture modifications may influence the results, affecting the comparison between both models. Furthermore, since the main

goal of this analysis is to evaluate the impact of the interconnection model on reliability, we consider that the presented analysis is adequate to validate our conclusions. For this reason, we did not perform a comprehensive design exploration nor presented any analysis related to performance, power and energy results when using the MIN model.

For the comparisons presented next, we used 32-input MINs with four extra stages, in a total of 9 stages and 144 switches (around 80% of extra area redundancy). The percentage of extra area due to the extra stages was used to add spare output multiplexers to the LOwER-FaT array. Figure 43 presents the reliabilities as a function of time for the reconfigurable architecture with both models.

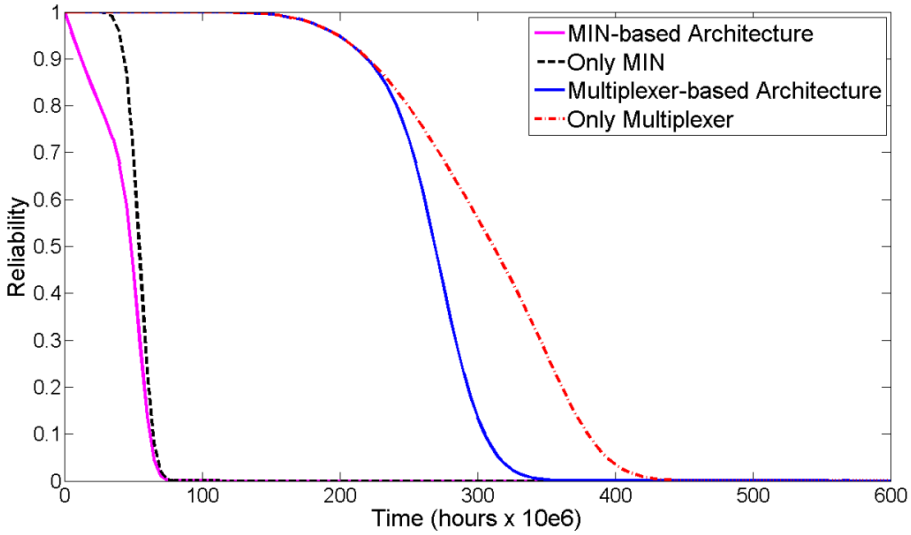


Figure 43. Multiplexer-based architecture *versus* MIN-based architecture

The curves in Figure 43 demonstrate that when adding the same percentage of redundancy to the interconnection models, the difference in time when reliability is 0.99999 is four orders of magnitude. This difference reduces to 40 times when reliability is 0.96. These results confirm that sharing switches between the extra stages is in fact an inefficient solution compared to the redundancy strategy implemented to the output multiplexers. To demonstrate this conclusion, the curves *Only MIN* and *Only Multiplexer* in Figure 43 also depicts the reliability curves of both models when the functional units and registers are not considered. As can be observed, the MIN-based model presents lower reliability than the Multiplexer-based model.

From the analyses presented above, we can conclude that to improve overall reliability of the reconfigurable architecture, the multistage interconnection model is an interesting solution when the spare multiplexers require the addition of a third multiplexer to select between the spares. At the same time, if redundancy can be applied with the output multiplexer strategy, one can have better results by adding spares multiplexers to the architecture.

Finally, it is important to mention that the analyses presented here were made in the context of the reconfigurable architecture and taking into consideration its characteristics. Although the analyses can be also applied for a more general case, it does not mean that multistage interconnection networks are not efficient solutions. In

the main works that propose MINs, they are used as solution to replace very expensive crossbars and not scalable single global bus (WU and FENG, 1980). Nevertheless, if the networks are used in the same context as the analysis presented in this work, then the multiplexer-based model appears to be better solution.

4.5.3 Performance and Energy Analysis

4.5.3.1 Graceful Degradation

Using the simulation environment described in the beginning of this section, we evaluated performance degradation in two different cases. In the first case, we considered the array with 16 context registers. In the second analysis, we doubled the number of context registers to compare the performance when the spare output multiplexers are included in the architecture.

Array with 16 context registers

Figure 44 presents the speedup degradation as a function of the fault rate. The initial speedup was calculated based on the applications running in the MIPS R3000.

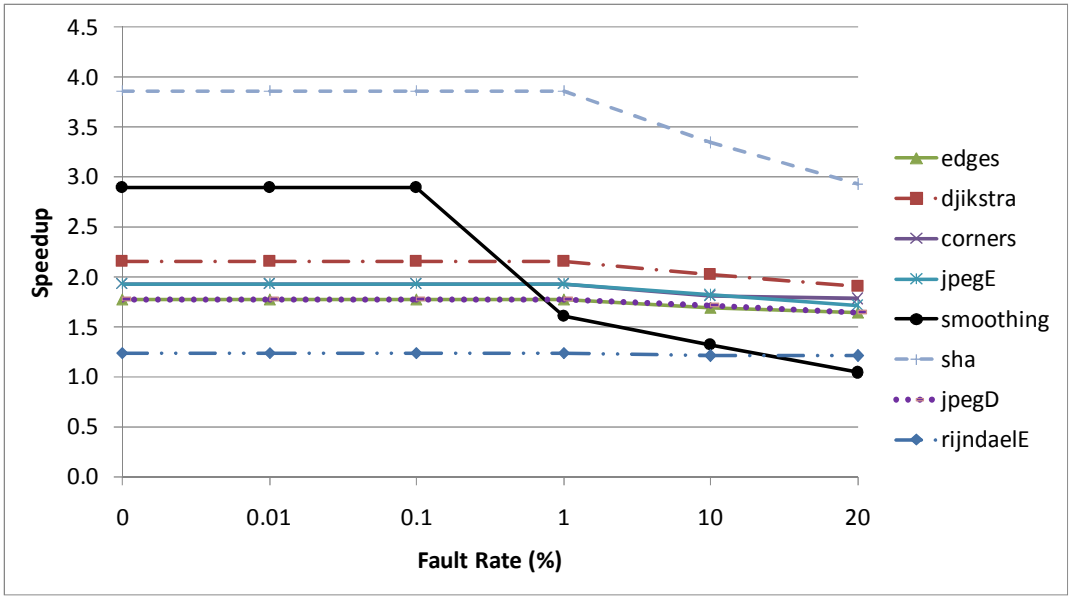


Figure 44. LOwER-FaT array speedup degradation – 16-context registers

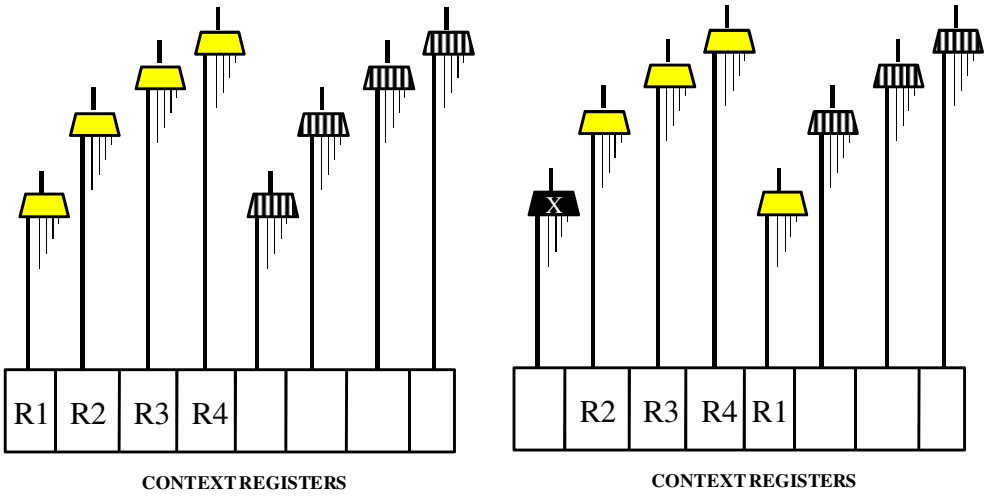
According to Figure 44, most applications presents a slight performance degradation, with exception of two applications, *smoothing* and *sha*. The former has a performance degradation of 64% and the latter 24.2%, under 20% of faulty resources. The mean performance degradation was 14.75%.

This result demonstrates that, in spite of the high fault rate, the LOwER-FaT array is still able to execute all applications and, with exception of *smoothing* application, accelerate execution when compared to the processor's execution.

The high performance degradation detected in the execution of *smoothing* and *sha* was due to the fact that these two applications require the largest amount of parallel

functional units in the architecture. For this reason, when many units are lost, there is no enough parallelism to execute the operations.

To compare the performance degradation when spare output multiplexers are added to the architecture, we have also simulated the architecture with 32 context registers. In this approach, more context registers were added. Therefore, if a data cannot flow through one context line due to a faulty multiplexer, the configuration generator selects a different context line. Figure 45 illustrated this strategy.



a) Fault-free multiplexers with spare context lines b) Faulty context line replaced by a spare one

Figure 45. Spare multiplexers in the LOwER-FaT array

In Figure 45.a, it is presented an example with four context registers in use and four as spare. For sake of simplicity, the example shows only one row of multiplexers. In Figure 45.b, the register allocated to the faulty context line is moved to a fault-free one (register R1).

Array with 32 context registers

Figure 46 presents the performance degradation of the LOwER-FaT array with 32 context registers. Before starting to analyze the performance degradation, it is important to mention that in some applications the initial speedup, when all the resources are fault-free, is higher than the initial speedup of the 16-context registers architecture. This happens because with more registers available, the configuration generator is able to allocate more registers in one configuration. In other words, there are configurations that require more than 16 registers. In the 16-context registers architecture, these configurations have to be broken in configurations with a small number of instructions. However, with the 32 context registers, the configurations are able to perform. Table 7 presents the speedup gains due to the addition of context registers.

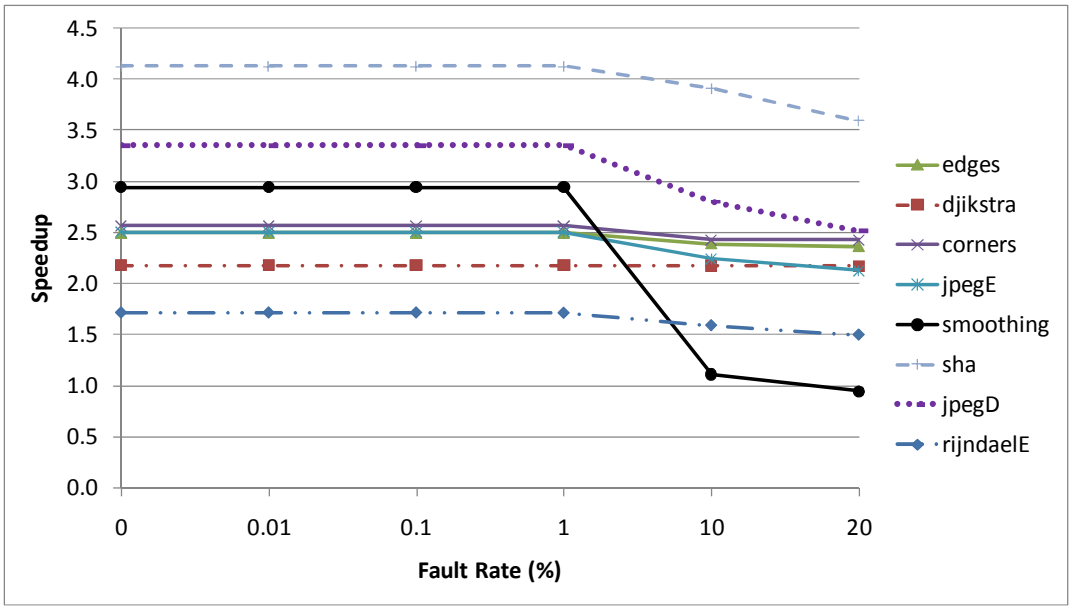


Figure 46. LOwER-FaT array speedup degradation – 32-context registers

According to Figure 46, *smoothing* was the application with highest acceleration degradation (67.6% of acceleration reduction under a 20% fault rate). Moreover, this architecture presented a higher mean performance degradation than the 16-context register one, 18% of performance degradation considering all the applications. This result is due to the fact that in this architecture, a higher percentage of output multiplexers was faulty, in comparison to the 16-context registers architecture. While the 16-context register architecture presented 48% of faults in functional units and 12% in the output multiplexers, the 32-context register one presented 42% of faulty functional units and 16% in the output multiplexers at 20% fault rate. Therefore, the simulation results also lead to the same conclusion presented in the reliability analysis. Interconnects are in fact the critical elements to system reliability.

Table 7. Percentage of speedup increase - 100% more context registers

Application	Speedup increase (%)
corners	32.6
djikstra	0.9
edges	40.9
jpegD	89.1
jpegE	29.8
rijndaelE	38.6
sha	6.9
smoothing	1.5
Average	**30.0**

Other shapes

We have also evaluated performance degradation considering the other two reconfigurable architecture organizations generated by ARISE tool (Table 4). For both architectures, we considered 32 context registers.

The analysis presented next considers the amount of ideal functional units based on the application requirements. This architecture, called Ideal Shape, was generated as the reference architecture. It achieves the higher performance possible with no area restriction. Therefore, the only reason not to execute parallel operations is the dependence between instructions. Figure 47 presents the performance degradation results.

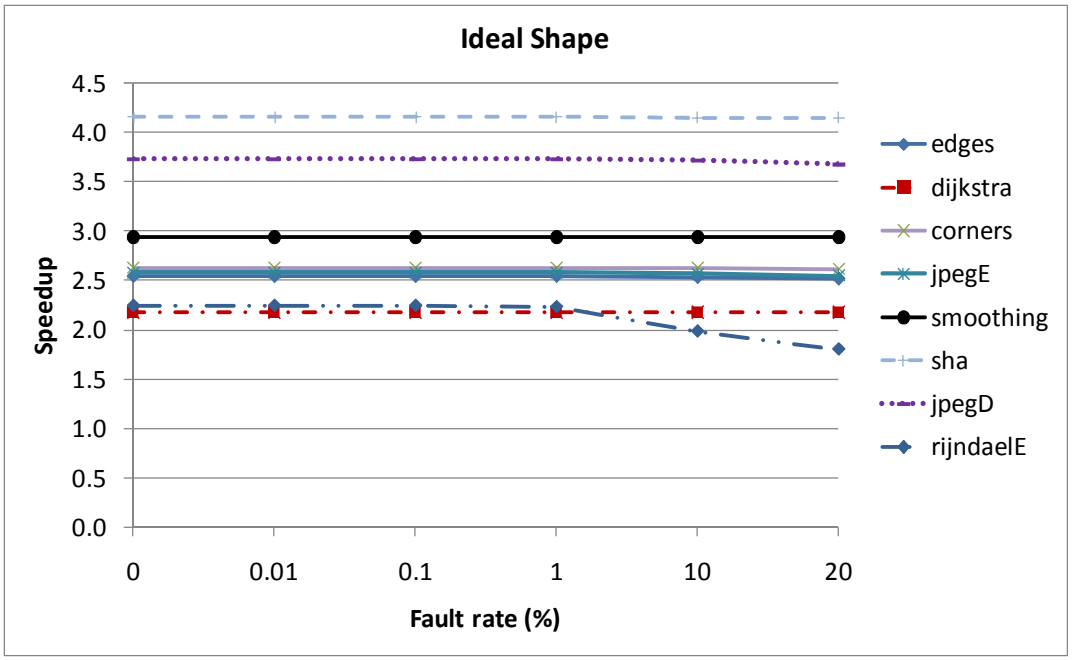


Figure 47. Ideal shape speedup degradation

As shown in Figure 47, the highest speedup degradation of ideal shape was presented in the execution of *rijndaelE*, with an average of 19.73% of acceleration loss under a 20% fault rate. This low degradation in the acceleration is a direct result of the substantial amount of functional units available in the ideal shape. Therefore, even loosing 20% of resources, there are still many resources.

The other architecture analyzed presents an extremely reduced amount of functional units (only 11% of the Ideal Shape resources). Figure 48 presents the performance degradation results.

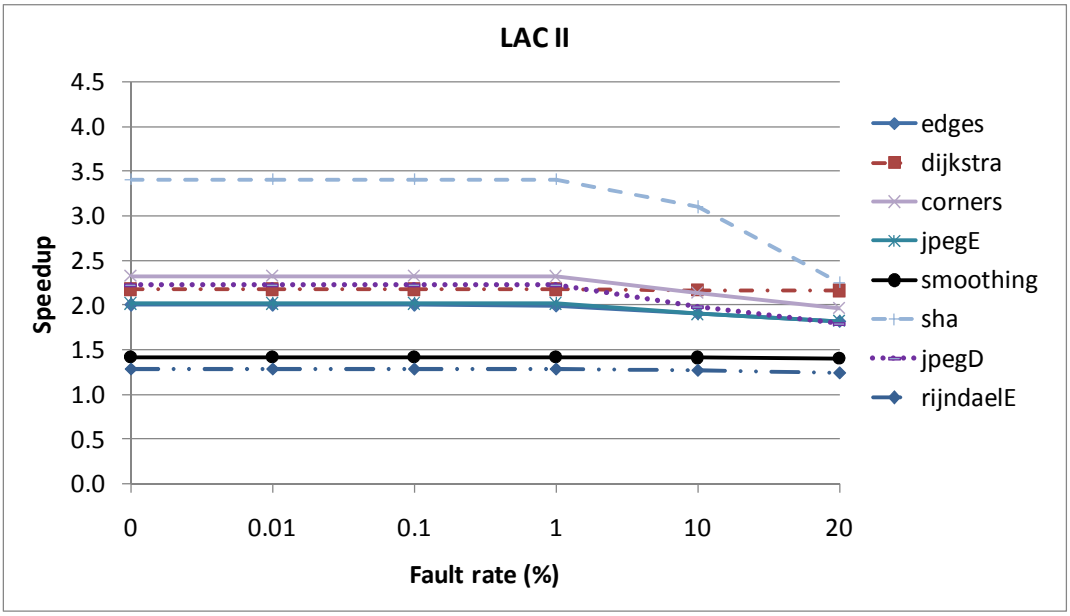


Figure 48. LAC II speedup degradation

As can be observed in Figure 48, since the LAC II architecture presents a reduced amount of units, it already starts with low performance, when compared to the other architectures. In spite of that, most applications did not present significant speedup degradation. This happened because with a small number of available functional units, only few instructions can be executed in the array. Consequently, most part of the applications are being executed by the processor. As one can observe, the most affected applications were the ones with speedup higher than 2. Some examples are: *corners,* with 15.5% of speedup degradation under 20% fault rate; *jpegD*, with 19.3% of degradation; *edges*, with 9.2% degradation, and *sha*, which had the highest speedup and for this reason presented the highest degradation, 33.9%.

4.5.3.2 Energy Reduction

In an attempt to reduce the power consumption of the reconfigurable architecture, we also proposed a solution that uses power-gating technique. The use of this technique in the reconfigurable architecture was already proposed in (RUTZIG, BECK and CARRO, 2009). However, in this analysis, we propose to extend the use of power-gating to be implemented also in faulty functional units.

Power-gating is one of the most effective leakage reduction methods (SHI and HOWARD, 2006). It consists of using sleep transistors to shut off the power supply of standby or idle resources.

In the original work (RUTZIG, BECK and CARRO, 2009), the authors proposed to add two sleep transistors in each functional unit, and during dynamic configuration, shut off the idle units. In next configuration, in case of the idle units start to operate, they can be shut on. Since the configuration generator allocates the functional units in each configuration, this same mechanism can be used to set the on/off bits in each functional unit. Therefore, according to the authors, the area overhead corresponds to

two transistors for each functional unit and only a few bits to turn on and off the sleep transistors.

To reduce even more power consumption, we propose to use the same mechanism that controls the sleep transistors of the idle functional units, to control the sleep transistors of the faulty ones. The difference is that sleep transistors of idle functional units can change between ON and OFF states according to the configuration, while sleep transistors of faulty units will always be set to the OFF state.

An evaluation of the energy consumption of the reconfigurable system and the MIPS R3000 demonstrates the effectiveness of the sleep transistor approach.

Figure 49 presents the energy consumption of the reconfigurable system and the standalone MIPS R3000. The Full Reconfigurable System bar represents the system without the sleep transistor technique and the Low Energy Reconfigurable System bar corresponds to the system with sleep transistor approach.

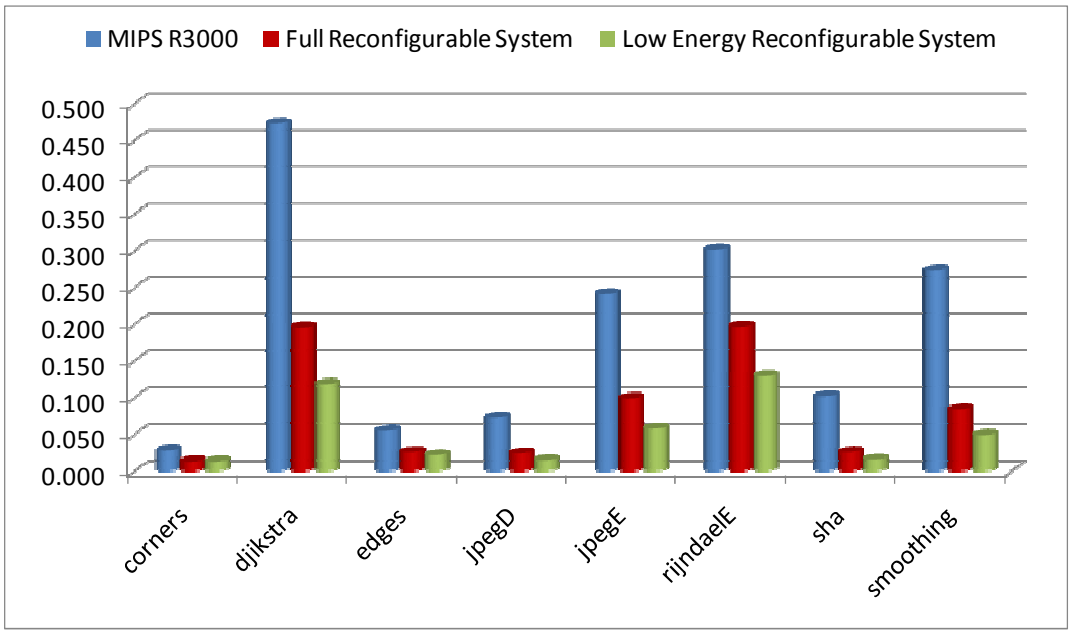


Figure 49. LOwER-FaT array energy consumption

Two observations can be made from the results presented in Figure 49. First, the application execution on the reconfigurable system resulted in a large amount of energy saving. This is a consequence of the speedup improvement achieved by the parallelism exploitation and by reducing the accesses to the processor instruction cache. The execution on the reconfigurable system resulted in at least 34.7% of energy saving.

Furthermore, the application of the sleep transistor approach reduced the energy consumption of the reconfigurable system in at least 16% in the *edges* execution and up to 41% in *smoothing* execution.

As mentioned before, sleep transistors can also be used to shut off faulty functional units. Although this approach is used to avoid power consumption of units that will never be used, this is not reflected in the energy consumption of the

reconfiguration system. Since the faults reduce the amount of functional units available, this directly affects the execution time, which increases the energy.

For this reason, the applications studied in this work presented a slightly increase in energy consumption when too many faults were presented. Ideal shape presented up to 6.2% of energy increase with respect to a fault-free array. Likewise, shape LAC I presented a 9.5% increase, and shape LAC II presented a 9.7% energy increase, all under a 20% fault rate.

5 PIPERENCH

PipeRench is a coarse-grained reconfigurable architecture firstly proposed in 1997 (SCHMIT, 1997), targeted to accelerate execution of streaming multimedia applications (CHOU, PILLAI, SCHMIT, *et al.*, 2000), (SCHMIT, WHELIHAND, TSAI, *et al.*, 2002) and (KAGOTANI and SCHMIT, 2003). (SINHA, KAMARCHIK and GOLDSTEIN, 2000) proposed a fault tolerance strategy to cope with permanent faults in logic, memory and interconnect elements.

We have opted for investigating this architecture because its fault tolerance strategy presents an interesting solution to avoid faulty processing elements and interconnects. The authors propose to use register files to bypass the entire faulty processing element (PE), which is composed of ALU, input and output multiplexers. The register files are directly connected in one another and have independent access. With this approach, the system does not have to rely on multiplexers to send data to other PEs. On the other hand, this strategy introduces the register files as new elements that may be critical to reliability.

Next, we present a brief architecture description, based on the details presented in (SCHMIT, WHELIHAND, TSAI, *et al.*, 2002). In section 5.2, we detail the fault tolerance technique proposed in (SINHA, KAMARCHIK and GOLDSTEIN, 2000). From the architecture and fault tolerance technique description, we modeled the reliability function, which is described in section 5.3.To conclude this chapter, the last section provides a reliability analysis comparing this architecture with LOwER-FaT array when area equivalence is assumed. Additionally, an evaluation of the fault tolerance technique is also presented.

5.1 Architecture Description

PipeRench relies on fast partial dynamic reconfiguration to provide hardware virtualization. The architecture consists of several processing elements connected to each other to form pipeline stages called *stripes*. A set of physical stripes is used to provide virtual stripes. The pipeline stage can be configured every cycle, while other stages are executing. Figure 50 illustrates the physical and virtual stripes.

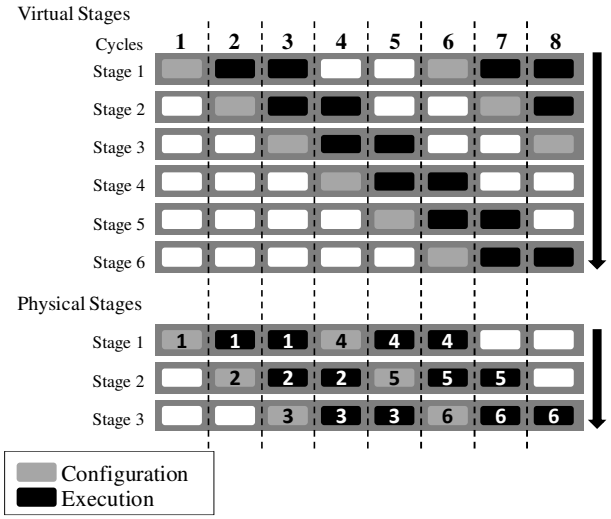


Figure 50. PipeRench physical and virtual pipeline

Each processing element (PE) consists of an ALU, registers and multiplexers. All PEs in one stripe can connect to each other through local buses. Moreover, a global bus allows connection between stripes. Connected to each PE there is a pass register file that sends and receives data from the PEs and from the register files of previous stripes. The connection among register files is independent from the PEs connection and it is done through the interstripe interconnect lines. Figure 51.a illustrates the PipeRench architecture and Figure 51.b illustrates the PEs and the register file, as well as their connections.

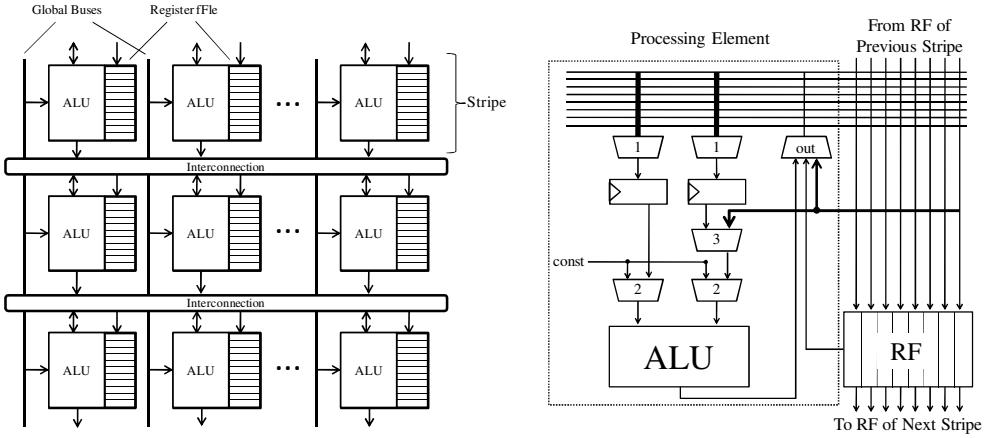


Figure 51. PipeRench block diagram a) system b) processing element

5.2 Fault Tolerance Technique

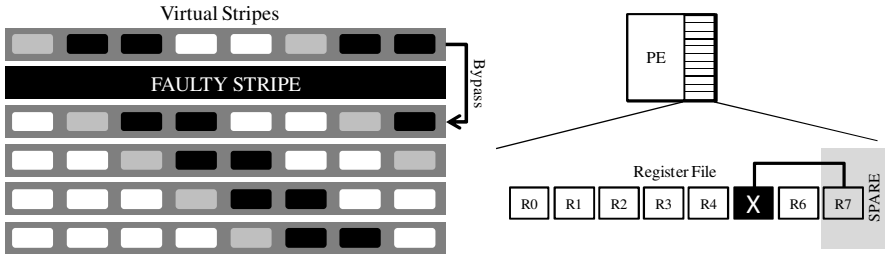
The fault tolerance strategy implemented in PipeRench consists of three steps: fault detection, fault isolation and fault tolerance.

To detect faults the authors proposed a BIST-like (Built-In Self-Test) algorithm that only tests for faults that can jeopardize the correct execution of the current configuration. For this reason, the authors call it Built-In Applicable Self-Test (BIAST).

The testing mechanism consists in adding extra virtual stripes to execute test configurations. Once the fault is detected, the next step isolates the fault. The isolation requires the execution of the test configurations alternating the tested stripes until the error can be isolated. Subsequently, the fault tolerance mechanism works around the fault to allow correct execution of the system.

There are two types of faults: non-interstripe interconnect faults and interstripe interconnect faults. The former consists of faults that affect the processing elements. Since there are no spare processing elements, the fault-tolerant mechanism sets the whole stripe as faulty. In this case, the pass register files are used to bypass data through the faulty stripe. This strategy is possible because the same data sent to the ALU is also sent to the register file using an individual input. The ALU's output can be sent to the register file or to the buses through an output multiplexer. There is no fault tolerance strategy to cope with faulty output multiplexers. Therefore, a fault in an output multiplexer will affect the processing element, consequently affecting the stripe that must be isolated. Additionally, the next stripe is configured with the function of the faulty one. Reducing the amount of available stripes causes a performance degradation of one stripe delay per stripe. Figure 52.a illustrates the fault-tolerant strategy to non-interstripe interconnect.

The interstripe interconnect faults affect the interstripe interconnect lines. Since each processing element has several interconnect lines (one interconnect line for each register of each register file in one stripe), if an interconnect line is faulty, there are no enough lines to move all register data among PEs. To solve this problem, the fault-tolerant mechanism leaves one register in each PE's register file with its respective interconnect line as spare. Therefore, it is possible to tolerate one faulty interstripe interconnect line per stripe. However, if multiple faults happen in the same stripe, the system will not work properly. Figure 52.b illustrates the fault-tolerant strategy to interstripe interconnect faults.



a) Non-interstripe interconnect b) Interstripe interconnect

Figure 52. PipeRench fault tolerance technique

5.3 Reliability Model

Because there is no enough detail about the overall area, processing elements and interconnects, to model and analyze reliability, we have assumed the same area of the LOwER-FaT array. Thus, we consider that the ALUs in PipeRench perform the operations and have same area of the LOwER-FaT array's ALUs. We also assume that the architecture is 32-bits.

To model PipeRench reliability we start with the minor subsystem in the hierarchy. Based on Figure 51, each processing element is composed of five input multiplexers and two registers connected to the ALU's input. The PE's output is sent to a multiplexer that also receives the register file's output. Since the PE and register file work independently, there are two subsystems connected in parallel, the PE subsystem and the register file subsystem.

Starting with the PE subsystem, we have to identify the atomic elements, which have reliability function given by equation (1). These units are the ALU, the register, and the multiplexers, with reliability functions given by equations (50) to (53),

$$R_{ALU}(t) = e^{-\lambda_{ALU}t}, \tag{50}$$

$$R_{Register}(t) = e^{-\lambda_{Register}t}, \tag{51}$$

$$R_{1bMux}(t) = e^{-\lambda_{Mux}t}, \tag{52}$$

$$R_{32bMux}(t) = \prod_{i=1}^{32}\left(R_{1bMux[i]}(t)\right). \tag{53}$$

The five input multiplexers can be split into three types, based on the amount of inputs. The first type is the multiplexers that select from which interconnect line the ALU will receive data. It is described in equation (54), where *im1* is the number of 32-bits 2:1 multiplexers required to select between *N* 32-bits interconnect lines. *N* is the number of processing elements in one stripe,

$$R_{inputMux1}(t) = \prod_{i=1}^{im1}\left(R_{32bMux[i]}(t)\right). \tag{54}$$

The second type has as input a register and all register file's input. Equation (55) describes the reliability function, where *im2* is the number of 32-bits 2:1 multiplexers required to select between the inputs,

$$R_{inputMux2}(t) = \prod_{i=1}^{im2}\left(R_{32bMux[i]}(t)\right). \tag{55}$$

Finally, the third type is the 2:1 32-bits multiplexer, with reliability function described in equation (56). In Figure 51.b, the input multiplexers are identified by their numbers.

$$R_{inputMux2}(t) = R_{32bMux}(t) \tag{56}$$

Since there is no fault tolerance implemented in the PE subsystem, all the components that compose this subsystem are connected in series. Equation (57) depicts the reliability function of the processing element subsystem,

$$R_{ProcessingElement}(t) = \prod_{i=1}^{2}\left(R_{inputMux1[i]}(t)\right) x \prod_{j=1}^{3}\left(R_{inputMux2[j]}(t)\right) x \prod_{k=1}^{2}\left(R_{Register[k]}(t)\right) x R_{ALU}(t). \tag{57}$$

The other subsystem that composes the processing element is the register file subsystem. According to the architecture description, the register file has one spare register to replace a working one in case of a fault. This is represented as a k-out-of-m system, where *k=m-1*. The function for the register file is given by equation (58),

$$R_{RF\_subsystem}(t) = \sum_{i=k\_reg}^{m\_reg} \binom{m\_reg}{i} \left(R_{Register}(t)\right)^{i} \left(1 - R_{Register}(t)\right)^{m\_reg-i}, \quad (58)$$

where *m_reg* is the total amount of registers in the register file and *k_reg* is *m_reg-1*.

Up to this point, the reliability model is completely predictable and no significant design strategies were made to actually increase reliability. The only strategy that may contribute to increase reliability is the fact that the register file has a spare register.

However, the next subsystems in the hierarchy present an interesting design strategy that we would like to reemphasize since it may be the solution to completely avoid the faulty interconnects and solve such a critical problem: the processing elements and register files were designed with independent outputs. As can be observed in Figure 51, the register file's output is sent to the output multiplexer and to the register file of the processing element in next stripe.

Therefore, in this solution, the system does not need to rely on multiplexers to keep the architecture working properly. If an ALU or an output multiplexer fails, it is possible to avoid these elements, and send the data to the next stripe through the register file.

To represent this strategy in the reliability model, firstly it is necessary to describe the dependence between the processing element subsystem and the output multiplexer. This dependence is represented as a series connection, with function given by equation (59).

$$R_{ProcessingElement+OutputMux}(t) = R_{ProcessingElement}(t)\ x\ R_{outputMux}(t) \quad (59)$$

Additionally, the output multiplexer is a 32-bits multiplexer that has as input the ALU, the register file output and the register file input, which corresponds to the output of the previous register file, with function given by equation (60). Where *o* is the number of 32-bits 2:1 multiplexers required to compose the output multiplexer,

$$R_{outputMux}(t) = \prod_{i=1}^{o} \left(R_{32bMux[i]}(t)\right) \quad (60)$$

Due to the interstripe interconnect strategy, all processing elements in one stripe must work so the stripe works. Otherwise, the entire stripe must be avoided. To represent this strategy, all processing elements (with their respective output multiplexer) in one stripe compose a series subsystem, with function given by equation (61), where *pe* is the amount of processing elements in one stripe,

$$R_{stripe}(t) = \prod_{i=1}^{pe} \left(R_{ProcessingElement+OutputMux[i]}(t)\right). \quad (61)$$

To complete the interstripe interconnect strategy model, it is necessary to connect the stripes subsystems. According to the authors, it is possible to bypass a faulty stripe and configure the next one to replace the faulty one. However, the authors do not mention how many stripes need to work before affecting the properly functioning of the system. Therefore, in this work, we assume that only one stripe needs to work, which is the best assumption we can make in favor of the architecture. Any other assumption will result in worse reliability results. Based on this assumption, the stripes compose a parallel subsystem with reliability function described in equation (62),

$$R_{StripesSubsystem}(t) = 1 - \prod_{i=1}^{st}\left(1 - R_{stripe[i]}(t)\right), \tag{62}$$

where *st* is the total number of stripes.

The last subsystem in the hierarchy is related to the register files. Because all the register files must work, so the system can work properly, the register files form a series subsystem, with function given by equation (63),

$$R_{RegisterFiles}(t) = \prod_{i=1}^{N}\left(R_{RF\_subsystem[i]}(t)\right), \tag{63}$$

where *N* is the total number of PEs (PEs multiplied by the number of stripes).

Finally, the register files and the stripes compose a parallel subsystem, where each subsystem can work independently. Equation (64) represents this connection between the subsystems and concludes the PipeRench model,

$$R_{PipeRench}(t) = 1 - \left(\left(1 - R_{RegisterFiles}(t)\right)x\left(1 - R_{StripesSubsystem}(t)\right)\right). \tag{64}$$

Based on the equations presented above, from a reliability point of view, we can affirm that the fault tolerance strategy implemented in PipeRench eliminated the system dependence on the output multiplexers. This means that, if all the output multiplexers fail, the system is still able to operate. Even if it only bypasses the inputs to the outputs of the architecture, without any computation by the ALUs.

In order to do that, another element was introduced to the system, the register file. If one register file fails, it is not possible to bypass the results to the next stripe. Consequently, the register files are the critical elements to system reliability. In bottom line, the fault tolerance strategy proposed by (SINHA, KAMARCHIK and GOLDSTEIN, 2000) transferred the dependence on the output multiplexer to the dependence on the register file. The analyses presented in next section evaluate if this trade was beneficial or detrimental to system reliability.

5.4 Experimental Results

Next, we present the reliability analysis considering the model described in last section. Because there are no details about the area of each component, and how the

performance results were obtained, a comparison considering area, performance and energy between the LOwER-FaT array and PipeRench was not possible.

For this reason, to evaluate reliability and compare the fault tolerance strategies of both architectures, we assume that both architectures used the same components, with the same failure rate.

5.4.1 Area

Based on the architecture description presented by the authors, we have estimated the area considering two different cases. In the first case, we considered the LOwER-FaT array and PipeRench with same area. In order to do this, we calculated the total area of the LOwER-FaT (with functional units, multiplexers and registers) and assumed this same area to PipeRench. The problem of this approach is the fact that PipeRench only has ALUs as functional units. Therefore, to assume the same area, we had to convert the LOwER-FaT array's area only in ALUs, multiplexers and registers.

The second case was based only on the amount of ALUs used in LOwER-FaT. In this case, the architectures do not have the same area, but present the same amount of ALUs and hence, can perform the same amount of operations.

Another problem related to PipeRench organization is the ALUs distribution along the stripes. This distribution influences reliability results, since large stripes increase the amount of processing elements in series. On the other hand, many stripes increase the amount of parallel stripes. For this reason, we based the amount of ALUs per stripe and the total amount of stripes according to the parallelism presented in the LOwER-FaT array.

5.4.1.1 Area equivalence

To calculate the amount of ALUs in each stripe, we used the amount of parallel functional units existing in LOwER-FaT array. According to section 4.6.1.1, there are 9 ALUs, 2 multipliers and 4 load/store units in parallel, in a total of 15 functional units. Considering the area ratio of the components described in Table 2, 15 functional units are equivalent to 31 ALUs. Therefore, there are 31 ALUs in each stripe.

The number of stripes is calculated based on the total area of LOwER-FaT array divided by the amount of ALUs per stripe. This calculation resulted in 170 stripes.

5.4.1.2 ALU equivalence

Taking into consideration only the same amount of ALUs, each stripe has only 9 ALUs and the total amount of ALUs is 459. Thus, the amount of stripes is 51.

In both cases, area and ALU equivalence, the register files have 8 registers. This number is based on the studies presented by the authors in (SCHMIT, WHELIHAND, TSAI, *et al.*, 2002).

5.4.2 Reliability Analysis

The analysis presented next compares three architectures: PipeRench with the same area of LOwER-FaT (area equivalence curve), PipeRench with the same amount of ALUs than LOwER-FaT array (ALU equivalence curve) and LOwER-FaT array. Figure 53 illustrates the reliabilities as a function of time for all three architectures.

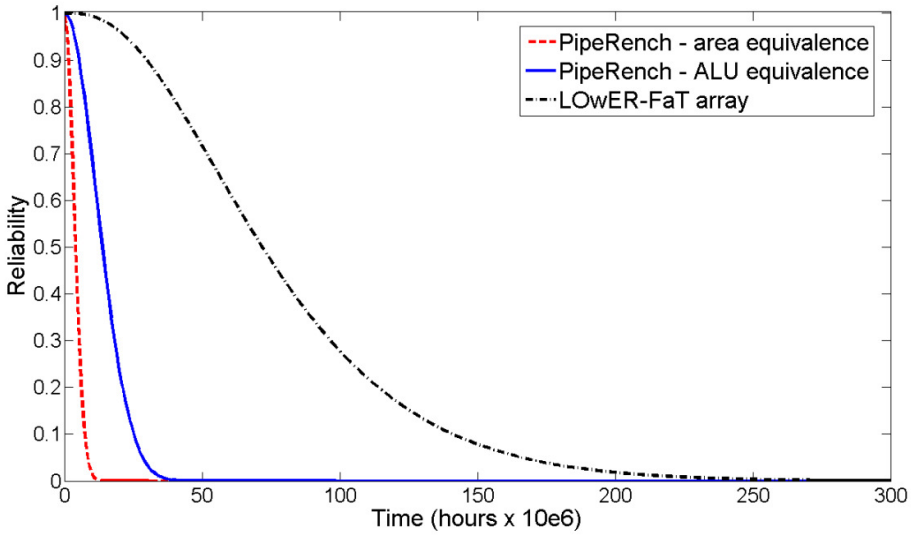


Figure 53. PipeRench *versus* LOwER-FaT array

As can be observed in Figure 53, even considering a reduced amount of resources in PipeRench, the LOwER-FaT array still presents higher reliability. For the area equivalence comparison, the results show a reliability of 0.99999 in around 11,000 hours for PipeRench against 860,000 hours achieved by the LOwER-FaT array. Moreover, a reliability of 0.96 is detected in approximately 969,000 hours, against 9,500,000 hours achieved by the LOwER-FaT array. This is a difference of approximately 78 and 9.8 times for the 0.99999 and 0.96 reliabilities, respectively.

When comparing both PipeRench architectures, a better reliability is achieved with ALU equivalence. This is due to the reduction in the amount of processing elements that also affects the amount of register files. The area equivalent PipeRench has 5,270 register files. On the other hand, the ALU equivalent one has only 459 register files. In spite of that, the reliability is lower than LOwER-FaT array. PipeRench with ALU equivalence achieves 0.99999 in 40,000 hours and 0.96 in 3,200,000 hours. This is around 21 and 3 times lower than LOwER-FaT array's reliability.

The analysis presented above gives an indication of which architecture presents better reliability. Nevertheless, it is not possible to make any assumption about the fault tolerance strategy implemented in PipeRench.

To evaluate if relying on independent register files is more effective than relying on multiplexers, a more comprehensive analysis of PipeRench reliability model is presented next.

Firstly, it is necessary to determine how much the register files impact on reliability. In order to do this, we start by showing the reliabilities of the register files subsystem and the stripes subsystems that compose PipeRench reliability model, as described in equation (64).

Figure 54 presents the reliabilities as a function of time for both subsystems, register files and stripes. Additionally, the overall PipeRench reliability for the area equivalent architecture is also presented. As can be observed in Figure 54, *stripes* subsystem reliability is higher than the *register files* subsystem. At the same time,

PipeRench presents same reliability as *register files* subsystem. This indicates that the register files are not only influencing the overall reliability but also preventing the system to achieve better results.

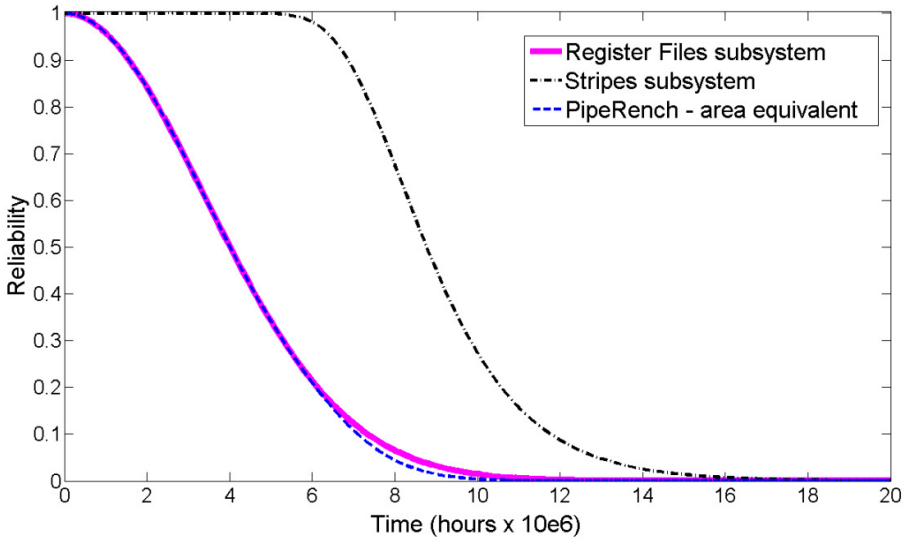


Figure 54. PipeRench reliability analysis

Based on this, a possible solution to improve reliability without changing architecture design is to increase the number of spare registers in each register file. The original fault tolerance strategy described in (SINHA, KAMARCHIK and GOLDSTEIN, 2000) proposes one spare register in each register file. In the reliability model, this strategy is described in equation (58).

In an attempt to improve reliability, we added more spare registers to each register file. Figure 55 presents the reliability curves of the original architecture with only one spare register and with 100%, 300% and 700% spare registers for the area equivalent architecture.

According to the results in Figure 55, by adding 100% spare registers the reliability presented a very significant improvement, achieving 0.99999 from 11,000 hours to 3.8 $x10^6$ hours. This is around a 345 times improvement. However, by adding more spare registers, the reliability starts to decline, as observed when 300% and 700% spares are added.

This reduction is a consequence of the increase in the area of the multiplexers that have the registers as input. Therefore, after some point, the *stripes* subsystem function (which includes the multiplexers) becomes the dominant term in reliability function. As already discussed during the analyses of the LOwER-FaT array in section 4.5.2.2 (Figure 28), adding spare multiplexers is not an efficient solution, since it requires the addition of an extra multiplexer to select between the spares.

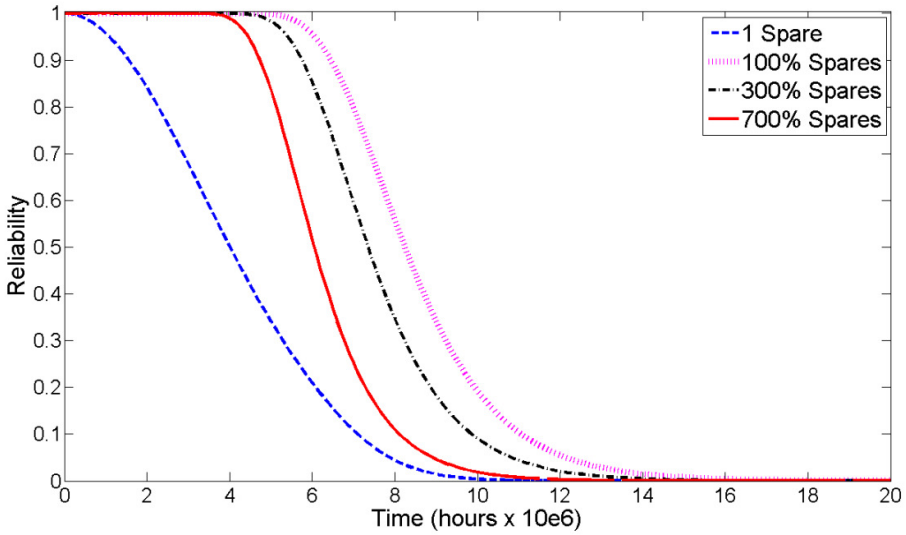


Figure 55. PipeRench reliability - spare registers

Therefore, based on the analyses presented above two conclusions can be made: 1) In the current architecture design, the register files are in fact the critical elements to system reliability. This was demonstrated with the comparisons between the register files and stripes subsystems reliability. In this analysis, it was showed that the overall system reliability was equivalent to the register files subsystem reliability (Figure 54).

2) Once again, we have demonstrated that there is a limit in improving reliability. This limit is determined by the fact that adding redundancy means area cost and in many cases, it also means adding new components that will directly influence reliability.

To conclude this analysis, we propose to eliminate the register files from the original PipeRench architecture and rely on the input and output multiplexers to send data throughout the architecture. This strategy allows to finally conclude if using independent register files is better than relying on the interconnects.

Because the register files are a type of redundancy that allows replicating data, to perform a fair comparison, we propose to replace the area of the register files by spare multiplexers.

To calculate the amount of spare multiplexers with same area of the register files, we used the component ratio presented in Table 2 and the amount of registers and multiplexers in the architecture. Moreover, assuming that each spare multiplexer adds a third multiplexer to select between the original and spare outputs, with the same area of the register files it is possible to add 18% of spare multiplexers. Figure 56 presents the reliability as a function of time when replacing the register files by spare multiplexers.

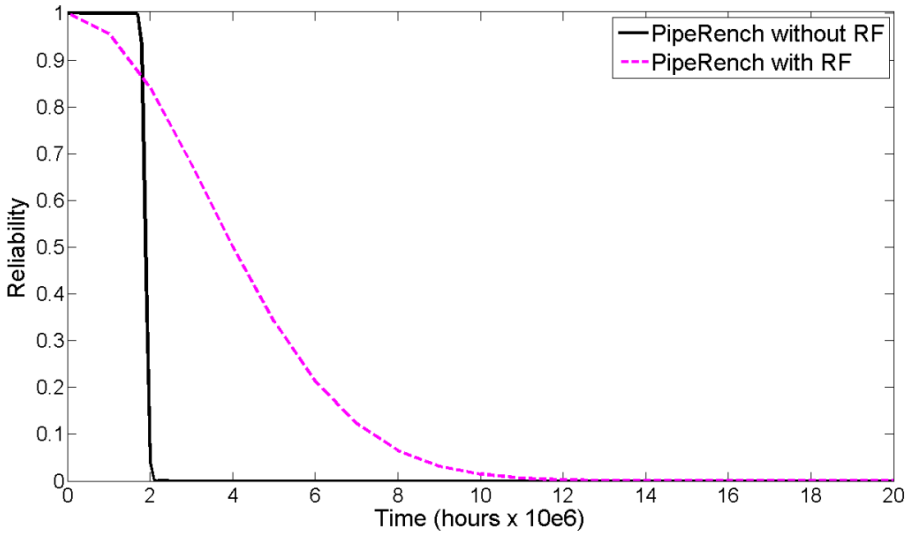


Figure 56. PipeRench without RF

According to Figure 56, the architecture without register files and with spare multiplexers presents a better result than the one with register files. The results show a difference of 148 times when reliability achieves 0.99999 and 1.84 times when it achieves 0.96.

This indicates that a better reliability improvement would be achieved if instead of having register files, the architecture had spare multiplexers. In order to do this, it would be necessary a complete change in the architecture design and execution. Since no more sequential path would exist between operations, this would affect the pipeline-based execution. Therefore, this hypothetical comparison was performed only to speculate if investments in register file were more efficient than investments in the interconnects. However, an important conclusion that can be made from these experiments is that it does not matter what type of strategy is implemented to provide communication among functional units, providing connection among function units is in fact the critical factor to reliability and consequently where the investments should concentrate.

6 RELATED WORK

This chapter is devoted to present some of the main works that relate to the proposed work. Since this work proposes a more comprehensive solution to assist in the design of reliable reconfigurable architectures, it is correlated to fault-tolerant strategies to increase yield and/or reliability in reconfigurable architectures.

In spite of that, it is not possible to provide a fair comparison to each previously proposed solution, since we are not proposing a specific solution to a specific architecture. Nevertheless, in this chapter we present some of the works that propose fault-tolerant strategies targeted to reconfigurable architectures. We also provide a comparison to the LOwER-FaT array used as case study.

6.1 Fault-Tolerant Reconfigurable Architectures

Reconfigurable architectures are potential candidates to cope with high fault rates predicted to future technologies. The inherent redundancy can be used to fault tolerance similar to the techniques used in memory devices (STOTT, SEDCOLE and CHEUNG, 2008). Moreover, the reconfiguration capability allows a high degree of flexibility that can be exploited to yield enhancement and reliability increase. In case of yield enhancement, the replacement of faulty resources can be performed at the manufacture by transparently reconfiguring the architecture, consequently avoiding discarding the faulty chips. As for reliability increase, reconfiguration can be used as an efficient fault-tolerant mechanism to reduce downtime and maintenance costs (HANCHEK and DUTT, 1996).

Despite fault tolerance has existed for 60 years, only in the early 90's fault tolerance approaches in reconfigurable architectures emerged as attractive solutions. This is a consequence of the consolidation of reconfigurable architectures as an efficient paradigm to balance the tradeoff between the low flexibility but high performance of ASICs (Application Specific Integrated Circuits) and the high flexibility and low performance of general-purpose processors.

The fault tolerance approaches in reconfigurable architectures differ in many aspects, such as fault type (transient or permanent faults); reconfiguration time (dynamic or static); targeted to yield enhancement or reliability increase; tolerance in logic blocks or interconnections; etc. Many of the approaches combine the different aspects to provide an efficient fault tolerance mechanism. For this reason, classifying the techniques in a specific category is very difficult and sometimes impracticable. Some works found in the literature divide the techniques between hardware level and

configuration level. The fault tolerance is considered hardware-level when the modifications in configuration targeted to fault tolerance are made during manufacturing or before device usage. On the other hand, configuration-level approaches implement the fault-tolerant techniques during the device lifetime, either at run-time or statically. We have chosen this classification as the way to present the techniques in this chapter. Later in this chapter, we summarize the techniques with their respective aspects presented in Table 8.

6.2 Hardware-Level Fault Tolerance

Hatori, Sakurai, Nogami, *et al.* (1993) were the first to propose a fault tolerance mechanism for yield enhancement of FPGAs, working at the early stages of circuit manufacturing. The approach consists in logically shifting the rows to avoid the faulty one. To shift the rows, first, the decoder for the row that has the faulty programming element is disabled and the spare row is enabled. Therefore, spare rows must be added to the design. In the experiments presented in (HATORI, SAKURAI, NOGAMI, *et al.*, 1993), one spare row is added. This allows tolerating only one faulty row. To allow interconnection between the spare rows and the fault-free ones, extra wiring segments are also added to the circuit. Figure 57 illustrates Hatori's approach.

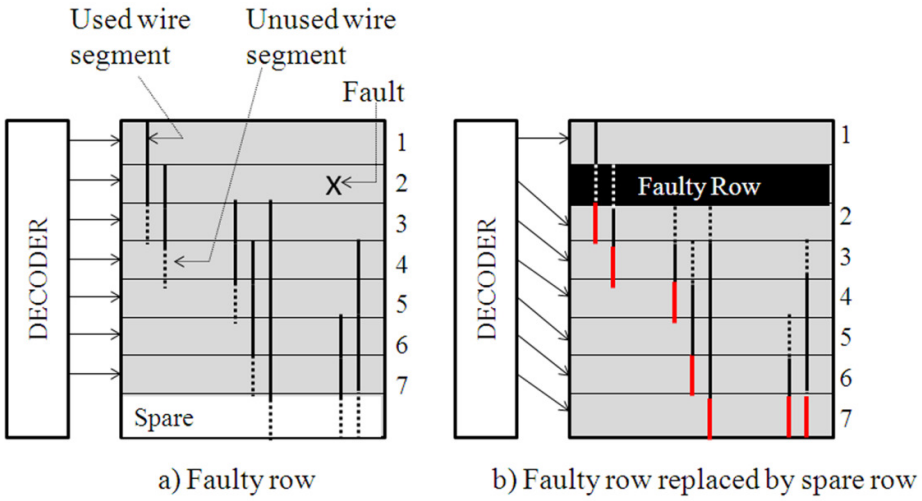


Figure 57. Hatori, Sakurai, Nogami, et al (1993)'s fault-tolerant mechanism

In Figure 57, a fault in one of the rows disables the decoder for the row and enables the spare row and extra wire segments in this row. The mechanism works in two steps. Firstly, a manufacturing test detects and diagnoses the faults. Then, the fault-tolerant mechanism uses the spare rows to permanently bypass the faulty one. The authors claim that with two spare rows it is possible to achieve up to 80% of yield, which is around 2.5 times more than the yield of unprotected circuits. To achieve this yield, the area overhead is around 2% for row selection logic, and there is no significant are overhead for the additional wiring. Moreover, the use of the spare wire segments requires the programming of the connections between the wires. These connections are called *programmable interconnect points* (PIPs). According to the authors, the fault-tolerant mechanism introduces a performance degradation of around 5% due to the programming of the extra PIPs. There is no reconfiguration overhead, since the approach works during manufacturing.

The main drawback of this approach is the fact that it only tolerates one fault per row. Furthermore, the fault-tolerant mechanism only tolerates logic faults, leaving interconnection faults uncovered.

Howard, Tyrrel and Allison (1994) also proposed a fault-tolerant mechanism to increase yield of FPGA circuits. To reduce the reconfiguration overhead and increase fault coverage, the authors proposed to group the logical elements in blocks (sub-arrays) with individual memories and reconfigure each block individually. Therefore, instead of shifting individual logic elements or rows, the fault-tolerant mechanism shifts an entire block. This requires less extra interconnects and consequently the impact on overall performance is reduced. Figure 58 illustrates this mechanism.

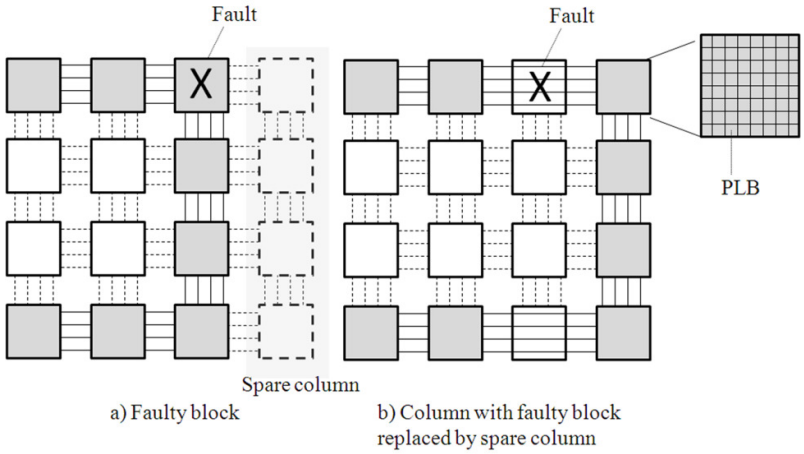


Figure 58. Howard, Tyrrel and Allison (1994)'s "blocking" mechanism

In Figure 58, the programmable logic blocks are grouped in larger blocks. When a fault is detected in one of the programmable logic blocks, the entire block is invalidated and bypassed. Long wires allow communication between non-adjacent blocks, which will be used in case of bypassing a faulty block. To tolerate the faulty blocks, alternate configurations are pre-computed and stored in memory. According to the faulty block position and the targeted layout, the appropriate alternate configuration is loaded from the memory.

The area overhead due to the addition of redundant blocks and routing elements is significant high when compared to redundancy in programmable logic blocks. The authors mention an increase of 47% overhead of redundant programmable logic blocks when 3 spare rows and columns of blocks are added in a 128x128 array. On the other hand, if instead of adding entire blocks, only three spare rows and columns of programmable logic blocks were added, the area overhead would be around 5%. Moreover, there is an 18% routing overhead.

Despite the increase in area, the authors claim that the performance overhead due to the fault-tolerant mechanism is significantly reduced when compared to programmable logic block redundancy techniques. Furthermore, since the alternate configurations are pre-computed, the fault-tolerant strategy adds a minor impact on reconfiguration time.

6.3 Configuration-Level Fault Tolerance

Hanchek and Dutt (1996) also proposed a fault-tolerant technique for yield enhancement that works at the manufacture. However, they show that the technique can be changed to work at runtime. The main contributions of Hanchek and Dutt's technique are reconfiguration around faulty logic blocks (called cells) without the need to generate new routing maps and no need to add extra switches in the channel wiring. Nevertheless, additional wiring segments are required. With this technique, the authors proposed to eliminate the need to perform new routing every time a new faulty logic block is detected and to reduce area overhead by avoiding the addition of extra switches. The technique is based on the node-covering approach proposed in (DUTT and HAYES, 1992), and consists in adding a spare logic block, called cover cell, to replace the faulty one in a chain-like method, where the adjacent cell replaces the faulty one, which in turn is replaced by its adjacent, until the spare block is reached. The technique tolerates only faults in logic cells. Figure 59 illustrates the node-covering technique.

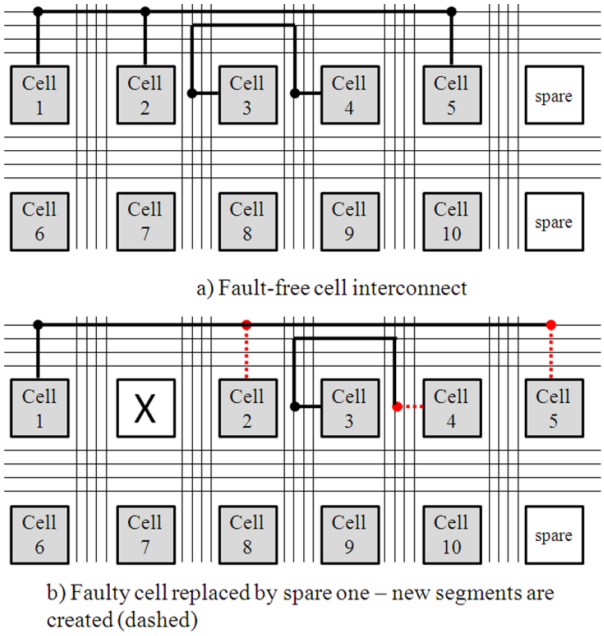


Figure 59. Hanchek and Dutt (1996)'s node-covering mechanism

To allow a cell to cover another cell, the cover cell must be able to replicate the functionality of the original one, as well as the connections to the rest of the array. Since the cells are identical, replicating the functionality is easily done through reconfiguration. Moreover, to replicate the connections, if possible, the fault-tolerant mechanism reuses wiring segments existent in the cover cell, otherwise, new cover segments must be added. In Figure 59.b, new segments were added (dashed lines) to allow the bypassing of the faulty cell and configure the remaining cells.

The authors present an analysis of a 16×16 array yield comparing the proposed technique with two other fault-tolerant techniques and the architecture without any fault-tolerant technique. The two other techniques are the spare column technique and the spare row/column technique. The results indicated a significant yield improvement

when the node-covering technique was used. Moreover, when comparing to the other fault-tolerant techniques, the node-covering presented a higher yield improvement with the same area overhead of the spare column technique and 50% less area overhead than the spare row/column technique.

Lach, Mangione-Smith and Potkonjak (1998) proposed to provide fault tolerance for time-constrained applications by eliminating the need of placement and routing before reconfiguration. The technique consists in dynamically tolerating faults by partially reconfiguring the FPGA to an alternate configuration that implements the same function while avoiding the faulty element. To ensure timing constraints, Lach's strategy consists in partitioning the physical design into tiles with a reserved spare logic block in each tile, and reconfiguring only the tiles that present faulty resources. Each tile consists of a set of logic blocks and interconnection elements. By reconfiguring only a specific tile, they reduce reconfiguration time and therefore, are able to attain timing constraints. For local interconnections, interconnect faults are treated as a fault in the logic block connected to it. Figure 60 illustrates Lach's approach.

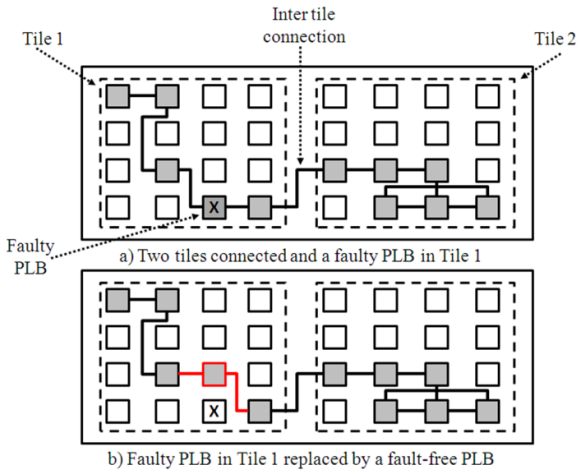


Figure 60. Tiling strategy proposed by (LACH, MANGIONE-SMITH and POTKONJAK, 1998)

In Figure 60.a, a logic block in tile 1 is faulty. Because this logic block is assigned with a function, it is necessary to replace it for another fault-free logic block and assign the same function. In Figure 60.b, a fault-free logic block replaces the faulty one.

To reconfigure around the faulty logic block, each tile has a set of alternate configurations, each one considering different fault location inside the tile. The alternate configurations are pre-computed and stored in the memory. Therefore, when a fault is detected the configuration that suits the fault location and interconnection constraints is loaded from memory. To replace the faulty logic block, spare logic blocks are added to the array. Lach's technique tolerates only one fault per tile and is limited to logic blocks.

To reconfigure around the faults, the technique relies on run-time reconfiguration. However, since all the configurations are pre-computed, the reconfiguration overhead is low and requires only loading the appropriate configuration from memory. Moreover, in this technique it is not necessary to reconfigure the entire FPGA, once the configuration

is specific to the faulty tile. This significantly reduces the reconfiguration time overhead.

According to the authors, experimental results using a 6x2 tiles XILINX 4000 FPGA presented a timing overhead between 14% to 45%, and a mean area overhead of 5.4%. Moreover, the design with the mentioned overheads presented 98% reliability against less than 1% reliability without the tile technique.

Lach, Mangione-Smith and Potkonjak (1999) presented an extended approach to handle faults in interconnection. The approach is the same applied to handle faults in logic blocks. Firstly, some routing resources are added as spare. Alternate configurations previously computed and stored are used to replace the original configuration. The alternate configurations have the same function implemented using different routing and logic elements. In this way, the faulty resources can be avoided. The drawback of the approach is tolerating only a limited amount of faults in each tile.

Although reducing reconfiguration overhead by using pre-computed configurations is an important contribution of the proposed fault-tolerant solutions, the main drawback of this strategy is related to memory requirements and fault tolerance efficiency. To cover all possible faults in all possible locations, a large amount of configurations must be generated, consequently requiring a large memory to store all the configurations. If the memory is limited, the only solution consists in limiting the number of faults that can be tolerated, consequently reducing the efficiency of the technique.

To cope with that, (ABRAMOVICI, STROND, HAMILTON, *et al.*, 1999), (ABRAMOVICI, EMMERT and STROUD, 2001) and (EMMERT, STROUD and ABRAMOVICI, 2007) proposed an on-line test, diagnosis and fault tolerance mechanism to handle faults that occur during circuit lifetime. To work during circuit usage, the authors proposed a mechanism that works in part of the circuit that is off-line, while the rest of the circuit continues its normal operation. In this way, they can reduce the reconfiguration overhead since it is still possible to perform operations in other parts of the circuit. The area that remains off-line and is tested and reconfigured is called self-testing area (STAR). When the test and reconfiguration is completed, the mechanism "roves" to another area and performs the same test, diagnostic and reconfiguration until the whole circuit has been scanned. The fault-tolerant strategy is illustrated in Figure 61.

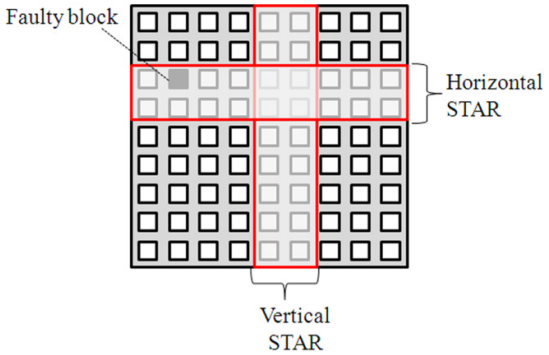


Figure 61. Roving STAR area approach proposed by (ABRAMOVICI, STROND, HAMILTON, *et al.*, 1999)

Figure 61 illustrates the STARs roving across the FPGA to test and tolerate faults. In each step, the STARs are tested by a BIST-like approach and the faulty resources are tolerated using two different approaches. The first approach consists in using spare resources to replace faulty ones. The amount and distribution of spares is based on logic block utilization. Depending on the application utilization, the spare logic blocks are distributed in a way that each logic block has at least one adjacent spare block or each spare block is located no more than one logic block away. The second approach is called *partially usable block* and consists in using partially faulty logic elements. The faulty elements are tested in all possible operations and if there are operations that can be performed correctly in spite of the fault, these logic elements are still reused. The same strategy is also used for interconnection elements. If an interconnect resource is faulty, than it is possible to route through this faulty segment.

To provide a more efficient fault tolerance, besides pre-computed alternate configurations, configurations can also be generated during run-time. According to the authors, this approach does not affect system operation because the configurations are computed for off-line resources, while the rest of the system is operating.

According to the authors, the programmable logic blocks reserved as spares and interconnects required for the BIST technique impose an area overhead that is dependent on the total FPGA area and decreases when the area increases. For an FPGA with 20x20 programmable logic blocks, the overhead is around 19%. On the other hand, for a 40x40 FPGA the area overhead is 10%. To evaluate the performance overhead, the proposed technique was implemented in an ORCA 2C series FPGA (LATTICE SEMICONDUCTOR CORPORATION, 2012) and the performance was obtained by evaluating individual benchmarks. In (ABRAMOVICI, EMMERT and STROUD, 2001), performance comparisons between the approach with and without pre-allocated spares are presented. For the tested benchmarks, the pre-allocated spares presented between 2.5% and 15.1% higher operating frequency than the non-allocated spares. However, performance comparisons between the FPGA with and without the proposed technique are not presented. Since roving from one area to another it is necessary to stop the clock, the authors also presented an analysis of the total time consumed by the clock interruptions, which is around 6.25% of the entire time. In relation to the fault-tolerant technique, the authors do not present analysis of the fault tolerance approach or reliability improvements.

Lakamraju and Tessier (2000) proposed a fault-tolerant technique targeted to cluster-based FPGA architectures. In such FPGAs, clusters are composed of pairs of logic blocks (in this case look up tables - LUTs) and flip-flops, called basic logic elements (BLEs). Such as blocks in (HOWARD, TYRREL and ALLISON, 1994) and tiles in (LACH, MANGIONE-SMITH and POTKONJAK, 1998), the main goal of cluster-based FPGAs is reducing configuration overhead, since the clusters can be manipulated individually. The technique proposed by Lakamraju and Tessier tolerates faults in logic clusters or global interconnection, and depending on the fault location, a different strategy is used. For faults in logic clusters, the main solution consists in remapping some (or all) the LUT's inputs to unused inputs in the same LUT. If this is not possible, the entire BLE is replaced by a spare one in the same cluster. Moreover, BLE replacement is also used when the fault affects resources outside the clusters, but still affects the clusters input, such as external input/output wiring and multiplexers. Figure 62 illustrates the BLE swap strategy, where the faulty BLE is replaced by the spare one. The strategy consists in simply using the multiplexer to select the spare BLE.

In case of global interconnections, the strategy consists in re-routing to an alternate route that still provides connection to the cluster. A special router is used in this strategy. To reduce timing overhead caused by the rerouting strategy, the technique uses information from the original circuit to create the new routes.

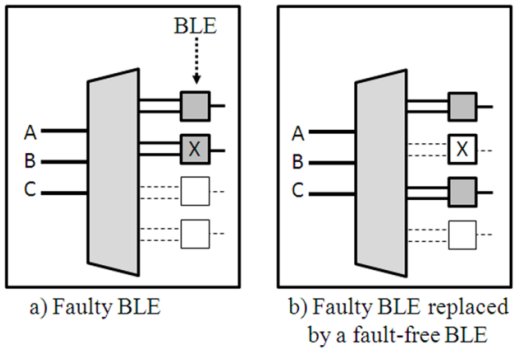


Figure 62. Lakamraju and Tessier (2000)'s BLE swapping strategy

According to the authors, fault injection tests applied in devices with 4 BLEs per cluster demonstrated that it was possible to recover from almost all the 10,000 cases of random single-fault injection. Moreover, the results also showed that the technique could cope with 500 simultaneous interconnection faults. To achieve these results, the technique presented a 20% area overhead for 4 BLEs/cluster when added two more cluster inputs and a spare BLE in each cluster. This overhead was reduced to 8% when the amount of BLEs per cluster increased to 8.

Sinha, Kamarchik and Goldstein (2000) proposed a fault-tolerant strategy to tolerate faults that occurs in a coarse-grained reconfigurable architecture called PipeRench. Since PipeRench is also used as case study of this work, a more detailed description about the architecture and the fault tolerance strategy can be found in chapter 5. In summary, PipeRench consists of several processing elements connected to each other to form pipeline stages. The physical pipeline stages, called stripes, are combined with fast partial dynamic reconfiguration to provide hardware virtualization. A fault in a processing element is tolerated by setting the entire stripe as faulty. To bypass the stripe, the mechanism utilizes the register file of the processing elements. Additionally, each register file contains a spare register.

Since the fault tolerance strategy implies in reducing the amount of available resources, the authors evaluated the performance degradation in two situations: 1) when the fault affects a processing element. In this case, the entire stripe is considered faulty and the fault-tolerant mechanism has to bypass this stripe. According to the authors, the performance degradation is $1/N_P$. Where N_P is the number of available physical stripes that remains after the fault tolerance mechanism bypasses the faulty ones. 2) when the fault occurs in the interconnect line that connects the stripes. In this case, a performance degradation of $3/N_P$ is perceived. The authors do not present any discussion about the efficiency of the fault tolerance strategy and system reliability.

6.4 The Proposed Approach

In this work, we address the particular drawbacks of the aforementioned approaches by proposing an architectural analysis to improve fault tolerance in general architectures.

Together with this analysis, we also present a fault-tolerant mechanism to increase reliability of a coarse-grained reconfigurable architecture. The main advantages of the proposed solution are:

- Low time and area overhead fault-tolerant strategy: the strategy to tolerate permanent faults in the architecture was designed to cause the minimum impact on performance and area. This was possible because the strategy was included in the on-line configuration mechanism that transparently generates the configuration. Moreover, the mechanism does not require additional spare resources.
  - As opposed to (HATORI, SAKURAI, NOGAMI, *et al.*, 1993), (HOWARD, TYRREL and ALLISON, 1994) and (HANCHEK and DUTT, 1996), our proposed fault tolerance approach is targeted to cope not only with yield enhancement but also with aging effects.
  - As opposed to (LACH, MANGIONE-SMITH and POTKONJAK, 1998), (LACH, MANGIONE-SMITH and POTKONJAK, 1999) and (LAKAMRAJU and TESSIER, 2000), our proposed solution generates the configuration during run-time. Therefore, it is not necessary to rely on pre-computed configurations that do not comprise all possible faults, and require a large amount of memory.
  - As opposed to (ABRAMOVICI, STROND, HAMILTON, *et al.*, 1999), our approach does not require turning off part of the device to apply the fault-tolerant strategy and stop the clock. Our fault-tolerant strategy works at run-time and does not add time overhead to the configuration mechanism.
  - As opposed to all fine-grained solutions, our approach uses a coarse-grained architecture, which reduces the time overhead required to reconfigure the system.

In Table 8, we summarize the works in fault-tolerant reconfigurable architectures mentioned along this chapter. The works are categorized according to the reconfiguration time, if it was proposed to yield enhancement or reliability increase, the part of the architecture where the fault tolerance is applied, and the granularity. It is important to highlight that all the mentioned works address manufacturing defects and/or permanent faults, which is the same type of faults addressed by the proposed work.

Table 8. Reconfigurable architecture fault-tolerant solutions

	Fault Type		Reconfiguration time		Goal		Target		Granularity
	Manufacturing defect	Permanent Fault	Static	Dynamic	Yield	Reliability	Functional Unit	Interconnect	
(HATORI, SAKURAI, NOGAMI, *et al.*, 1993)	x		x		x		X		Fine-grained
(HOWARD, TYRREL and ALLISON, 1994)	x		x		x		X		Block-based
(HANCHEK and DUTT, 1996)	x[1]		x		x		X		Fine-grained
(LACH, MANGIONE-SMITH and POTKONJAK, 1998)		x		x		x		x	Tile-based
(LACH, MANGIONE-SMITH and POTKONJAK, 1999)		x		x		x		x	Fine-grained
(ABRAMOVICI, STROND, HAMILTON, *et al.*, 1999)		x		x[2]		x	X	x	Fine-grained
(LAKAMRAJU and TESSIER, 2000)		x		x		x	x	x	Cluster-based
(SINHA, KAMARCHIK and GOLDSTEIN, 2000)		x		x		x	x	x	Coarse-grained
LOwER-FaT Array	**x**	**x**		**x[2]**	**x**	**x**	**x**	**x**	**Coarse-grained**

[1]Hanchek and Dutt proposed a solution to yield enhancement. However, they also mention the possibility of modifying the approach and using dynamic reconfiguration to tolerate permanent faults.

[2]In all dynamic approaches, the alternate configurations are generated statically. The exceptions are Abramovici's approach and LOwER-FaT array. The former, besides pre-computed configurations, also generates configuration at run-time. The latter, generates all configurations at run-time.

7 CONCLUSIONS

In this work, the design of fault-tolerant reconfigurable architectures targeted to increase reliability of future reduced feature size devices has been investigated. In order to do that, we have presented a reliability analysis based on a mathematical model that hierarchically connects each resource as a set of series and parallel subsystems. The reliability modeling is a powerful mechanism to evaluate the most critical resources to system reliability and provides a better direction towards fault tolerance investments and design modifications to improve reliability.

7.1 Summary of Contributions

In this section, the main contributions of the research work started in 2008 are presented.

7.1.1 Reliability Model of Reconfigurable Architectures

In this work, we have performed a comprehensive reliability analysis of two reconfigurable architectures.

To perform the analyses, we have used a mathematical model together with the block diagram representation to describe the reconfigurable architectures. In the representation, each individual component is determined by a reliability function, called exponential failure law. Additionally, the components relate to each other forming subsystems that are hierarchically connected among each other.

The analyses presented in this work demonstrated the validity of using this reliability model to represent reconfigurable architectures and to evaluate design modifications and fault tolerance solutions in order to improve reliability.

7.1.2 Interconnection

From the performed reliability analyses, we have found that the communication among functional units is the part of the architecture with highest impact on reliability.

In terms of reliability model, the functional units are completely dependent on the elements that allow data moving throughout the architecture. The analyses have demonstrated that this is true not only for communication based on interconnection model, but also when the communication is provided by register files, which is the case when the register files are directly connected in each other.

This means that the advantage of having several identical functional units that can work independently and in case of a fault, having their operations executed in another

functional unit, is diminished by the fact that the functional units have to rely on the communication elements to move data throughout the architecture.

7.1.3 Limits on Reliability Improvement

The analyses have also shown that there is a threshold in reliability improvement when hardware redundancy is used as solution.

This conclusion was made based on the fact that adding hardware redundancy in the architecture, e.g. more multiplexers or more registers, directly affects the area. Since reliability is also function of area, if the subsystem where redundancy is being added has other subsystems or components that rely on it, the area increase may affect negatively on reliability. Furthermore, in many cases, to add redundancy implies in including extra components or indirectly increasing the area of other components that will become critical to reliability. For example, adding extra registers implies in increasing the amount of inputs in the multiplexers that select among the registers.

7.1.4 Ad hoc solution

The last conclusion based on the reliability analysis is related to the solutions to increase reliability.

The analyses have demonstrated that there is no a general solution to increase reliability in reconfigurable architectures. Since there are many different designs, each one with a specific interconnection model and configuration strategy, it is not possible to find one single solution that will fit in all architectures. Even considering the fact that most architectures share a common characteristic: having several identical functional units connected by an ample interconnection model.

Nevertheless, one conclusion is shared among the solutions analyzed: the communication among functional units is the critical point to reliability. Therefore, one should concentrate the investments in this part of the architecture.

7.1.5 Fault-Tolerant Reconfigurable Architecture

In the context of this thesis, we have proposed a fault tolerance approach to be implemented in a reconfigurable architecture used as case study.

The LOwER-FaT (**L**ow **O**verhead **w**ithout **E**xtra **R**edundancy **Fa**ult-**T**olerant) mechanism transparently and at run-time generates the configuration considering only fault-free resources, avoiding the faulty ones. Moreover, to avoid increasing time and area overheads, the fault-tolerant strategy was kept as simple as possible. In the strategy, we exploited the regularity characteristic of the reconfigurable architecture and used the replicated resources as spare. Therefore, there is no need to add extra resources. For this reason, the number of available resources decreases as the number of fault resources increase, consequently introducing a performance penalty. Moreover, to avoid faulty functional units, the proposed strategy requires only the elimination of the faulty units, preserving all the good units. For faulty multiplexers, the approach proposes the elimination of functional units only if the faulty multiplexer selects one of the inputs to the respective unit. For all other cases, it is possible to bypass the multiplexer in an attempt to find another path between input and output.

The fault-tolerant approach was implemented in the reconfigurable architecture simulator and fault injection simulations were performed to evaluate the performance degradation in function of the fault rate. The results presented a mean performance degradation of 14.75% in the execution of benchmarks under a 20% fault rate. This

result demonstrated that, in spite of the high fault rate, the LOwER-FaT array is still able to execute all applications and, with exception of one application, accelerate execution when compared to the processor's execution.

7.2 Proposed Research Topics for Future works

In this section, we discuss some possible research topics that can be addressed as continuation to this work.

7.2.1 Investigating different interconnect mechanisms

The investigated solutions demonstrated that the communication mechanisms achieved high reliability only when a large amount of redundancy was applied. Consequently, significant area overhead was introduced in the solutions. Therefore, a potential research topic is the investigation of other interconnection models in order to find better solutions to improve system reliability.

7.2.2 Extending reliability analysis to FPGAs

Because of the particular characteristics of FPGAs, extending the reliability analysis to these fine-grained reconfigurable architectures requires considerations about the functional units and interconnects.

The functional units in FPGAs, usually called programmable logic blocks, are composed of lookup tables (LUTs), which are 1-bit memory elements that implement truth tables where the combination of inputs generates an output based on the logic implemented by the truth tables (BROWN, FRANCIS, ROSE, *et al.*, 1992). Because of this particular characteristic, first it is necessary to evaluate how to model the reliability of the lookup tables, as well as the reliability of the programmable logic blocks, which have several lookup tables and flip-flops. To connect all the look-up tables inside a logic block as well as all logic blocks, FPGAs present a dense and complex interconnection model that comprises most of the circuit's area (BROWN, FRANCIS, ROSE, *et al.*, 1992). Extending the reliability model to this type of interconnect also deserves a previous analysis to take into consideration the specific characteristics of this interconnection model and the solutions to increase reliability.

7.2.3 Extending reliability analysis to other high performance architectures

Differently from reconfigurable architectures, architectures such as VLIWs and super-scalars do not present such a complex interconnection model. This leads to some questions about what are the critical elements to reliability and what is the reliability degree of these architectures. These answers would help not only to make the appropriate investment to increase reliability considering the critical elements, but also could give some indications about what kind of architecture should be considered in a design targeted to reliability.

7.2.4 Extending reliability analysis to MPSoCs with Reconfigurable Architectures

Another advantage that the aggressively scaling has brought is related to the transistor density increase and consequently the higher integration capability of processors in one chip (HILL and MARTY, 2008). However, such as reconfigurable architectures, multiprocessor system-on-chip (MPSoCs) is also susceptible to the same reliability problems already discussed in this work, that are also a consequence of scaling.

Reconfigurable architectures and MPSoCs have in common the fact that both architectures have large amount of identical processing elements and a complex interconnection model. However, while reconfigurable architectures are at ALU or LUT granularity, MPSoCs are at processor level.

Therefore, one can extend the reliability assessment presented in this work to investigate reliability of MPSoCs. In (PEREIRA and CARRO, 2011), we have presented a comparison between LOwER-FaT array and a MIPS-based MPSoC with area equivalence. In the comparison, we show that the reconfigurable architecture sustains performance even under a 20% fault rate, while all the cores of the MPSoC fail under 15% fault rate. For this reason, solutions to improve fault tolerance in MPSoCs are necessary. To allow a comprehensive study of possible solutions to improve MPSoC`s reliability, an analysis of different networks-on-chip (NoCs) should be performed. The analysis should take into consideration topology, routing algorithm, network traffic, etc.

7.2.5 Extending LOwER-FaT array fault tolerance strategy to fine-grained reconfigurable architectures

The main advantages of the fault tolerance strategy proposed in this work are the run-time replacement of faulty resources and the elimination of faulty functional units without the elimination of fault-free ones. Therefore, the fault tolerance approach can be extended to FPGAs, by using the partial dynamic reconfiguration capability available in commercial devices.

In (PEREIRA, BRAUN, HÜBNER, *et al.*, 2011), a collaboration with Karlsruhe Institute of Technology research group proposed the combination of the fault tolerance approach proposed in this work and a run-time dynamic resource instantiation technique proposed by (BRAUN and BECKER, 2010). However, the theoretical work did not present any quantitative analysis. Therefore, this work can be extended to FPGAs taking into consideration aspects such as area, performance, reliability, etc.

8 PUBLICATIONS

8.1 Journals

PEREIRA, M. M.; CARRO, L. Dynamic Reconfigurable Computing: The Alternative to Homogeneous Multicores under Massive Defect Rates. **International Journal of Reconfigurable Computing**, 2011. 17p.

FERREIRA, R.; BUENO, C.; LAURE, M.; PEREIRA, M. M.; CARRO, L. A Dynamic Reconfigurable Super-VLIW Architecture for a Fault Tolerant Nanoscale Design. **Transactions on High-Performance Embedded Architectures and Compilers**, 2011. 18p.

8.2 Conferences, Symposia and Workshops

PEREIRA, M. M.; CARRO, L. **Using a Reliability Analysis to Design Low Cost Reliable-Aware Coarse-Grained Reconfigurable Architectures.** 4th Workshop on Design for Reliability (DFR). Paris, 2012.

PEREIRA, M. M. et al. **Run-time Resource Instatiation for Fault Tolerance in FPGAs**. 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS). 2011. p. 88 - 95.

FERREIRA, R.; VENDRAMINI, J. G.; MUCIDA, L.; PEREIRA, M. M.; CARRO, L. **An FPGA-based heterogeneous coarse-grained dynamically reconfigurable architecture**. Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems (CASES), Taipei: ACM, 2011.

FERREIRA, R. ; BUENO, C. ; LAURE, M. ; PEREIRA, M. M.; CARRO, L. **A Dynamic Reconfigurable Super-VLIW Architecture for a Fault Tolerant Nanoscale Design**. Proceedings of 4th HiPEAC Workshop on Reconfigurable Computing (WRC), Pisa., 2010. p. 7-16.

PEREIRA, M. M.; CARRO, L. **Dynamic Reconfigurable Computing: the Alternative to Homogeneous Multicores under Massive Defect Rates**. Proceedings of the 5th International Workshop on Reconfigurable Communication-centric Systems on Chip (ReCoSoC), Karlsruhe. 2010.

PEREIRA, M. M.; BRAUN, L. ; HÜBNER, M. ; BECKER, J.; CARRO, L. **Use of Partial Dynamic Reconfiguration for Fault- Tolerance Processor Design**. 1st Brazilian-German Workshop on Micro- and Nano- Electronics, Porto Alegre, 2010.

PEREIRA, M. M.; CARRO, L. **Dynamically Adapted Low-Energy Fault Tolerant Processors**. NASA/ESA Conference on Adaptive Hardware and Systems (AHS). 2009. p. 91-97.

PEREIRA, M. M.; LO, T.; CARRO, L. **A Self-adaptive Approach for Fault-Tolerance in Future Technologies**. The 1st HiPEAC Workshop on Design for Reliability (DFR). 2009.

PEREIRA, M. M.; CARRO, L. **A Self-Adaptive Approach to Increase Reliability of Processors**. Proceedings of the 17th IFIP/IEEE International Conference On Very Large Scale Integration (PhD Forum - VLSI-SoC'09), Florianópolis, 2009.

PEREIRA, M. M.; CARRO, L. **A Dynamic Reconfiguration Approach for Accelerating Highly Defective Processors**. Proceedings of the 17th IFIP/IEEE International Conference On Very Large Scale Integration (VLSI-SoC'09), Florianópolis, 2009.

REFERENCES

ABRAMOVICI, M.; BREUER, M. A.; FRIEDMAN, A. D. **Digital Systems Testing and Testable Design**. 1.ed. ed. Hoboken: Wiley-IEEE Press, 1994.

ABRAMOVICI, M.; EMMERT, J. M.; STROUD, C. E. Roving STARs: An Integrated Approach to On-Line Testing, Diagnosis, and Fault Tolerance for FPGAs in Adaptive Computing Systems. PROCEEDINGS OF THIRD NASA/DoD WORKSHOP ON EVOLVABLE HARDWARE. [S.l.]: IEEE Computer Society Press. 2001. p. 73-92.

ABRAMOVICI, M.; STROND, C.; HAMILTON, C.; WIJESURIYA, S. et al. Using Roving STARs for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications. PROCEEDINGS OF INTERNATIONAL TEST CONFERENCE. [S.l.]: IEEE Computer Society Press. 1999. p. 973-982.

ADAMS, G. B. I.; AGRAWAL, D. P.; SIEGEL, H. J. A Survey and Comparison of Fault-Tolerant Multistage Interconnection Networks. **Computer**, Jun. 1987. 14-27.

ALTERA, 2012. Available at: <http://www.altera.com/>.

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic Concepts and Taxonomy of Dependable and Secure Computing. **IEEE Transactions on Dependable and Secure Computing**, Jan.-Mar 2004. 11-33.

BECK, A. C. S.; CARRO, L. **Dynamic Reconfigurable Architectures and Transparent Optimization Techniques:** Automatic Acceleration of Software Execution. 1.ed. ed. Dordrecht/Heidelberg: Springer, 2010.

BECK, A. C. S.; RUTZIG, M. B.; GAYDADJIEV, G.; CARRO, L. Transparent Reconfigurable Acceleration for Heterogeneous Embedded Applications. PROCEEDINGS OF THE DESIGN, AUTOMATION & TEST IN EUROPE. [S.l.]: IEEE Computer Society Press. 2008. p. 1208-1213.

BECK, A. C. S.; RUTZIG, M. B.; GAYDADJIEV, G.; CARRO, L. Run-Time Adaptable Architectures for Heterogeneous Behavior Embedded Systems. PROCEEDINGS OF THE 4TH INTERNATIONAL WORKSHOP ON RECONFIGURABLE COMPUTING: ARCHITECTURES, TOOLS AND APPLICATIONS. Berlin/Heidelberg: Springer-Verlag. 2008b. p. 111-124.

BENES, V. E. Mathematical Theory of Connecting Networks and Telephone Traffic. **Academic Press**, 1965. 319.

BORKAR, S. Microarchitecture and Design Challenges for Gigascale Integration. Keynote address, 37TH ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE. Los Alamitos: IEEE Computer Society Press. 2004. p. 3.

BORKAR, S. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. **IEEE Micro**, Los Alamitos, Nov.-Dec. 2005. 10-16.

BRAUN, L.; BECKER, J. Two-Dimensional Dynamic Multigrained Reconfigurable Hardware. PROCEEDINGS OF THE 2010 IEEE ANNUAL SYMPOSIUM ON VLSI. Washington, DC: IEEE Computer Society Press. 2010. p. 475-476.

BROWN, S. D.; FRANCIS, R. J.; ROSE, J.; VRANESIC, Z. G. **Field-Programmable Gate Arrays**. Dordrecht: Kluwer Academic Publishers Group, v. 180, 1992.

CHOU, Y.; PILLAI, P.; SCHMIT, H.; SHEN, J. P. PipeRench Implementation of the Instruction Path Coprocessor. PROCEEDINGS OF 33RD ANNUAL ACM/IEEE INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE. New York: ACM. 2000. p. 147-158.

CLOS, C. A Study of Non-Blocking Switching Networks. **Bell System Technical Journal**, Mar. 1953. 406-424.

COMPTON, K.; HAUCK, S. Reconfigurable Computing: A Survey of System and Software. ACM COMPUTING SURVEYS. New York: ACM. 2002. p. 171-210.

CONSTANTINESCU, C. Trends and challenges in VLSI circuit reliability. PROCEEDINGS OF THE 36TH ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE. Washington, DC: IEEE Computer Society Press. 2003. p. 14-19.

DEHON, A.; NAEIMI, H. Seven Strategies for Tolerating Highly Defective Fabrication. **IEEE Design & Test**, Los Alamitos, Jul.-Aug. 2005. 306-315.

DUTT, S.; HAYES, J. P. Some practical issues in the design of fault-tolerant multiprocessors. **IEEE Transactions on Computers**, Los Alamitos, May 1992. 588-598.

EMMERT, J. M.; STROUD, C. E.; ABRAMOVICI, M. Online Fault Tolerance for FPGA Logic Blocks. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, Los Alamitos, Feb. 2007. 216-226.

EPSMA. **Guidelines to Understanding Reliability Prediction**. European Power Supply Manufacturers Association. [S.l.], p. 29. 2005.

ERICSON, C. Fault Tree analysis - A History. PROCEEDINGS OF THE 17TH INTERNATIONAL SYSTEM SAFETY CONFERENCE. [S.l.:s.n.]. 1999.

FAJARDO, J. J.; RUTZIG, M. B.; BECK, A. C. S.; CARRO, L. Towards an Adaptable Multiple-ISA Reconfigurable Processor. PROCEEDINGS OF THE 7TH INTERNATIONAL CONFERENCE ON RECONFIGURABLE COMPUTING: ARCHITECTURES, TOOLS AND APPLICATIONS. Berlin/Heidelberg: Springer-Verlag. 2011.

FERREIRA, R.; BUENO, C.; LAURE, M.; PEREIRA, M. M. et al. A Dynamic Reconfigurable Super-VLIW Architecture for a Fault Tolerant Nanoscale Design. **Transactions on HiPEAC: Volume 5, Issue 3**, 2011. 18.

FERREIRA, R.; VENDRAMINI, J. G.; MUCIDA, L.; PEREIRA, M. M. et al. An FPGA-Based Heterogeneous Coarse-Grained Dynamically Reconfigurable Architecture. PROCEEDINGS OF THE 14TH INTERNATIONAL CONFERENCE

ON COMPILERS, ARCHITECTURES AND SYNTHESIS FOR EMBEDDED SYSTEMS. New York: ACM. 2011. p. 195-204.

GALEY, J. M.; NORBY, R. E.; ROTH, J. P. Techniques for the Diagnosis of Switching Circuit Failures. PROCEEDINGS OF THE SECOND ANNUAL SYMPOSIUM ON SWITCHING CIRCUIT THEORY AND LOGICAL DESIGN (SWCT). [S.l.]: IEEE Computer Society Press. 1961. p. 152-160.

GOKE, L. R. Banyan Networks for Partitioning Multiprocessor Systems. PROCEEDINGS OF THE 1ST ANNUAL SYMPOSIUM ON COMPUTER ARCHITECTURE. New York: ACM. 1973. p. 21-28.

GUTHAUS, M. R.; RINGENBERG, J. S.; ERNST, D.; AUSTIN, T. M. et al. MiBench: A Free Commercially, Representative Embedded Benchmark Suite. PROCEEDINGS OF THE IEEE INTERNATIONAL WORKSHOP ON WORKLOAD CHARACTERIZATION. Los Alamitos: IEEE Computer Society Press. 2001. p. 3-14.

HANCHEK, F.; DUTT, S. Node-Covering Based Defect and Fault Tolerance Methods for Increased Yield in FPGAs. PROCEEDINGS OF 9TH INTERNATIONAL CONFERENCE ON VLSI DESIGN. Washington, DC: IEEE Computer Society Press. 1996. p. 225-229.

HARTENSTEIN, R. A Decade of Reconfigurable Computing: A Visionary Restrospective. PROCEEDINGS OF THE CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE. Piscataway: IEEE Press. 2001. p. 642-649.

HATORI, F.; SAKURAI, T.; NOGAMI, K.; SAWADA, K. et al. Introducing Redundancy in Field Programmable Gate Arrays. PROCEEDINGS OF THE IEEE 1993 CUSTOM INTEGRATED CIRCUITS CONFERENCE. Washington, DC: IEEE Computer Society Press. 1993. p. 711-714.

HEIJMEN, T. **Radiation Induced Soft Errors in Digital Circuits: A Literature survey**. Philips Electronics Nederland - Report 2002/828. Netherlands, p. 93. 2002.

HILL, M. D.; MARTY, M. R. Amdahl's Law in the Multicore Era. **Computer**, Washington, DC, Jul. 2008. 33-38.

HOWARD, N. J.; TYRREL, A. M.; ALLISON, N. M. The Yield Enhancement of Field-Programmable Gate Arrays. **IEEE Transactions on Very Large Scale Integration**, Washington, DC, Mar. 1994. 115-123.

INTEL. Intel Expands Customer Choice with First Configurable Intel® AtomTM-based Processor, 2010. Available at: <http://newsroom.intel.com/community/intel_newsroom/blog/2010/11/22/intel-expands-customer-choice-with-first-configurable-intel-atom-based-processor>. Accessed in: January 2012.

INTEL. Intel Solid-State Drive 320 Series Product Specification, 2012. Available at: <http://www.intel.com/content/www/us/en/solid-state-drives/ssd-320-specification.html>. Accessed in: January 2012.

ITRS. **International Roadmap for Semiconductors: Executive Summary**. [S.l.]. 2011.

ITRS. **International Roadmap for Semiconductors: Process Integration Devices and Structures**. [S.l.]. 2011b.

KAGOTANI, H.; SCHMIT, H. Asynchronous PipeRench: Architecture and Performance Evaluations. 11TH ANNUAL IEEE SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES. Washington, DC: IEEE Computer Society Press. 2003. p. 121-129.

KANE, G.; HEINRICH. **MIPS RISC Architecture**. [S.l.]: Prentice Hall, 1992.

KUO, W.; ZUO, M. J. **Optimal Reliability Modeling:** Principles and Applications. Hoboken: John Wiley & Sons, Inc, 2003.

LACH, J.; MANGIONE-SMITH, W. H.; POTKONJAK, M. Efficiently Supporting Fault-Tolerance in FPGAs. PROCEEDINGS OF THE 1998 ACM/SIGDA SIXTH INTERNATIONAL SYMPOSIUM ON FIELD PROGRAMMABLE GATE ARRAYS. New York: ACM. 1998. p. 105-115.

LACH, J.; MANGIONE-SMITH, W.; POTKONJAK, M. Algorithms for Efficient Runtime Fault Recovery on Diverse FPGA Architectures. PROCEEDINGS OF 1999 IEEE INTERNATIONAL SYMPOSIUM ON DEFECT AND FAULT TOLERANCE IN VLSI SYSTEMS. Washington, DC: IEEE Computer Society Press. 1999. p. 386-394.

LAKAMRAJU, V.; TESSIER, R. Tolerating Operational Faults in Cluster-Based FPGAs. PROCEEDINGS OF THE 2000 ACM/SIGDA EIGHTH INTERNATIONAL SYMPOSIUM ON FIELD PROGRAMMABLE GATE ARRAYS. New York: ACM. 2000. p. 187-194.

LAPRIE, J.-C. Dependable Computing and Fault Tolerance: Concepts and Terminolgoy. THE TWENTY-FIFTH INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING. Los Alamitos: IEEE Computer Society Press. 1996. p. 2.

LATTICE SEMICONDUCTOR CORPORATION. ORCA Series 2 FPGA, 2012. Available at: <http://www.latticesemi.com/products/maturedevices/orcaseries2fpga.cfm>. Accessed in: January 2012.

LAWRIE, D. H. Access and Alignment of Data in an Array Processor. **IEEE Transactions on Computer**, Los Alamitos, Dec. 1975. 1145-1155.

LEE, I.; BASOGLU, M.; SULLIVAN, M. **Survey of Error and Fault Detection Mechanisms**. The University of Texas at Austin. Austin, p. 24. 2011. (TR-LPH-2011-002).

LO, T. B.; BECK, A. C. S.; RUTZIG, M. B.; CARRO, L. A Low-Energy Approach for Context Memory in Reconfigurable Systems. PROCEEDINGS OF THE 2010 IEEE INTERNATIONAL SYMPOSIUM ON PARALLEL & DISTRIBUTED PROCESSING - WORKSHOP AND PHD FORUM. Washington, DC: IEEE Computer Society Press. 2010. p. 1-8.

MATHWORKS. MATLAB - The Language Of Technical Computing, 2012. Available at: <www.mathworks.com/products/matlab/>. Accessed in: January 2012.

MENTOR GRAPHICS. LeonardoSpectrum, 2012. Available at: <http://www.mentor.com/products/fpga/synthesis/leonardo_spectrum/>. Accessed in: January 2012.

MISHRA, M.; GOLDSTEIN, S. Defect Tolerance at the End of the Roadmap. PROCEEDINGS OF IEEE INTERNATIONAL TEST CONFERENCE. Washington, DC: IEEE Computer Society Press. 2003. p. 1201-1210.

PAUL, B. C.; KUNHYUK, K.; KUFLUOGLU, H.; ALAM, M. A. et al. Impact of NBTI on the temporal performance degradation of digital circuits. **IEEE Electron Device Letters**, Washington, DC, Aug. 2005. 560 - 562.

PEREIRA, M. M.; BRAUN, L.; HÜBNER, M.; BECKER, J. et al. Run-time Resource Instatiation for Fault Tolerance in FPGAs. PROCEEDINGS OF THE 2011 NASA/ESA CONFERENCE ON ADAPTIVE HARDWARE AND SYSTEMS. Los Alamitos: IEEE Computer Society Press. 2011. p. 88 - 95.

PEREIRA, M. M.; CARRO, L. Dynamically Adapted Low-Energy Fault Tolerant Processors. PROCEEDINGS OF THE 2009 NASA/ESA CONFERENCE ON ADAPTIVE HARDWARE AND SYSTEMS. Los Alamitos: IEEE Computer Society Press. 2009. p. 91-97.

PEREIRA, M. M.; CARRO, L. Dynamic Reconfigurable Computing: The Alternative to Homogeneous Multicores under Massive Defect Rates. **International Journal of Reconfigurable Computing**, New York, 2011. 17.

PEREIRA, M. M.; LO, T.; CARRO, L. A Self-adaptive Approach for Fault-Tolerance in Future Technologies. In: THE 1ST HiPEAC WORKSHOP ON DESIGN FOR RELIABILITY. Cyprus: [s.n.]. 2009. p. 10.

PETERSON, W. W.; WELDON, E. J. **Error-correcting codes**. Cambridge: The MIT Press, 1980.

PRADHAN, D. K. **Fault-Tolerant Computer System Design**. [S.l.]: Prentice Hall, 1996.

RUTZIG, M. B.; BECK, A. C. S.; CARRO, L. Balancing reconfigurable data path resources according to application requirements. PROCEEDINGS OF THE IEEE INTERNATIONAL SYMPOSIUM ON PARALLEL DISTRIBUTED PROCESSING. Los Alamitos: IEEE Computer Society Press. 2008. p. 1-8.

RUTZIG, M. B.; BECK, A. C. S.; CARRO, L. Dynamically Adapted Low Power ASIPs. PROCEEDINGS OF THE 5TH INTERNATIONAL WORKSHOP ON RECONFIGURABLE COMPUTING: ARCHITECTURES, TOOLS AND APPLICATIONS. Berlin/Heidelberg: Springer. 2009. p. 110-122.

RUTZIG, M. B.; BECK, A. C. S.; CARRO, L. CReAMS: An Embedded Multiprocessor Platform. PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON APPLIED RECONFIGURABLE COMPUTING. Berlin/Heidelberg: Springer. 2011. p. 118-124.

SCHMIT, H. Incremental Reconfiguration for Pipelined Applications. PROCEEDINGS OF THE 5TH ANNUAL IEEE SYMPOSIUM ON FPGAs FOR CUSTOM COMPUTING MACHINES. Los Alamitos: IEEE Computer Society Press. 1997. p. 47-55.

SCHMIT, H.; WHELIHAND, D.; TSAI, A.; MOE, M. et al. PipeRench: A Virtualized Programmable Datapath in 0.18 Micron Technology. PROCEEDINGS OF THE IEEE 2002 CUSTOM INTEGRATED CIRCUITS CONFERENCE. Los Alamitos: IEEE Computer Society Press. 2002. p. 63-66.

SCHUSTER, S. E. Multiple Word/Bit Line Redundancy for Semiconductor Memories. **IEEE Journal of Solid-State Circuits**, Washington, DC, Oct. 1978. 698 - 703.

SHI, K.; HOWARD, D. Challenges in Sleep Transistor Design and Implementation in Low-Power Designs. PROCEEDINGS OF THE 43RD ANNUAL CONFERENCE ON DESIGN AUTOMATION. New York: ACM. 2006. p. 113-116.

SINHA, S. K.; KAMARCHIK, P. M.; GOLDSTEIN, S. C. Tunable Fault Tolerance for Runtime Reconfigurable Architectures. PROCEEDINGS OF THE 2000 IEEE SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES. Washington, DC: IEEE Computer Society Press. 2000. p. 185-192.

SMITH, D. J. **Reliability, Maintainability and Risk:** Practical Methods for Enginners including Reliability Centred Maintenance and Safety-Related Systems. Oxford: Elsevier Butterworth-Heinemann, 2007.

SRINIVASAN, J.; ADVE, S. V.; BOSE, P.; RIVERS, J. A. The Impact of Technology Scaling on Lifetime Reliability. PROCEEDINGS OF THE 2004 INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS. [S.l.]: IEEE Computer Society Press. 2004. p. 177–186.

SRINIVASAN, J.; ADVE, S. V.; PRADIP, B.; RIVERS, J. A. Lifetime Reliability: Toward an Architectural Solution. **IEEE Micro**, Washington, DC, May-Jun. 2005. 70-80.

STOTT, E.; SEDCOLE, P.; CHEUNG, P. Fault Tolerant Methods for Reliability in FPGAs. PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS. Los Alamitos: IEEE Computer Society Press. 2008. p. 415-420.

STOTT, E.; SEDCOLE, P.; CHEUNG, P. Fault Tolerance and Reliability in Field Programmable Gate Arrays. **IET Computers & Digital Techniques**, May 2010. 196-210.

SYNOPSYS. Power Compiler , 2012. Available at: <http://www.synopsys.com/tools/implementation/rtlsynthesis/pages/powercompiler.aspx>. Accessed in: January 2012.

US DEPARTMENT OF DEFENSE. **Military Handbook - Reliability Prediction of Electronic Equipment**. Washington. 1986. (MIL-HDBK-217E).

VAHID, F.; STITT, G.; LYSECKY, R. Warp Processing: Dynamic Translation of Binaries to FPGA Circuits. **Computer**, Los Alamitos, Jul. 2008. 40-46.

VIGRASS, W. J. Calculation of Semiconductor Failure Rates. **Harris Semiconductor**, 2012. Available at: <http://rel.intersil.com/docs/index.html>. Accessed in: Jan. 2012.

WADSACK, R. L. Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits. **The Bell System Technical Journal**, 1978. 1449 -1474.

WEAVER, C.; AUSTIN, T. A Fault Tolerant Approach to Microprocessor Design. PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS. Los Alamitos: IEEE Computer Society Press. 2001. p. 411-420.

WEY, C.-L.; LOMBARDI, F. On the Repair of Redundant RAM's. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, Washington, DC, Mar. 1987. 222 - 231.

WHITE, M.; CHEN, Y. **Scaled CMOS Technology Reliability Users Guide**. Jet Propulsion Laboratory. Pasadena, p. 27. 2008.

WILTON, S. J. E.; JOUPPI, N. P. CACTI: an enhanced cache access and cycle time model. **IEEE Journal of Solid-State Circuits**, Washington, DC, May 1996. 677-688.

WU, C.-L.; FENG, T. Y. On a Class of Multistage Interconnection Networks. **IEEE Transactions on Computers**, Washington, DC, Aug. 1980. 694-702.

XILINX. FPGA, CPLD, and EPP Solutions from Xilinx, Inc. **Xilinx**, 2012. Available at: <http://www.xilinx.com/>. Accessed in: January 2012.

YAN, G.; HAN, Y.; XIAOWEI, L. A unified online Fault Detection scheme via checking of Stability Violation. PROCEEDINGS OF THE DESIGN, AUTOMATION & TEST IN EUROPE CONFERENCE & EXHIBITION. [S.l.]: IEEE Computer Society Press. 2009. p. 496 - 501.

APPENDIX A - RELIABILITY RESULTS

This appendix presents the reliability results for all cases analyzed in this work. The results are presented time (hours x 10^6) for two reliability values, 0.99999 and 0.96, and separated according to the respective section and figure legends.

4.5.2.1 Reference reconfigurable architecture

Figure 23. Reconfigurable array reliability - without fault tolerance

	0.99999	**0.96**
All Levels	0.000000008	0.00006
60% Levels	0.000002672	0.02100

4.5.2.2 LOwER-FaT array

Figure 24. LOwER-FaT array reliability

	0.99999	**0.96**
All Levels	0.86	9.50
60% Levels	0.86	9.50

Figure 25. Spare output multiplexer strategy

	0.99999	**0.96**
No Spares	0.86	9.5
30% Spares	21.34	89.8
50% Spares	41.42	130.4
80% Spares	78.51	192.2
100% Spares	101.07	215.3

Figure 26. Spare output multiplexer strategy – reliability

	0.99999	0.96
300% Spares	101.2	128.0
400% Spares	83.6	105.4
500% Spares	71.2	89.7
700% Spares	54.8	69.1

Figure 27. Spare output multiplexer strategy – justification

	0.99999	0.96
$R_{outputMuxes}$ - 100% Spares	101.1	225.0
$R_{inputMuxes}$ - 100% Spares	0.0005	3.0
$R_{outputMuxes}$ - 700% Spares	546.1	748.7
$R_{inputMuxes}$ - 700% Spares	0.0002	0.8

Figure 29. Spare input multiplexer strategy

	0.99999	0.96
No Spares	101.2	128.0
100% Spares	163.3	192.7
300% Spares	211.0	238.0
1500% Spares	246.0	282.0
3100% Spares	177.3	223.0

Figure 30. LOwER-FaT array reliability with spare multiplexers strategy – different technologies

	0.99999	0.96
90nm	211.0	238.0
32nm	133.7	153.2
18nm	74.5	85.3
11nm	48.1	55.0

4.5.2.3 Multistage Interconnection Network as Interconnection Model

Figure 41. 16:1 Multiplexer-based model *versus* 16-input Omega network with 3 extra stages

	0.99999	0.96
Multiplexer-based model1	0.00001001	0.034
Multiplexer-based model 2	61.01000000	75.000
MIN-based model	11.01000000	38.300

Figure 43. Multiplexer-based architecture versus *MIN-based*

	0.99999	**0.96**
MIN-based architecture	0.00110000	4.69
Multiplexer-based architecture	78.5000	192.1
Only MIN	18.9000	39.6
Only Multiplexer	78.5000	192.2

5.4.2 Reliability Analysis

Figure 53. PipeRench *versus* LOwER-FaT array

	0.99999	**0.96**
PipeRench - area equivalence	0.011	0.969
PipeRench - ALU equivalence	0.040	3.200
LOwER-FaT array	0.860	9.500

Figure 54. PipeRench reliability analysis

	0.99999	**0.96**
Register Files subsystem	0.011	0.969
Stripes subsystem	4.100	5.900
PipeRench - area equivalent	0.011	0.969

Figure 55. PipeRench reliability - spare registers

	0.99999	**0.96**
1 Spare	0.011	0.969
100% Spares	3.800	5.600
300% Spares	3.400	5.000
700% Spares	2.940	4.100

Figure 56. PipeRench without RF

	0.99999	**0.96**
PipeRench with RF	0.011	0.969
PipeRench without RF	1.632	1.784

APPENDIX B - LOWER-FAT ARRAY WITH REDUCED AREA

This appendix reproduces the reliability analysis for the LOwER-FaT array described in chapter 4, considering a architecture with smaller amount of functional units. The main goal of this study is to evaluate the reliability when the architecture presents a significant reduction in the amount of available units.

The architecture analyzed next is around five times smaller than the architecture analyzed in chapter 4. To differentiate between both architectures, the reduced one is called LAC II and the one evaluated in chapter 4 is called LAC I. The description in amount of functional units is presented in Table 1.

Table 1. Number of resources

	LAC I	LAC II
ALU	459	68
Load/Store	68	49
Multiplier	34	1
Input Multiplexer 16:1	1122	236
Output Multiplexer 16:1	816	576

The curves present next follow the same analysis presented in chapter 4. To allow a comparison between the two architectures, LAC I (presented in chapter 4) and LAC II (presented next), we designated the legend of the figures with the same description presented in the equivalent figure for LAC I architecture described in section 4.5.2.

Figure 1 presents the system reliability without fault tolerance approach. Figure 2 presents the curves for the LAC II when the fault tolerance mechanism is in place. Both figures illustrated the architecture when all the functional units are in use (All Levels) and when only 60% of the architecture is in use (60% Levels). The remaining curves reproduce the same analysis presented in section 4.5.2, regarding the addition of spare interconnection elements. In Figure 3, we evaluated reliability improvement when spare output multiplexers were added to the architecture. As can be observed, the reliability increase starts to saturate and from 80% the reliability starts to decrease. This behavior is also observed when the amount of spares is extrapolated to 300% up to 700%. This is the same behavior observed with the LAC I architecture, demonstrated in Figure 25 and Figure 26. The reason for this behavior is the fact that in order to increase the amount of output multiplexers, it is necessary to increase the area of the input multiplexers that have as input the output multiplexers (Figure 10).

Following the same analysis presented in section 4.5.2, the next step consists in adding spare input multiplexer in an attempt to increase reliability. However, as demonstrated with the LAC I architecture, the reliability increase also saturates. The reason for this is the fact that extra input multiplexers must be added to select between the original multiplexer and the spares (Figure 28). This is demonstrated in Figure 5, where reliability decreases after the addition of 3100% spare input multiplexers (total of 64 input multiplexers in each functional unit input).

Finally, the last analysis consists in evaluating the reliability when considering different technologies. Once again, as already demonstrated with LAC I architecture, reliability decreases when technology shrinks. This is demonstrated in Figure 6, considering 300% spare output multiplexers and 1500% spare input multiplexers (the same amount of spares used in LAC I architecture - Figure 30).

An overall analysis of the results demonstrates that, in spite of some different in the absolute results, the reliability curves for LAC II architecture behave the same way as the reliability curves for LAC I architecture. This result shows that, in spite of having a significant reduced amount of available resources, the ratio between the amount of interconnection elements and functional units remains the same. In other words, as long as the architecture relies on a large and complex interconnection model to send data throughout the architecture, this model will be the critical element to system reliability.

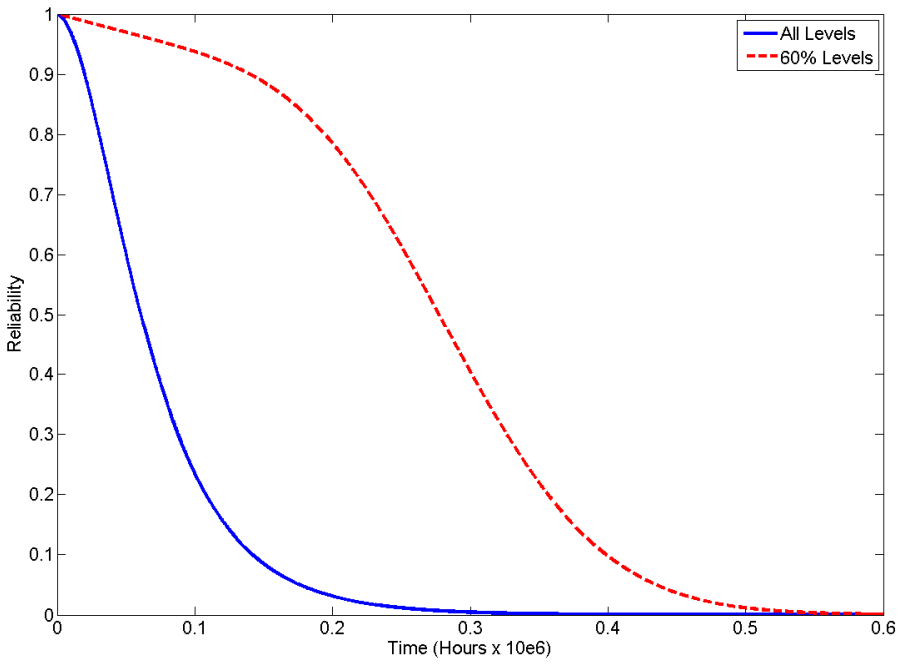


Figure 1. Reconfigurable array reliability - without fault tolerance

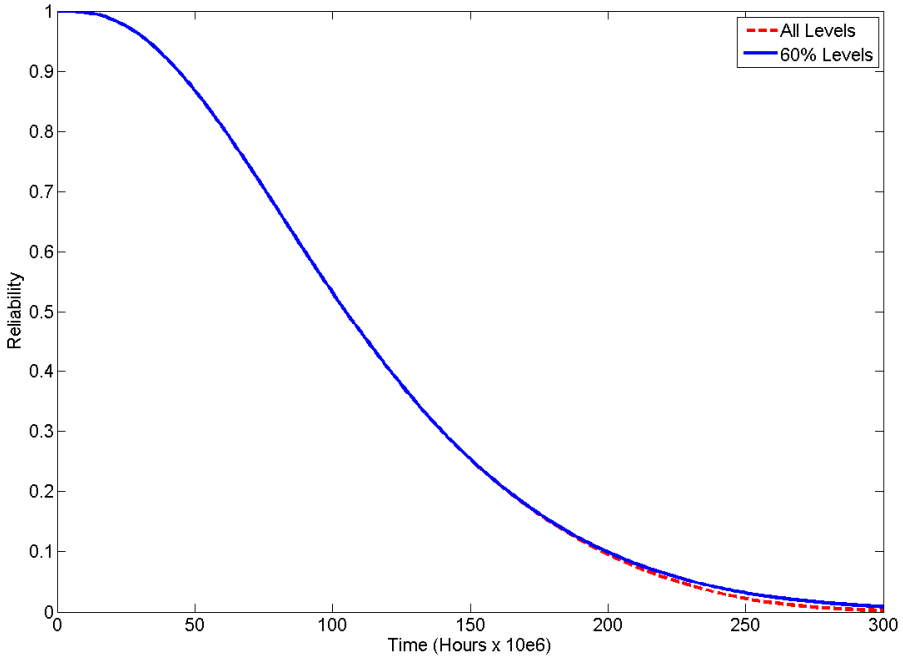


Figure 2. LOwER-FaT array reliability

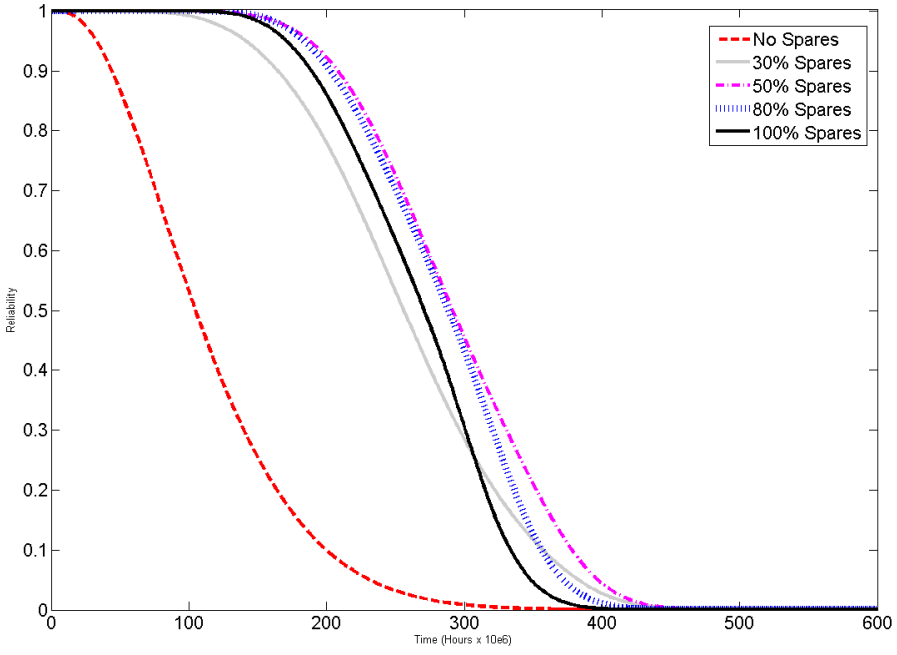


Figure 3. Spare output multiplexer strategy

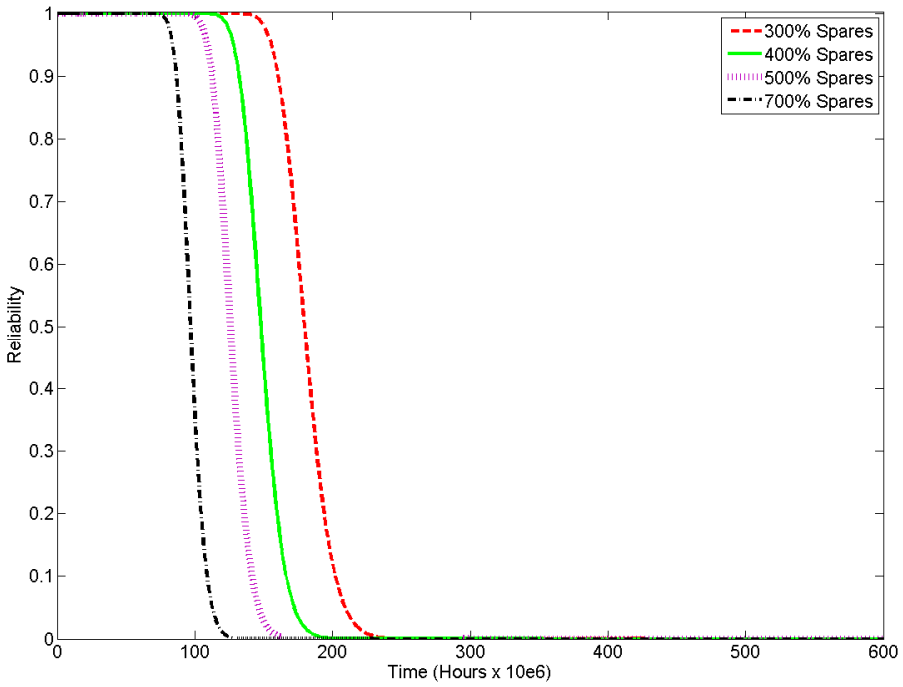


Figure 4. Spare output multiplexer strategy – reliability saturation

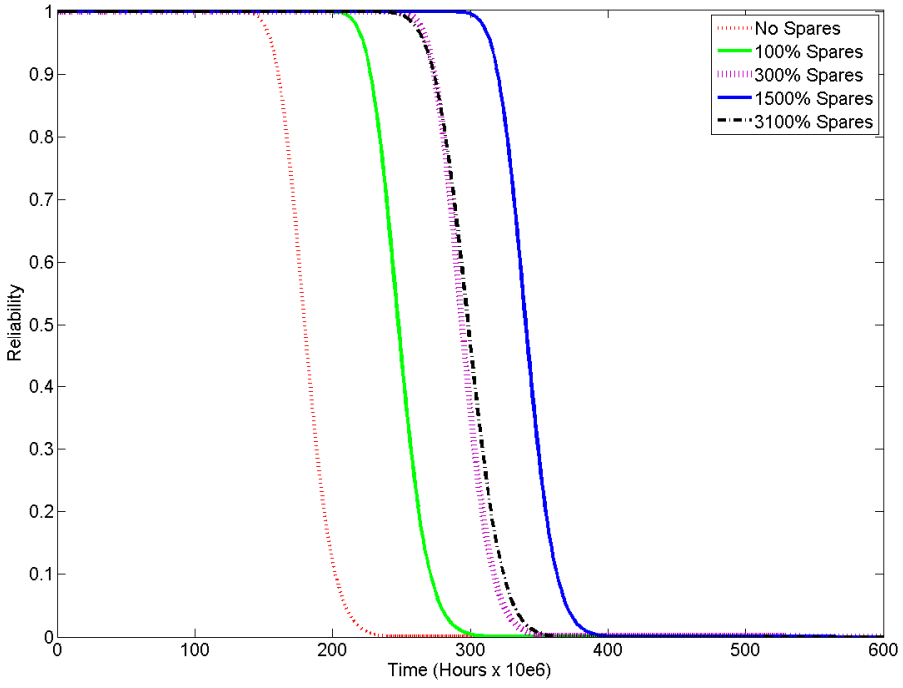


Figure 5. Spare input multiplexer strategy

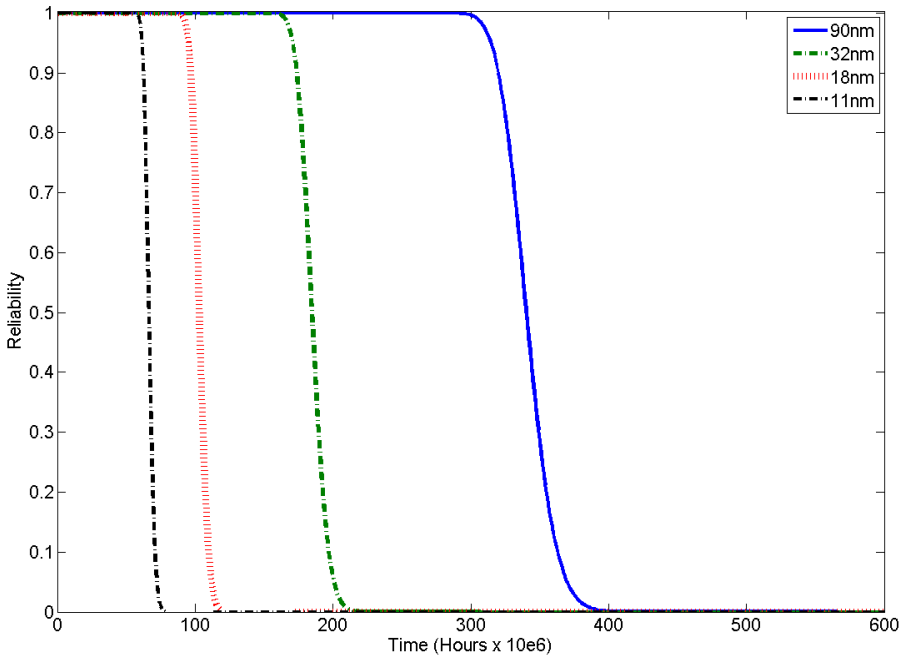


Figure 6. LOwER-FaT array reliability with spare multiplexers strategy – different technologies

APPENDIX C - BRIEF OVERVIEW IN PORTUGUESE

Este apêndice apresenta uma descrição resumida em língua portuguesa da tese descrita nesse documento. A descrição em português consiste de um resumo de toda a obra com os principais tópicos abordados na tese e com uma descrição das contribuições e conclusões do trabalho. As figuras e resultados apresentados são os mesmo descritos na versão em inglês do documento e são apenas parte do que foi obtido durante a pesquisa dessa tese.

Introdução

A evolução na pesquisa dos semicondutores tem feito várias contribuições para o processo de fabricação. Os esforços para atender a demanda e reduzir o custo de fabricação podem ser concentrados em aumentar o rendimento (*yield*) e melhorar o desempenho dos equipamentos, aumentar o tamanho do *wafer*, entre outras estratégias. Porém, a principal contribuição para reduzir o custo de fabricação é a redução do tamanho do circuito.

A miniaturização dos circuitos permite a inclusão de mais dispositivos em um *wafer*, com o aumento do desempenho do dispositivo. Em nano escala, é possível fabricar elementos lógicos com funcionalidade universal, como *arrays* programáveis, núcleos de memórias e interconexões programáveis.

Apesar disto, a redução proporcionada pelo processo de miniaturização também leva a altas densidades de falhas[1]. Em *nanowires*, por exemplo, a redução do diâmetro dos fios torna-os mais frágeis e suscetíveis a quebras. Além disso, como as áreas de contato entre os fios e os dispositivos também é reduzida, fica mais difícil garantir a integridade do contato, o que torna os circuitos mais suscetíveis a falhas no processo de fabricação. Isso implica em taxas de falhas muito altas que devem ficar entre 1% e 15% para fios e conexões.

Técnicas de redundância tradicionais que usam replicação podem ser extremamente custosas, não apenas por causa da grande quantidade de área necessária para tolerar altas densidades de falhas, mas principalmente pela excessiva dissipação de potência que essas técnicas introduzem. Uma solução para restringir os custos de potência e área das técnicas de redundância clássicas é o uso de partes ativas do circuito para substituir os recursos com falhas. Entretanto, uma consequência dessa solução é uma degradação de desempenho causada pela redução dos recursos disponíveis.

---

[1] Neste texto a palavra "falha" foi usada com sentido de *fault*. Apenas quando explicitado, falha terá o sentido de *failure.*

Assim, existem duas soluções principais para lidar com o projeto de processadores usando novas tecnologias. A primeira é modificar e melhorar o processo de litografia, mas esta ainda é uma área em desenvolvimento e muitas modificações complexas serão necessárias. A segunda é adaptar a arquitetura para que esta continue executando aplicações mesmo com altas densidades de defeitos.

Arquiteturas reconfiguráveis são possíveis candidatas para tolerância a falhas. Uma vez que a lógica reconfigurável geralmente consiste de diversos elementos idênticos, essa regularidade pode ser explorada para substituir os recursos com falha sem a necessidade de adicionar recursos extras e a capacidade de reconfiguração pode ser utilizada para alocar apenas os recursos livres de falhas. Além disso, a degradação de desempenho causada pelo uso de recursos operacionais para substituir os recursos com falha pode ser amortizada pelo alto desempenho atingido por essas arquiteturas. Adicionalmente, a reconfiguração dinâmica pode ser usada para evitar a necessidade de parar o sistema no momento da substituição de recursos, uma vez que a configuração é gerada e realizada em tempo de execução, sem comprometer o desempenho.

Baseado nisso, o foco desse trabalho consiste em buscar soluções de tolerância a falhas, mais especificamente de redundância de hardware, com o objetivo de aumentar a confiabilidade de arquiteturas reconfiguráveis, levando em consideração o impacto na área e no desempenho do sistema quando a redundância é aplicada.

As seções a seguir apresentam de forma resumida as principais contribuições desse trabalho e uma breve descrição do principal estudo de caso utilizado nessa tese e os resultados obtidos durante a pesquisa.

Contribuições

Considerando todas as motivações mencionadas anteriormente, esse trabalho apresenta três contribuições principais. A primeira contribuição consiste em determinar que existe um limite na quantidade de redundância de hardware que pode ser adicionada as arquiteturas reconfiguráveis de forma a aumentar a confiabilidade. Dependendo de onde e quanta redundância é aplicada ao sistema, isto pode impactar negativamente, e até mesmo reduzir a confiabilidade. Esse limite está relacionado ao custo de área que é introduzido quando a redundância de hardware é aplicada. Por essa razão, uma análise compreensiva deve ser realizada para encontrar a melhor estratégia de redundância para aumentar a confiabilidade.

A segunda contribuição desse trabalho é identificar qual parte da arquitetura reconfigurável deve ser investida de forma a aumentar a confiabilidade. Existe um consenso na comunidade de pesquisa de arquiteturas reconfiguráveis que o modelo de interconexão é crítico para o sistema em muitos aspectos, como área, potência e tolerância a falhas. Em alguns aspectos, essa consideração geral é baseada em análises críticas como o resultado de vários estudos práticos realizadas ao longo de muitos anos de pesquisa. Esse é o caso dos aspectos de área e potência. Entretanto, em relação a tolerância a falhas, não existe um estudo abrangente que leve em consideração todas as características da arquitetura e quantifique exatamente em quanto o modelo de interconexão influencia o confiabilidade da arquitetura em questão. Nesse sentido, esse trabalho apresenta uma análise detalhada de arquiteturas reconfiguráveis, estimando o impacto do modelo de interconexão na confiabilidade do sistema. Nessa análise, nós mostramos que as interconexões são de fato os elementos críticos para a confiabilidade do sistema. Nós também discutimos alternativas para superar esse problema e continuar aumentando a confiabilidade.

A terceira contribuição está relacionada com os meios utilizados para alcançar as primeiras duas contribuições. Para realizar uma análise de confiabilidade compreensiva, nós propomos o uso de uma modelagem de confiabilidade combinada com a análise de outros aspectos, como área e desempenho. para modelar a arquitetura reconfigurável, nós propomos o uso de uma representação matemática onde cada componente da arquitetura apresenta uma confiabilidade individual. Adicionalmente, a confiabilidade total do sistema é uma função das confiabilidades de todos os componentes que compõem o sistema. A análise é voltada para avaliar os efeitos das falhas permanentes na arquitetura e encontrar a melhor estratégia para atenuar os efeitos das falhas de forma a permitir o uso do dispositivo.

Portanto, a análise tem o objetivo de auxiliar o projeto de arquiteturas reconfiguráveis confiáveis dando algumas indicações sobre o tipo de investimento e onde este deve ser feito. Consequentemente, encontrando a melhor estratégia para o projeto de uma arquitetura confiável e evitando alto custo de área, desempenho e/ou potência.

No contexto desse trabalho, nós também propomos uma nova estratégia de tolerância a falhas implementada em uma arquitetura reconfigurável. Essa estratégia dinamicamente gera a configuração através da seleção de recursos livres de falhas para executar as operações. A estratégia explora a regularidade da arquitetura reconfigurável, evitando a inclusão de recursos extras somente para tolerância a falhas. Portanto, nessa abordagem, todas as unidade funcionais são usadas para acelerar execução, e somente quanto uma falha afeta um recurso, este recurso é eliminado.

Modelagem de Confiabilidade

Confiabilidade é a probabilidade que um sistema trabalhe corretamente durante um período de tempo sob condições específicas. Em outras palavras, confiabilidade é a medida que indica por quanto tempo o sistema pode trabalhar corretamente mesmo quando falhas afetam partes do seu sistema.

Técnicas de tolerância a falhas implementadas em sistemas computacionais impactam diretamente na confiabilidade. Um sistema tolerante a falhas é capaz de tomar ações para prevenir que as falhas levem a erros e consequentemente a total falha (aqui falha tem o sentido de *failure*) do sistema que afeta o funcionamento correto do sistema. Portanto, o sistema pode trabalhar corretamente por um período de tempo maior apesar das falhas.

Para descrever a confiabilidade do sistema é necessário modelar esse sistema levando em consideração alguns aspectos importantes relacionados ao projeto do sistema e as características físicas que influenciam a probabilidade de falhas.

Dessa forma, duas informações essenciais devem ser incluídas para garantir uma descrição correta do sistema. A primeira informação consiste na descrição dos componentes e do relacionamento entre esses componentes. Nesse trabalho, nós descrevemos os componentes e os relacionamentos utilizando a representação de diagrama de blocos. Como o nome indica, os componentes são representados por blocos e o relacionamento entre blocos representam a forma que os componentes se conectam uns com os outros no sistema. Além disso, também é necessário especificar a lei de probabilidade que governa as falhas. Para dispositivos eletrônicos, a lei de probabilidade adotada consiste em uma distribuição exponencial de falhas chamada lei exponencial de falhas (*exponential failure law*), que é função da taxa de falhas (λ) e do

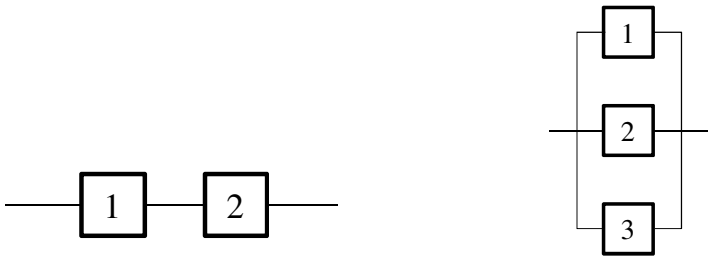
tempo. A taxa de falhas descreve a quantidade de erros que irão ocorrer no tempo. É uma constante determinada pelo modelo do dispositivo, que leva em consideração parâmetros que descrevem as condições físicas e operacionais no dispositivo, e as condições ambientais em que o dispositivo opera. A função que define a lei exponencial de falhas é descrita na equação (1).

$$p(t) = e^{-\lambda t}. \quad (1)$$

Um sistema eletrônico consiste de vários componentes eletrônicos independentes, cada um com uma confiabilidade diferente determinada pela equação (1). A confiabilidade total do sistema é função das confiabilidades de todos os componentes eletrônicos que compõem o sistema.

Após definir a confiabilidade individual dos componentes, é necessário determinar suas conexões uns com os outros. Na representação de diagrama de blocos, cada componente é representado com um bloco e as conexões entre os componentes são representadas pelas arestas.

Em um sistema sem redundância, todos os componentes são essenciais para o funcionamento adequado do sistema. Para representar este relacionamento, os componentes são conectados por uma única aresta, formando a conexão em série. Sistemas compostos por conexão em séries são chamado sistemas em série. Por outro lado, quando redundância de hardware é adicionada para aumentar a confiabilidade do sistema, os componentes redundantes não são essenciais para a correta operação do sistema. Para representar isto, os componentes são conectados entre si de forma paralela, formando um sistema paralelo. A terceira e última representação também usada em sistemas redundantes descreve a situação quando de todos os componentes, apenas uma parte deles é necessária para o correto funcionamento do sistema. Esse é chamado sistema k-de-m (*k-out-of-m*), onde m é o número total de componentes e k é o número de componentes necessários. Para um sistema totalmente paralelo k=1, e para um sistema em série k=m. A Figura 1 ilustra os dois primeiros tipos de sistemas, onde Figura 1.a representa o sistema em série, Figura 1.b representa o sistema em paralelo, respectivamente. Devido a particularidade do sistema k-de-m, este não possui representação gráfica. Além disso, as funções de confiabilidade que descrevem os três sistemas, série, paralelo e k-de-m, são descritas nas equações (2), (3) e (4), respectivamente.



a) Exemplo de sistema em série b) Exemplo de sistema em paralelo

Figura 1. Representação de diagrama de blocos

$$R_{serie}(t) = \prod_{i=1}^{n}(p_i(t)), \quad (2)$$

$$R_{paralelo}(t) = 1 - \prod_{i=1}^{n}(1 - p_i(t)), \quad (3)$$

$$R_{k-de-m}(t) = \sum_{i=k}^{m} \binom{m}{i} p(t)^i (1 - p(t))^{m-i}, \quad (4)$$

Onde R vem do nome em inglês *reliability, n* é o número total de componentes nas equação (2) e (3), *m* é o número total de componentes na equação (4) e por fim, *k* é o número de componentes necessários na equação (4).

A função de confiabilidade de sistemas complexos é definida pela combinação de vários subsistemas que podem estar em série, paralelo ou k-de-m.

Na seção seguinte é apresentada a arquitetura reconfigurável utilizada como principal estudo de caso desse trabalho, bem como o seu modelo de confiabilidade.

Arquitetura Reconfigurável

O sistema reconfigurável consiste de uma arquitetura reconfigurável de granularidade grossa fortemente acoplada a um processador MIPS R3000; um mecanismo para gerar a configuração e a memória de reconfiguração que armazena a configuração. A Figura 2 ilustra o sistema reconfigurável representado em diagrama de blocos.

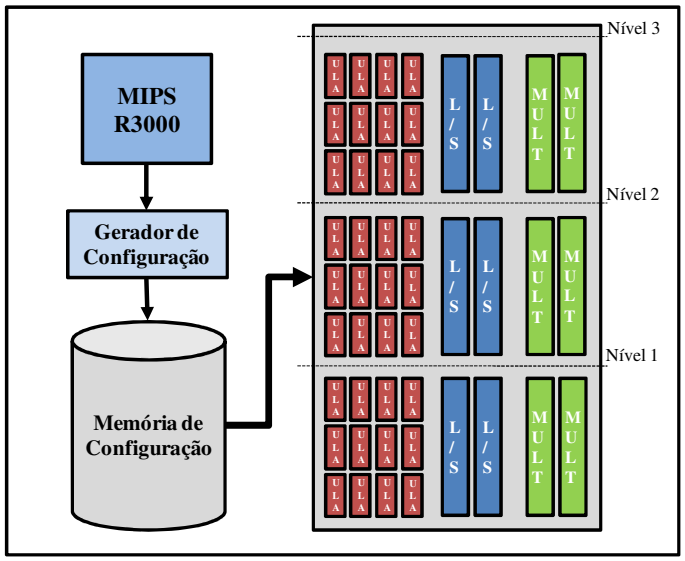


Figura 2. Sistema Reconfigurável

A arquitetura reconfigurável é um circuito combinacional formado por três grupos de unidades funcionais: o grupo de unidades de lógica e aritmética (ULA), o grupo de unidades de armazenamento (*load/store*) e o grupo de unidades de multiplicação. A Figura 3 apresenta o diagrama de blocos da arquitetura reconfigurável.

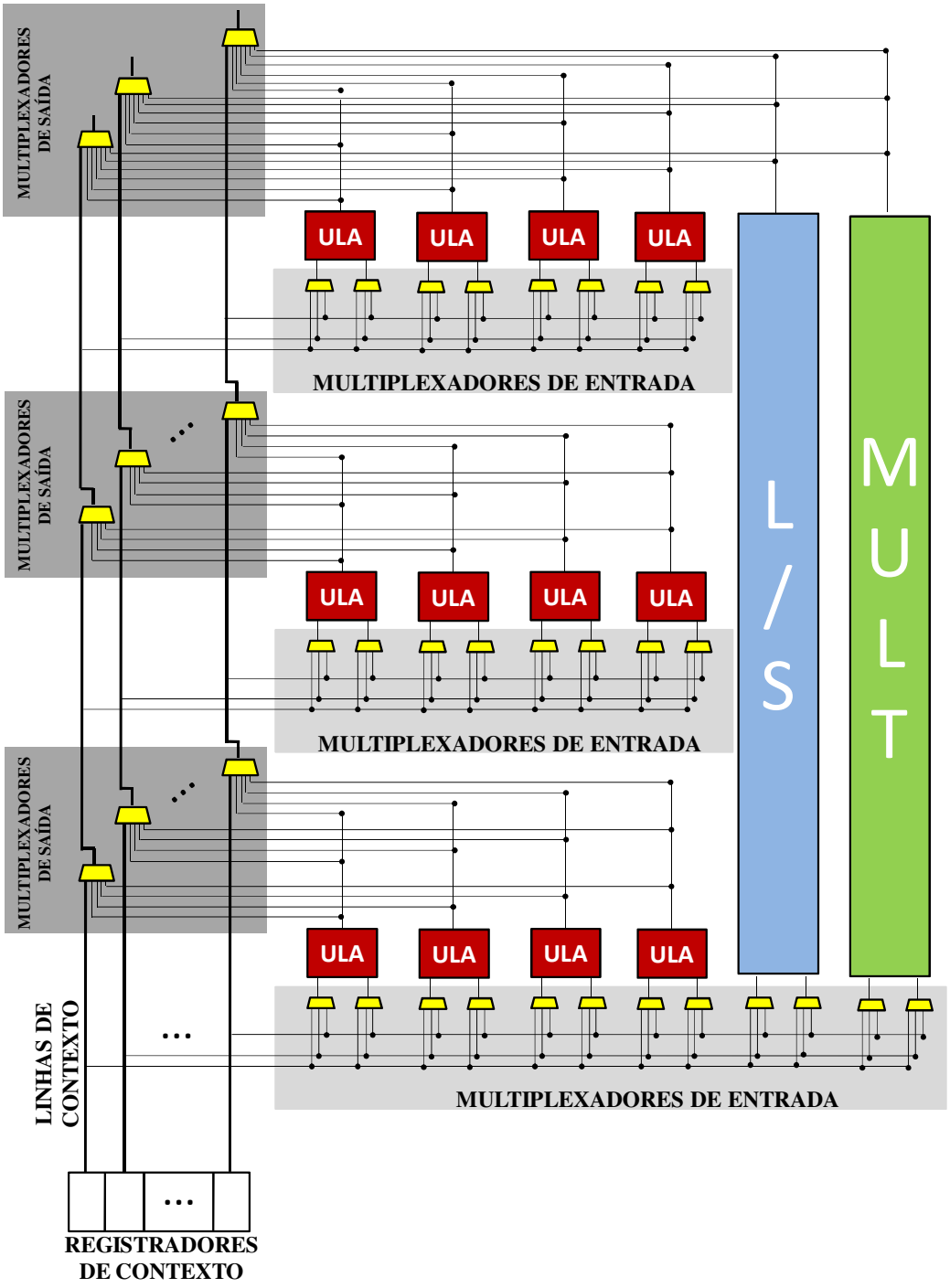


Figura 3. Arquitetura Reconfigurável

É importante ressaltar que cada grupo de unidades funcionais pode ter um tempo de execução diferente, que depende da tecnologia e da estratégia de implementação. Baseado nisso, nesse trabalho, o grupo de ULAs pode executar até três operações em um ciclo de processador, enquanto os outros dois grupos executam uma operação cada em um ciclo de processador. Para diferenciar o ciclo do processador do ciclo da

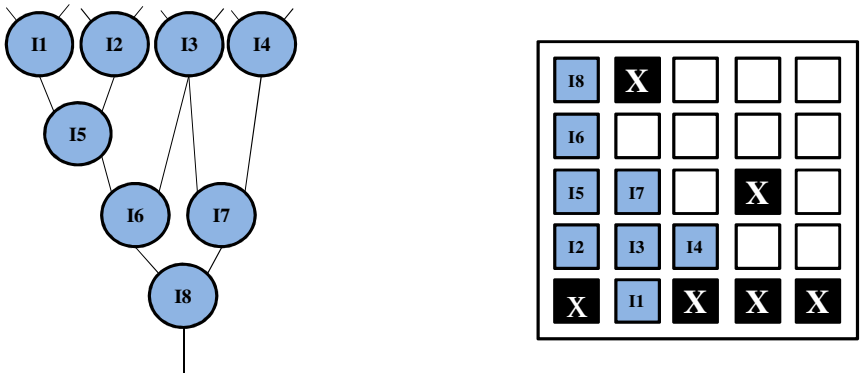
arquitetura reconfigurável, este último é chamado de nível. Os diferentes tempos de execução apresentados por cada grupo de unidades funcionais permite a execução de mais que de uma operação por nível. Portanto, a arquitetura reconfigurável pode executar até três operações lógicas e/ou aritméticas que apresentam dependência de dados entre si em apenas um ciclo. Essa característica da arquitetura permite que a mesma acelere a execução de código sequencial, quando comparado com a execução no processador MIPS R3000.

LOwER-FaT Array

Para poder aumentar a confiabilidade da arquitetura reconfigurável apresentada na seção anterior, nesse trabalho foi proposto um mecanismo de tolerância a falhas para evitar que falhas em unidades funcionais e elementos de interconexão afetem a execução correta do sistema.

O objetivo do mecanismo é explorar a redundância intrínseca da arquitetura reconfigurável através do uso de unidades funcionais e interconexão operantes para substituir os recursos com falha. Nessa abordagem não é necessário adicionar recursos extras específicos para tolerância a falhas. Com isso, em uma situação em que não existam falhas, todas as unidades funcionais são usadas para acelerar a execução da aplicação e somente em caso de falha, uma unidade funcional e/ou elemento de interconexão é substituído. Além disso, a capacidade de reconfiguração da arquitetura é explorada para mudar a alocação de recursos baseado nos recursos com falha, e a reconfiguração dinâmica é utilizada para gerar uma nova configuração em tempo de execução. Portanto, falhas de envelhecimento que ocorram durante a vida útil do dispositivo também podem ser toleradas.

Para exemplificar como o mecanismo de tolerância falhas funciona, a Figura 4 apresenta um exemplo de alocação de recursos na arquitetura reconfigurável. A Figura 4.a ilustra o grafo de dependência de dados que deve ser executado pela arquitetura. Para alocar as instruções nas unidades funcionais, inicialmente é buscada a primeira unidade funcional disponível. Como pode ser observado na Figura 4.b, a primeira fileira de unidades funcionais tem apenas uma unidade funcional disponível, e as demais com falha (representadas com um X). Neste caso, apenas a primeira instrução é alocada na primeira fileira, e as instruções seguinte são alocadas nas próximas fileiras. Uma vez que as instruções 2, 3 e 4 podem ser executas em paralelo, todas são alocadas na segunda fileira. Por outro lado, a instrução 5 que depende do resultado das instruções 1 e 2, deve ser alocada na fileira acima da instrução 2. Bem como a instrução 6, que depende da instrução 5 e, consequentemente, deve ser alocada na fileira seguinte. Já a instrução 7, que depende das instruções 3 e 4, pode ser alocada na mesma fileira da instrução 5. Por fim, a instrução 8 é alocada na última fileira, pois esta depende do resultado da instrução 6.



a) Grafo de dependência de dados b) Alocação de intruções na arquitetura com falhas

Figura 4. Exemplo de alocação de instruções com mecanismo de tolerância a falhas

Resultados

Baseado nas descrições da arquitetura reconfigurável e do mecanismo de tolerância a falhas, o modelo de confiabilidade do sistema reconfigurável foi gerado. A partir desse modelo, descrito no capítulo 4 desse documento, diversos estudos de confiabilidade foram realizados. O objetivo dos estudos foi de avaliar quais as estratégias de redundância de hardware mais adequadas para aumentar a confiabilidade, levando em consideração o custo de área, quando novos componentes foram acrescentados.

A Figura 5 apresenta a curva de confiabilidade em função do tempo da arquitetura reconfigurável com e sem o mecanismo de tolerância a falhas (Figura 5.a e Figura5.b respectivamente). Os resultados obtidos nessa comparação apresentaram uma diferença de 8 ordens de magnitude entre os resultados, demonstrando um grande aumento da confiabilidade quando o mecanismo de tolerância a falhas é adicionado a arquitetura. Esse resultado pode ser visto nos gráficos ao observar o eixo do tempo que apresenta escalas diferentes.

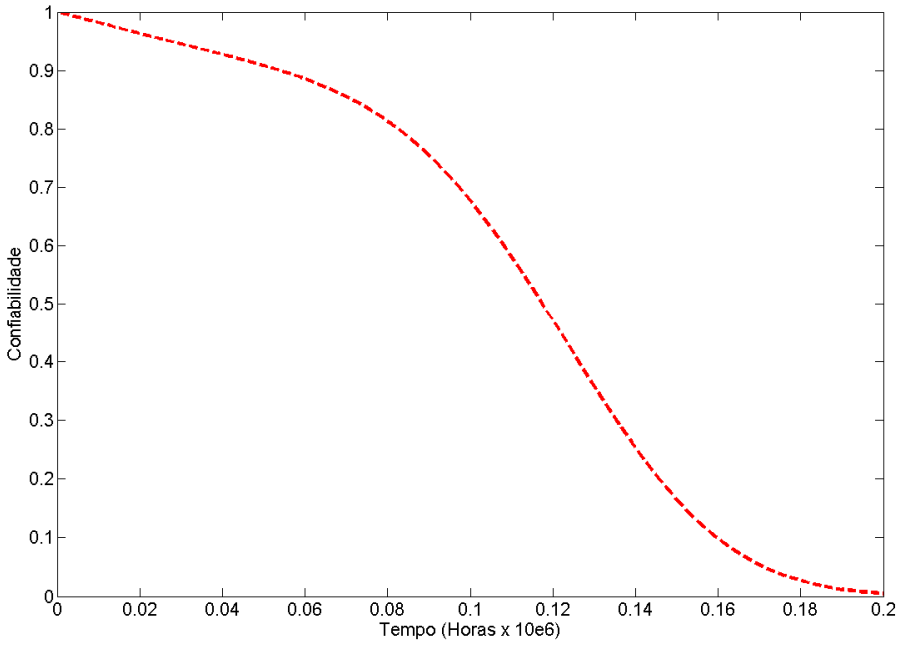


Figura 5.a) Confiabilidade da arquitetura reconfigurável sem tolerância a falhas

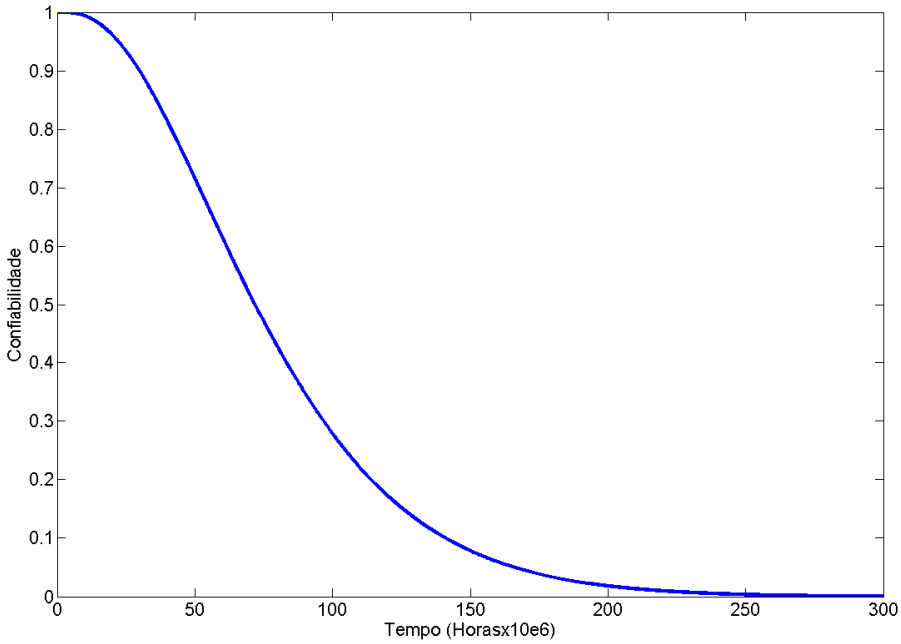


Figura 5.b) Confiabilidade da arquitetura reconfigurável com tolerância a falhas

Outro resultado relevante, também apresentado no capítulo 4, consiste na confiabilidade da arquitetura reconfigurável estimada para diferentes tecnologias. Nessa análise, além do mecanismo de tolerância a falhas também foram acrescentadas mais interconexões para substituírem as interconexões com falha. Essa estratégia foi

resultado de um estudo abrangente que avaliou diversas estratégias de redundância, e possibilitou identificar a quantidade de redundância ideal para ser acrescentada na arquitetura de forma a maximizar a confiabilidade. Para a obtenção da confiabilidade apresentada reproduzida na Figura 6 foram acrescentados 300% de multiplexadores de saída e 1500% de multiplexadores de entrada. A partir do resultado apresentado na Figura 6 também é possível observar que a redução do tamanho do circuito integrado para tecnologias menores está de fato impactando de forma negativa na confiabilidade.

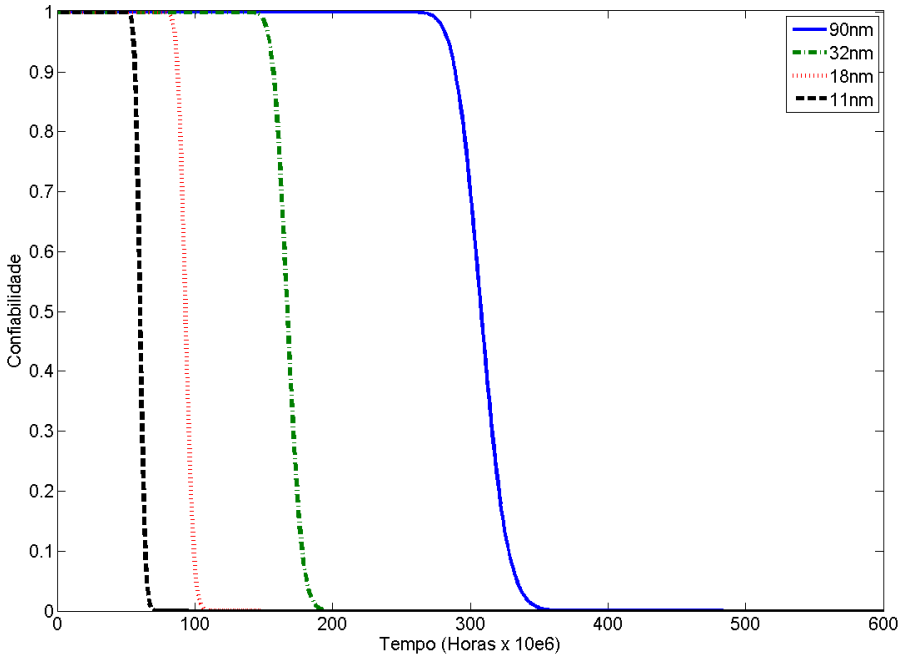


Figura 6. Confiabilidade da arquitetura reconfigurável para tecnologias diferentes

Outros resultados de confiabilidade incluindo o estudo de um modelo de interconexão alternativo ao modelo de barramento e multiplexador, além da análise de outra arquitetura reconfigurável de granularidade grossa podem ser encontrados nos capítulos 4 e 5 desse documento.

Conclusões

Esse trabalho investigou o projeto de arquiteturas reconfiguráveis tolerantes a falhas visando aumentar a confiabilidade e considerando o aumento da taxa de falhas esperado para tecnologias do futuro com tamanho do transistor reduzido. Para tal, foi apresentada uma análise de confiabilidade baseada em um modelo matemático que hierarquicamente conecta cada recurso como um conjunto de subsistemas, onde cada subsistema possui sua própria confiabilidade.

A modelagem de confiabilidade é um poderoso mecanismo para avaliar os recursos mais críticos para a confiabilidade do sistema e permite o direcionamento dos investimentos e modificações no projeto do hardware de forma a aumentar a confiabilidade e levando em consideração custos como área, desempenho, dentre outros.

Dentre as principais contribuições desse trabalho destacam-se a utilização de um modelo de confiabilidade para estudar a confiabilidade de arquiteturas reconfiguráveis;

a identificação dos elementos de interconexão como parte mais crítica na confiabilidade de arquiteturas reconfiguráveis; a descoberta de um limite de redundância de hardware que pode ser acrescentada para aumento de confiabilidade; a conclusão de que não existe uma solução genérica que pode ser aplicada a todas as arquiteturas e por fim, a proposta de um mecanismo de tolerância a falhas para a arquitetura reconfigurável utilizada como principal estudo de caso desse trabalho.

Dentre os tópicos de pesquisa futuros que podem dar continuidade a esse trabalho estão a investigação de diferentes modelos de interconexão visando encontrar modelos mais confiáveis para arquiteturas reconfiguráveis. Aplicar a modelagem de confiabilidade para arquiteturas reconfiguráveis de granularidade fina (*FPGAs - Field Programmable Gate Arrays*) com o objetivo de avaliar a confiabilidade desse tipo de arquitetura. Estender a análise de confiabilidade para outras arquiteturas de alto desempenho, como arquiteturas VLIW (*Very Long Instruction Word*) e super-escalares, de forma a avaliar qual das arquiteturas apresenta uma melhor relação confiabilidade *versus* desempenho. Aplicar a modelagem de confiabilidade para o estudo da confiabilidade de arquiteturas multiprocessadas, como MPSoCs (*Multiprocessor System-on-Chip*) bem como o modelo de interconexão utilizado nessas arquiteturas, como rede em chip (NoC - *network on chip*). E implementar a arquitetura reconfigurável e o mecanismo de tolerância a falhas proposto como estudo de caso desse trabalho em FPGA.