POLITECNICO DI TORINO

DOCTORATE SCHOOL

Ph.D. in Computer And Control Engineering – XXVII cycle

PhD Thesis

# New Test and Fault Tolerance Techniques for Reliability Characterization of Parallel and Reconfigurable Processors


**Davide Sabena**

**Advisor**
Prof. Luca Sterpone

**PhD Program Coordinator**
Prof. Matteo Sonza Reorda

February 2015

*"Dedicata a tutti coloro che nella vita hanno scelto di vivere come uomini giusti."*

# Acknowledgements

First of all, I would like to thank my tutor Prof. Luca Sterpone that with its support, help and patience guided my work during these years.

I would also like to thank the Prof. Matteo Sonza Reorda for the countless ideas and opinions suggested me throughout the PhD course.

I consider it an honour to have worked in the CAD group; in particular, many thanks to Ernesto Sanchez, Giovanni Squillero and Paolo Bernardi for the interesting discussions we had in these years.

Special thanks go to my fellow lab members at Politecnico di Torino: Anees Ullah, Boyang Du, Lyl Ciganda, Marco Desogus, Marco Gaudesi, Mauricio De Carvalho, Riccardo Cantoro, Ronaldo Ferreira, and all the students who came and went during these years; all of them made my work much more fun. Moreover, I would like to thank my friends and PhD colleagues Diego Chiabrando, Giulio Gambardella, and Daniele Rolfo for the countless discussions we had in these years: they have been an essential source of motivation.

I would also like to thank Prof. Paolo Rech (from Universidade Federal do Rio Grande do Sul) for the great collaborations we had in these years; his support has been essential for the execution of the radiation experiment we performed together at ISIS in December 2013.

From a personal point of view, I would like to thank my girlfriend (as well as my future wife) Valentina, for the constant support and generosity that she provided me during these years. Without her I would not have achieved all the important results obtained in the PhD course.

Last but not least, special thanks to my parents Marina and Giorgio for their essential help and support over the years; and, finally, many thanks to all my family members.

# Summary

Integrated electronic systems are more and more used in a wide number of applications and environments, ranging from mobile devices to safety-critical products. This wide distribution is mainly due to the miniaturization surrounded by an increasing computing power of semiconductor devices. However, there are many complex and arduous challenges associated to this phenomenon.

One of these challenges is the reliability of electronic systems. Nowadays, several research efforts are aimed at improving the semiconductors reliability. Manufacturing processes, aging phenomena of components and environmental stress may cause internal permanent defects and damages during the lifetime of a device; in the other side, the environment in which these devices are employed could introduce soft errors (i.e., errors that do not damage the device but a data during the computation) in their internal circuitry, thus compromising the correct behavior of the whole system. Consequently, in order to guarantee product quality and consumer satisfaction, it is necessary to discover faults as soon as possible (both, in the manufacturing process and during the devices lifetime); moreover, it is equally important to provide the electronic systems with fault tolerance equipments aimed to assure a correct functioning in every condition.

Despite the reliability requirements, modern electronic systems require also an increasing computational power to satisfy the customers needs. In order to face to this demand, in the last two decades different powerful computational devices have been designed and developed. They are mainly based on architectures allowing the execution of multiple computations in parallel at the same time.

Among the others, the Very Long Instruction Word (VLIW) processors are a particular type of multicore and reconfigurable processors; they have been developed to perform several operations in parallel, where the scheduling of the operations themselves is completely demanded at the compiler: VLIWs are suitable for systems requiring high computational performance maintaining a reduced power consumption. Another interesting type of multicore computational units are the General Purpose Graphics Processing Units (GPGPUs): their very high computational power, combined with low cost, reduced power consumption, and flexible

development platforms are pushing their adoption not only for graphical applications, but also in the High Performance Computing (HPC) market and in embedded devices. Moreover, GPGPUs are increasingly used in some safety-critical embedded domains, such as automotive, avionics, space and biomedical.

The main in common feature of VLIWs and GPGPUs is that they can be used in a System-on-Chip (SoC) as computational co-processors: in a typical SoC, in fact, the main Central Processing Unit (CPU) is in charge of demand and supervise the execution of data intensive operations to these architectures; in this way, the workload of the CPU itself is lower. As an example, in the NASA labs, VLIWs have been evaluated to efficiently perform image analysis on board a Mars rover for future space missions, while the main CPU of the system is available to perform other real-time control operations. In the other hand, the Advanced Driver Assistance Systems (ADASs) which are increasingly common in cars, uses GPGPUs or GPGPU-like devices to analyze images (or radar signals) coming from external cameras and sensors to detect possible obstacles, requiring the automatic intervention of the breaking system.

In this PhD thesis, several new techniques have been developed with the common goal of improving the reliability characteristics of multicore processing units. More in particular, considering VLIW processors, new test and diagnostic methods have been studied and implemented in order to detect permanent faults; they are mainly based on the Software-Based Self-Test (SBST) technique. The final goal is to reduce the time required to perform the test of a generic VLIW processor, and to efficiently localize the faulty module. On the other hand, the present dissertation focus on the effects introduced by soft errors in GPGPU devices; this works have been done through the execution of several neutron radiation tests. At the end of these analysis, new techniques finalized to the fault tolerance enhancement of GPGPU applications have been proposed.

As industrial case, the validation of a programmable timing multicore co-processor module (i.e., the Generic Timer Module manufactured by Bosch) used in the today automotive Electronic Control Units (ECUs) has been designed and implemented. More in particular, an FPGA-based validation platform has been developed, where one of its main feature is the ability to efficiently verify the behavior of the module under test, thus ensuring a correct implementation of the software running on it. This work has been done in collaboration with General Motors Powertrain Europe (site of Torino, Italy).

By implementing the techniques presented in this PhD thesis, several interesting data about the reliability of the treated devices have been acquired; they are shown in 15 papers, published in conference proceedings, book chapters, and international journals.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction: reliability in parallel architectures and System-on-Chip

Integrated electronic systems are more and more used in a wide number of applications and environments, ranging from mobile devices to safety-critical products. In the last three decades, in fact, information technology has been integrated in most of the devices used every day by million of people. This wide distribution phenomenon is mainly influenced by a continous miniaturization surrounded by an increasing computing power of semiconductor devices.

Originally expressed by Gordon Moore in 1965, Moore's Law states that the number of transistors on an integrated circuit doubles every one to two years. The result over the last fifty years has been incredible improvements in computational capabilities, communications, entertainment, and all aspects of electronic technology, all at ever lower cost. Literally, Moore's Law has brought about a radical change in society, with huge implications for the everyday life of people [1].

Alongside to this scenario, there are many complex and arduous challenges. One of these is the reliability of the electronic systems. Nowadays, several research efforts are aimed at improving the semiconductors reliability. Continuing decrease in the feature size of transistors leads to increases in susceptibility to permanent and transient faults: the technology scaling down to the nanometer domain, shrinking transistor sizes, lower power voltages, and higher operating frequencies seriously affect the reliability of CMOS VLSI circuits [2][3]. In general, the reliability of semiconductor devices may depend on manufacturing processes, aging phenomena of components, and environmental stress, that could be source of internal permanent defects and damages during the lifetime of a device; in the other side, the environment in which these devices are employed could introduce soft errors (i.e., errors that do not damage the device but a data during the computation) in their internal circuitry, thus compromising the correct behavior of the whole system [4]. Consequently, in order to guarantee product quality and consumer satisfaction, it

is necessary to discover faults as soon as possible (both, in the manufacturing process and during the devices lifetime); moreover, it is equally important to provide the electronic systems with fault tolerance mechanism aimed to assure a correct functioning in every condition.

Beside the reliability challenge, the power consumption is another parameter that has to be considered designing innovative computational units and System-on-Chip (SoC). To face the power problem, in the last two decades chip designers have resorted to multicore architectures, that provide a way to continue improving performance with a low increase in the power consumption [4] (if compared with the power consumption required by a further increase of the functioning frequency). An advantage of this phenomenon is that multiprocessors devices are inherently suitable for reliability, due to the availability of several computational units on which redundant computations can be executed in order to error detection and/or correction.

Among the various multicore systems developed until now, the Very Long Instruction Word (VLIW) processors and the General Purpose Graphic Processing Units (GPGPUs) are of significant interest. Both these architectures are based on a lot of parallel computational units able to perform a huge number of operations in parallel at the same time. One of the main in common feature of VLIWs and GPGPUs is that they can be used in a SoC as computational co-processors: the Central Processing Unit (CPU) of the system is in charge of demand and supervise the execution of data intensive operations to these architectures, lowering its workload.

In this PhD thesis, innovative techniques are proposed addressing different device scenario, where the main goal is to improve the reliability features of the most used parallel architectures today embedded in SoC as computational co-processing units, i.e., the VLIW processors and the GPGPUs. As industrial case, the validation of a programmable timing multicore co-processor module (i.e., the Generic Timer Module by Bosch) used in the today automotive Electronic Control Units (ECUs) has been designed and implemented. More in particular, an FPGA-based validation platform has been developed, where one of its main feature is the ability to efficiently verify the behavior of the module under test, thus ensuring a correct implementation of the software running on it. This work has been done in collaboration with General Motors Powertrain Europe (site of Torino, Italy).

By implementing all the techniques presented in this PhD thesis, several interesting data about the reliability of the treated devices have been acquired; these data are presented in 15 papers published in conference proceedings, book chapters, and international journal.

The rest of this PhD thesis is structured as follow: the other four sections of this introductory chapter present an overview on the addressed devices (from a reliability point of view); in the Chapter 2 the reliability characterization and the

State-Of-The-Art are presented; in Chapter 3, instead, the proposed SBST and diagnostic methods for VLIW processors are shown, along with some experimental results and the activity conclusions; in Chapter 4 the reliability works developed in this PhD thesis related to GPGPU devices are presented; in Chapter 5 the industrial case treated in this PhD thesis is described; finally, the Chapter 6 concludes this dissertation presenting the general conclusions.

## 1.1 Parallel architectures: different needs, different devices

As general concept, modern electronic systems require an increasing computational power. In order to face with this need, in the last two decades, different powerful computational devices have been designed and developed. They are mainly based on architectures allowing the execution of multiple computations in parallel at the same time. In the past, applications could simply rely on system performance improvements from advances in semiconductor manufacturing and single-thread architecture. Today, nearly all major microprocessor vendors offer multicore processors. Instead of scaling performance with increased frequency, multicore processors offer higher performance through more processing cores. As main motivation behind this phenomenon there are the efforts to reduce the power consumption and the thermal dissipation.

Multicore processors differ from traditional processors in many ways, such as higher core counts, simpler core architecture, and more elaborate on-chip interconnections. The increase in core count and compute density is the most notable difference between multicore architectures and traditional single-thread architectures [5]. As a consequence of this radical change, the applications running on new multicore devices have to be parallelized to fully exploit the computational power provided.

The VLIW processors have been demonstrated to be a viable solution especially for applications demanding high performance while exposing a considerable amount of parallelism, such as several Digital Signal Processing (DSP) algorithms used in multimedia and communication applications [6]. In VLIW architectures, the management of the parallelization of the applications is completely demanded at the software compiler; in fact, unlike superscalar processors, VLIW processors do not include any significant control logic, since instruction scheduling is completely performed at compile time. This implies that the hardware complexity is far lower than for superscalar processors, while the compilation steps become more complicated. VLIWs offer the possibility to run a typical sequential program in a multicore architecture without changing the original source code; the compiler, in fact, is in charge of understanding and exploiting as much as possible the Instruction Level

Parallelism (ILP) intrinsic of the software application. Moreover, VLIW processors are characterized by a pipelined architecture with multiple Functional Units (FUs). Considering the instruction format, VLIW processors are characterized by grouping several instructions (named *micro-instructions*) into one large *macro-instruction* (also called *bundle*), where each micro-instruction within the bundle is executed in parallel distinct computational units, referred to as *Computational Domains (CD)* [7].

Recently, new devices known as GPGPUs made their appearance on the market. Their very high computational power, combined with low cost, reduced power consumption, and flexible development platforms are pushing their adoption not only for graphical applications, but also in the High Performance Computing (HPC) and in the embedded systems markets [8]. This phenomenon is motivated by the fact that GPGPUs offer tremendous computational power by running a huge number threads concurrently. For example, the NVIDIA GPU Kepler [9] allows concurrent execution of over 30,000 active threads and delivers an aggregate performance of 4.6 teraflops on single-precision floating-point computation [10]. In order to manage the scheduling of the threads execution, the GPGPUs contain dedicated hardware modules (i.e., the thread schedulers). The programming model is based on the Single Instruction Multiple Data (SIMD), where the user has to design ah ad-hoc software compliant with the SIMD paradigm.

As industrial case, in this PhD thesis the reliability of a new multicore timer co-processor, today used in automotive applications, has been addressed. The name of this module is Generic Timer Module (GTM), and it has been developed by Bosch Semiconductors [11]. This computational unit has been designed with the aim of executing the tasks typically related to the management of the fuel injection in a thermal engine. The GTM can be seen as an autonomous co-processor; in fact, it contains several hardware sub-modules allowing it to directly manage the engine fuel injection signals without any interaction with the main control system (it unloads the CPU from handling Interrupt Service Requests (ISR) as much as possible), thus ensuring the real-time constraints required in automotive ECU. More in particular, the GTM is composed of several independent parallel computational units that could be programmed to efficiently manage different real-time tasks in parallel.

## 1.2 Test methods for VLIW processors

VLIWs are today used in several data intensive applications; they are also employed in mission critical systems. As an example, the processor Tilera TILE64, composed of several VLIW cores, has been evaluated (in the Jet Propulsion Lab., California Institute of Technology, Pasadena, CA, USA) to efficiently perform image analysis on-board a Mars rover, in support of autonomous scientific activities [12][13]. Given

these motivations, test methods are required, in order to discover the presence of permanent faults (both at the start-up than during the operational life) in VLIW processors.

In literature few test approaches have been proposed aimed to properly test VLIW processors against permanent faults. In this PhD thesis the generation of effective test programs for the whole VLIW processor is addressed, characterized by minimal duration, minimal size, and maximal fault coverage. The proposed method is mainly based on the Software-Based Self-Test (SBST) approach [14], and it starts from existing functional test algorithms developed for each single FU type embedded into the processor (e.g., ALUs, adders, multipliers, memory units). Although the characteristic of the FUs used within a VLIW processor are similar to those used in traditional processors, generating optimized code to effectively test these units is not a trivial task: in fact, by exploiting the intrinsic parallelism of VLIW processors it is theoretically possible to reduce the increase in duration and size of the test programs when the VLIW processor size grows. For example, testing the ALUs in the different Computational Domains can be performed in parallel, forcing them to perform the required computations in parallel at the same clock cycle. However, generating an optimized test program with minimal size and duration may require a significant manual effort, taking into account both the test algorithms for each FU and the specific VLIW processor configuration. The proposed test generation procedure provides an automatic solution for optimized test program generation, once the processor configuration and the test algorithms for each FU are known. VLIW processors do not include any specially designed hardware module (as it happens for other processor types), but are rather based on a combination of common Functional Units: exploiting this characteristic, our solution allows test program generation and optimization to be performed autonomously and automatically, without any manual effort. The test programs generated by the proposed method are highly optimized and exploit the VLIW processor features in order to minimize the test time and the test program size [7].

## 1.3 Fault tolerance enhancement for GPGPUs

Reliability is a big issue for GPGPU cores. While in their original application domain (i.e., video processing) wrong pixels caused by either soft or hard errors have a negligible effect on the user experience, when GPGPUs are exploited in HPC applications such as financial or scientific computations, correctness and high dependability become a primary requirement [15]. Moreover, the dependability requirements are the same if GPGPUs for embedded systems are considered: the Advanced Driver Assistance Systems (ADASs) which are increasingly common in cars, uses GPGPUs or GPGPU-like devices to analyze images (or radar signals)

coming from external cameras and sensors to detect possible obstacles, requiring the automatic intervention of the breaking system.

In literature, several issues about the reliability of GPGPUs have been raised [16]. Given their high degree of parallelism, many assume that GPGPUs could intrinsically provide a good degree of fault tolerance; however, their size and complexity could make them particularly sensible to soft errors. Moreover, while hardening techniques already exist for systems based on traditional CPUs, similar solutions for GPGPU-based systems are still in their infancy [17]. The programming paradigm adopted by GPUs (i.e., SIMD) can provide some advantages when designing hardening strategies, but requires effective solutions to combine detection and correction capabilities with the required high performance characteristics. When assessing the GPGPUs reliability, a commonly adopted solution is performing radiation experiments with accelerated particles, counting the number of errors they trigger.

The main topic of the research work presented in this PhD thesis, in the GPGPUs reliability context, is focused on investigating the sensitivity to soft-errors induced by terrestrial radiation effects. This evaluation has been performed through three different neutron-based radiation tests; they have been executed in the VESUVIO lab at the ISIS facility (Didcot, UK), and in the LANSCE lab at Los Alamos Neutron Science Center (Los Alamos, USA). The goals of this research activity are mainly three: initially, an evaluation of the radiation sensitivity of GPGPUs memories has been performed; then, several traditional soft error hardening techniques have been applied at different GPGPU benchmark algorithms, in order to assess the validity of these methods; finally, an analysis aimed at providing data about the reliability of different GPGPUs configuration is proposed. The final goal of the latter activity is to provide at the designers of GPGPU applications (running in safety-critical environments ) a set of guidelines to improve their reliability against soft errors.

## 1.4 Validation and test of automotive timing multicore co-processor module

The today automotive development processes are characterized by an increasing complexity in mechanic and electronic. However, electronic devices have been the major innovation driver for the automotive systems in the last decade [18]. In this context, the requirements in terms of comfort and safety lead to an increasing number of on-vehicle embedded systems, with more and more software-dependent solutions using several distributed Electronic Control Units (ECUs). Sophisticate engine control algorithms require performance enhancement of microprocessors to satisfy real-time constraints [19]. Parallel architectures are a promising solution to improve performances without an huge increase of power consumption. Moreover,

the code generation, the verification, and the validation of the code itself, become key part in the automotive domain: the software component development processes have to be as efficient and effective as possible. Moreover, without a reliable validation procedure, the automotive embedded software can lead to a lot of errors and bugs, decreasing the quality and the reliability of the applications.

Electronic devices managing the fuel injection in today engines have a key role, in order to guarantee efficient and powerful vehicles. Within this context, in this PhD thesis an efficient FPGA-based platform to test and validate a multicore timing co-processor (i.e., the GTM, introduced in the previous section) is presented. This platform is a sort of embedded validation platform to verify the correct behavior of the module under test. The high flexibility, combined with the capability of extreme precise measurements, make the platform very suitable for the developers of automotive applications during the parallel software development.

# Chapter 2

# Reliability background

Reliability is the predisposition of a product or system to perform the tasks for which it has been designed without failures, respecting its specified performance limits for a specified time, and in its life-cycle environment. Reliability characterization refers in general to all methods and procedures to measure how reliable a device is [20, 21, 22].

In this chapter, the reliability definition (within the electronic systems dependability domain) is introduced, along with other terms commonly used in this research field; then, several state-of-the-art techniques and method addressing the related reliability enhancements (for the devices treated in this PhD thesis) are presented.

## 2.1 Dependablity and related definitions

In general, the **dependability** of a computing system is the ability to deliver services that can be trusted in a justifiable way. Dependability is a comprehensive term used to describe the availability performance and its influencing factors: the reliability performance, the maintainability performance and maintenance support performance. As shown in the dependability tree proposed by Avizienis et al. in [20, 21](Figure 2.1), the concept of dependability consists of three main parts: the threats to, the attributes of, and the means by which dependability is attained.

### 2.1.1 The threats: faults, errors, and failures

An **error** is that part of the system state that may cause a subsequent failure: a **failure** occurs when an error reaches the service interface and alters the service itself. A **fault** is the hypothetical or established cause of an error. A fault is said "active" when it produces an error; otherwise, a fault without any related error is said "latent". Moreover, a system does not always fail in the same way: the **failure modes** describe the ways in which a system can fail [21].

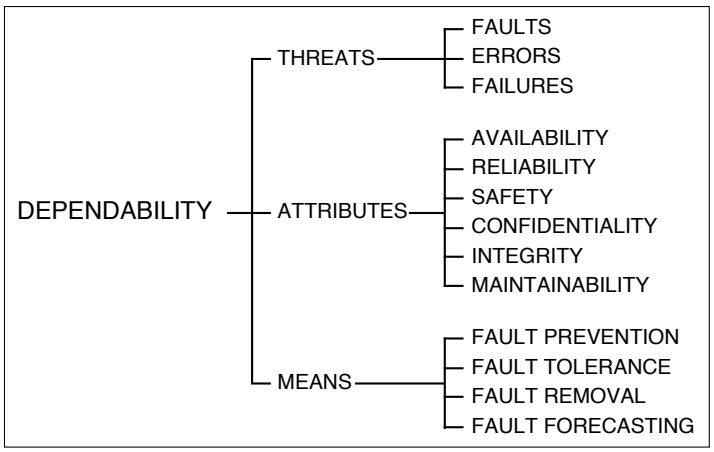



Figure 2.1. The dependability tree, proposed by Avizienis et al. in [20, 21].

Faults and their sources are several: their classification, according to six major criteria proposed by Avizienis et al. in [20, 21], is presented in Figure 2.2. Considering the research activities described in this PhD thesis, the persistence is an important criterion, discriminating which reliability method has to be implemented in the electronic systems domain. For example, in case of **permanent fault** (i.e., hardware malfunction that always occurs when particular conditions exist), the system has to detect it (by means of testing techniques), localize the fault itself (or the module containing it) through diagnostic procedures, and finally avoid the usage of the faulty module or of the complete system, substituting it with a spare one. In the other hand, a **transient fault** is a kind of fault that does not damage the device: alpha and beta particles from packaging material and/or neutrons from cosmic rays can invert a bit in SRAM cell, dynamic latch, or gate. This phenomenon has become relevant in the last decade, due to the increasing number of transistors combined with a decreasing feature size of the transistors themselves, and a reduced chip voltages and noise margins.

Finally, the errors prodced by intermittent faults (Figure 2.3) are usually called **soft errors**; the **intermittent faults** are the result of a class of faults composed of elusive developmental faults and transient physical faults.

### 2.1.2 The attributes (focus on reliability)

The most important attributes of the dependability definition addressed in this PhD thesis are reliability, availability, and safety.

In the IEEE Standard Computer Dictionary the **reliability** has been defined as "the ability of a system or component to perform its required functions under stated

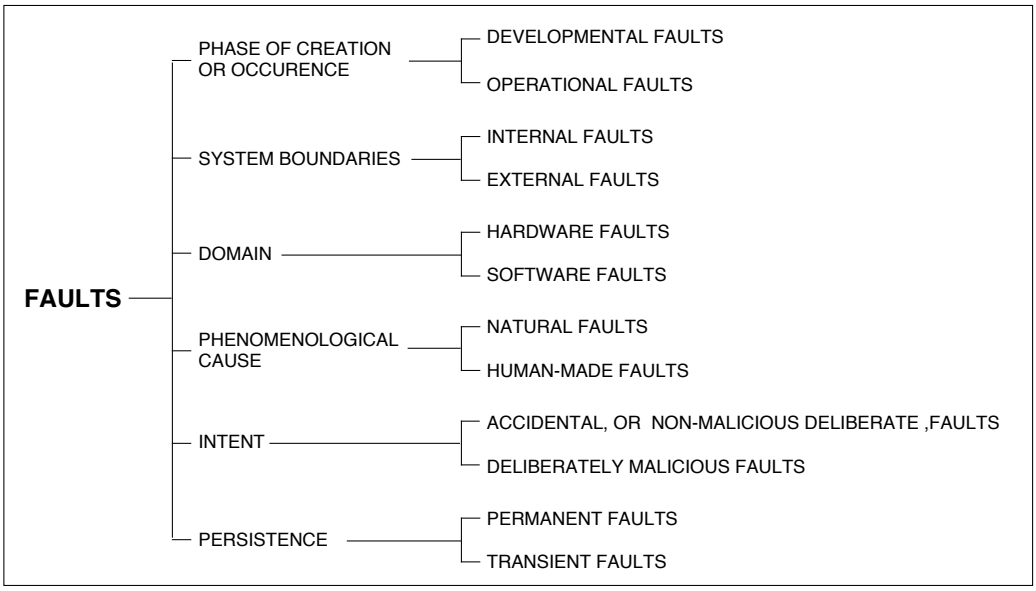



Figure 2.2. Elementary fault classes, proposed by Avizienis et al. in [20, 21].

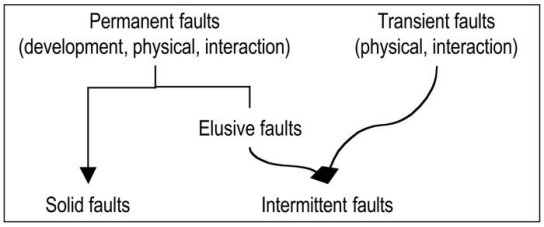



Figure 2.3. Intermittent fault sources, as proposed by Avizienis et al. in [20, 21].

conditions for a specified period of time" [23]. Many research activities have been proposed in the last years in order to improve the reliability of electronic devices; in this context, *reliability growth* (i.e., the improvement in reliability that results from correction of faults), and *reliability model* (i.e., model used to estimate, measure, or predict the reliability of a system ) are the main metrics that should be considered [23].

From a mathematical point of view, the reliability R is the likelihood of a system to work in a proper way up to time *t*. A commonly used model to express the reliability is the exponential distribution: the failure rate $\lambda$ is assumed as constant and the reliability R(t) is given by the equation 3.1.

$$R(t) = e^{-\lambda t} \tag{2.1}$$

From a purely statistical point of view the exponential distribution is a natural choice for representing times-to-failure of a unit during its useful life period [24]. One of the major feature of this model relies in the fact that it is a memoryless model, i.e., the future behavior is not influenced by the past events. Consequently, the probability that a component fails in the near future is always the same, and does not depend on its current age. Every instant is like the beginning of a new random period, which has the same distribution regardless of how much time has already elapsed. Exponential distribution is also very convenient because it is easy to combine failure rates of independent components (e.g., belonging to the same SoC) to find a reliability model of a complex system [22].

One of the drawback associated to the exponential distribution is that it not always appropriate to model the overall lifetime of technical devices, because their failure rates are not constant: more failures occur when the considered systems are very young or very old. The life cycle of a population of semiconductor devices, for example, can be graphically represented with a curve called *bathtub curve* (Figure 2.4), which models the failure rate vs. time [24]. This curve is created by mapping the rate of the so called "early infant mortality", the rate known as useful life (normal life) or random failure, and the rate of failures due to the wear out of the device. The earliest period, with steepest part of the curve, has the highest but decreasing failure rate; the lowest failure rate, instead, is related to the central part of the chart, corresponding to the normal life period of the device; finally, the rightmost part of the graph, where the curve goes up quickly, represents the increasing failure rate when reaching the end of life, due to intrinsic material issues and accumulative electrical or mechanical stresses.

Another attribute related to the dependability definition is the **availability**: often expressed as a probability, it represents the degree to which a system or component is operational and accessible when required for use. In high availability applications, a metric known as nines, corresponding to the number of nines following the decimal point, is used. For example, "five nines" means 0.99999 (or 99.999%) availability [23][25].

Finally, the **safety** is defined as the absence of disastrous consequences on the users and the environment, when a system is employed.

As general consideration, several other dependability attributes have been defined that are either combinations or specializations of the six basic attributes listed above. For example, **security** is the concurrent existence of availability for authorized users only, confidentiality, and integrity. Moreover, the characterization of a system reaction to faults could be done, for example, implementing **robustness**, i.e. dependability with respect to erroneous inputs.

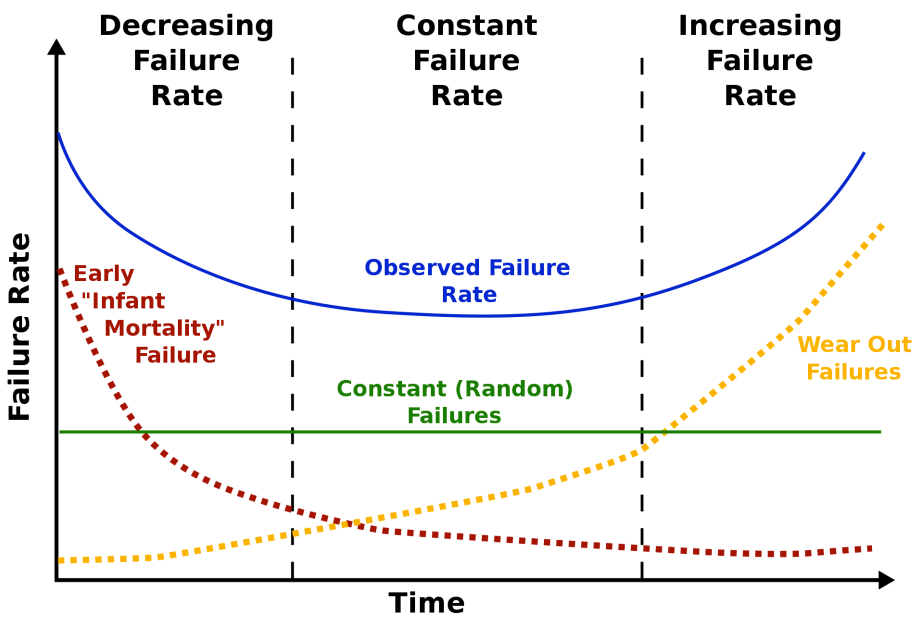



Figure 2.4. The Bathtub curve, representing the failure rate vs. the time of electronics devices.

### 2.1.3 The means

The development of a dependable computing system requires a combined utilization of a set of four techniques: **fault prevention**, to prevent the occurrence or the introduction of faults; **fault tolerance**, to deliver correct service in the presence of faults; **fault removal**, to reduce the number of faults; and **fault forecasting** to estimate, among the others, the present number, the future incidence, and the consequences of faults.

In the context of this PhD thesis, we address only the fault tolerance. IEEE defines **fault tolerance** in two complementary ways: (1) the ability of a system or component to continue normal operation despite the presence of hardware or software faults. (2) The number of faults a system or component can withstand before normal operation is impaired [23].

Fault tolerance is a recursive concept: it is essential that the mechanisms that implement fault tolerance should be protected against the faults that might affect them. Popular examples are the voter modules employed in redundancy-based fault tolerance systems, self-checking checkers systems, etc.

## 2.2 Reliability characterization methods

Nowadays, testing methods are essential for electronic systems to detect both latent hardware fault and new faults appearing in logic resources and/or in memory modules. In general, **testing** is the traditional procedure aimed at measuring the output response to specific signals applied at the input of the Device Under Test (DUT), comparing the obtained results with the known fault-free response (also called golden response). In this context, the input signals are called **test patterns** or **test vectors**, while the golden response is traditionally obtained through simulations. Different testing strategies exist, along with different applicability moments within the system device lifetime, in order to characterize its reliability.

### 2.2.1 Structural and functional test

One of the major test classification is between structural and functional test [26]. **Structural test** category includes all the techniques that take advantage of deep knowledge of the internal structure of the DUT; this kind of test check that each element of the device is working as expected. In general, the test patterns are simple, since they target one element, so they can highly benefit from automation. The main drawbacks of structural test are the lack of an overall view of the entire DUT, and the issues to assess the device performance. **Functional test** methods, instead, do not need any information about the internal architecture, but they use functional specification of the DUT; in practice, a functional test control that the entire device behaves as expected, assessing also the performance. The main drawback is related to the test stimuli generation, which require a lot of expertise that relies on the test engineer [22].

### 2.2.2 Manufacturing and on-line testing

Another possible test classification is based on the lifetime phase in which a test is applied.

**Manufacturing testing** allows the evaluation of the performances of the target device at production time. The today semiconductor devices are very sensitive to impurities and particles: to manufacture these devices it is necessary to manage many processes while ensuring an optimal "cleaning level". Consequently, even if the device design was verified to be correct, there is no guarantee that the manufactured device is compliant with the design requirements. Therefore, manufacturing testing is required. The manufacturing test main goal is to classify faulty from good devices. Additionally, it can help in determining if there is any phase of the fabrication process that systematically introduce a defect in the produced chips, by performing some kind of diagnosis [22]. Tests are executed in various stages, in order

to prevent expending money, time and efforts in realizing a faulty device. According to these features, two are the most important test approaches used today in the manufacturing process:

- *Wafer Sort & After Packaging Final Test*: wafer test (also called as wafer sort) is performed to all dies present on the wafer, looking for functional defects; this is typically done by applying special test patterns to them. As a consequence, the faulty dies are marked, so only known good dies will be packaged. Once the fault-free dies have been packaged, a final test is performed aimed at verifying that the packaging process itself does not affect the devices, and also that the pin connections were correctly wired.

- *Burn-in*: In general, semiconductor manufacturers want to avoid introducing to the market devices that will fail in an early stage (i.e., in the "Infant Mortality Failure" period described in the Bathtub curve - Figure 2.4). The Burn-in test is performed to discover this phenomenon: it consists in subjecting the devices to particularly stressing conditions (e.g., extreme temperatures and supply voltage) for a specified period of time. These stressing conditions work as a "time machine" for the chips under test. After the burn-in period, the final test is performed again to the devices. In this way, chips that were subject to infant mortality, will not go into the market. In order to characterize the Bathtub curve (Figure 2.4), the complete burn-in process, that is covering all three stages of a device lifecycle, is to be performed on a statistical sample of products.

The two manufacturing tests, described in the previous paragraph, are executable by means of a special machines in charge of applying the tests at the DUT. These machines are commonly called **Automatic test Equipment (ATE)** and are composed by several parts, where the controlling of the different instruments is demanded to a computer. ATEs are widely used in electronic industry where automation is needed for performing measurements and evaluation of test results during manufacturing and maintenance [27].

The main drawbacks of ATEs are related to the current semiconductor development trend: with more than 1 billion transistors in a 22 nm technology microprocessor, the complexity of the DUT is increasing dramatically. Related to this phenomenon, the minimum number of test patterns is increasingly with the same tendency; consequently, even with the most fast and efficient ATE, apply these test patters to Very Large Scale Integration (VLSI) circuits it may take thousands of years to fully excite all possible states of a DUT. Moreover, due to complexity, not all possible states may be reachable just manipulating the primary inputs of a device [22].

A solution to this problem is the **Design for Test (DfT)**: the circuits are designed in a way that make them efficiently and easily testable. More in particular, this test enhancements can be done exploiting the sequential parts of the DUT. This design modifications help the circuits to be tested with an acceptable fault coverage, in an acceptable time, and, in addition, to overcome the problem of test access. The added special features can allow the control and observation of deeply embedded parts of the circuit under test to verify circuit functionality and detect fabrication defects. Performance loss due to their inclusion must be minimal in normal (non-test) operating mode. The most popular DfT techniques are structured scan-based approaches (e.g., scan chains [28]), but there are also other useful special purpose techniques to be applied at the design stage (e.g., test point insertion [29]).

Finally, a further step in aiding manufacturing testing is based on generating test patterns and evaluating their results directly on-chip; this approach is called **Built-In Self-Test (BIST)**, and it requires some dedicated hardware modules (within the DUT) aimed to generate the patterns, to apply the patterns themselves, and to analyze the results. In practice, in BIST the typical functions performed by an external testers (i.e., the test generation and response analysis) are carried out on-chip. Consequently, the main role of the tester is related to the management of the test enable signals, aimed to indicate at the chip under test that the BIST procedure has to be executed; then, a signal is generated by the chip under test in oder to specify at the tester that the chip itself pass or fail the test procedure [30].

**On-line testing** has been defined as the process where faults are detected and/or corrected while the system is working in its natural environment. Nowadays, on-line testing is essential for modern microprocessors to detect both latent hardware defect and new defects appearing both in logic and memory modules. On-line test is required by most safety-critical applications, since a faulty behavior could lead to customers' inconveniences, economic loss and even casualties. In **concurrent on-line** testing the detection of operational faults is performed at the same time the device is working. This means, more precisely, that the detection of operational faults must be performed keeping the system in normal or safety operational state. Concurrent online testing usually exploits specific hardware, such as data redundancy and voters, with a costly overhead for the final system. In the other hand, in **non-concurrent on-line** testing, the detection of operational faults is performed while the normal operations are temporarily suspended. The test procedure is either interruptible or composed of small subparts, and a test manager schedules it in order to minimize its intrusiveness. Non-concurrent testing usually needs less additional hardware. Finally, another kind of on-line test is **start-up** testing, that is executed when a system is booting, before putting it to actual service. This implies no consequence for the application performance, as it is not yet running; moreover, on-line testing is useful for systems that are consistently restarted throughout their mission time, like the Power-On Self-Test (POST) of a computer memory [31].

## 2.3 Different devices, different reliability aspects

In this section, different test methods and procedure are presented, in order to overview the today state-of-the-art techniques applied to different electronic devices. The common features of these approaches is the effort to reduce cost. In fact, with the advent of SoC, the low-cost concept has become a common denominator among test generation and test application. With the last generation of semiconductors devices, test cost is becoming increasingly percentage of Cost of Build (COB). This is even critical in low cost markets like consumer devices. In order to reduce this phenomenon, different strategies could be adopted: they are mainly based on DfT features included on a chip to reduce test costs without impacting its effectiveness [32]. The costs of SoC test procedure involve many factors, that are primarily test pin count, application frequency, and tester memory depth for pattern and data storage.

In recent years, many efforts have been done in order to decrease test cost [33]. Two are the most important approaches that have to be considered: the low-cost scan-based methods and the low-cost self-test techniques. Low-cost scan-based test approaches rely on design techniques that allow the minimization of the number of tester channels and the tester frequency requirements [34]. In addition to (or in substitution of) traditional scan cells, these techniques adopt suitable DfT features such as decoders and phase-locked loop (PLL)-based circuitries; the former addresses pin count minimization, while the latter permits moving deterministic patterns in the chip at reduced speed, thus applying them at higher frequency [22].

In the other hand, self-test procedures address frequency requirement mitigation, since it normally exploits internal or independent clock supply resources that do not request any external intervention. In particular, low-cost self-test approaches may be based on infrastructure intellectual property (I-IP) or may employ functional parts of the DUT itself [35]. The key point is that, once launched, a self-test procedure is autonomously applied until it ends. The two main categories of low cost self-test are: Software-Based Self-Test (SBST) and BIST. The details related to the SBST approaches are listed in the Section 2.4, while the BIST features have been described in Section 2.2.2. Moreover, there exist also test procedures exploiting both SBST and BIST principles; they consist of at least three parts: (1)a preliminary initialization phase aimed at loading at low-frequency the test microcode and/or setting parameters; (2) a self-test execution at high frequency; and (3) result download at low frequency.

Finally, the test pattern compression is another strategy aimed at saving test costs. Such technique is intended to reduce the overall size of the test vectors to be applied to the DUT, thus reducing the test time. Several techniques have been developed in this direction: they mainly consist in encoding the test vectors using as few bits as possible. Compressed data are then reconstructed, or decompressed,

by ad hoc hardware decoders/decompressors placed on a chip or on the tester [22].

### 2.3.1 Microprocessors

Among the different issues related to the test of electronic devices, testing embedded microprocessors is one of the most challenging tasks to be performed at the end of the SoC production cycle. The key point is that execute an exhaustive testing is not a feasible solution: in fact, apply all the possible instructions, in all their feasible addressing modes, with all the potential data combinations, in all possible orders, starting from all the reachable initial states, may take extremely long testing times, thus making impossible this test approach. In order to decrease these issues several approaches are today used.

As already stated in the previous section, the today testing strategies are largely based on the introduction of additional DfT hardware devoted to perform structural testing; scan chains and BIST are well known and very popular solutions. However, functional methods, such as SBST are today increasingly used.

Whens the testing of a microprocessor is addressed, there are several aspects to be taken onto account: the fault coverage (i.e., the percentage of fault that can be detected during the test), the ability of the test to be executed at maximum speed, an acceptable test time, the guarantee of independence of the different cores to be tested in the system, the area overhead introduced by the DfT module, etc... Moreover, the execution of an on-line test implies others constraints to be considered: the preservation of the previous microprocessor state, the ability to provide the diagnostic information, and the fulfilling of the timing constraints considering that generally the time slice assigned to test execution is shorter than the test program itself.

Nowadays, SBST and, in general, microprocessor functional testing approaches, are gaining again popularity after many years in which scan-based approaches have been largely preferred [36]. This phenomenon is due to many reasons: the latest technologies show timing-related faulty behaviours that can be investigated in a more accurate manner using SBST, since this technique allows executing a test at the chip speed; moreover, SBST techniques are cost and time effective, since they request few tester channels and limited memory amount on the tester; finally, the self-test program can be stored in a non-volatile memory and activated also during the component lifetime to perform on-line testing. As an example, in the automotive fields, emerging standards and regulations (like the ISO 26262 [37]) require high fault coverage and in-field testing (e.g., periodic on-line testing, power-up testing, etc..) that can be more easily implemented through SBST.

Beside the analysis presented above regarding the microprocessors test, a particular attention must be given at the problem of testing embedded processors cores: their wide diffusion is increasing the challenges in the test arena. Modern designs

include complex architectures that further increase test complexity, as pipelined and superscalar designs. Single sub-components in a microprocessor may be autonomously tested by accessing their inputs and outputs through specific test buses built in the chip, and by applying specific test patterns, or resorting to integrated hardware, such as with Logic Built-In Self-Test (LBIST) [38]. Within this framework, a critical issue is the system integration test, where the whole processor and the interconnection between different modules have to be checked. At this level, one suitable possibility is to let the processor execute carefully crafted test programs [22].

### 2.3.2 Memories

Semiconductor memories are today commonly used to store programs and huge volume of data in all the digital systems. For many current applications, the today system performance is strictly related to the number, size, and speed of its embedded memory modules [41]. In fact, microprocessors rely more and more on efficient memory devices as a support of the extreme computational demand. This phenomenon is further highlighted when multicore computational units are considered. As an example, in [39] is stated that, when multithreaded multicore processors are considered, caches are organized in multiple levels and multibank architectures that occupy almost 90% of the relative chip area. Thus, high-quality memories on-line testing in modern processors is essential. As countermeasure, Memory Built-In Self-Test (MBIST) schemes [40] are integrated in embedded memories for manufacturing testing purposes; anyway, they can also be reused for on-line testing.

In literature, several research works have been proposed in the last decade addressing the memories test. In this section the main aspects related to the Software-based memory tests are presented. In general, the term Software BIST is used to denote a test solution targeting memories embedded in a SoC, based on performing the test through a suitable program executed by a processor inside the SoC itself. This program is in charge of executing the sequence of accesses to memory (for both read and write operations) according to a given test algorithm, typically known as March algorithm [42]. The software-based test approaches are suitable to address the Back-to-Back (BtB) test execution: several published works underline, in fact, that various memory defects existing in new technologies require the test accesses to be performed at the maximum speed (or at least at-speed). A typical software-based test procedure is based on the usage of suitable instructions (i.e., LOAD and STORE instructions); moreover, implementing a March element in any assembly language requires addressing some critical issues, e.g., the loops management, the result evaluation, and the generation of the memory address [22].

Among the various types of memories, Static Random Access Memories (SRAM) are widely used in embedded and in high speed applications (e.g., the cache memories

of microprocessors). The main challenge of testing SRAMs consists in providing realistic fault models and practical test solutions with minimal application time. Two are the most important fault models associated to the memory environment: *static faults*, such as stuck-at, transition, and coupling faults which require at most one read or write operation to be sensitized, and *dynamic faults*, that are sensitized by more than one read or write operations. In [41] the authors stated that in order to address the latter fault category, an elaborated combination of operations, data background, and addressing sequence is required. Consequently, classic SRAM test solutions are not sufficient to cover a new type of faults that are emerging in the last nanometer technologies and that are a consequence of the shrinking geometries (i.e., the dynamic faults).

## 2.4 Software-Based Self-Test and Software-Based Diagnosis

As already explained in the previous Section, nowadays, SBST and, in general, microprocessor functional testing approaches are increasingly adopted. Two are the main reasons behind this phenomenon: (1) the latest technologies show timing-related faulty behaviors that can be investigated in a more accurate manner using SBST, since this technique allows executing a test at the chip speed; moreover, (2) SBST techniques are cost and time effective, since they request few tester channels and limited memory amount on the tester.

In general, the principle of Software-Based Self-Test (SBST) is to run functional test patterns, based on the processor Instruction Set Architecture (ISA), i.e., exploiting processor resources to test the processor itself and the components around it [14, 22]. Practically, SBST consists in forcing the embedded processor to execute a special purpose test program, i.e., a sequence of instructions capable of deeply exciting possible device faults and propagating the fault effects to some observable locations. The SBST program can either be stored in a non-volatile memory, or uploaded in a RAM immediately before the test execution. SBST does not require circuit modifications (therefore, making it particularly suitable for the test of third-parties cores, which can hardly be modified and which hardware model is not always available) and may offer excellent defect coverage, since it is executed at the same speed of the normal applications. Moreover, it can be performed both at the end of the production process, and during the operational phase (e.g., for periodical on-line testing).

Many efforts have been invested in the past three decades on the development of functional and SBST topic. Academy [14, 43] and industry [44] have proposed many techniques solving general test problems and giving solutions to functionally

reach the highest possible fault coverage. Considering the applicability fields, SBST is included in the manufacturing flow of microprocessors, as described in [45] where an industrial case study has been presented. Moreover, SBST is also used to identify faults during normal operation of the device by performing SBST on-line testing. Besides the advantages of SBST on-line testing explained previously, some additional aspects have to be taken into account when dealing with test program generation for on-line purposes. The test program must first be able to properly excite the considered processor modules, and then, once the results have been produced, it must turn them observable to observable locations; the latter operation has to be executed in a transparent way that does not affect the normal operation of the mission application. The most important constraints for on-line testing are (among the others):

- Preserving the processor status: the status of the interrupted mission (i.e., the processor status register content) has to be saved and restored at the end of the test.
- Execution time: duration must be as short as possible in order to, for example, avoid the real time features of the device.
- Memory content: it is crucial to prevent test programs from overriding information belonging to other processes (e.g., the mission program). Code and data memory belonging to the test procedures must be clearly defined and limited considering the system memory map of the device [22].

Current state-of-the-art techniques include different strategies able to generate test programs resorting to manual and automatic approaches. In general, the new literature methodologies are generic enough to be easily adapted to various processors belonging to different application domains. As an example, new grading techniques to rapidly characterize test programs have been proposed in [44]. However, reducing costs related to both test program fault grading and test application is still an open issue[22]. The efficient generation of the SBST programs is the main critical issue; an alternative approach aimed at reducing the efforts related to the test program development is based on the usage of particular tools able to generate pseudo random pattern [45, 46].

Another recent trend in SBST research is to scale SBST techniques in multithreaded multicore architectures. In [47] the authors propose multithreaded (MT) SBST methodology able to generate an efficient multithreaded version of the test program, and able to schedule the resulting test threads into the hardware threads of the processor, in order to reduce the overall test execution time and, at the same time to increase the overall fault coverage. This methodology has been demonstrated in the OpenSPARC T1 processor model which integrates eight CPU cores, each one

supporting four hardware threads. More in general, the effective application of SBST to multithreaded multicore architectures creates significant challenges:

1. porting of existing test programs from the single-threaded, unicore case to efficiently test all the individual cores;

2. providing sufficient fault coverage for the thread-specific control logic, which is a significant portion of the control logic in the multithreaded architectures;

3. exploitation of thread-level and core-level parallelism to reduce test execution time;

4. avoiding the scaling of test program memory footprint with the number of cores.

Given their high computational power, General Purpose Graphics Processing Units (GPGPUs) are increasingly adopted: GPGPUs have begun to be preferred to CPUs for several computationally intensive applications, not necessarily related to computer graphics. In this context, new efficient test methodologies are mandatory. In [15] a new SBST aimed at detecting and localizing hardware faults in GPGPUs is presented. This method is based on a set of GPU kernel programs containing SBST procedure; once that a kernel is executed, a signature of the test and an identifier of the Streaming Multiprocessor (SM) in which it has been executed, are provided at the CPU. In this way, the CPU is able to avoid the usage of the faulty SM for the future computations.

The SBST technique can be even used for diagnostic purpose [48, 49, 50]. The main common concept that drives several research work within this context is the following: "Given a set of test patterns, two faults can be distinguished if they exhibit different behaviors at the Primary Outputs (POs)." In general, with a software-based diagnosis approach, using functional information, it is easier to generate test programs to distinguish certain faults without structural analysis or fault simulation. For example, faults on the data bus may be distinguished by loading and storing specific values; faults in the instruction decode unit may be distinguished by executing different instructions. However, it should be pointed out that due to the complex structure of a processor, it is impossible to foresee all faults detected by a test program. Thus, given an observed test response, the fault candidates may be several (some expected and some other unexpected). This is the main issue related to the generation of software-based diagnosis methods, and this also represents the main motivation behind the research works that have been developed in the area.

## 2.5 Overview on the Fault Tolerance Techniques

In general, the fault tolerance techniques aim to achieve robustness and dependability in a system through the usage of error detection and system recovery methods. Among the different fault tolerance approaches, the first classification that has to be introduced is between **proactive** and **reactive** techniques. The proactive ones are aimed to detect a possible fault before it appears in the system; the reactive ones, instead, are aimed to manage the fault in order to reduce as much as possible the effects introduced in the system. Moreover, the reactive techniques could be further classified in **error processing** (i.e., remove errors from the computational state) and **fault treatment** (i.e., prevent faults from being reactivated) [51].

In the rest of this section and of this PhD thesis the reactive techniques finalized to error processing and to fault treatment are addressed.

The basic idea behind fault tolerance is the use of **redundancy**. The idea of using redundancy to construct reliable systems from unreliable components was first described by Von Neumann in 1956 [52]. Redundancy (based on multiple copies) is used to detect faults and mask failures: Avizienis and Kelly [56] suggest that the different types of redundancy possible in a computation are repetition, replication (hardware), and logic (software) [54].

In Figure 2.5 the main techniques involved in fault tolerance are shown. The choice of error detection, error handling and fault handling techniques, and of their implementation is directly related to and strongly dependent upon the underlying fault assumption: the class(es) of faults that can actually be tolerated depend(s) on the fault assumption that is being considered in the development process and, thus, relies on the independence of redundancies with respect to the process of fault creation and activation. A (widely used) method of achieving fault tolerance is to perform multiple computations through multiple channels, either sequentially or concurrently [21, 22]. When tolerance of physical faults is required by the system, the channels may be of identical design, based on the assumption that hardware components fail independently. However, this approach is not completely suitable for the tolerance of solid development faults, which necessitates that the channels implement the same function via separate designs and implementations methodologies [55], i.e., through **design diversity** [56].

**Fault masking** (a.k.a. **masking**), results from the systematic usage of compensation. Such masking will conceal a possibly progressive and eventually fatal loss of protective redundancy. So, practical implementations of masking generally involve error detection (and possibly fault handling), leading to masking and recovery [21].

**Preemptive error detection and handling**, possibly followed by fault handling, is commonly performed at system power up. It also comes into play during operation, under various forms such as spare checking, memory scrubbing, audit programs, or so-called software rejuvenation, aimed at removing the effects of software

aging before they lead to failure [21].

The provision within a component of the required functional processing capability together with concurrent error detection mechanisms leads to the notion of **self-checking component**, either in hardware or in software; in the context of this PhD thesis, an increasingly used self-checking software techniques, a.k.a. Software-Based Self-Test is introduced and applied to different research fields (Section 2.4).

Finally, fault tolerance could be considered also as a recursive concept: it is essential that the mechanisms that implement fault tolerance should be protected against the faults that might affect them. Examples of such protection are voter replication (in case of redundancy with a majority voter mechanism is used), self-checking checkers, "stable" memory for recovery programs and data [21].

Upon the fault tolerance techniques and approaches summarized above, a very important term that should be introduced when the fault tolerance is address is **coverage**: in fact, not all fault tolerance techniques are equally effective; the measure of effectiveness of any given fault tolerance technique is called its coverage [21].

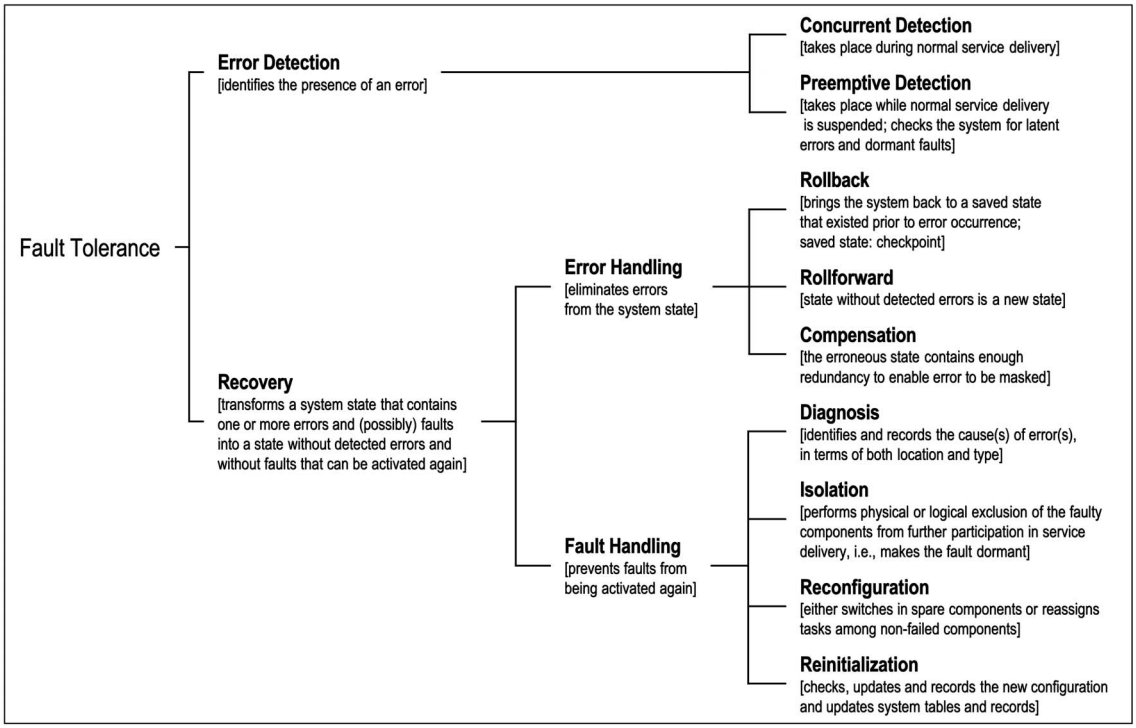



Figure 2.5. The Fault Tolerace techniques, as proposed by Avizienis et al. in [20, 21].

## 2.6 Automatic Test Equipment (ATE)

The **Automatic Test Equipments (ATE)** are machines that perform tests on a device, using automatic approaches finalized to quickly perform measurements and evaluate the test results. An ATE can be a simple computer controlled digital multimeter, or a complex system containing several test instruments (real or simulated electronic test equipment), capable of automatically testing and diagnosing faults in sophisticated electronic packaged parts or on Wafer testing, including System-On-Chips and Integrated circuits [57]. In addition to the semiconductor industry, ATE is used in the automotive, medical equipment, airplane, and other manufacturing industries. ATE also conducts stress testing with minimal human interaction; moreover it is considered cost efficient only for high-volume testing: in fact, the wide application domain and the complexity of its equipments make ATE to be expensive devices.

Considering all the manufacturing tests described in the previous sections of this thesis, an ATE is in charge of applying some of these tests to the DUTs: typically, it is able to perform the testing process. More in particular, the basic component of a generic ATE are:

- a computer in charge of controlling the process and the different instruments that will be connected to the DUT;
- a variable number of instruments, which performed the desired measures, applying stimulus and collecting results;
- a fixture that is the physical place holder for the DUT, where it connects to the ATE;
- eventually, a handler to place the packaged chips in the fixture; or probes that connect directly to the DUT when wafer testing is performed. It can be from as simple as computer controlling a multimeter, to a complex equipment, performing many different analogue and digital measurements [22].

Among the different ATE types, the most used are listed below [58]:

**PCB inspection system** PCB inspection is a key element in any production process and particularly important where automatic pick and place machines are involved. Manual inspection has been used for many years, but it was always unreliable and inconsistent. Nowadays, considering the printed circuit boards that are considerably more complicated, manual inspection is not a viable option. **Automatic Optical Inspection (AOI)** is widely used in many manufacturing environments. It is essentially a form of inspection, but achieved automatically. This provides a much greater degree of repeatability

and speed when compared to manual inspection. AOI is particularly useful when situated at the end of a line producing soldered boards. Here it can quickly locate production problems including solder defects. As AOI systems are generally located immediately after the PCB solder process, any solder process problems can be resolved quickly and before too many printed circuit boards are affected. Moreover AOI takes time to set up and for the test equipment to learn the board; consequently, It is ideal for high volume production. **Automated X-Ray inspection (AXI)** has many similarities to AOI. However, with the advent of BGA packages it was necessary to be able to use a form of inspection that could view items not visible optically. AXI systems can look through IC packages and examine the solder joints underneath the package to evaluate the solder joints.

**ICT In circuit test** ICT not only looks at short circuits, open circuits, component values, but it also checks the operation of ICs. Although ICT is a very powerful tool, it is limited by lack of access to boards, as a result of the high density of components in most designs.

**JTAG Boundary scan testing** Boundary scan is a form of testing that has come to the fore in recent years. Also known as **Joint Test Action Group (JTAG)**, or by its standard IEEE 1149.1 [59], boundary scan offers significant advantages over more traditional forms of testing and it has become one of the major used tools in automatic testing. The main reason behind the boundary scan testing developed was to overcome the problems of lack of access to boards and integrated circuits for testing. Boundary scan overcomes this, by having specific boundary scan registers in large integrated circuits. With the board set to a boundary scan mode, serial data registers in the integrated circuits have data passed into them. The response and hence data passing out of the serial data chain enables the tester to detect any failures. As a result of its ability to test boards and even ICs with very limited physical test access, Boundary Scan / JTAG has become very widely used.

**Functional testing** Functional test can be considered as any form of electronics testing that exercises the function of a circuit. There are different approaches that can be adopted, dependending on the type of circuit (RF, digital, analogue, etc), and on the degree of testing required. The main approach today used is **Functional Automatic Test Equipment (FATE)** [60].

**Combinational test** Nowadays, no single method of testing is able to provide a complete solution; in order to overcome this issue, various ATE systems incorporate a variety of test approaches. These combinational testers are generally used for printed circuit board testing. As a consequence, a single electronics

test is able to gain a much greater level of access for the printed circuit board test, and the test coverage is much higher. Additionally, a combinational tester is able to integrate a variety of different types of test without the need of moving the board from one tester to another: in this way a single suite of tests may include In-circuit testing as well as some functional tests and then some JTAG boundary scan testing.

In the context of this PhD thesis, a sort of ATE has been designed and implemented; the final goal of this system is the automatic test of an automotive timer module, i.e., the Generic Timer Module (GTM). The developed ATE is based on an FPGA device, and it ensures the possibility to test the real time characteristics of the timer module under test. The detail of this system are explained in Section 5.

# Chapter 3

# The proposed SBST and diagnostic methods for VLIW processors

In this section, a comprehensive explanation of the main features of a generic VLIW processor are proposed, along with the details of the VLIW processor used as case study (i.e., the VEX processor, developed by HP [61]). Then, the most important SBST and diagnostic methods developed in the context of this PhD thesis are presented, together with the most important experimental results gathered by applying the proposed techniques.

## 3.1 Motivation and introduction

Mainly due to the continuous scaling in the semiconductor manufacturing process and to the increasingly high operation frequency of integrated circuits, processor chips face growing testability problems. Moreover, since the production processes are highly stressed, phenomena like metal migration or aging of the circuit may increase the occurrence of permanent faults into the systems, even during the circuit operational phase. For these reasons, new test solutions are being investigated in order to provide high fault coverage with acceptable costs (e.g., in terms of test time, silicon area overhead and required test infrastructure).

A promising approach for processors and processor-based systems (e.g., Systems on a Chip, or SoCs) corresponds to the so called Software-Based Self-Test (SBST) [36]: the basic idea is to generate test programs to be executed by the processor and able to fully exercise the processor itself or other components in the system, and to detect possible faults by looking at the produced results. One of the main advantages of SBST lies in the fact that it does not require any extra hardware;

therefore, the test cost is reduced and any performance or area penalty is avoided. Moreover, SBST approaches allow at-speed testing, and can be easily used even for on-line test. For these reasons, SBST is increasingly applied for processors and SoC testing, often in combination with other approaches.

Among the various microprocessor architectures, Very Long Instruction Word (VLIW) processors have been demonstrated to be a viable solution especially for applications demanding high performance while exposing a considerable amount of parallelism, such as several Digital Signal Processing algorithms used in multimedia and communication applications [62]. VLIW processors are currently adopted in several products, in particular for embedded applications, and the problem of testing them is increasingly relevant.

VLIW processors are characterized by a pipelined architecture with multiple Functional Units (FUs). Unlike superscalar processors, VLIW processors do not include any significant control logic, since instruction scheduling is completely performed by the compiler. This implies that the hardware complexity is far lower than for superscalar processors, while the compilation steps become more complicated. Consequently, the control hardware of the processor is much more easily testable than in other processors (e.g., the superscalar ones). Another key feature of VLIW processors is the instruction format. In fact, VLIW processors are characterized by grouping several instructions (named micro-instructions) into one large macro-instruction (also called bundle), where each micro-instruction within the bundle is executed in parallel distinct computational units, referred to as Computational Domains. In VLIW architectures the scheduling of the operations is fully performed at compile time: the compiler is responsible for allocating the execution of each instruction to a specific Functional Unit. Due to these characteristics, VLIW processors are also suitable for safety-critical systems adopted in space, automotive or rail-transport fields which require computationally intensive functionalities combined with low power consumption. As an example, the processor Tilera TILE64TM, composed of several VLIW cores, has been evaluated to efficiently perform image analysis on-board a Mars rover in support of autonomous scientific activities [12, 13].

A few SBST approaches have been proposed in the literature in order to properly test VLIW processors against permanent faults: some of them rely on suitable instructions belonging to the processor instruction set to apply the test patterns previously generated by a conventional automatic test pattern generator (ATPG) targeting each internal component [63]. Although effective, these methods have several drawbacks: first of all, transforming the test patterns generated by the ATPG into test programs is not always straightforward; secondly, the resulting test programs are far from being optimized, especially in terms of test length; finally, the attainable fault coverage is not always as high as it may be required.

In [64] a specific issue which must be faced when testing a VLIW processor is addressed: the register file characteristics are different than in other processors, since

it must be accessed from different domains. In the same paper an effective solution to the test of VLIW register files is presented. In this section the generation of effective SBST test programs for the whole VLIW processor is addressed, characterized by minimal duration, minimal size, and maximal fault coverage. The proposed method starts from existing functional test algorithms developed for each single FU type embedded into the processor (e.g., ALUs, adders, multipliers, memory units). Although the characteristic of the FUs used within a VLIW processor are similar to those used in traditional processors, generating optimized code to effectively test these units is not a trivial task: in fact, by exploiting the intrinsic parallelism of VLIW processors it is theoretically possible to reduce the increase in duration and size of the test programs when the VLIW processor size grows. For example, testing the ALUs in the different Computational Domains can be performed in parallel, forcing them to perform the required computations in the same clock cycle, provided that a sufficient number of registers are available, and an effective method to check the results is devised. However, generating an optimized test program with minimal size and duration may require a significant manual effort, taking into account both the test algorithms for each Functional Unit, and the specific VLIW processor configuration: our test generation procedure provides an automatic solution for test program generation, once the processor configuration and the test algorithms for each FU are known.

VLIW processors do not include any specially designed hardware module (as it happens for other processor types), but are rather based on a combination of common Functional Units: exploiting this characteristic, our solution allows test program generation and optimization to be performed autonomously and automatically, without any manual effort. The test programs generated by the proposed method are highly optimized and exploit the VLIW processor features in order to minimize the test time and the test program size. Moreover, since the method is totally functional, it does not require the usage of any ATPG tool, nor the adoption of any Design for Testability (DfT) technique. In principle, the proposed scheduling technique is based on the same approach typically used by the VLIW compilers for optimization purposes. However, the use of a compiler is not feasible for optimizing a test program: in fact, in our case the optimization should maintain unchanged the Fault Coverage of the original test program, while a compiler typically optimizes the code by analyzing the function performed by the code (which in the case of a test program is meaningless) and selecting the most suitable resources to be used at each time step. For example, a typical VLIW compiler tries to optimize the parallelism of the instructions exploiting the VLIW resources without any external constraints; in the context of this PhD thesis, test programs are considered and thus the instructions composing each piece of code have to be executed in a specific Computational Domain and cannot be moved from one to another without modifying the corresponding Fault Coverage. More in general, the test of a specific unit

in a VLIW processor requires performing a well-defined sequence of instructions in a well-defined Computational Domain. If the test program is encoded in a high-level language and then the compiler is launch, there isn't any way for forcing it to generate a code which executes the given sequence of instructions on the functional units of a given Computational Domain. Consequently, it is not possible to use a VLIW compiler to generate the machine code for testing the processor, nor to use it to optimize the test code.

The proposed method has been experimentally evaluated on a VLIW platform based on the Delft University $\rho$-VEX VLIW processor [65, 66] which supports most of the features of industrial VLIW architectures. The achieved results clearly demonstrate the effectiveness of our approach on three different VLIW configurations: the required test time for the 4, 6 and 8 Computational Domains configurations of the $\rho$-VEX processor decreased of about 54%, 56% and 59% with respect to the corresponding non optimized solution, respectively; considering, instead, the size of the test programs, the reductions are 58%, 60% and 63%, respectively. The reached fault coverage, for all the processor configuration is about 98%.

Beside the test methods, a second research work developed in the context of this PhD thesis is related to the design and implementation of new diagnostic approaches. In fact, reconfigurable processors [79] are increasingly used in different domains. Their key characteristic lies in the fact that they can be easily configured to match the specific requirements of the target application, e.g., in terms of performance, size, and power consumption, thus possibly making them more convenient than traditional processors. VLIW processors [62] represent a popular choice among reconfigurable processors.

When dependability is a concern, dynamic reconfigurability is sometimes exploited: in this case the processor undergoes some test during the operational phase, aiming at detecting possible faults with minimal latency. The test can be activated either at a specific moment in time (e.g., at power on), or periodically. As soon as a permanent fault is detected, a diagnostic procedure is activated to locate the faulty partition, so that it can be substituted with a spare one, thus restoring the system integrity. This scheme requires first of all the availability of an effective test procedure, able to detect the highest percentage of possible faults while matching the requirements of a test performed during the operational phase (e.g., in terms of duration, size, invasiveness); when considering VLIW processors, some previous works in the area [72, 74] showed that these goals can be achieved resorting to an approach, in which a suitable test program is executed and the behavior of the processor during the execution (e.g., in terms of produced results) is observed. As already explained in the previous paragraphs, test based on such an approach is sometimes referred to as SBST, and some authors proved that effective SBST test programs can be generated even starting from RT-level descriptions [80]. The regular structure of VLIW processors may ease the task of generating the test program,

which is often cumbersomely hard for conventional processors. Some recent work demonstrated that test program generation can be even automated in the case of VLIW processors. The SBST approach is often preferred for the above purpose to the structural approach (e.g., based on scan) mainly due to its easier usage during the operational phase.

Once a fault has been detected, the application execution is typically suspended on the faulty processor and a diagnostic procedure is activated, whose goal is to identify the faulty partition out of those composing the processor: in this context each partition represents the minimal unit that can be substituted if faulty. Once again, diagnosis may be performed resorting to SBST, i.e., to the execution of a suitable test program, whose results allow identifying the faulty partition [69].

In this PhD thesis the issue of writing a diagnostic SBST test program for a VLIW processor is also addressed. The proposed approach is based on exploiting an existing test program (targeting to fault detection, only), and on applying a set of techniques for improving it so that it can hold sufficient diagnostic properties. The basic idea behind these techniques is to exploit the regularity and parallelism characterizing a VLIW processor: in particular, the technique proposed in this PhD thesis is based on splitting the original test program in small pieces (called fragments), and then modifying each fragment in such a way that it performs the same operation using different resources (e.g., different registers, or different ALUs). By checking which ones of the replicas of the original fragment (called brother fragments) generate a misbehavior, the faulty module could be identified. Since the basic motivation for this work is to support the design of highly dependable systems based on dynamic reconfiguration, the goal of our diagnostic approach is to identify the faulty module, rather than the specific fault responsible for a given misbehavior, as done in other works (e.g., [81]). A similar approach was followed in [82], where the issue of self-adapting the test so that it takes into account possible units which have been already labeled as faulty is considered. However, no one of the previous works give a systematic method to generate diagnosis-oriented test programs, as done in this PhD thesis.

The proposed method has been experimentally evaluated resorting to a sample VLIW processor [66]: an existing test program (developed with the approach explained in the next sections) has been modified and improved, thus obtaining a diagnostic test program whose characteristics (in terms of size, duration and diagnostic capabilities) have been evaluated and compared with those of the original test program. It has been proved that the method is able to significantly improve the diagnostic capabilities of the test program, and to allow the identification of the faulty unit in a high percentage of cases.

In the next sections the details of this research works are presented.

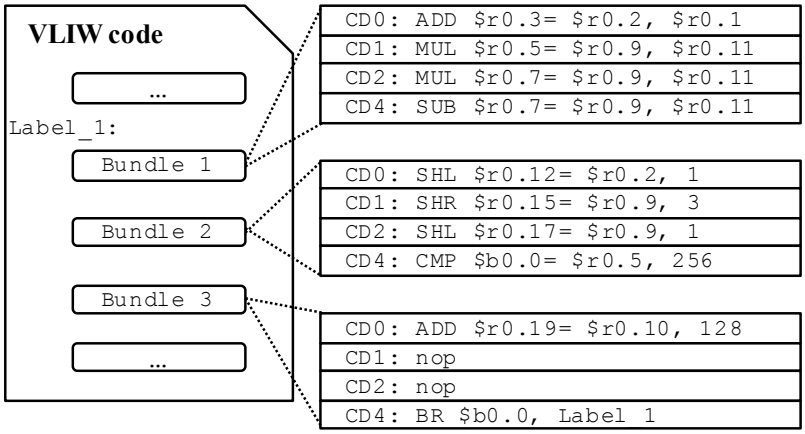



Figure 3.1. An example of VLIW code [61].

## 3.2 VLIW processors main features

A VLIW processor is characterized by the fact that all the operations are executed by parallel Computational Domains (CDs), each characterized by its own Functional Units (FUs); the scheduling is completely static, being fully defined at the compile time, as described in Figure 3.1. As illustrated in Figure 3.2, the assembly code for a VLIW processor is rather different from the point of view of the machine code from that for a superscalar processor: several instructions are grouped together in a single macro-instruction (also called Bundle) and each instruction is assigned for execution to a specific Computational Domain (CD). Consequently, in a VLIW processor there isn't any hardware instruction scheduler, and the tasks typically performed by this component are done by the compiler. In this way the power consumption is reduced and the silicon area occupied is far less that the area occupied by a traditional superscalar processor, while the design complexity is dramatically reduced; on the other side, the Instruction Level Parallelism can be adequately exploited (at least in the case of data intensive applications), since for many Digital Signal Processing applications a good compiler is able to understand which instructions can be executed in parallel checking the whole program at compile time [64] and generating the optimized program accordingly.

Finally, given their high regularity, VLIW architectures can be easily customized to efficiently perform any given application. In fact, a generic VLIW processor parametric architecture may have a variable number of CDs and FUs, so that different options, such as the number and type of functional units, the number of multi-ported registers (i.e., the size of the register file), the width of the memory buses and the type of different accessible FUs, can be modified to best fit the application

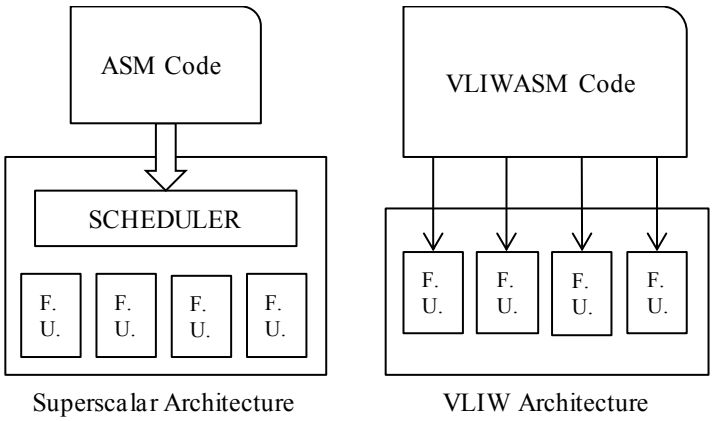



Figure 3.2. Difference between a superscalar and a VLIW processor.

requirements [36, 66]. All the features of a VLIW processor are grouped together and listed in the so called VLIW manifest. The manifest specifies the number of CDs, the number and type of FUs embedded into each CD, the size and the access mode of the register file and any other feature that must be taken into account when developing the code, such as the memory size and the memory access mode. Moreover, this file contains also the description of the Instruction Set Architecture (ISA) of the considered VLIW processor, which is clearly crucial in order to write the corresponding assembly code.

From the software point of view, the machine code is very different with respect to the code of traditional superscalar processors: the VLIW code is composed of a sequence of bundles, each of which contains a number of instructions equal to the number of CDs composing the VLIW processor; each instruction is assigned to a CD which is responsible for its execution. In Figure 3.1, an example of VLIW code is reported, based on the assembly code of a VLIW processor having four distinct CDs [61]: the assumption behind this work is that for each bundle there are four instructions. Considering these features, the VLIW code is more complex than a traditional assembly code, and the size of the code is typically larger. The mapping of instructions to CDs is entirely performed at compile time. It is also up to the compiler to identify the most suitable instructions to be included in each bundle, while guaranteeing that they can be executed in parallel by the different CDs taking into account any possible dependency; if for some reasons there are not enough independent instructions to be assembled into a bundle, the remaining slots are filled with NOP instructions.

Considering the particular architecture of the VLIW processor, several solutions have been proposed in order to produce the assembly code suitable for this kind

of processors: when generating code for a VLIW processor, the programmer or the compiler is faced with the issue of extracting parallelism from a sequential code and scheduling independent operations, concurrently, to the embedded functional units. For this reason the scheduling algorithms are critical to the performance of a VLIW processor. Many compilers for the first-generation of VLIW processors used a three phases method to generate code: first of all they generate a sequential program, then they analyze each basic block (a basic block is a sequence of instructions with a single entry point and a single exit point) in the sequential program looking for independent operations, and finally they schedule independent operations within the same block in parallel. The main problem of this approach is that in many cases the instructions in a basic block are dependent on each other: hence, insufficient ILP may be available within a single basic block, especially considering the large number of parallel resources of a typical VLIW processor. The Trace Scheduling [67, 68] is the most important scheduling algorithm, if VLIW processors are considered: it is a profile driven method where a set of commonly executed sequence of basic blocks embedded in the control flow is gathered together into a trace and the whole trace is scheduled together. In this way, the probability of assigning an operation to each functional unit increases since in a trace the possibility to find instructions that can be executed together at the same clock cycle is greater than when considering a single basic block.

## 3.3 Case study: the $\rho$-VEX processor

In this section the main features of the $\rho$-VEX processor, used as case study, are described. The $\rho$-VEX processor is a generic and reconfigurable VLIW processor whose VHDL description has been released by researchers from Delft University of Technology [65, 66]; it includes most of the features of the VLIW processors used by industry. The processor standard configuration (Figure 3.3) consists of a four stages pipeline organization: fetch, decode, execute and write-back. The fetch unit fetches a VLIW macro-instruction from the attached instruction memory and splits it into micro-instructions, that are passed, in parallel, to the decode unit. In this step the decoding operations are performed and the registers used as operands are fetched from the register file. The micro-instructions are then forwarded to the parallel execution units; as shown in Figure 3.3, for each execution unit there is an ALU unit (A); within the second and the third execution units there is also a MUL unit (M).

In order to prove the effectiveness of the proposed test and diagnostic methods, three different configurations of the $\rho$-VEX VLIW processor have been implemented: the key difference between these versions is the number of Computational Domains composing the processor itself, which has been varied from 4 to 8. Each of these

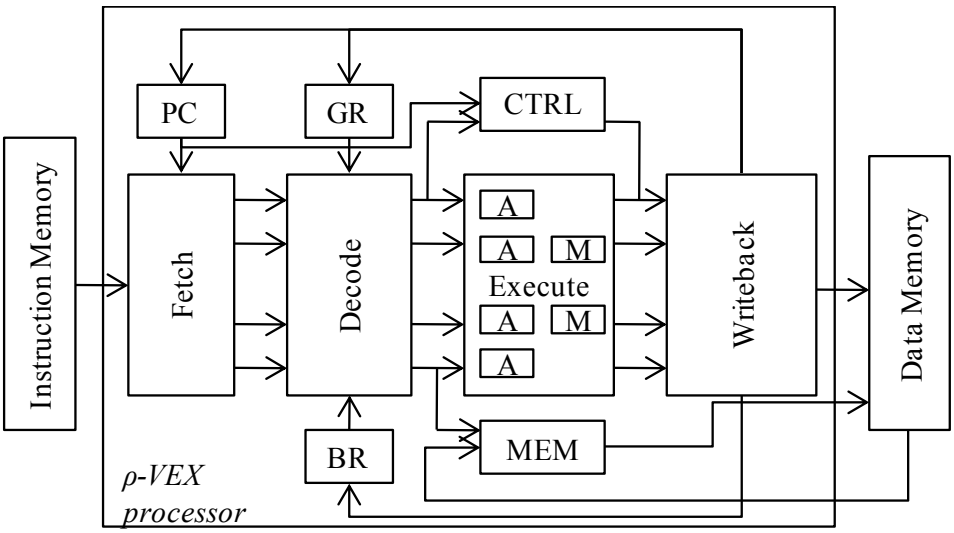



Figure 3.3. The standard configuration of the $\rho$-VEX VLIW processor.

processor configurations is described in a manifest, corresponding to a ASCII file containing the features of the resources and the ISA of the processor.

## 3.4 The new SBST method

In this section the new developed SBST techniques, along with some related works and experimental results, are presented.

### 3.4.1 Related Works

Popular techniques to test processor chips and Systems-On-Chip are Built-In Self-Test (BIST) and Software-Based Self-Test (SBST). Some methodologies may require very expensive Automatic Test Equipment (ATE); however, the increasing gap between maximum ATE frequencies and device operating frequencies makes external at-speed testing problematic and expensive; at-speed testing is needed because of failures detectable only when the test is performed at the device operating frequency.

BIST moves the testing task from external resources (ATE) to internal hardware: additional hardware is integrated into the circuit to allow it to perform self-testing. The use of this technique leads to decrease the test time, maintaining or improving the fault coverage, at the cost of additional silicon area [64].

A special type of on-chip testing is SBST [36], that is a non-intrusive methodology, since it adopts existing processor resources and instructions to perform self-testing. A major advantage of this technology is that it uses only the processor functionality and its instruction set for both test pattern application and output data evaluation, and thus does not introduce any hardware overhead in the design. However, software-based self-test methods may suffer from long program sequences to achieve high fault coverage [36, 43], and require effective techniques for generating suitable test programs.

In the literature there are many papers related to functional self-test of processors, but only a few of them refer to the test of VLIW processors [64, 69, 70, 71, 72]. In [64] a novel SBST algorithm aimed at testing the Register File of a generic VLIW processor is resented; the register file of a VLIW processor has a multi-port architecture, since this component must be accessed by all the Computational Domains. In detail, a single Computational Domain can access each register both in writing and reading through the use of write-ports and read-ports; this means that internally the register file of a generic VLIW processor has the architecture of a complex cross-bar. Considering this structure, a new SBST algorithm able to achieve high fault coverage with respect to stuck-at faults has been developed. The gathered experimental results show that this algorithm achieves up to 97.12% of fault coverage with respect to stuck-at faults. The method proposed in [64] has been developed addressing stuck-at faults, but it is extensible to deal with different fault models, such as transition delay faults.

Another technique aimed at testing VLIW processors combining scan and SBST, in order to obtain a good diagnostic resolution with a low hardware overhead, is proposed in [70]. The peculiarity of that approach, aimed at detecting faults in the functional units of the processor, is that the same test patterns are loaded directly into the fetch registers of all Computational Domains. The proper functioning of each domain is tested by comparing the test response of all domains, which should be the same in the fault-free case. This solution involves a hardware overhead of about 6% and requires that the processor runs in a special self-test mode.

In [69] the authors propose a Built-In Self-Repair (BISR) strategy for VLIW processors based on SBST; this approach is able both to detect faults and to identify the most convenient configuration able to tolerate them, if faults are located in components that have redundant elements. Considering the SBST technique, the main idea of that approach is to use the Instruction Set Architecture (ISA) in order to apply the test patterns generated off-line by an automatic test pattern generator (ATPG) for the units embedded into the processor.

In [71] a method is proposed that exploits the idle cycles of the VLIW Functional Units to run test instructions for detecting permanent faults, without increasing the hardware cost or significantly impacting the performance. In that approach the authors assume that for each functional unit a test set is available for testing

permanent faults (the test set may consist of pseudorandom patterns or deterministic ones from the literature [13]) but no details are provided about their generation for the considered VLIW architecture.

In [72] a software methodology is proposed for VLIW processors for detecting faults based on the execution of each operation twice on two different Functional Units exploiting the idle computational resources and checking the redundant results through control instructions. This approach is a valid solution for detecting both permanent and temporary faults exploiting only the ISA of the VLIW processor, but the performance degradation, mainly due to the checking instructions, and the code growth (more than 100% on the considered benchmarks) are significant.

Considering the test program generation process, in [73] the method for generating SBST programs for a whole VLIW processor is presented, starting from existing test algorithms developed for traditional processors. In particular, the method addresses the Functional Units (such as the ALUs and the MULs), embedded into a VLIW processors and explains how to combine them in a single test program for the whole processor.

### 3.4.2 The SBST method

The new SBST method proposed in this PhD thesis aims at automatically generating the test program for a given VLIW processor starting from the VLIW manifest and from a library of existing SBST programs. The generation of the test program is automatically performed on the basis of the VLIW configuration, and is therefore autonomously tuned depending on the VLIW manifest features. The flow supports some optimizations concerning the execution time, the fault coverage and the code length, and these optimizations are not correlated to the characteristics of the original test routines.

The execution flow is based on four main steps: the Fragmentation, the Customization, the Selection and the Scheduling, as illustrated in Figure 3.4. The flow has three initial inputs, that include two global requirements (the VLIW manifest and the library of generic SBST programs), and a specific input (the SBST program for testing the VLIW register file). The VLIW manifest contains all the features of the processor under test, while the SBST library contains a set of programs able to test the different modules within the processor itself. These two requirements are defined as global since they are configurable depending on the characteristics of the addressed VLIW processor. In particular, the library is a collection of generic SBST programs based on literature test algorithms: it contains the functional test code able to test the most relevant Functional Units of a generic VLIW processor. The test codes stored into the library are purely functional (i.e., do not require any DfT feature) and are completely independent on any physical implementation of the Functional Unit they refer to; these codes may be based on the techniques used

to test the same Functional Units when used in conventional processors. Their description exploits a pseudo-code based on C-language. On the other side, when the register file is considered, the algorithm proposed in [64] can be successfully used, since the structure of a register file of a superscalar processor is very different with respect to the register file of a VLIW processor, especially considering the access mode to the registers: in a typical VLIW processor, the register file is composed of a variable number of read and write ports, depending on the number of Computational Domains embedded into the processor [64]; consequently a special algorithm must be used in order to reach high fault coverage. These requirements are elaborated by the several steps of the developed flow; hereafter the details of each of these steps are described.

### The Fragmentation phase

The purpose of the Fragmentation phase is to minimize the number of test operations in order to create efficient and optimized test programs. The Fragmentation phase, illustrated in Figure 3.4 - Step A, performs two main tasks: the first is the selection from the library of the test programs needed to test the VLIW processor under test, ignoring those which refer to Functional Units that are not part of the processor itself. The second task performed by this step is to fragment each selected test program into a set of small pieces of code, called Fragments, containing few test operations and the other instructions needed to perform an independent test. The result of the fragmentation phase is a set of unique test Fragments, where each Fragment is normally built around a single test instruction and includes some preliminary instructions, required to correctly perform it, and some additional instructions to forward the produced results into observable locations [78]; a Fragment is described through architecture-independent code. A test program is typically composed of a set of test operations enclosed in a loop; the Fragmentation phase simply separates them in a series of short test programs using the Loop Unrolling technique, as shown into the pseudo-code reported in Figure 3.5. The Fragmentation phase is required in order to optimize the code, since VLIW processors are composed of multiple parallel Computational Domains that perform the execution of the operations in parallel, as described in section II. Due to this feature, when a SBST program is executed with the aim of testing a particular unit, at the same time other operations are also executed on the other parallel units. As a consequence, as it is shown in Figure 3.6for a sample VLIW processor, by applying the SBST program for the test of the VLIW register file [64], several faults belonging to other Functional Units, (e.g. the adders and the MEM unit), are also covered. The main idea behind test program fragmentation is to divide the original programs in atomic test units in order to effectively evaluate each one of them; in this way multiple fault coverage is avoided and the test code can be optimized in terms of test time and used resources. At the

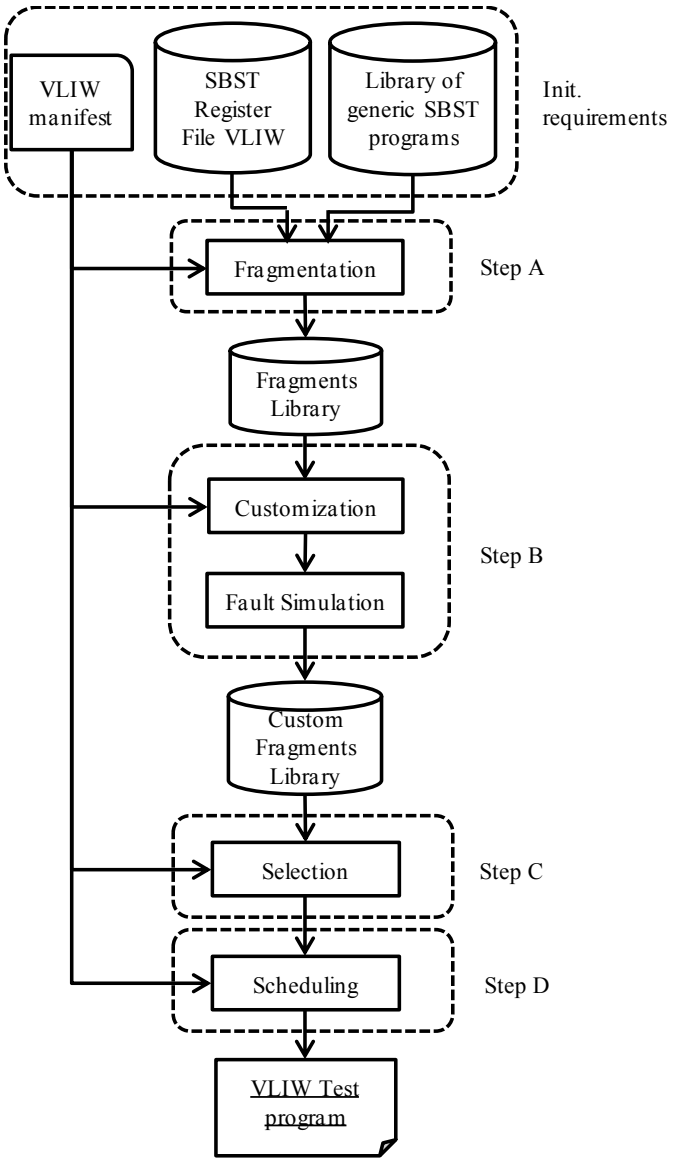



Figure 3.4. The flow of the proposed method.

end of the Fragmentation phase a new library called Fragments Library is obtained, that contains the set of architecture-independent Fragments. A simple example of Fragment is show in Figure 3.7: it is composed of a single test instruction, that performs an addition between two values, two preliminary instructions, that assign a value at the registers used by the test instruction, and an additional instruction used to forward the result of the test instruction into the memory.

```
1. for each cycle C of the loop L {
   1.1. S = set of performed operations;
   1.2. PI = input pattern applied to S into
        the cycle C;
   1.3. R = expected results performing S
        using PI as input pattern;
   1.4. GENERATE_NEW_FRAGMENT (PI, S, R); }
```

Figure 3.5. The pseudo-code of the Fragmentation phase.

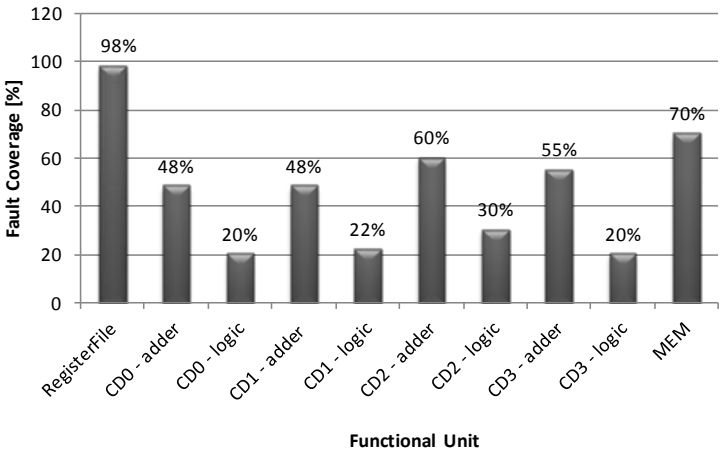



Figure 3.6. The coverage of the SBST program for the register file with respect to the other modules of the processor.

### The Customization phase

The customization step, illustrated in Figure 3.4 - Step B, is responsible for the translation of the generic architecture-independent test programs into the VLIW

```
1. R_src1 = All 0's;
2. R_src2 = All 1's;
3. R_dst = add (R_src1, R_src2);
4. Store (R_dst, memory);
```

Figure 3.7. An example of Fragment, where the instructions 1 and 2 set the registers used by the test instruction (3), and instruction 4 makes the result observable.

code, exploiting the Instruction Set Architecture (ISA) of the considered processor. In particular, starting from the Fragments Library and from the VLIW manifest, the method translates each generic Fragment in a Custom Fragment, that can be executed by the processor under test. A Custom Fragment is defined as a set of instructions belonging to the ISA of the processor under test, which perform several operations in order to test the addressed Functional Unit. An example of the Customization process is shown in Table I, where the code of a Fragment before and after the Customization phase is reported.

The two most relevant tasks performed by the Customization phase are the definition of the resources needed to execute the code (such as the registers and the memory areas required) and the introduction of the information, into the code, that assign the execution of an instruction to a defined VLIW CD. An example of instructions, before and after the customization, is illustrated in Figure 3.8; after the customization, the instructions are grouped in macro-instructions, where each macro-instruction is divided in CDs in charge of executing the addressed instruction.

Each Fragment is translated independently from the others; moreover, one architecture independent fragment can be translated into several architecture-dependent fragments, according to the features listed in the VLIW manifest, such as the number of the CDs and the type of the functional units contained by each of them: for example, if in the VLIW processor under test there are 4 adder units, one for each of the 4 CDs, the generic Fragments related to the test of a generic adder are translated into 4 architecture-dependent fragments, one for each adder unit embedded into the CDs.

At the end of the Customization phase, each architecture-dependent Fragment is fault simulated in order to get a detailed list of the faults covered by the specific test program considering all the resources of the VLIW processor. Finally, a library called Custom Fragments Library is obtained: it contains all the architecture-dependent Fragments needed to test the processor under test and the list of faults covered by each of them.

### The Selection of the Custom Fragments phase

The selection of the custom fragments, illustrated in Figure 3.4 - Step C, consists in the choice of the test fragments that optimize a set of rules dependent on the requirements desired for the final SBST program. The optimization is performed by the execution of the algorithm described in Figure 3.9; the algorithm is able to implement two alternative rules. The former aims at selecting the minimum number of Custom Fragments that allow to reach the maximum fault coverage with respect to all the resources of the processor under test. During this phase all the fragments are filtered depending on their fault coverage on the full VLIW processor. The filtering of the fragments is performed by the execution of multiple algorithm

**Before Customization**

```
R = add (All 0's, All 0's);
Store(R , memory);
```

**After Customization**

```
;;----Macro-instruction 1----
CD0 : mov R1 = 0;
CD1 : mov R2 = 0;
;;----Macro-instruction 2----
CD0 : add R3 = R1, R2;
;;----Macro-instruction 3----
CD0 : stw 0[R0] = R3; //R0 is the stack pointer
;;--------------------------------
```

Figure 3.8. An example of the translation performed by the customization step.

iterations. At each iteration the algorithm adds to the selected fragment list the fragments that maximize the fault coverage with respect to all the resources of the processor under test. In this way, at the end of the execution, several Custom Fragments are not selected, since the faults covered by these fragments are already covered by the fragments chosen by the algorithm.

The second rule aims at optimizing the number of resources used by the selected Custom Fragments. Generally, this rule is aimed at reducing the number of used hardware resources, in terms of registers and memory locations. For the purpose of our method, we implemented the automatic flow allowing the user to specify the area constraints for each type of resource. On the basis of these constraints, the algorithm selects the Custom Fragments that allow to achieve the maximum fault coverage without using extra resources than those specified. In this way, the proposed method is able to generate test programs depending on the final requirements: for example, if the final goal is to generate test programs oriented to on-line testing, with the use of the proposed algorithm it is possible to generate test codes that uses only a limited set of registers and memory locations.

### The Scheduling phase

The last step of the proposed automatic test program generation flow is the Scheduling, illustrated in Figure 3.4 - Step D. The scheduling phase first elaborates the selected Custom Fragments obtained from the Selection phase. This process is responsible for the integration of the Custom Fragments in order to obtain an optimized and efficient final test program. In order to reach this goal, we developed a scheduler that optimizes and merges the codes contained into the Custom Fragments exploiting the VLIW features; in particular, it compacts the test programs trying to maximize the ILP of the VLIW processor by an optimal usage of the parallel CDs.

```
1. FL = Fault List of the considered processor;
2. CFL = Custom Fragments Library;
3. SFL = Selected Fragments List;
4. while ( CFL is not empty AND found) {
   4.1. select Fragment F that allows to
        maximize the coverage of FL;
   4.2. if (F exists){
        • put F into SFL;
        • remove F from CFL;
        • found = TRUE;
   4.3. } else
        • found = FALSE; }
```

Figure 3.9. The pseudo-code of the algorithm for the selection of the Custom Fragments.

In order to optimize the execution of the test instructions composing the Custom Fragments, we developed a new scheduling algorithm based on the Trace Scheduling algorithm described in [67, 68].

The developed scheduling algorithm aims at optimizing the execution of the testing code in a generic VLIW architecture, taking into account the possibility of computing several instructions in a single clock cycle while maintaining the fault coverage capabilities of the compacted code unaltered with respect to the generated Custom Fragments. Our solution organizes the code belonging to the Custom Fragments in traces, which are loop-free sequences of basic blocks (a basic block is a sequence of instructions with a single entry point and a single exit point) and then squeezes a trace into few VLIW instructions. The scheduling algorithm we developed is restricted with respect to the original version of the trace scheduling algorithm, since in our test code (which is composed of several Custom Fragments that must be performed only once) we neglected the loop management. First of all the selected Custom Fragments are analyzed, looking for data dependencies among the instructions: for each Fragment an Instruction Dependency Graph (IDG) is created. More in details, a node exists in the IDG for each instruction in the fragment code, while each edge between two nodes corresponds to a data dependency between the corresponding instructions: hence, each edge corresponds to a physical resource (i.e., a register or memory location) used to store the data produced by one instruction and used by the other. During this phase, it is possible that two or more instructions, belonging to different Custom Fragments and related to the same Computational Domain, are identified as operations that perform the same job (e.g., they write the same value into the same register); if this behavior is detected, a unique IDG will be defined for the considered Custom Fragments, where only one

of these micro-instructions will be considered, while the others will be neglected; in this way the code functionality of the Custom Fragments remains unchanged, while the number of instructions is reduced.

Aside from the Instruction Dependency Graph generated, a table called Resource Usage Table, containing the details of the instructions executed is created. For each instruction, a new entry in the table is instantiated, containing the instruction ID and the identifier of the CD where the instruction must be executed; finally, each entry has a priority field, which is initialized to 1. An example of the Dependency Graph and of the Resource Use Table is reported in Figure 3.10.

The Dependency Graph and the Resource Use Table are then used by the developed trace scheduling algorithm in order to parallelize the code and provide an optimal usage of the VLIW parallel resources. The flow of the trace scheduling algorithm is illustrated in Figure 3.11. The managing of the instruction priority represents the main difference between the developed algorithm with respect to the original Trace Scheduling, since in the original solution the priority is used to optimize the frequently executed instructions if there are loops in the code, by a static prediction of the loops. Since the Custom Fragments are generated with the loop unrolling technique, as explain in Section 3.4.2, in the code composing each of them there isn't any loops; consequently, we can neglect the priority concept of the original Trace Scheduling.

In the proposed solution the priority values are used to avoid that an instruction goes in a starvation situation: when an instruction enters in the ready-set (which is the set containing the instructions ready to be executed), it is allocated in a given CD; in case the addressed CD is already occupied, the instruction remains in the ready-set but its priority value is increased at each iteration of the algorithm; with this method we guarantee that the priority value is higher for the instructions that first entered in the ready-set; therefore, the longer is the persistence of an instruction within the ready-set, the higher is its priority value. At the beginning of the scheduling algorithm, when all the instructions have the same priority value (fixed to 1 for default), the resource assignment is carried out randomly.

Considering the Instruction Dependency Graph and the Resource Use Table reported in Figure 3.10, in Figure 3.12 few basic steps of the developed trace scheduling algorithm are described. The considered VLIW processor is characterized by four CDs. The algorithm tries to maximize the parallelism by assigning to each resource an instruction; in our case, since the goal is to address the test of the resources, we have to guarantee that the test coverage of the customized fragments remain unchanged; therefore, it is not always possible to maximize the processor utilization, since the instructions must be executed in a specific CD and the trace scheduling algorithm avoids to change the execution units of an instruction with another one: changing the FU for an instruction may lead to a change in the fault coverage value computed in the previously described Customization step. The description of the

performed steps is provided here:

**Cycle 1** The instructions A, E, G are in the Ready-Set; A must be executed by the Computational Domain (CD) 0, E by CD3 and G by CD0. A and G compete for the same resource: since this is the first step and all the resources have by default the same priority value (the default value is 1), the choice of which instruction (between A and G) must be executed by CD0 is made randomly and in this case A wins. Consequently, A is assigned to CD0 and E to CD3 (E is the only instruction addressed on CD3); the instruction G remains in the Ready-Set and its priority value is incremented.

**Cycle 2** The instructions G, B, F are in the Ready-Set; G and B must be executed by CD0 and F by CD3; since the priority value of G is higher that the priority value of B, G is assigned to CD0, while F is assigned to CD3; the instruction B remains in the Ready-Set and its priority value is incremented.

**Cycle 3** The instructions B and H are in the Ready-Set; B must be executed in CD0 and H in CD2. In this case there is no resource competition; consequently, B is assigned to CD0 and H to CD2.

**Cycle 4** The instructions C, D and I are in the Ready-Set; C must be executed in CD1, D in CD3 and I in CD2. In this case there is no resource competition; consequently, C is assigned to CD1, I is assigned to CD2 and D to CD3.

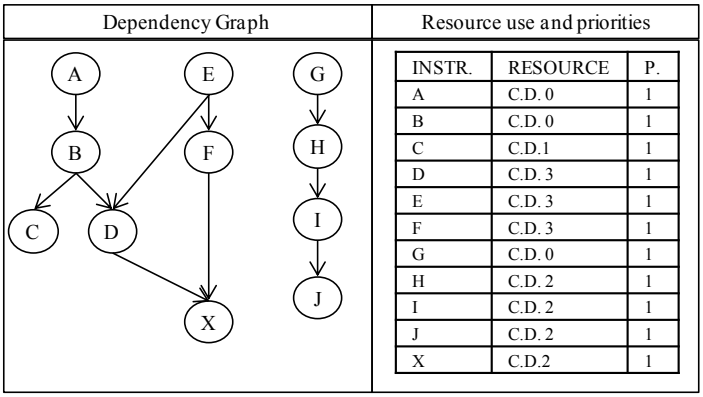
**Cycle 5** The instructions X and J are in the Ready-Set; both X and J must be executed in CD2. The priority value of both instructions is equal and consequently the choice of which instruction must be executed by CD2 is made randomly; in this case X wins. X is assigned to CD2 while J remains in the Ready-Set and its priority value is increased.

**Cycle 6** The instruction J is in the Ready-Set; J must be executed in CD2; J is assigned to CD2.

As illustrated in Figure 3.12 , at the end of the algorithm execution, the column *M. Instr.* contains the instructions that can be packed together in a single macro-instruction executed in a single clock cycle.

### 3.4.3 The experimental results

As already explained in Section 3.3, in order to prove the effectiveness of the proposed method, three different configurations of the $\rho$-VEX VLIW processor have been generated; where the key difference between these versions is the number of Computational Domains composing the processor itself (which has been varied from




| INSTR. | RESOURCE | P. |
|---|---|---|
| A | C.D. 0 | 1 |
| B | C.D. 0 | 1 |
| C | C.D.1 | 1 |
| D | C.D. 3 | 1 |
| E | C.D. 3 | 1 |
| F | C.D. 3 | 1 |
| G | C.D. 0 | 1 |
| H | C.D. 2 | 1 |
| I | C.D. 2 | 1 |
| J | C.D. 2 | 1 |
| X | C.D.2 | 1 |

Figure 3.10. An example of Dependency Graph (a) and of the Resource Use Table (b), where P stands for Priority.

```
1. The first instruction of every dependency
   graphs is added at the Ready-Set;
2. CC = Clock-Cycle = 0;
3. While (Ready-Set contains entries) {
   3.1. For each Computational Domain CD{
        • If exist an instruction I in the
          Ready-Set addressed to CD {
          ▪ Assigns I at CD;
          ▪ Remove I from Ready-Set; }
   3.2. Instructions free from data
        dependences are added to Ready-Set;
   3.3. Update the priority of the instruction
        of the Ready-Set; }
```

Figure 3.11. The developed scheduling algorithm based on the Trace Scheduling algorithm.

4 to 8). As a consequence, one test program for each of these configurations has been generated exploiting the method previously described. In particular a prototypical tool implementing the proposed method has been developed: it is composed of about 3,000 lines of C code.

As reference point, 6 SBST programs from the literature [64, 75, 43, 76, 79] have been selected, aimed at testing the Functional Units embedded in the processor: each of them has been encoded in architecture-independent pseudo-code and has been inserted in the starting library. The total time required to manually prepare the three input files required by the method was approximately 30 hours. At the

| Cycle | Ready-Set | C.D. 0 | C.D. 1 | C.D. 2 | C.D. 3 | M. Instr. |
|---|---|---|---|---|---|---|
| 1 | A E G | A | | | E | (A,E) |
| 2 | G B F | G | | | F | (G,F) |
| 3 | B H | B | | H | | (B,H) |
| 4 | C D I | | C | I | D | (C,I,D) |
| 5 | X J | | | X | | (X) |
| 6 | J | | | J | | (J) |

Figure 3.12. Few steps of the proposed scheduling algorithm.

end of the fragmentation step we obtained a Fragments Library (see Figure 3.4) composed of 520 architecture-independent Fragments.

In order to evaluate the stuck-at fault coverage achieved by the generated test programs, the $\rho$-VEX processor has been synthesized and implemented using a standard ASIC gate library. The assembly code generated following the described method has been inserted into the instruction memory; then, a fault simulation experiment has been performed. For all the fault simulation experiments, the Synopsys TetraMAX ATPG tool has been used [86].

In Table 3.1 the obtained results, considering the standard configuration of the processor, are reported; in Table 3.2 the results obtained adding 2 further Computational Domains (1 ALU and 1 MUL) are reported; finally, in Table 3.3 the results related to the $\rho$-VEX processor composed of 8 Computational Domains are shown, where the number of embedded ALUs has been increased to 8 and that of MULs to 4. As the reader can notice, the reached fault coverage is almost the same for the three configurations of the VLIW processor used as case study.

The details about the test programs generated through the proposed method and used to test the $\rho$-VEX processor are reported in Table 3.4, where the required number of clock cycles, the reached coverage and the size of the test programs are shown. The generation time for each test program was approximately 40 hours, of which about 95% used for the fault simulation of the Custom Fragments. The computational time has been evaluated on a workstation with a quad-core processor with frequency of 3 GHz and 8 GB of RAM.

The test programs (denoted as TP) generated implementing the proposed approach have been compared with a traditional test program (denoted as Plain TP), consisting of several test programs developed using some algorithms taken from the literature for the functional units of traditional processors; in the Plain TP these test programs have simply been queued in a unique test program, without performing any selection or scheduling steps, therefore adopting a realistic test estimation of what can be achieved with previously developed test algorithms without any optimization method. This is the only possible approach to have a comparison for the

proposed method: in the literature there is no method aimed at optimizing the SBST routines for VLIW processors exploiting the parallelism that characterizes these architectures. In order to fairly evaluate the two solutions, these test programs have been applied using the loop-unrolling technique, as it is common for any VLIW application in order to fully exploit the parallel FUs. From Table 3.4 it is possible to notice that with the proposed method it is possible to generate test programs that allow to reduce the needed clock cycles to perform the test (see Figure 3.13) and to lower the test program size (see Figure 3.14), while the fault coverage remains the same. More in general, the results show that thanks to the proposed optimization method, the duration of the test program grows less than linearly when increasing the number of CDs of the target VLIW processor. Moreover, the user?s effort is minimized, since it is enough to specify the processor features in the manifest in order to generate the appropriate test program for the generic VLIW processor.

Finally, the advantage of the proposed approach is significantly greater with respect to traditional scan test techniques: in fact, in order to reach 100% of stuck-at fault coverage, traditional scan test technique requires a number of clock cycles greater than 3 orders of magnitude on the average, with respect to our solution, for all the three configurations of the considered VLIW processor.

Table 3.1. Fault Simulation results for the version of the $\rho$-VEX processor composed of 4 CDs

| $\rho$-VEX Component | | Faults[#] | Fault Coverage |
|---|---|---|---|
| Fetch | | 2,156 | 99.2% |
| Decode | | 249,196 | 98.1% |
| Execute | 4 ALUs | 75,554 | 98.3% |
| | 2 MULs | 37,244 | 98.6% |
| | 1 MEM | 1,730 | 97.2% |
| Writeback | | 1,420 | 98.1% |
| *Total* | | *387,290* | *98.2%* |

## 3.5 The new Diagnosis method

In this section the new developed Diagnosis techniques for VLIW processors, along with some related works and experimental results, are presented.

Table 3.2. Fault Simulation results for the version of the $\rho$-VEX processor composed of 6 CDs

| $\rho$-VEX Component | | Faults[#] | Fault Coverage |
|---|---|---|---|
| Fetch | | 3,250 | 98.9% |
| Decode | | 290,135 | 98.0% |
| Execute | 6 ALUs | 113,316 | 98.3% |
| | 3 MULs | 55,866 | 98.6% |
| | 1 MEM | 1,730 | 97.2% |
| Writeback | | 2,130 | 98.1% |
| *Total* | | *466,427* | *98.1%* |

Table 3.3. Fault Simulation results for the version of the $\rho$-VEX processor composed of 8 CDs

| $\rho$-VEX Component | | Faults[#] | Fault Coverage |
|---|---|---|---|
| Fetch | | 4,312 | 98.8% |
| Decode | | 312,234 | 98.2% |
| Execute | 8 ALUs | 151,088 | 98.3% |
| | 4 MULs | 74,488 | 98.6% |
| | 1 MEM | 1,730 | 97.2% |
| Writeback | | 2,842 | 98.1% |
| *Total* | | *546,694* | *98.3%* |

Table 3.4. Comparison between the results obtained by applying some plain test test programs and the proposed method

| $\rho$-VEX config. | Test Program | Clock Cycles [#] | Fault Coverage | Size [kB] |
|---|---|---|---|---|
| 4 CDs | Plain | 18,540 | 98.2% | 3,894 |
| | Proposed Method | 8,447 | 98.2% | 1,612 |
| 6 CDs | Plain | 25,619 | 98.1% | 5,841 |
| | Proposed Method | 11,139 | 98.1% | 2,304 |
| 8 CDs | Plain | 32,699 | 98.3% | 7,788 |
| | Proposed Method | 13,132 | 98.3% | 2,832 |

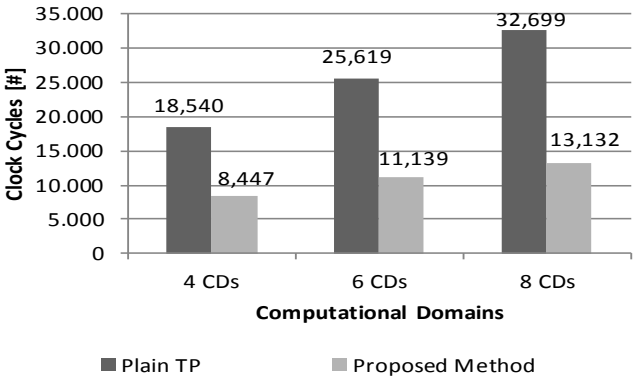



Figure 3.13. The reduction of the clock cycles using the proposed method wrt the plain test-program.

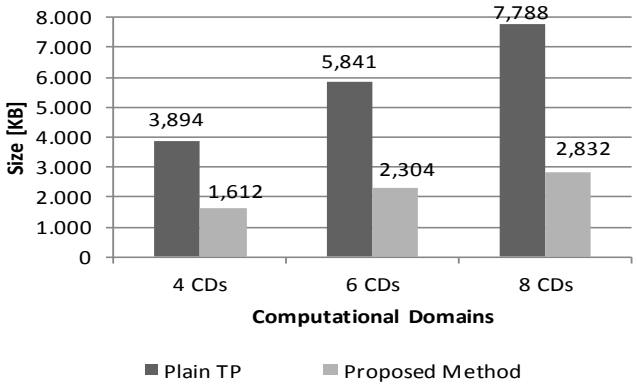



Figure 3.14. The reduction of the size of the test program using the proposed method wrt the plain test-program.

### 3.5.1 Basics on Diagnosis

Let call

$$F = f_0, f_1, ..., f_{n-1} \tag{3.1}$$

the set of $n$ faults that can affect the considered Unit Under Test (UUT). Each of these faults causes the UUT to produce a given output behavior $b$ (also called *syndrome*) when a given sequence of *Input Stimuli* $I$ is applied; let denote by $b_i$ the output behavior produced by fault $f_i$, and $b_g$ the output behavior of the fault-free circuit. Clearly, $b_i = b_g$ for all undetected faults $f_i$.

When SBST is considered, the assumption is often made, that the output behavior corresponds to the set of values left by the program in memory at the end of its

execution. This assumption is make in the context of the diagnostic work presented in this PhD thesis. The key rationale behind it is the ease of its implementation in practice, when test (or diagnosis) are run during the operational phase. Therefore, $b_i = b_j$ iff the two faults $f_i$ and $f_j$ produce the same output values in memory at the end of the execution of the test (or diagnosis) program. From a practical point of view, storing a signature of the values produced by each fault may allow to easily identify the existing faults. Alternative solutions avoiding the storage even of this compressed form of fault dictionary can also be considered [84].

A given pair of faults *($f_i$, $f_j$)* is said to be distinguished by a given sequence of *Input Stimuli I* iff $b_i \neq b_j$. Otherwise, they are said to be equivalent wrt *I*. All faults that are equivalent wrt to a give sequence of input stimuli *I* are said to belong to the same *Equivalence Class (EC)* wrt I. A detected fault fi is said to be fully diagnosed by a sequence of input stimuli I iff any couple of faults *($f_i$, $f_j$)* including $f_i$ is distinguished by *I*. Since two faults $f_i$, $f_j$ can never be distinguished if they are functionally equivalent, the number of fully diagnosed faults in a circuit is typically rather low.

Several possible metrics can be adopted to measure the diagnostic capabilities of a sequence of input stimuli *I* [85]. When diagnosis is used in a reconfigurable system for identifying the partition including the fault, the precision required is lower than in other situations where diagnosis is required (e.g., for yield ramp-up): in fact, the final goal in this case is to be able to distinguish all pairs of faults belonging to different partitions, while distinguishing pairs of faults belonging to the same partitions is not of interest. Hence, in the context of the diagnostic methods proposed in this PhD thesis a metric called *Diagnostic Capability*, or *DC(I)*, which corresponds to the percentage of faults belonging to an EC wrt *I* composed of faults all belonging to the same partition. In the ideal case in which DC(I) is 100%, this would mean that *I* is able to always identify the partition where the fault is located. Finally, the notion of *Fully Diagnosed Fault with respect to Partitions (FDP)* is also used, which is a fault belonging to an Equivalence Class composed of faults all belonging to the same partition. Clearly, DC(I) is the percentage of FDP faults with respect to the total number of faults.

### 3.5.2 The method

In this section the proposed method aimed at generating the diagnostic programs for a generic VLIW processor, once its specific configuration is known, is presented.

The proposed method, illustrated in Figure 3.15, is composed of two parts denoted as *classificatio*n and *brother fragment generation*. Moreover, it requires two main inputs. The former is the manifest of the VLIW processor under analysis, which contains all the features of the processor itself (which is supposed to be organized into a few partitions). The latter is a collection of small test programs aimed

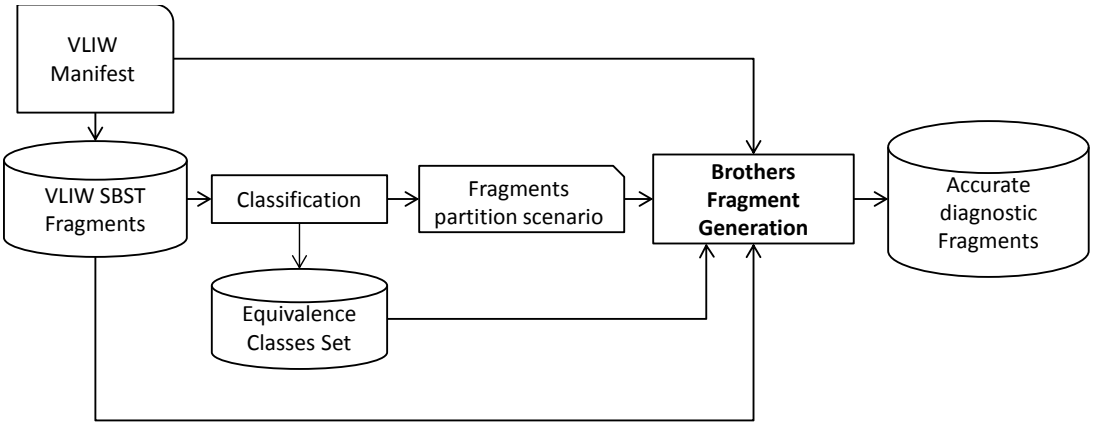



Figure 3.15. The flow of the proposed diagnostic method.

at fault detection, called fragments: each fragment performs a few test instructions (aimed at exciting a specific fault or group of faults) plus some other instructions needed to prepare the required parameters and make the results of the test instruction observable. The fragments have been generated splitting the original SBST programs (see Section 3.4.2 of this thesis): the fragments should contain the lowest possible number of instructions and detect the lowest possible number of faults (while still maintaining the same total fault coverage). Consequently, the diagnostic capability of the fragments set is improved. The set of the initial fragments is called *Initial Test Program*.

### Classification

The classification part aims at computing the Equivalence Classes with respect to the Initial Test Program. This task can be easily performed resorting to commercial Fault Simulation tools (i.e., the Sinopsys TetraMAX tool [86]) and its final result (which requires some further custom post-processing) is the assignment of each fault either to an Equivalence Class composed of faults belonging to a single partition (in which case the fault is labeled as FDP) or to an Equivalence Class including faults belonging to different partitions. In practice, this phase requires performing Fault Simulation of each fragment, and then processing the data base storing the syndrome of each fault, computing the Equivalence Classes. The result of this part of the method is the *Fragment Partition Scenario* which consists of a database where for each partition the list of faults belonging to it, their syndrome and the associated partition are stored.

### Brother Fragment Generation

The brother fragment generation phase is a flow oriented to the generation of new diagnostic fragments capable to improve the overall custom fragment diagnostic capability, thus increasing the DC(I) metric of the addressed VLIW partitions. The flow, illustrated in Figure 3.16, is composed of four phases: (1) analysis of multiple partitions, (2) couple faults extraction, (3) module identification, and (4) creation of new fragments. The 4 phases are repeated until a given stopping condition (e.g., based on maximum computational time, or on the achieved diagnostic capabilities) is reached.

The *Analyze Multiple Partition* phase elaborates the fragment partition scenario database comparing equivalence classes including faults belonging to two partitions. In details, for each couple of equivalence classes $i$ and $j$, it compares all the fault syndromes and provides the list of faults not distinguishable between $i$ and $j$. Once the list of faults is generated, the *Couple Faults Extraction* phase selects each couple of two fault locations, one belonging to the partition $i$ and the other belonging to $j$.

The *Module Identification* phase identifies the location of the two faults analyzing the fault location hierarchy with respect to the VLIW manifest information; the result of this phase is the identification of the VLIW circuit resources involved by each fault.

Finally, the *Create New Fragments* phase is executed. This phase is based on the pseudo-code reported in Figure 3.17: basically, it elaborates the original test fragments involved into the VLIW resource module identified by the Module Identification phase and generates a new set of fragments modifying the resource used by the original test instructions. In this way, the final test program includes two different fragments, which are supposed to fail alternatively, depending on whether one or the other of the two partitions we want to distinguish are faulty.

The algorithm needs the code of the original test fragment (OF), the VLIW manifest (VM) and the selected rule (R) which is provided by the module identification phase. There are two main rules that can be used for the generation of the new fragments: the first, denoted as R1, is a *register re-allocation rule* and it implies that the brother fragment will contain the same instructions of the original one, but each instruction will use different registers. In this way, by checking the results of the two fragment execution, we are able to understand if the fault is the register file (in case the two fragments results are both wrong) or one of the other VLIW module involved by the two fragments. The second rule, denoted as R2, is a *resource re-allocation rule*: simply, the new brother fragment will use a different VLIW Functional Unit to execute the test instruction of the fragment.

According to OF, VM, and R the algorithm analyzes the original test fragment considering the used test instruction (TI), the VLIW functional unit (FU), the registers used as operands (RI), and the registers used to forward the produced

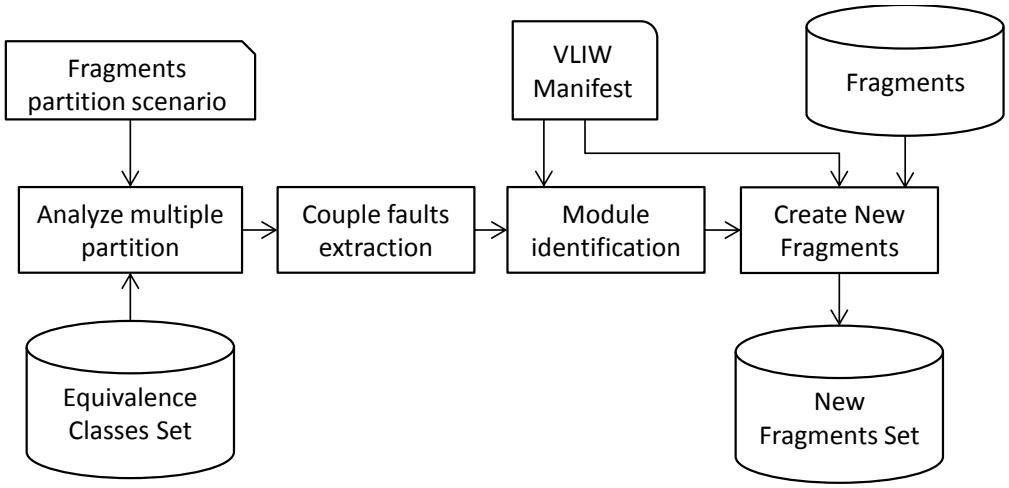



Figure 3.16. The flow of brother fragment generation phase.

results to observable locations (RO). Finally, it selects a new set of resources and on the basis of the defined rules it generates a new fragment.

In Figure 3.18, an example of original fragment and two corresponding brother fragments is shown; in this example a fragment in which the test instruction aims at the adder functional unit embedded in the Computational Domain 0 (referred as CD0) is addressed. The first brother fragment has been generated according to the rule R1 (i.e., the register re-allocation rule), in order to dismember an equivalence class containing faults embedded in the register file and in the adder functional unit of CD0. Consequently, the new brother fragment will be generated changing all the registers used to perform the test instruction and to forward the result in the data memory, without changing the functionality of the original fragment. The second brother fragment, instead, has been generated with the rule R2 (i.e., the resource re-allocation rule): practically, the test instruction of the original fragment has been moved from the computational domain 0 to the computational domain 1, leaving unaltered the other instructions composing the original fragment. In this way, if the results of the two fragments are both wrong, the fault is definitely not embedded in one of the two functional units executing the test instructions, but it belongs to another module used by the two fragments.

### 3.5.3 The experimental results

In this section the experimental results obtained using the $\rho$-VEX VLIW processor as a case study are presented. For the purpose of this research work, the stuck-at

```
1. OF = Original Fragment;
2. VM = VLIW Manifest;
3. R  = Selected Rule;
4. FL = Faults List;
5. Analysis of the Original Fragment OF -
   Identification of:
   5.1. A: Test Instruction (TI);
   5.2. B: Instructions that set-up the
        registers used by TI;
   5.3. C: Instructions that forward  the
        produced results to observable
        locations;
   5.4. The registers used by A, B and C;
6. Selection of the new resources, according
   with:
      • the selected rule R;
      • the VLIW Manifest VM;
7. Brother Fragment Generation: Assigns to A, B
   and C the new resources, according with:
      • the selected rule R;
      • the VLIW Manifest VM;
8. FL = Fault Simulation of the Brother
   Fragment.
```

Figure 3.17. The pseudo-code for the create new fragments phase.

fault model has been considered, although the method can be easily extended to deal with other fault models. The total number of stuck-at faults related to the $\rho$-VEX processor is 335,336. Moreover, the $\rho$-VEX processor has been divided in 10 logic partitions: the fetch unit, the decode unit, the general-purpose register file, the branch-management register file, the write-back unit, and the four Computational Domains in which the functional units are embedded. Clearly, these partitions are not uniform (in terms of number of contained resources).

Considering the diagnosis goal, in this Section only the most relevant partitions of the $\rho$-VEX processor are considered, i.e., the register file and the four Computational Domains (CD0 to CD3). The number of faults enclosed in each of the four Computational Domains is not exactly the same, since some of the functional units embedded in each of them are different: for example, CD0 includes a branch unit, while CD3 embeds a memory access unit, while all the CDs include an ALU unit.

A program (composed of about 1,200 lines of C++ code) able to implement the proposed method has been written: it is able to compare the fault lists generated by the fault simulation step; the main goal of this program is to implement the classification phase, i.e., performing the computation of the equivalence classes with

**Original Fragment**

```
---< macro-Instruction 1 >---
CD0: mov $r0.1 = 11111…1
CD1: mov $r0.2 = 00000…0
---< macro-Instruction 2 >---
CD0: add $r0.3 = $r0.1, $r0.2  /*Test instr.*/
CD1: nop
---< macro-Instruction 3 >---
CD0: stw 0[$r0.63] = $r0.3
CD1: nop
-----------------------------
```

**1st Brother Fragment – Rule R1**

```
---< macro-Instruction 1 >---
CD0: mov $r0.7 = 11111…1
CD1: mov $r0.8 = 00000…0
---< macro-Instruction 2 >---
CD0: add $r0.9 = $r0.7, $r0.8  /*Test instr.*/
CD1: nop
---< macro-Instruction 3 >---
CD0: stw 0[$r0.63] = $r0.9
CD1: nop
-----------------------------
```

**2st Brother Fragment – Rule R2**

```
---< macro-Instruction 1 >---
CD0: mov $r0.1 = 11111…1
CD1: mov $r0.2 = 00000…0
---< macro-Instruction 2 >---
CD0: nop
CD1: add $r0.3 = $r0.1, $r0.2  /*Test instr.*/
---< macro-Instruction 3 >---
CD0: stw 0[$r0.63] = $r0.3
CD1: nop
-----------------------------
```

Figure 3.18. An example of the generation of a brother fragment from an original fragment.

respect to the adopted test programs. The tool is also able to identifies FDP faults, and provides information about the remaining faults.

As a starting test program the set of fragments used for the optimized generation of an SBST program addressing the $\rho$-VEX processor has been used; it has been generated with the method proposed in Section 3.4; this set is a selection, from an exhaustive set of possible fragments, of the fragments that allow to maximize the stuck-at fault coverage, minimizing the test size and length.

The gathered experimental results are reported in Table 3.5, which includes the percentage of FDP faults with respect to the total number of faults of each partition, i.e., the Diagnostic Capability. The first column of Table 3.5 (denoted as Optimized SBST) is the original test set, composed of 244 fragments; its diagnostic level is

rather low for all the considered partitions, since this is optimized in terms of size and length, which are often conflicting goals with respect to diagnosis. The stuck-at fault coverage reached by this test program is 98.2% with respect to all the resources of the considered VLIW processor.

The first step towards the improvement of the Diagnostic Capability is the use of the whole fragments set generated resorting to the method described in Section 3.4. The results obtained with this approach are shown in the second column of Table 3.5 . The improvement of the diagnosis resolution is greater when the register file is considered (the improvement for this partition is more than 21%), while it is limited for the Computational Domains. This is mainly because the considered set of fragments is composed of 748 fragments, and 68% of them target the test of a portion of the register file itself.

The final step of the proposed flow is the evaluation of the diagnostic capabilities of an ad-hoc fragments set, composed of the fragments of the Exhaustive Fragments Set with an additional set of fragments brothers, developed with the proposed method. For the purpose of this research work, the brother fragments for the fragments addressing the test of the ALUs have been generated: in fact, these components are the most relevant of each CD in terms of number of stuck-at faults. Moreover, we the brother fragments have been developed also for the memory unit (which is embedded in CD3), since this unit is used by all the fragments in order to save the results of the test instructions in the data memory; consequently, there are many equivalence classes containing a fault belonging to this unit, and an efficient diagnostic of this module is required. The resulting set of fragments is composed of 1,056 fragments, of which 308 are brother fragments. The CPU generation time for the brother fragments was approximately 21 hours, of which about 85% used for the fault simulation; the computational time has been evaluated on a workstation with an Intel Xeon Processor E5450. As shown in Table 3.5, the improvements due to this approach are evident if the partitions containing the ALUs (referred as CD1, CD2, CD3 and CD4) are considered: the capability to recognize if a fault is enclosed in one of these partitions is improved of about 8% with respect to the previous approaches. The resulting diagnosability is not uniform for all the four CDs since, as explained previously, the functional units embedded into these partitions are not the same.

An analysis about the Equivalence Classes wrt the last test set has been performed, focusing on those that include faults belonging to more than one partition (i.e., neglecting all FDP faults). Analyzing these equivalence classes, only, it is possible to notice that about 95% of them are classes only including faults belonging to the same partition; moreover, if the remaining classes are considered, about 60% of them are equivalence classes enclosing faults belonging to 2 partitions, while about 35% are classes enclosing faults belonging to 3 different partitions, as shown in the graph of Figure 3.19. These results show that even when the diagnostic resolution

of our method is not enough to identify the single partition including a fault, still it is able to identify the couple of candidate partitions in about 60% of the cases. Finally, in Table 3.6 some more information about the size and the execution time of the final fragments set are shown. These results confirm that optimizations, in terms of size and length, are often conflicting goals with respect to diagnosis.

Table 3.5. Diagnostic Capability

| Partition | Optimized SBST | Exhaustive Fragments Set | Proposed Approach |
|---|---|---|---|
| Register File | 62.82% | 84.23% | 87.17% |
| CD0 | 77.12% | 77.79% | 83.74% |
| CD1 | 80.12% | 81.56% | 88.39% |
| CD2 | 79.99% | 80.34% | 88.23% |
| CD3 | 70.80% | 72.14% | 81.65% |

Table 3.6. Size and duration of the different test sets.

| Method | Size [kB] | Execution time [clock cycle] |
|---|---|---|
| Optimized SBST | 1,926 | 10,601 |
| Exhaustive Fragments set | 3,429 | 17,049 |
| Proposed Approach | 4,899 | 24,356 |

As a conclusion of this research activity, in this Section the first method able to generate optimized Software-Based Self-Test programs for VLIW processors (by exploiting the intrinsic parallelism existing in the VLIW processors) has been proposed. The method is fully automatic and allows to drastically reduce the effort of generating test programs for VLIW processors. The method has been experimentally validated resorting to a representative VLIW processor and generating test programs using a prototypical tool: the obtained results clearly demonstrate the efficiency of the method, that allows to reduce significantly both the number of clock cycles and the memory resources with respect to test programs generated by applying generic SBST methods to the specific case of the VLIW processors. More in particular, the method shows that it is possible to develop test programs whose duration and size grows less than linearly with the VLIW parallelism.

Another research work presented in this Section shows how the test programs could be used for the diagnosis of the main Functional Units of a generic VLIW processor. In this PhD thesis a new diagnostic method has been presented: it starts

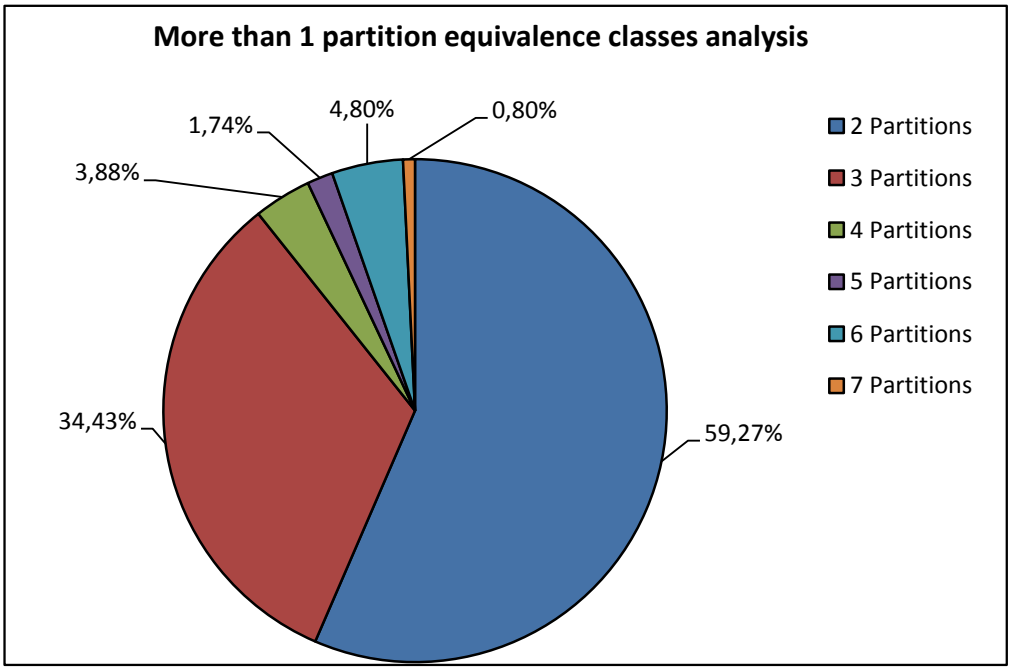



Figure 3.19. Analysis of Equivalence Classes including faults belonging to more than one partition.

from existing detection-oriented programs, and generates a diagnosis-oriented test program for a generic VLIW processor. The method exploits the parallelism (and the presence of several alternative resources) intrinsic in VLIW processors to enhance the original test program. The resulting diagnostic program is thus able in most cases to identify the faulty module and is therefore highly suitable for being used within reconfigurable systems.

# Chapter 4

# Reliability evaluation and mitigation of GPGPUs

In this chapter, the research works, developed in the context of this PhD thesis, finalized to the reliability evaluation and enhancement of General Purpose Graphics Processing Units (GPGPUs) are presented; moreover, an exhaustive explanation of the radiation tests performed to evaluate the proposed methods are explained, along with the details of the NVIDIA devices used as case study.

In the last decade, new devices known as General Purpose Graphics Processing Units (GPGPUs) made their appearance on the market. Their very high computational power, combined with low cost, reduced power consumption, and flexible development platforms are pushing their adoption not only for graphical applications, but also in the High Performance Computing (HPC) market. Moreover, GPGPUs are increasingly used in some safety-critical embedded domains, such as automotive, avionics, space and biomedical [87]. As an example, Advanced Driver Assistance Systems (ADASs), which are increasingly common in cars, make an extensive usage of the images (or radar signals) coming from external cameras and sensors to detect possible obstacles requiring the automatic intervention of the breaking system.

However, several issues about the reliability of GPGPUs have been raised [16, 88]. Given their high degree of parallelism, many assume that GPGPUs could intrinsically provide a good degree of fault tolerance; however, their size and complexity could make them particularly sensible to soft errors. Moreover, while hardening techniques already exist for systems based on traditional CPUs, similar solutions for GPGPU-based systems are still in their infancy [17]. The programming paradigm adopted by GPUs (i.e., Single Instruction Multiple Data) can provide some advantages when designing hardening strategies, but requires effective solutions to combine detection and correction capabilities with the required high performance characteristics.

When assessing the sensitivity of GPGPUs to radiation, a commonly adopted

solution is performing radiation experiments with accelerated particles, counting the number of errors they trigger. A major target of radiation effects are GPGPU's memories, both standard and caches. Recently, manufacturers adopted Error Correction Code (ECC) mechanisms against Soft Errors affecting all GPGPU memory modules. In particular, some manufacturers like NVIDIA have recently introduced the ECC scheme for GPGPUs oriented to High Performance Computing (HPC), which is characterized by looser constraints on power consumption and area cost than the embedded computing market. Vice versa, in GPGPUs designed for embedded systems the ECC mechanism is still not available. Radiation errors on memories may then significantly reduce the reliability of the device. Finally (and more generally), few data are still available on the GPGPU memory soft-error rate, so that quantitatively evaluating their reliability is still a hard task.

In this PhD thesis three different research works finalized to the reliability evaluation of GPGPUs are presented. The goals of these works are the investigation of the sensitivity to soft-errors induced by terrestrial radiation effects on GPGPUs, thus evaluating their capability to produce correct results even when used for long and massive computations in HPC data centers, and to work in harsh environments and/or for safety-critical applications.

When adopting GPGPUs in these applications, a major target of radiation effects is represented by the caches, due to their size and their impact in increasing the performance of the GPGPU [89]. Any radiation campaign focused on testing cache soft-error sensitivity requires first forcing the memories to a given value, then exposing the device to a given radiation fluence letting errors to accumulate, and finally checking whether all the bits in the cache(s) are still holding the initial value. The first and last steps are particularly critical, since caches are not directly accessible, and relatively few information is delivered about the cache organization and architecture if commercial-off-the-shelf GPGPUs are considered. One of the motivations of this work is that, traditionally, the manufacturers of electronic devices perform several tests in order to measure the reliability of their devices ; however, for industrial and confidentiality reasons normally this kind of data is not publicly available. Only few manufacturers (e.g., Xilinx) provide the user with actual radiation-induced error rate; to the best of our knowledge, no GPGPU manufacturer does so. Moreover, the GPU producers, like NVIDIA, are highly likely to know the soft-error sensitiveness of the memory array they use, since this data may be provided by the silicon manufacturer. Nevertheless, the operative soft error reliability of the memory array when embedded in the final products may significantly differ from the stand alone memory array. With the proposed method, it is possible to evaluate the cross section and the FIT of the main memories used in the NVIDIA GPUs. In this PhD thesis the method to successfully evaluated the GPGPU caches reliability is presented; it is mainly based on specially written programs, which are run immediately before and after irradiating the GPGPU device with a given particles fluence. The

proposed approach has also the feature of detecting anomalies in the hit/miss mechanism of data caches caused by cache tag corruptions: while turning a hit into a miss mainly causes performance degradation, the reverse, even if less likely to occur, may have serious effects on results correctness [90]. To the best of our knowledge, this is the first attempt to exploit carefully written programs (such as those proposed in [91]) to support radiation experiments and extract specific information about the embedded memories reliability.

The second research work presented in this PhD thesis moves on the direction of understanding the reliability of embedded GPGPUs, giving novel insight on their behaviors when exposed to ionizing radiation. The main contribution of this work is the reliability analysis of a FFT algorithm designed for embedded GPGPUs through neutron beam radiation experiments. The analysis is performed giving particular attention to caches and thread scheduler corruption. As explained in the previous paragraph, caches have been demonstrated to be a critical module in terms of reliability [8], since these memories have a very important role during the parallel algorithms execution. Caches memories have a key role on GPGPUs architectures as they significantly improve the performance allowing parallel tasks to share data. Consequently, caches represent a critical component of GPGPUs computing, since an error in a data induced by a radiation could affect the computation of all the cores, thus compromising the whole algorithm execution. Clearly, this behavior is not acceptable if GPGPUs are employed for safety critical applications. This represents the motivation of the proposed work: the soft error sensitiveness data of a typical parallel algorithm, executed with different GPGPUs cache configurations is presented. Moreover, the related data about the same algorithm, but executed with different threads distribution are shown. In order to be fully compliant with the Degree of Parallelism (DOP) adopted by embedded GPGPUs, the progressive reduction of the FFT parallel tasks from iteration to iteration is considered, depicting both overall GPGPU cross-section and the error rate for all the FFT stages. The progressive reduction of the parallel tasks is a common behavior of several GPGPUs algorithms (e.g., the Breadth-First Search - BFS algorithm, where the 30% of the GPGPU execution time is managed by one thread), and it is called underutilized parallelism in GPGPU computing [120]. Consequently, even if in this method the analysis is performed only on the FFT algorithm, the results and discussion are extendable to any other similar parallel algorithms for embedded GPGPUs. The obtained results show that, if the L1 cache of the considered GPGPU is disabled, the FFT algorithm execution has the lowest cross section. Instead, when the number of parallel threads managing the same algorithm is reduced, the execution is less reliable. This effect has been correlated to a different usage of the GPGPU caches due to a different number of threads running in the GPGPU itself. Finally, a greater number of errors when the algorithm exploits all the parallel resources of the considered GPGPU, i.e., in the first stages of the FFT algorithm, has been

experienced, in particular when the number of parallel tasks is higher.

Finally, the last research work in this area focus on the evaluation of two software redundancy techniques aimed at soft-error detection in GPGPUs; these techniques are completely algorithm independent; for the purpose of this work they have been applied to a benchmark application, frequently used to evaluate the performances of multi-core architectures (i.e., matrix multiplication algorithm), running on a commercial-of-the-shelf GPGPU. In particular, two redundancy techniques have been implemented, both based on Duplication-with-Comparison: (1) time redundancy, in which the code running on the GPGPU is executed twice and the obtained results are then compared, and (2) thread redundancy, in which a high number of threads is used, where half of them are devoted to compute the results and the other half is used to compute the replica; moreover, each of the two redundancy techniques has been implemented in two different versions: in the first the computation is based on a single instance of the input data, while in the second, two instances of the input data exist.

The proposed methods have been implemented on a device by NVIDIA with Fermi Architecture and validated in several extensive radiation campaigns at the ISIS facility in the Rutherford Appleton Laboratories (RAL) in Didcot, UK, and in the LANSCE facility at Los Alamos in USA. The gathered experimental results demonstrate their effectiveness, and provide interesting data about the sensitivity to radiation of GPGPU devices.

## 4.1 NVIDIA GPGPUs Fermi-Based Architecture

The GPGPU architecture addressed in the research works described in this PhD thesis is the NVIDIA Fermi Architecture [97]. As shown in Figure 4.1, the Fermi GPGPU family is composed of an array of Streaming Multiprocessors (SMs), each of which has the ability of execute several threads in parallel. The SMs are composed of several computational units, called NVIDIA CUDA cores or Streaming Processors (SPs), where each core manages a thread at a time. The Thread Blocks Scheduler assigns the thread blocks to the SMs while the Thread Scheduler inside a SM assigns a thread to a SP (as shown in Figure 4.2).

From the software point of view, NVIDIA CUDA extends C language by allowing the programmer to define C-based functions, called *kernels*, that, when called, are executed in parallel by N different CUDA threads; the NVIDIA CUDA programming model assumes that the threads execute on a physical separate device (i.e., the GPGPU) that operates as a coprocessor of the host (i.e., the CPU) running a controlling program. CUDA is particularly suitable for embedded GPGPUs as the user can easily define directly on the code the portion of the application to be executed on the host and the portion that requires GPGPU acceleration [97, 102].

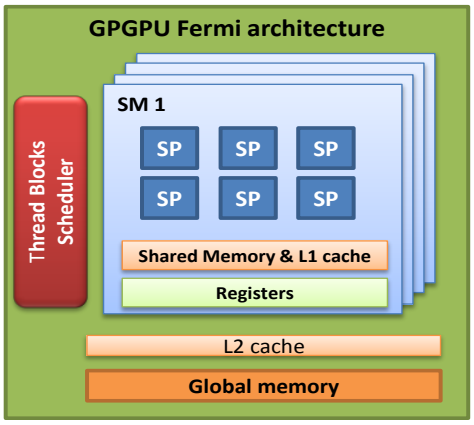



Figure 4.1. Fermi-based GPGPU Architecture: it is mainly composed of several SMs

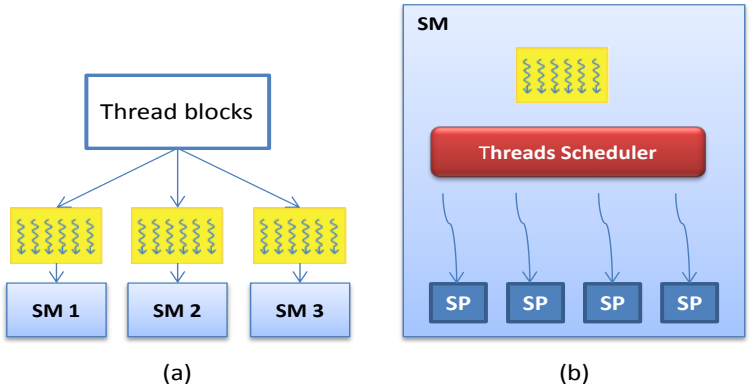



Figure 4.2. A block of thread is forwarded to each SM by the ThreadBlock scheduler (a); each SP receives one thread at a time (b).

### 4.1.1 Memory hierarchy

The cache memories have a key role in fully exploiting the high computation capabilities of the GPGPU [87]. In general, cache memories speed up the microprocessor memory access by storing recently used data values. Cache internal organization is based on two memory arrays (data array and directory array) managed by a cache controller circuitry, in which the minimum allocation unit of the data array, called cache line, stores a set of memory words. Finally, a shared memory exists for each streaming multi- processor, storing the data in common to the threads running on it.

As shown in Figure 4.1 the main modules composing the GPGPU are the SMs

and the DRAM memory in which the global memory of the device is mapped. Each SM can access each global memory location and both the reading and the writing operations are cached in the L2 cache. The maxmum L2 cache size is 768 KB. Moreover, in the SM, the L1 cache and the shared memory are embedded in the same physical memory device and allocated in the same block; the typical dimension of this memory for a Fermi-based GPU block is 64 KB, with two possible configurations defined by the final user: (1) 16 KB of shared memory and 48 KB of L1 cache, and (2) 48 KB of shared memory and 16 KB of L1 cache.

In the SM, each CUDA core can access each global memory location, but only the read operations are cached in the L1 cache; moreover, all the shared memory locations are accessible by all the CUDA cores embedded in the same SM and these accesses are not cached. Finally, the shared memory is expected to be much faster than the global. The global memory accesses are cached and they can be configured at compile time to be cached in both L1 and L2 or in L2 only. A cache line is composed of 128 Bytes and maps to a 128 Bytes wide aligned segment in the device memory. Memory accesses that are cached in both L1 and L2 are serviced with 128-Byte memory transactions, whereas memory accesses that are cached in L2 only are serviced with 32-Byte memory transactions. Both L1 and L2 caches are set associative and implement a LRU strategy. Moreover, the default store instruction cache operations are write-back, both for the L1 and L2 cache [99]. This is taken into account in the development of the algorithms proposed in this PhD thesis.

Please note that the works proposed in this PhD thesis address a Fermi Architecture GPGPU oriented to embedded systems, in which the ECC scheme is not applied to the memory modules. The ECC scheme proposed by NVIDIA for HPC devices, even if effective in reducing the impact of ionizing radiation on the GPGPU memory structures, has several limitations, as its activation reduces the GPGPU performance of up to 30% and memory availability of up to 15% [100]. Consequently, it is essential to carefully evaluate the cache sensitivity to understand when the ECC can be conveniently adopted: for example, in case the user needs extreme performance and for this reason ECC is disabled, a careful evaluation of the consequences on the reliability of the GPGPU memories should be conducted.

## 4.2 GPGPU reliability background

Radiation effects are a concern for the reliability of electronic devices not only in harsh radiation environments such as space or avionic, but also at ground level [8]. Today, device technology shrinking has led to a drastic reduction of critical charges in logic gates and memory cells that results in a higher sensitivity to soft-errors induced by ionizing radiation.

In the last years, an increasing research interest has been devoted to the soft-error sensitiveness evaluation of GPGPUs [17] . A first method for the evaluation of their radiation sensitivity has been proposed in [92], where authors present an analytical model for the evaluation of Single Event Upset (SEU) occurrences on GPGPUs depending on the memory and register usage of the running application.

Recently, GPGPUs have been evaluated using spallation neutron sources that provide the user with an atmospheric-like spectrum. A preliminary experimental setup for the execution of neutron radiation test of a GPGPU has been proposed in [93]. The authors describe a low-cost but effective setup providing some guidelines on how to test GPGPUs, focusing on the constraints imposed by the radiation source and the device connections with the host computer controlling the experiment. The first radiation test results demonstrate that both memory and logic resources of a GPGPU may be corrupted by atmospheric neutrons. Being characterized by high-performance computational units such as fixed and floating point units, a further evaluation of the probabilities that radiation-induced errors may affect the mantissa, the exponents or the sign has been evaluated in [96]. A strong variation on the output error rate has been observed when different types of data are elaborated showing a higher sensitivity for the resources used by the mantissa. Moreover, radiation evaluations also addressed the distribution of the computational workload on GPGPUs such as the thread distribution and their relation to multiple output errors [95, 121].

Aside from testing approaches, software-based hardening solutions have also tentatively addressed in [95], where authors developed an optimized software-based hardening strategy exclusively oriented to matrix multiplication applications. Researches have also investigated the possibility of using Software-Based Self-Test (SBST) programs in order to detect and localize permanent faults in GPGPUs: a preliminary work proposing a possible SBST solution has been presented in [**?**].

Recent previous works have also been done to evaluate the reliability of HPC-oriented GPGPUs [118, 96]. Embedded low-power GPGPUs have a similar architecture than HPC devices, but have a different programming paradigm. If on one side, HPC applications have performances as a major concern, so the GPGPU is likely to be always fully loaded with parallel processes; on the contrary, embedded GPGPUs have to take care on the power consumption and must respect tight constraints on area and device resources. For this reason GPGPU devices should have limited area usage and, in order to save energy, optimized computational cores usage (e.g., a temporarily not used core is switched off in idle state).

In the HPC fields, supercomputers (like *TITAN* at the Oak Ridge National Laboratory [127], or *Moonlight* at the Los Alamos National Laboratory [128]) are composed of thousands of Graphics Processing Units that work in parallel. Titan, world's second fastest supercomputer for open science in 2014, consists of more than 18,000 GPUs that scientists from various domains such as astrophysics, fusion,

climate, and combustion use routinely to run large-scale simulations. Scientific applications that run on these large-scale machines are typically very long-running. A single simulation may take from a few hours to a couple of days. Due to the large-scale and the long duration, leadership scientific applications may encounter interruptions due to system failures as well as Silent Data Corruption (SDC) in the output. Therefore, while the performance improvement achieved via inherent parallelism available in GPUs is necessary to expedite the scientific discovery process, it is equally critical that applications are able to cope with system failures during a run, without losing all of the work.

As documented in the works cited before, the newest GPU cores are sensitive to radiation-induced errors, including those from the terrestrial neutron radiation environment. Field data obtained from more than 18 months of controlled operation of both TITAN and Moonlight will be correlated with beam experiments data. Moreover, novel code optimizations to reduce the time-to-solution of specific parallel algorithms are continuously implemented. Unfortunately, while the performance and efficiency of code optimizations is well established, their impact on energy consumption and resilience characteristics has not been fully evaluated. In this contact, a current research topic is based on studies to provide a thorough understanding of the effects of code optimizations on GPU's performance, energy consumption, and reliability will be presented.

## 4.3 Case Study: Seco CARMAKIT board for embedded GPGPUs development

In order to evaluate the reliability techniques addressed in this PhD thesis, three different radiation experiments have been performed. The evaluation board used for the execution of these radiation test campaigns is the CARMA-Kit - SECOCQ7-MxM [101, 102], whose simplified architecture is shown in Figure 4.3.

The CARMA DevKit features a Qseven NVIDIA Tegra 3 Quad-core ARM A9 CPU and the NVIDIA Quadro 1000 M GPU with 96 CUDA cores. The ARM A9 CPU and the GPGPU are connected through a PCI express bus. In the NVIDIA QUADRO 1000 M there are 128 KB of L2 cache. The device includes two SMs, each of which contains 48 CUDA cores; for each SM, the size of the memory module in which the shared memory and the L1 cache are mapped is 64 KB; the configuration in which 16 KB are dedicated to the L1 cache and 48 KB are for the shared memory has been chosen.

In Figure 4.4 a picture of the SECO board used during the radiation test campaigns is shown.

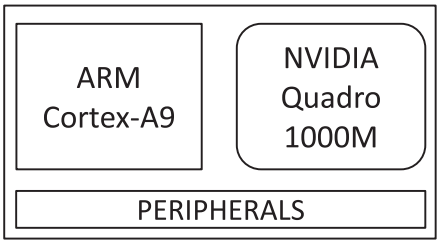



Figure 4.3. A simplified representation of the SECO evaluation board.

Figure 4.4. A picture of the SECO board used for the radiation test campaigns.

## 4.4 GPGPU Caches reliability evaluation: the proposed approach

The purpose of this research work is the investigation of the sensitivity to soft-errors induced by terrestrial radiation effects on GPGPUs caches, thus evaluating their capability to produce correct results even when used for long and massive computations in HPC data centers, and to work in harsh environments and/or for safety-critical applications.

### 4.4.1 Developed method

In this section, the algorithms designed and implemented to efficiently support the reliability evaluation of the caches memories of a typical GPGPU are presented. The

goal of the proposed algorithms for the Shared Memory, L1 and L2 caches is twofold. First, before radiating the device the load into each word of the target memory of a given value (typically all 0's or all 1's) is required; this operation causes a sequence of cache miss, if the L1 or L2 cache are addressed. Secondly, after radiation a read back operation of the data from the memory is required, checking their correctness (i.e., whether any bit changed its value due to a possible radiation effect); clearly, in the ideal case, this operation causes a sequence of cache hits, if the L1 or L2 cache are addressed, since the data are already present in these two memories.

Considering the L1 and L2 cache, the proposed algorithms are also able to measure the time required to access each addressed data, thus they allow to understand if a reading operation causes a cache hit (if the data is already present in the memory) or a cache miss (if the data needs to be loaded from the main memory). The way in which these temporal data are acquired is explained in the following sections. These data allow to understand if a radiation particle can force a cache hit to become a cache miss, e.g., by corrupting a bit in the cache tag. This behavior can be detected by checking the time required to read a data after the exposure to radiation: since the data is already present in the considered cache, the expected access time should correspond to a cache hit time. Instead, if the calculated time corresponds to a cache miss time, it means that a soft error affected the corresponding cache tag, invalidating the corresponding data.

### Algorithm for L1 cache and shared memory

As already explained in the previous sections, each SM includes a memory block containing both the L1 cache and the shared memory; as the goal of the proposed method is to write a specific data pattern into these memories and to read their content, it is sufficient that one thread in each SM executes the kernel functions corresponding to the proposed algorithm.

The algorithm basically allocates a properly initialized vector whose size is equal to that of the target memory, and performs an access to the vector, which moves it to the target memory itself. After irradiation, the vector elements are read back to check whether they still are in memory, and whether they still hold the expected values. Moreover, when working on a cache the algorithm also checks whether any access produces the expected behavior (i.e., hit or miss).

In Figure 4.5 the algorithm proposed for the L1 cache is shown in details; the algorithm is parametric, since the user can choose the value that is written into each memory bit; this value can be all 0's, if the goal is the detection of SEUs turning a 0 into a 1, or all 1's, if the goal is the detection of SEUs turning a 1 into a 0; this value is specified by the input parameter T. The algorithm is expected to be repeated twice with opposite values of T and is independent on the original content of the cache. The steps from 6 to 10 are the most important in the algorithm; V is

a vector defined into the global memory of the device (through the CUDA keyword *__global__*) whose size is equal to the size of the L1 cache; each element of V is of type CACHE_LINE, i.e., a data structure defined for this purpose, whose size is equal to 128 Bytes (the cache line size). In step 6 the vector V is initialized with the value T; in this step the L1 cache is not involved since writing operations are not cached. In step 7, instead, the vector V is read and each element is saved into registers (registers since they are not cached and consequently they cannot interact with the L1 cache); since V is located in the global memory and since the size of V is equal to the size of the L1 cache (which uses the LRU substitution policy), at the end of the step 6 the L1 cache is fully loaded with V (i.e., it is full of 0's or 1's). Step 7 causes cache misses, only. The number of clock cycles needed to manage each access to V in this phase can be computed with the steps 7.1, 7.3 and 7.4, in which the clock count before and after each reading operation is saved into an element of a proper vector L1_t_miss, stored in the shared memory since the shared memory accesses are not cached. During step 8 the device is exposed to radiation. The duration of this step is tuned by the user, and it should be carefully engineered when performing a radiation test. The exposure time, in fact, depends on both the particles flux provided by the facility and on the sensitivity of caches, which may not be known a priori. Step 8 should be long enough to consider negligible the time required to perform steps 5, 6, and 9 but short enough to consider negligible the probability of having more than one particle to corrupt the memory in a single test. This latter consideration is essential to measure the occurrences of Multiple Bit Upsets (MBUs). Moreover, by matching this condition, the probability of a soft-error in the addressed cache location is maximized with respect to the probability of a soft-error in the combinational logic. In the step 10 the vector V is read again, in order to check if during the step 8 any SEU arose in the L1 cache; the performed operations are the same of the step 7, plus a bitwise check (Bitwise Check) to compute the number of bits corrupted by radiation. These read operations cause a sequence of cache hits, since all the elements of vector V are in the L1 cache already; the number of clock cycles needed to manage a L1 cache hit is computed by the steps 10.1, 10.3 and 10.4, and saved in a proper vector L1_t_hit stored in the shared memory.

By reading the content of L1_t_hit and L1_t_miss at the end of the experiment, checking whether they contain any value significantly different from the others, it is possible to detect any fault that caused a miss to be turned into a hit, or vice versa. Hence, this mechanism allows the detection of cache tag malfunctions that lead to unexpected miss or hit behavior: in the former situation, data will be loaded in the cache even if not necessary, impacting the code performance, while, in the latter, wrong or obsolete data will be considered as corrected, impacting seriously the code reliability.

Figure 4.6 shows the algorithm proposed to manage the radiation experiment targeting the shared memory embedded in each SM. The algorithm is similar to

the one proposed for the L1 cache. The data structure used for this algorithm is a vector V defined into the shared memory of the addressed SM (through the CUDA key-word *__shared__*) whose size is equal to the size of the shared memory; each element of V is composed of a type S_MEM_LINE, i.e., a data structure defined for this purpose, whose size is equal to 128 Bytes. This algorithm is composed of two major steps; in the step 5, the shared memory is initialized with the value T (all 0's or all 1's) chosen by the user; in the step 8 all the shared memory locations are read, and the values are controlled with a bitwise check: in this way it is possible to understand if during the step 6 a radiation damaged one or more bit values of the shared memory.

```
T = all 0's | all 1's;
Radiation_test_L1 (T){
1.   L1_size = size of the L1 cache;
2.   Define type CACHE_LINE; /*128 byte*/
3.   V_size= L1_size / sizeof (CACHE_LINE);
4.   Define a vector V of type CACHE_LINE with
     dimension V_size into the Global Memory;
5.   Define Integer vectors L1_t_hit and L1_t_miss
     with dimension V_size into the Shared Memory;
6.   For each element v[I] of the vector V
     6.1.  Write T into V[I];
7.   For each element v[I] of the vector V
     7.1.  t1 = clock();
     7.2.  Read the value of V[I] and save it into a
           register R;
     7.3.  t2 = clock();
     7.4.  L1_t_miss[I] = t2 - t1;
8.   Wait for exposition to neutrons;
9.   err = 0;
10.  For each element v[I] of the vector V {
     10.1. t1 = clock();
     10.2. Read the value of V[I] and save it into a
           register R;
     10.3. t2 = clock();
     10.4. L1_t_hit [I] = t2 - t1;
     10.5. err += BitwiseCheck(R , T);  }
11.  print (err); }
```

Figure 4.5. The pseudo-code of the algorithm for the test of the L1 cache.

```
T = all 0's | all 1's;
Radiation_test_Shared Memory (T){
1.  S_MEM_size = size of the shared memory;
2.  Define type S_MEM_LINE;   /*128 byte*/
3.  V_size= S_MEM_size / sizeof (S_MEM_LINE);
4.  Define a vector V of type S_MEM_LINE with
    dimension V_size into the Shared Memory;
5.  For each element v[I] of the vector V
    5.1.  Write T into V[I];
6.  Wait for exposition to neutrons;
7.  err = 0;
8.  For each element v[I] of the vector V{
    8.1.  Read the value of V[I] and save it into a
          register R;
    8.2.  err += BitwiseCheck(R , T);  }
9.  print (err); }
```

Figure 4.6. The pseudo-code of the algorithm for the test of Shared memory.

### Algorithm for L2 cache

Figure 4.7 shows the proposed algorithm, finalized to manage the radiation experiments for the L2 cache. As said for the L1 and Shared memory test, the algorithm can be executed by a single thread instantiated in a single SM. The L1 cache must be disabled during this test in order to prevent the functionalities of L2 cache to be influenced by the L1 cache (especially when the number of clock cycles needed to manage a cache hit or a cache miss are measured). L1 cache can be easily disabled adding the CUDA parameters *-Xptxas-dlcm = cg* at compile time.

The algorithm designed for the radiation test of the L2 cache is pretty similar to the algorithm addressing the L1; the only difference is the step 7, in which a cleaning operation of the L2 cache is introduced: since both the reading and the writing operations involving the global memory are cached in the L2, the content of the L2 cache has to be cleared after the step 6, before reading for the first time the content of the vector V defined into the global memory. In this way, with the first read operation (step 8 of the algorithm) it is possible to compute the number of clock cycles needed to manage a L2 cache miss (steps 8.1, 8.3, and 8.4); these values are saved into a proper vector L2_t_miss stored in the shared memory. In the step 11, the vector V is read again, in order to check if during the step 9 radiations corrupted the values stored in the L2 cache; the performed operations are the same of the step 8, plus a bitwise check for computing the number of bit values corrupted by radiation. These read operations cause cache hits, since all the data of vector V are in the L2 cache; the number of clock cycles needed to manage a L2 cache

hit are computed by the steps 11.1, 11.3 and 11.4, and are saved into a proper vector L2_t_hit stored in the shared memory. Also in this case the gathered timing information allows the detection of cache tags malfunctions, and the usage of these data will be further discussed in Section 4.1.

```
T = all 0's | all 1's;
Radiation_test_L2 (T){
1.   L2_size = size of the L2 cache;
2.   Define type CACHE_LINE;   /*128 byte*/
3.   V_size= L2_size / sizeof (CACHE_LINE);
4.   Define a vector V of type CACHE_LINE with
     dimension V_size into the Global Memory;
5.   Define Integer vectors L2_t_hit and L2_t_miss
     with dimension V_size into the Shared Memory;
6.   For each element v[I] of the vector V
     6.1.  Write T into V[I];
7.   Clear the L2 Cache;
8.   For each element v[I] of the vector V
     8.1.  t1 = clock();
     8.2.  Read the value of V[I] and save it into a
           register R;
     8.3.  t2 = clock();
     8.4.  L2_t_miss [I] = t2 - t1;
9.   Wait for exposition to neutrons;
10.  err = 0;
11.  For each element v[I] of the vector V {
     11.1. t1 = clock();
     11.2. Read the value of V[I] and save it into a
           register R;
     11.3. t2 = clock();
     11.4. L2_t_hit [I] = t2 - t1;
     11.5. err += BitwiseCheck(R , T);  }
12.  print (err); }
```

Figure 4.7. The pseudo-code of the algorithm for the L2 cache.

### Algorithms execution mode

In this section, the execution mode of the proposed algorithms is presented, considering their execution in a GPGPU environment. When the goal of the test is the evaluation of the soft-error sensitiveness of the L1 cache and of the shared memory, it is necessary that only one thread in a SM composing the GPGPU under test executes the proposed algorithms. In this way, it is possible to acquire the data about the radiation sensitiveness of a single memory (i.e., the L1 cache or the shared

memory) at a time. This is what that has been done in the experiments described in Section 4.1 of this PhD thesis. Moreover, if the goal is to evaluate the radiation sensitiveness of all the L1 caches and of all the shared memories embedded in the GPGPU under test, it is required to activate a thread (executing the algorithms proposed in Sections 4.4.1) in each SM composing the GPGPU. These threads will be executed in parallel by the GPGPU and each of them will use a different input data vector to load into the addressed memory the user defined pattern.

On the other hand, considering the test of the L2 cache, since this memory is a unique memory shared by all the SMs composing the addressed GPGPU, it is necessary that only one thread activated in one SM executes the algorithm proposed in Section 4.4.1.

### 4.4.2 Experimental Results

In this section the most important results obtained applying the proposed method are presented, together with some details about the experimental environment. Finally, some conclusions and future works related to this activity are explained.

As concern the neutron test setup, radiation experiments were performed in May 2013 at the VESUVIO neutron facility at ISIS, Rutherford Appleton Laboratories (RAL), Didcot, UK. The device under test has been irradiated with the available spectrum, that has been already demonstrated to be suitable for emulating the atmospheric neutron flux [103]. The available flux for energies above 10 MeV was of about $3.4 * 10^5 n/(cm^2 * s)$ at 137 cm from the beam source, where the CARMA DevKit was placed. Since for the same energy spectrum the neutron flux at sea level has been measured to be of about $13n/(cm^2 * h)$ [105], ISIS provides an acceleration factor of about $10^8$.

Irradiation was performed at room temperature with normal angle of incidence and the beam was focused on a spot with a diameter of 2 cm plus 1 cm of penumbra. The side of the spot is sufficient to uniformly irradiate the GPGPU chip, leaving the ARM processor, the control circuitry, and critical peripherals out of the beam. This is essential for preventing neutron-induced errors on power switches, which may compromise the experiment, and on the PCI express protocol that manages the data communications between the CPU and the GPGPU. The ARM processor, running a Linux Ubuntu embedded operative system, has been linked through an Ethernet cable to an external controlling computer located in the control room. Moreover, leaving the ARM core out of the beam allowed to safely supervising the experiments from the control room without any error occurrences on the experimental data transmission.

Neutrons are known to generate also permanent failures in electronic devices (e.g., latch-ups) that may cause severe effects on the device functionalities. A latch-up is a radiation-induced short circuit that, if not promptly detected, may destroy

the device. During the test execution the devices under test functionalities have been monitored while the beam was turned off, and never observed any misbehavior: this allow to state that the tested GPGPUs are not prone to experience single event latch-ups.

The performed analysis is based on three experimental campaigns, one related to each memory block (L1 cache, L2 cache and shared memory). Each experimental campaign consists of the execution of the related monitoring programs (i.e., the algorithms presented in Section 4.4.1) in the considered GPGPU. Each monitoring program is supported by the Linux Embedded Operating System (Ubuntu 10.04), running on the ARM processor: it provides a continuous transmission of the experimental data to the external controlling computer; please note that the transmission is performed by the ARM core without any intrusiveness to the GPGPU computation: the Operating System first collects and then analyzes the data provided by the GPGPU execution of the monitoring program, without interfering with its computation. In details, during the execution of each program on the GPGPU the Operating System running in the ARM processor sends to the controlling computer the iteration ID, thread ID, Streaming Multiprocessor ID, and the number of errors possibly identified by the monitoring program.

Furthermore, when L1 and L2 caches are considered, also the cache miss and hit times, measured by the monitoring program during each cache access, are forwarded to the external controlling computer. This allows identifying any error happening in the cache tags or control logic, by checking whether the cache hit times are homogeneous and coherent with the expected values: if a fault occurs in a cache tag, the data stored in the corresponding cache line cannot be loaded by the processor and, consequently, a cache miss is generated, while the expected behavior is a cache hit. The opposite behavior (a cache hit instead of a cache miss) is extremely unlikely.

When performing L1 cache or shared memory radiation experiment, a single thread has been activated, executing the corresponding algorithm on each SM; if the test of the L2 cache is executed, one thread in one SM executes the algorithm. The exposure time for the three algorithms (referred in the Figure 4.5, Figure 4.6, and Figure 4.7 as step *Wait for exposition to neutrons*) has been set equal to about 10s, in order to make negligible (1) the time required to load the memories with the specific values, and (2) the time required to check the content of the memories after the neutron exposition. At the end of the radiation test campaign the whole number of errors observed per GPGPU memory module has been counted. In order to properly compare the sensitivity of each memory module, the GPGPU cross-section has been computed by dividing the number of observed errors per time unit in each module by the number of exposed bits and the average neutrons flux (number of particles hitting the device per unit area and unit time). The main obtained results are reported in Table 4.1: the cross-section data is computed considering all the error events experienced during the whole testing time. Considering the

estimation of the uncertainty, the 95% confidence interval of the cross section data reported in Table 4.1 is estimated to be equal to the 10% of the value in the worst case due to a statistical error. Moreover, the experiments have been performed by applying homogeneous pattern (e.g., all 0s and all 1s) since these patterns allow a right characterization of the memory cell transitions 0 to 1 and 1 to 0 [104].

The ISIS beam features a 1/E spectrum, which is similar to the terrestrial one with an acceleration factor between $10^7$ and $10^8$ in the energy range 10 - 100 MeV [103]. As the neutron spectrum available at ISIS resembles the atmospheric one, it is also possible to predict the Failures In Time (FIT) (i.e., error every $10^9$ hours of operation) of each considered memory resource on a realistic application; this is achieved by multiplying the experimental cross section (see the second column of Table 4.1) by the flux of neutrons with energies higher than 10 MeV that reach the ground (about $13n/cm^2 * h$) [105]. The FIT figures calculated for each of the considered memory are reported in the second and in the last column of Table 4.1. In [107] the error rate at New York City for several SRAM cells built with various design rules are shown to be in the range between $10^{-4}$ and $10^{-2}$ FIT. Unfortunately, there are only few data presented for advanced technologies like the ones considered in this work. The values measured with the proposed test platform are lower than the ones reported in [107], in agreement with prediction on future technology reliability trends. In fact, novel technologies are expected to have an increased reliability with respect to mature ones. A reduced transistor dimension lowers the device cross section, as the exposed area becomes smaller. Nevertheless, reducing the feature size typically reduces the device node capacitance. It is possible that a neutron hitting a bit cell in an advanced device built with a 40 nm technology node (like the tested devices) or smaller may have a higher probability of generating a failure with respect to a neutron hitting a bit cell in a more mature device (like the one presented in [107]). The combination of reduced sensitive area and capacitance is expected to bring an overall benefit in reducing the radiation sensitivity for future technologies [108]. The supply voltage plays a key role in the radiation sensitivity of memory cells. Reducing the memory cells supply voltage has the drawback of linearly increasing the device radiation sensitivity [109]. For current and future devices, the voltage reduction per generation is limited to 5-10% [110]. Hence, no significant radiation sensitivity increase is expected in advanced devices with respect to mature ones due to a reduced supply voltage. As supply voltage reduction in future technology generations is expected to be small, it is possible to anticipate that increases in circuit sensitivity will be relatively small.

As the reader can notice, the shared memory cells appear to be less sensitive than the L1 and L2 cache ones. However, the reported figure is comparable with the sensitivity of L1 cache, whose cells are about twice as sensitive as the shared memory ones. Vice versa, the L2 cache appears to be an order of magnitude more sensitive if compared to the shared memory and the L1 cache. In fact, the L1

cache and the shared memory have a similar function in a GPGPU and are built with similar architectures as they both need to be extremely fast to reduce memory access latencies [102]. On the contrary, the L2 cache needs to be compact and dense to reduce silicon area and is likely to have a different architecture than the L1 and shared memory [102].

If the GPGPU whole memories area is considered, the device FIT ratio is far from being negligible. Considering the differences in terms of SER between the two different patterns, it is possible to highlight that about 70% of errors due to a transition from 0 to 1 (i.e., applying the pattern all 0s), and 30% due a transition from 1 to 0 (i.e., applying the pattern all 1s) for the three addressed memories. Such a great difference among 1s and 0s sensitivities has already been observed in other technologies and devices and relies mostly on the asymmetry of SRAM cells [104].

If the assumption that the three memories are implemented with the same VLSI technology is considered, the sensitivity difference may be due to the different architectural organization of the memory cells. On the basis of the achieved results, the L1 cache and the shared memory are implemented using the same type of cells and that the architectural organization is homogeneous. This is also supported by the fact that the shared memory and the L1 cache dimension can be multiplexed by the user at compilation time. Vice versa, with the obtained results it is possible to support the hypothesis that the L2 cache is implemented using heterogeneous cells according to the RAM scratchpad or Local Store principle [106] which results in a higher radiation effect sensitivity.

The proposed method and the acquired data are extremely important since they are useful to understand the soft-error reliability of the main memories embedded in a NVIDIA GPGPU, since these data are not always published by the manufacturer of the device. Moreover, the GPU producers, like NVIDIA, are likely to know the soft- error sensitiveness of the memory array they use, since this kind of data is provided by the manufacturer. Nevertheless, the operative soft error reliability of the memory array when embedded in the final products may significantly differ from the stand alone memory array.

During the experiments, several Single Event Functional Interrupts (SEFIs) occurred. The effects due to this kind of errors have not been considered, since it is not in the scope of the work: in case of a SEFI during the experiment, the complete experiment has not been considered. This decision was taken because with the current setup it is not possible to understand if the SEFI was caused by a memory misbehavior or by other sensitive elements.

In this particular set of experiments, MBUs have not been observed. To the best of my knowledge, MBUs due to a single particle have not been experienced since the flux provided by the ISIS facility is relatively low, while the MBUs caused by multiple particles have not been experienced due to (1) the exposure time for the three algorithms (referred in the Figure 4.5, Figure 4.6, and Figure 4.7 as step *Wait*

*for exposition to neutrons*) has been set to a relatively short duration (10s, as explained previously), and (2) at the end of each iteration of the proposed algorithms, the addressed memory has been completely reload with the selected patterns, thus avoiding the accumulation of errors in subsequent experiment iterations.

Finally, analyzing the collected access timing data about the L1 and L2 cache hit and miss, 367 cases of L1 cache access misbehavior and 66 cases of L2 cache access misbehavior have been observed; the number of cache line accesses performed in the experiments is the same for both the caches, and it is equal to $10^6$ accesses each. None of the observed SEUs affecting the cache tags caused a miss to turn into a hit: clearly, in this scenario a corruption of one of the bit composing a cache tag could only cause a cache hit to become a cache miss. Instead, it is not possible that a cache miss becomes a cache hit, since all the data to be read at the end of the neutron exposure are already present in the cache itself; consequently, the expected behavior, in case the cache tags are not corrupted by the radiation exposure, is a sequence of cache hits, while in case a cache tag has been corrupted by a neutron, the only possible behavior is that a cache hit becomes a cache miss. This is the reason why in this experiments any transformation from a cache miss to a cache hi has been observed; hence, only performance degradation was experienced as a result [90].

Table 4.1. GPGPU memory error cross-section and FIT. The 95% confidence interval is estimated to be equal to the 10% of the value in the worst case due to a statistical error.

| Module | Soft error cross-section ($cm^2/bit$) | F.I.T. (per bit) | Soft error cross-section ($cm^2/device$) | F.I.T. (per device) |
|---|---|---|---|---|
| L1 cache | 5.40E-15 | 7.02E-5 | 1.42E-09 | 1.84E01 |
| Shared memory | 3.15E-15 | 4.09E-5 | 2.48E-09 | 3.22E01 |
| L2 cache | 1.29E-14 | 1.60E-4 | 1.35E-08 | 1.68E02 |

As a conclusion of this research activity, in this section a new method allowing to effectively evaluate and analyze the radiation effects in GPGPU L1 and L2 data caches and shared memory has been presented. Despite of the few information delivered about the memory architecture and the difficulties in accessing them directly, the proposed algorithms allow to efficiently face these problems, so that the addressed memories can be properly initialized and read when radiation experiments are performed. The proposed algorithms have been validated through neutron-based radiation test experiments. The collected results provide some first

data about the sensitivity to radiation effects of memory modules within GPGPU devices. The reported figures can be crucial for evaluating the reliability of applications resorting to GPGPUs and to support the application engineer in decisions related to the possible usage of the addressed memories. As future works, an extension of the presented analysis will be executed, in order to address the internal logic modules controlling the memories, either shared and cache, and to devise methods to increase the GPGPU memories reliability.

## 4.5 Evaluation of Embedded GPGPU algorithms

In this section the soft error sensitiveness data of a typical parallel algorithm, executed with different GPGPUs cache configurations is presented. Moreover, the data about the same algorithm, but executed with different threads distribution are also highlighted, in order to provide an estimation of the most reliable GPGPU configuration when running a common algorithm. In order to be fully compliant with the Degree of Parallelism (DOP) adopted by embedded GPGPUs, the progressive reduction of the FFT parallel tasks from iteration to iteration has been considered, depicting both overall GPGPU cross-section and the error rate for all the FFT stages. this algorithm has been selected for this evaluation since the progressive reduction of the parallel tasks is a common behavior of several GPGPUs algorithms (e.g., the Breadth-First Search - BFS algorithm, where the 30% of the GPGPU execution time is managed by one thread), and it is called *underutilized parallelism in GPGPU computing* [120]. Consequently, even if the analysis is performed only on the FFT algorithm, the results and discussion presented in tho PhD thesis within this context are extendable to any other similar parallel algorithms for embedded GPGPUs. The proposed analysis has been implemented on a device by NVIDIA with Fermi architecture and the data has been acquired in an extensive radiation campaign at the ISIS facility in the Rutherford Appleton Laboratories (RAL) in Didcot, UK. The gathered experimental results provide interesting data about the sensitivity to radiation of GPGPUs device when different configurations are considered.

### 4.5.1 Proposed method

The main goal of the proposed research work is to evaluate how parallel algorithms behave when executed in embedded GPGPUs. Additionally, an investigation on how the radiation-induced errors propagate in the parallel algorithm till reaching the output is also performed. Such a study is of great interest as it highlights the weaker parts of the code that should be hardened to increase the device reliability. Additionally, the effects of different threads and caches distributions on the

GPGPUs output error rate are analyzed. In this way it is possible to understand which GPGPU configuration ensures the highest level of reliability under several constraints.

The choosen benchmark is the Cooley-Tukey algorithm [122], the most common FFT implementation for embedded applications. Cooley-Tukey algorithm allows to reduce the computations complexity from $O(N^2)$ to $O(N * log_2 N)$, where N is the number of the input data. The input data of the tested algorithm is composed of 65,536 complex numbers, each of which represented with 2 float values.

First of all, a CUDA version of that algorithm has been implemented, taking into account the parallel resources, the memory usage, and the programming paradigm provided by the NVIDIA environment. In Figure 4.8, the Degree Of Parallelism (DOP) of each stage is outlined. At each stage a FFT butterfly unit [123] combines the results of two smaller Discrete Fourier Transforms (DFTs) into a large DFT [124]. The butterfly units on a stage can be executed in parallel, while to start the next stage computation it is necessary to wait for the current stage computation to be completed. The number of parallel tasks required for computation, then, decreases exponentially from a stage to the following one. In the tested code the number of stages required to solve the problem is 16, since the input data are 65,536 complex numbers. The stages management is a task of the host device that controls the GPGPU, which computes, per each stage, the optimal number of threads and of thread blocks to be instantiated (considering the addressed GPGPU configuration). In the ideal solution for performances each thread should compute only one butterfly unit of the Cooley Tukey algorithm [123]. Nevertheless, since the required number of threads would be too high (almost 32K in the case of 65,536 input numbers), in the initial stages of the computation each thread is asked to compute one or more butterfly units. In the initial stages of the algorithm the GPGPU is then fully loaded, while in the final stages the number of the parallel processes needed decreases, becoming lower than the number of computing units available in the NVIDIA Quadro GPGPU. In the last stage, for instance, a single process is instantiated (see Figure 4.8). It is worth noting that such a behavior of having the GPGPU fully loaded only in the early stage of computation is typical of embedded GPGPUs. On the contrary, GPGPUs used in High Performance Computing (HPC) usually are continuously loaded to take full advantage of their computational capabilities.

In the FFT algorithm execution addressed in the context of this PhD Thesis, the L1 caches are always fully exploited (when they are enabled), since the number of input data is big (65,536 complex numbers); consequently, the reliability impact of the L1 caches in the different steps of the algorithm execution is the same.

To characterize the behavior of embedded GPGPUs the FFT algorithm has been developed in order to instantiate a maximum of 2 thread blocks and 64 threads per block (FFT_64), and then 2 thread blocks and 32 threads per block (FFT_32). As 2 SMs are available in the tested GPGPU, the block scheduler is not activated.

FFT_32 does not require thread scheduling, since the 32 instantiated threads can run at the same time in parallel given that 48 are available per SM in the Quadro 1000M GPGPU. Vice versa, for the FFT_64 configuration the thread scheduler is mandatory since the parallel threads instantiated exceed the maximum threads that can be executed in parallel. Moreover, the FFT algorithm with 2 thread blocks, 64 threads per block and the L1 cache of each SM disabled (FFT_64_NOL1) has been also tested. Disabling the L1 may sensibly reduce the number of errors observed at the output. Even if performances are likely to be affected when L1 is disabled (see Figure 4.9, and Figure 4.12), on safety critical applications the execution time is not as critical as reliability.

The considered configurations have different execution times. In Figure 4.9, the total GPGPU Kernel time for the different GPGPU configurations is shown: FFT_32 kernel time increases of about 21.8% with respect to FFT_64, while disabling the L1 cache increases execution time of only about 4.5%. The performances degradation is not remarked as the L1 cache, although completely filled during computation (the tested FFT uses about 1MB of data), is used only for reading operations. Consequently, the writing operations (about 65,536 for each stage) performed by the FFT algorithm don't involve the L1 cache. As reported in Figure 4.12, the FFT_32 and FFT_64 have the same kernel time after the stage 9, but in the stages 1 to 9, the kernel time of the FFT_32 is almost the double if compared with the FFT_64 configurations.

In Figure 4.10 and Figure 4.11 the profile of the used Thread Blocks and of the number of threads used in each block, considering the FFT stages, is shown. More in particular, the number of the parallel tasks decreases after the stage 9, in case of the FFT_32 configuration is used, and after the stage 10, in case the used configuration is FFT_64. In these figures, the profile of the FFT_64_NOL1 configuration is not reported, since this configuration is equal, in terms of used threads and blocks, to the FFT_64.

Considering the GPGPU utilization shown in Figure 4.11, Figure 4.11, and Figure 4.12, an higher number of errors in the first stages of the FFT algorithm execution, i.e., when all the computational resources of the GPGPU under test are fully exploited, is expected. Finally, considering the FFT behavior, the proposed analysis is valid for all the GPGPU algorithms with a similar usage of the L1 cache, i.e., the algorithms that initially acquire the required data from the L1 cache, and then make a long computation (in terms of clock cycles) without interacting more with the cache itself. It is clear that the data reported in this section are not completely suitable for the algorithms that continuously access the data stored in the considered cache.

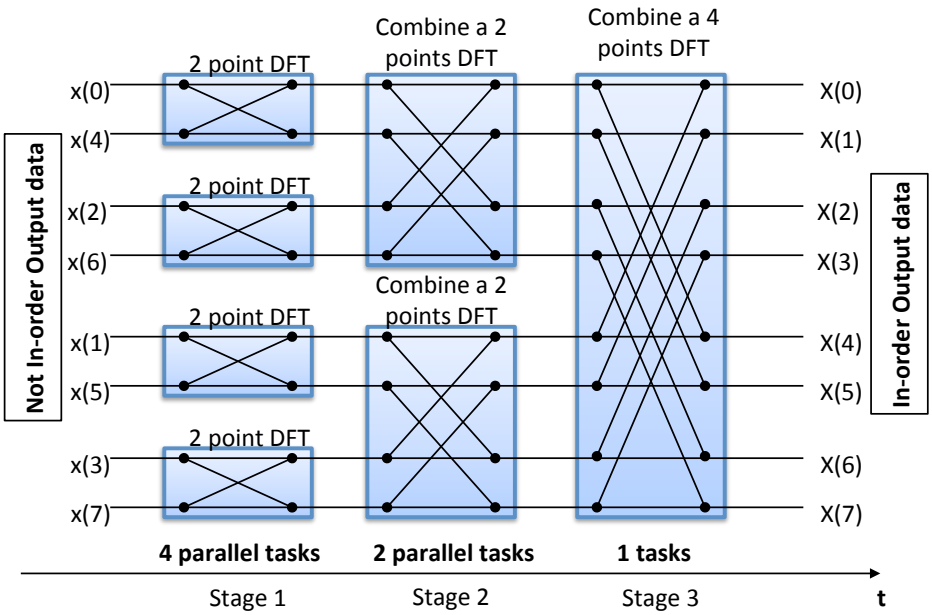



Figure 4.8. The representation of the butterflies for each stage of the FFT algorithm, independently from the GPGPU configuration. The number of parallel tasks at each stage (highlighted with boxes in the figure) decreases exponentially from a stage to the following one.

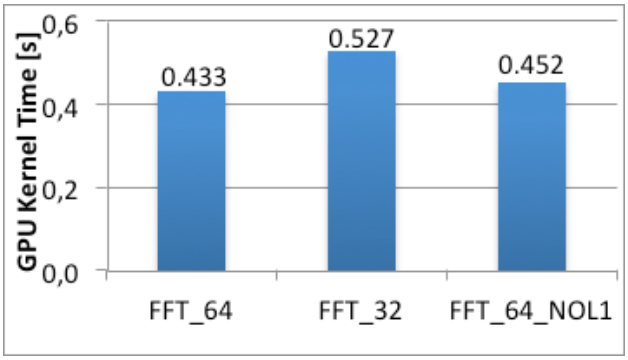



Figure 4.9. The total GPGPU Kernel Time increases in the cases of 32 thread (FFT_32) and disabling the L1 cache (FFT_64_NOL1).

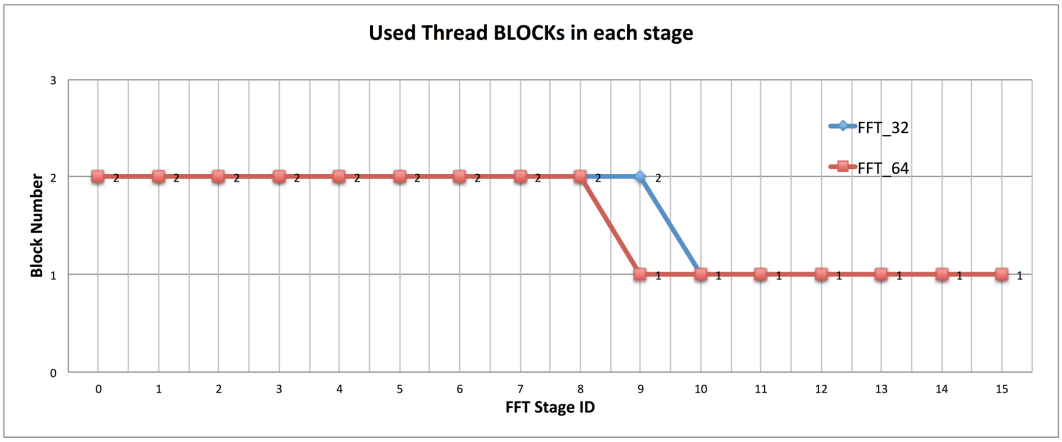



Figure 4.10. The number of the used Thread Blocks per each FFT Stage .

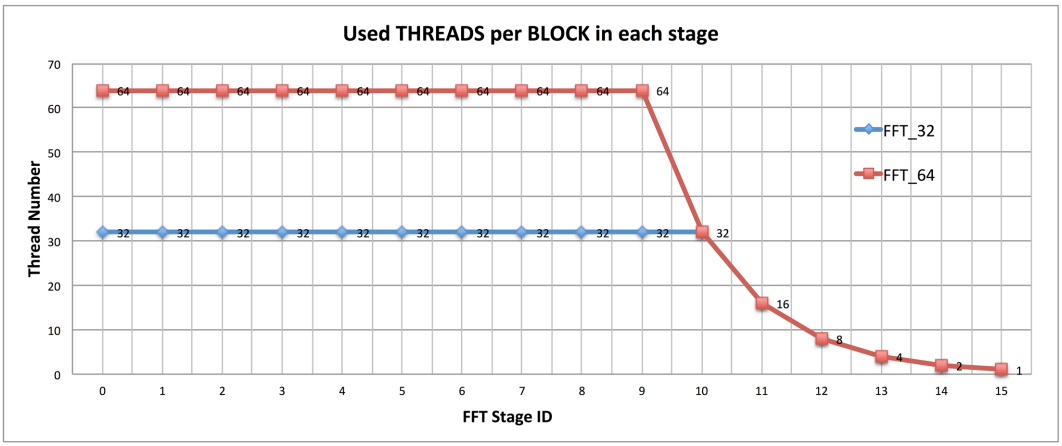



Figure 4.11. The number of the used Threads per Block in each Stage of the FFT algorithm.

### 4.5.2 Experimental setup

Radiation experiments were performed at the VESUVIO neutron facility at ISIS, Rutherford Appleton Laboratories (RAL), Didcot, UK in December 2013. The device has been irradiated with the available spectrum that has been already demonstrated to be suitable for emulating the atmospheric neutron flux [103]. The available flux was of about $3.89 * 10^4 n/(cm^2 * s)$. Irradiation was performed at room temperature with normal angle of incidence and the beam was focused on a spot with a diameter of 2 cm plus 1 cm of penumbra. Spot size is sufficient to uniformly irradiate the GPGPU chip, leaving the ARM processor, the control circuitry, and

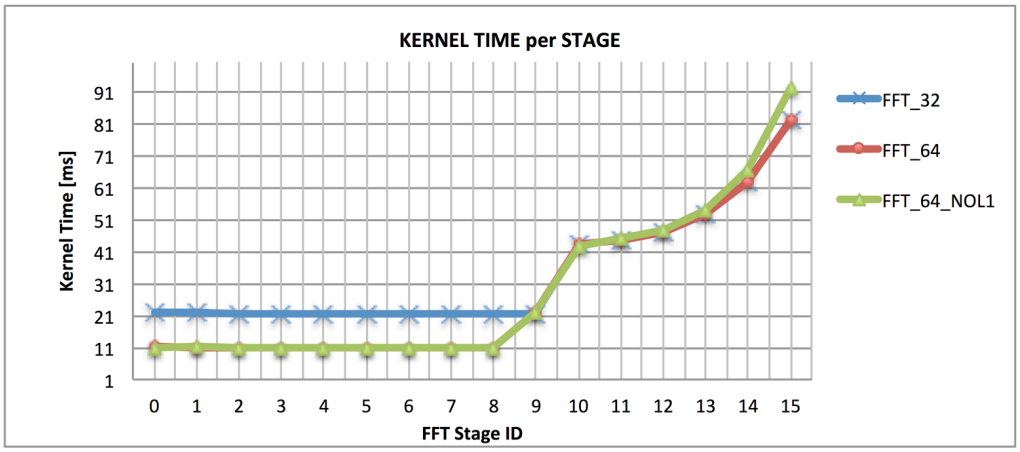



Figure 4.12. The GPU Kernel time per each FFT Stage.

critical peripherals out of the beam. This is essential for preventing neutron-induced errors on power switches, which may compromise the experiment, and on the PCI express protocol that manages the data communications between the CPU and the GPGPU.

### 4.5.3 Experimental results

Figure 4.13 shows a detailed analysis of the observed errors propagation in the FFT algorithm reporting the error rate of each stage of the FFT algorithm (i.e., the number of errors generated in a stage divided by the time required to compute the stage itself). As it can be noticed, in the initial stages of the algorithm (from stage 1 to stage 10) the error rate is much greater than in the later stages. In stages from 0 to 10 the GPGPU is fully loaded since the instantiated parallel processes exceeds the GPGPU parallel resources and the thread schedulers is active. From stage 11 to 15 the number of threads instantiated is lower than the number of available SPs, till the extreme case of stage 15 in which a single thread is active (see Figure 4.8). The area exposed to radiation is then the whole GPGPU in the first 11 stages, and then decreases making it more likely for the GPGPU to be corrupted when fully loaded. Moreover, in some stages of the FFT algorithm, any error has been detected during the execution of the experiments (in all the considered configurations). This is mainly due to the fact that the error rate was low if correlated to the area exposed to neutron radiation.

Figure 4.14 shows the experimentally obtained cross sections for the three different tested GPGPU configurations (FFT_64, FFT_64_NOL1, and FFT_32). The cross section was measured dividing the number of observed error per second by the

average neutron flux. For each cross section value the confidence interval (drawn with oblique lines) calculated considering a statistical error of 10% in the worst case is also reported. As it can be noticed, even if the algorithm and thus the workload is the same, the three configurations show different cross sections. Such a different sensitivity to radiation relies on the amount of resources required to compute the solution more than in the workload that, again, is constant in the three tested cases.

It is clear from Figure 4.14 that, as expected, disabling the L1 caches reduces the cross section of the algorithm of 38%, while the performance overhead is not so relevant (about 4.5%, as already explained in the previous section and in Figure 4.9). As said, the significant decrease of the cross section is mainly due to the fact that the L1 cache is a very sensitive memory [8]. The contribution of L1 caches corruption in the overall SM radiation sensitivity is not negligible, especially when the algorithms perform a large amount of memory accesses. Obviously, when disabling L1 caches the execution time is increased (see Figure 4.9). Nevertheless, in safety critical applications, performance may be sacrificed to gain higher reliability. It is worth noting that even if performances are not a major concern, having a longer execution time will force the GPGPU to be irradiated for a longer time before solving the assigned task. As demonstrated in [126], it is essential to carefully evaluate if disabling caches actually improve the device reliability.

FFT_32 has a 25% higher cross section than FFT_64. Limiting the number of parallel threads per block seems to increase the GPGPU sensitivity. This behavior is mainly due to a different data distribution in the GPGPU caches. The 32 instantiated threads per block in FFT_32 are always active in a SM (each SM is composed of 48 SP); consequently, the data used by the threads in FFT_32 is present in the L1 cache for all computation. On the contrary, in FFT_64, the number of instantiated thread (i.e., 64) imposes the thread scheduler to continuously swap the active threads. The data in the L1 cache is then refreshed frequently. If data is refreshed frequently, the Time Vulnerability Factor [121] (TVF, i.e., the portion of time during which data is critical and a corruption propagates to the output) becomes smaller, and possible error caused by a radiation is more likely to corrupt obsolete data. When threads scheduling is enabled, as in FFT_64, a refresh of the data stored in the L1 cache is expected and, thus, the cross section is lower.

As already highlighted in the previous paragraph of this section, comparing the FFT_32 and FFT_64 configurations, it is possible to notice that FFT_32 has an higher cross section and an higher kernel time. It must be underlined that these two behaviors are not correlated, since the cross section values have been calculated dividing the number of observed errors per second by the average neutron flux (about $3.89 * 10^4 n/(cm^2 * s)$) present during our experiments. Consequently, the higher execution time of the FFT_32 configuration is not the reason of the higher cross section. The reason of this effect is the different Time Vulnerability Factor of the data present in the L1 caches due to the different threads number executing the

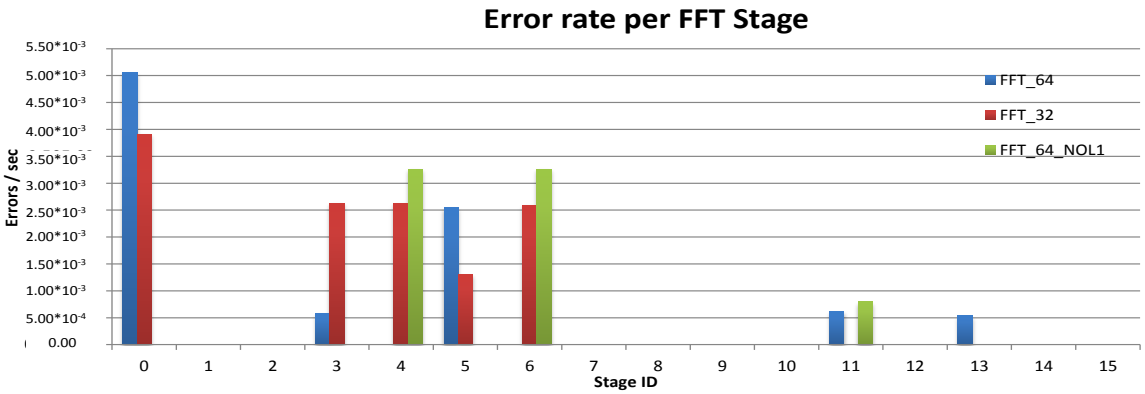



Figure 4.13. The error rate of the different stages of the FFT algorithm, for each GPGPU configuration.

same FFT algorithm.

The most interesting point highlighted by the reported data is that, when the reliability of a GPGPU is a concern, all the stages of the algorithms should be carefully designed. The designers of safety critical applications to be executed on GPGPUs should, in fact, take into account that the reliability of the GPGPU applications varies according to the number of parallel tasks running at the same time in the GPGPU itself, as shown in Figure 4.8. Moreover, hardening techniques should be tuned taking into account that most errors occurs in the early stages of the algorithm, when the GPGPU is fully loaded.

From data depicted in Figure 4.8 and Figure 4.14, it is clear that disabling the L1 caches (contained in the SMs embedded in the considered GPU), the impact in terms of performance is low, but the reliability of the GPU device increases significantly. This is a very important aspect when the design of safety critical applications is addressed: the tradeoff between the reliability improvement and the introduced overhead is one of the most important parameter that has to be considered. With the presented analysis, it is possible to understand that, if the algorithm has to be used in safety critical GPGPU applications, disabling the L1 caches represent a good solution in terms of reliability and performance overhead.

As a conclusion of this research activity, in this section the reliability of embedded GPGPUs has been addressed, and the cross section of different embedded GPGPU configurations running the FFT parallel algorithm based on the Cooley Tukey method has been experimentally determinated. The results show that the most reliable configuration is obtained disabling the L1 cache memory. Moreover, on embedded GPGPUs oriented to safety critical applications the number of parallel processes to instantiate should be carefully design to avoid undesired behaviors.

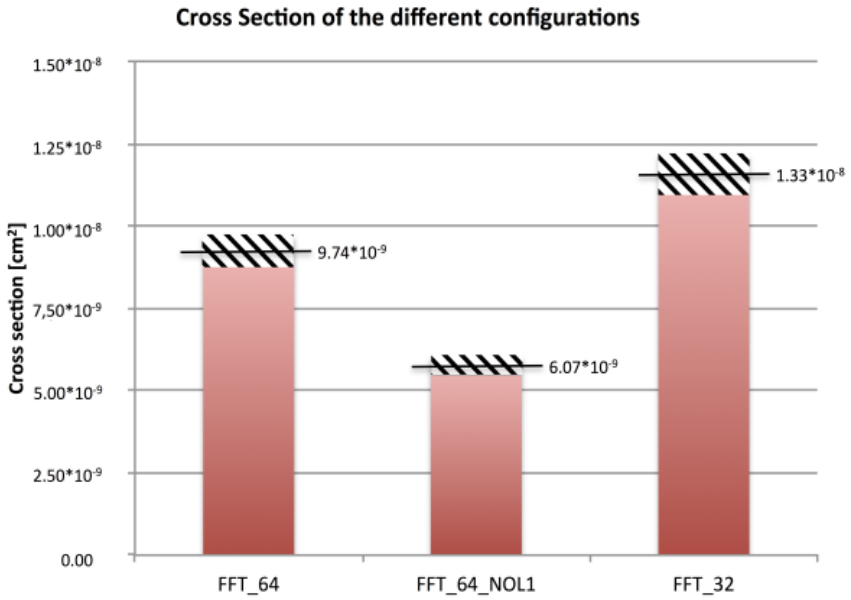



Figure 4.14. The cross section of the different configurations of the FFT algorithm. The confidence interval (drawn with oblique lines) has been calculated considering a statistical error of 10% in the worst case.

## 4.6 Fault tolerance techniques evaluation

Due to their internal structure complexity, GPGPUs show a relatively high sensitivity to soft errors. Hence, there is some interest in devising and applying software techniques able to exploit their computational power by just acting on the executed code. In this section some preliminary results obtained by applying two different software redundancy techniques aimed at soft-error detection are presented; these techniques are completely algorithm independent, and have been applied on a sample application running on a Commercial-Off-The-Shelf GPGPU. The results have been gathered resorting to a neutron testing campaign. Some experimental results, explaining the capabilities of the methods, are presented and commented, along with some activity conclusions.

Considering the related works in this area, in [115] three redundancy techniques are presented, applied on different GPGPU architectures, with a particular emphasis on the evaluation of the effects on performance. In this work the authors propose three different methodologies based on software addressing soft-error reliability based on redundancy; moreover, they also discuss how these approaches can be enhanced by adding ECC/parity protection in off-chip global memory and in on-chip memory [115]. Finally, a detailed evaluation of the performance reduction due

to the introduced redundancy is presented. This research work represents the first work in the field of the software soft-error detection related to GPGPUs, but the proposed techniques have not been validated with a fault injection experiment campaign. As a consequence, the research work proposed in this section provides the first experimental data about the evaluation of algorithm-independent techniques aimed at the detection of soft-errors on GPGPUs; the considered hardening solutions have been evaluated with neutron radiation experiments: the main scientific contribution provided by this research work is the possibility of effectively evaluating the impact of soft-error detection techniques applied to a common benchmark application running on a Commercial-Off-The-Shelf GPGPU. As a benchmark application the matrix multiplication algorithm has been selected, since it is very commonly used for the experiments in the field [95] [97].

### 4.6.1 Developed method

In this section the soft error detection techniques addressed in this work are presented. The choosen techniques are based on time and thread redundancy [115]. In the following sections the steps required to apply the redundancy techniques to applications running on a NVIDIA GPGPU based on Fermi Architecture are shown. Moreover, for each technique two different approaches are presented:

- redundancy-based with common input data;
- redundancy-based with different instances of the input data.

The goal is the evaluation of the soft-errors detection capability of these four methods. The proposed redundancy methods are completely algorithm-independent and are easily applicable to any algorithm running on a GPGPU. The soft error detection capability of these methods has been validated through neutron injection campaigns; a detailed report of the detection capability and of the introduced temporal overhead of the considered methods is discussed in Section 4.6.2 .

#### Time redundancy fault detection approach

The time redundancy fault detection approach is based on forcing the GPGPU to execute twice the same code on the same data, and then checking the two sets of results for consistency. In Figure 4.15, the diagram representing the sequence of steps performed by both the host computer (denoted as CPU) and the GPGPU is shown. In particular, the sequence is composed of 5 main steps: initially, the host initializes the device memory, writing into it the source data D and allocating the result memory area; in the second step the selected algorithm is executed in the GPGPU, and then the result R is acquired by the host; in the third step, the

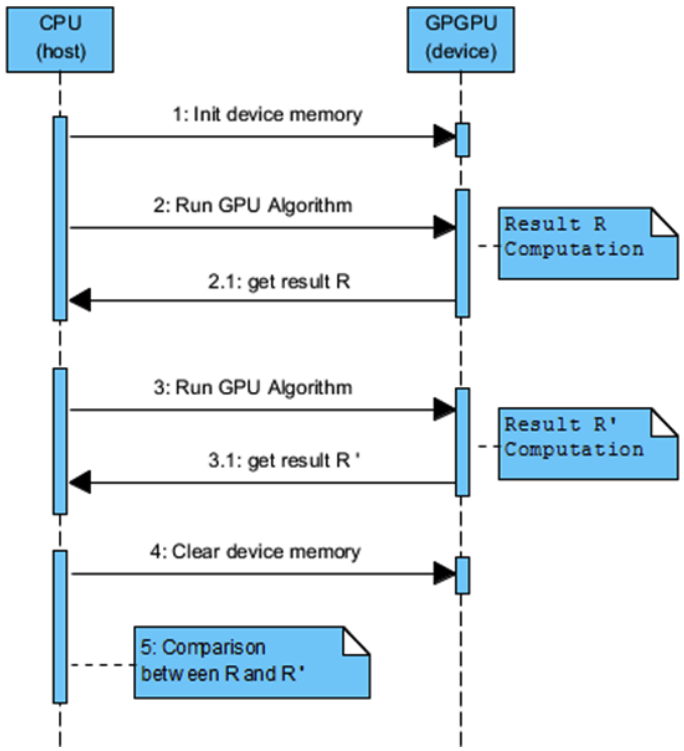



Figure 4.15. Sequence diagram of the time redundancy approach with common input data.

algorithm is executed again, generating the second instance of the result R', using the same source data D used in the step 2. Then, the device memory is cleared. Finally, the host performs the comparison between the two obtained results R and R'.

In Figure 4.16, the same diagram is shown for the same time redundancy approach, now implemented with two input data instances. In this case the two algorithm computations produce the results R (normal computation) and R' (replica computation) operating with two different instances of the input data D and D'. If compared with the time redundancy method with common input data (see Figure 4.15), there are two additional steps, i.e., steps 3 and 4; these two new iterations are required to clear and then re-initialize the device memory with a new instance of the source data D' at the end of the first algorithm computation (which produces the result R). In the step 5, the algorithm is executed again using D' as input data and producing R' as a result. Finally, the host performs the comparison between R and R'.

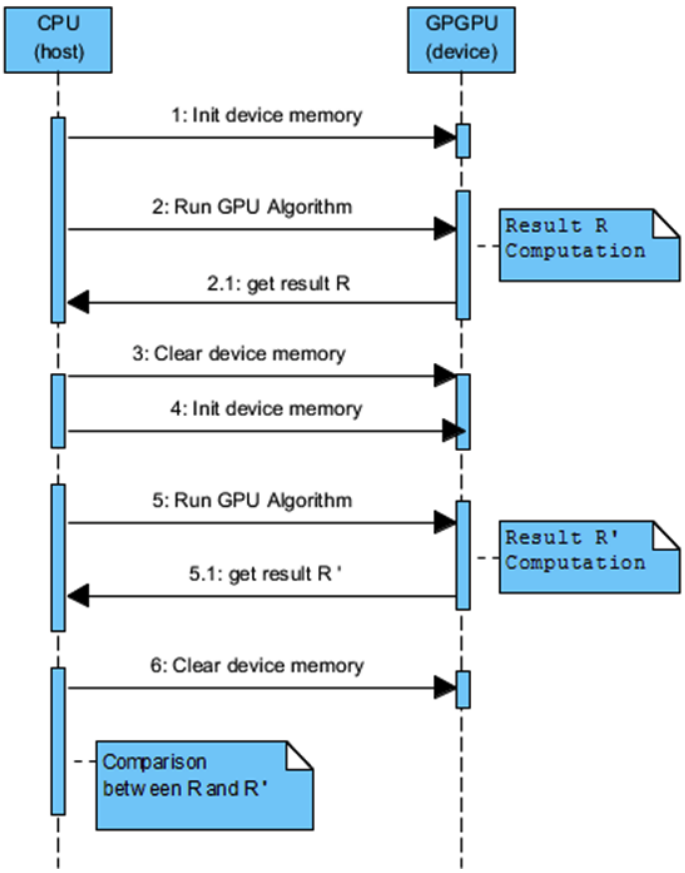



Figure 4.16. Sequence diagram of the time redundancy approach with different input data.

### Thread redundancy fault detection approach

The purpose of the thread redundancy method is to exploit the GPGPU capability of managing a high number of threads in parallel at the same time in order to implement the redundancy and minimizing the introduced overhead.

In general, the NVIDIA programming model implies that a GPU algorithm is executed as a grid of thread blocks, each one containing several threads. In the proposed method, a thread block is internally subdivided in two sets of threads: the first is executed by a defined set of SPs and produces the algorithm result R, while the second is executed by a different set of SPs and produces the replica result R'. The partitioning of the thread block is performed using the block index and the thread index, which are available within a kernel function running in a GPU; finally, the proposed method works properly if only one thread warp is executed by a SM at

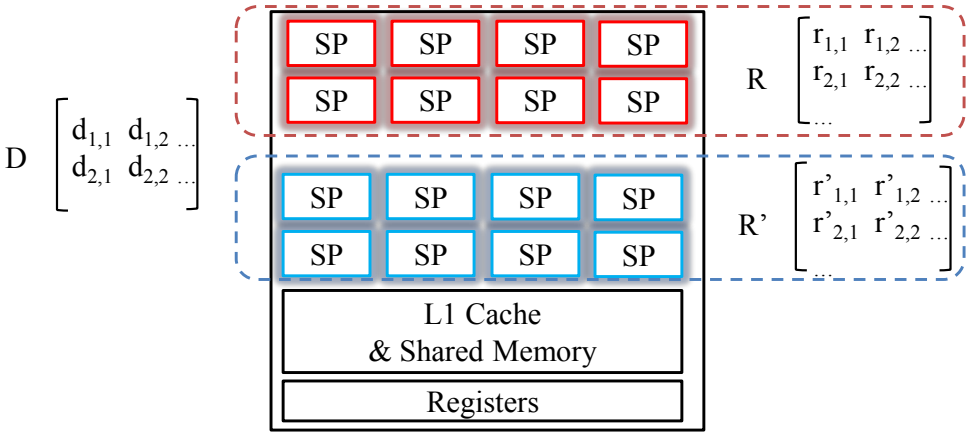



Figure 4.17. A simplified representation of the thread redundancy approach with common input data.

a time, in order to ensure that the threads are not executed in time slice way [98].

In Figure 4.17, a simplified representation of the thread redundancy approach with common input data is shown; in this case, the two instances of the result R and R' are computed using the same instance of the input data D. At the end of the result computation, R and R' are read by the host which then performs their comparison, in order to detect possible errors generated during the computations.

In Figure 4.18, a second version of the same thread redundancy approach is proposed, in which two instances of the input data (D and D') are used; consequently, the two sets of threads operate on a proper instance of the input data, and produce the results R and R' in an independent way. Finally, as in the previous cases, the host reads R and R' and performs the comparison.

### 4.6.2 Results

In this section the gathered results and the neutron testing setup are presented.

#### Neutron testing setup

Experiments have been performed at Los Alamos National Laboratory's (LANL) Los Alamos Neutron Science Center (LANSCE) Irradiation of Chips and Electronics House II, called ICE House II, in August 2013. The ICE House II beam line is a new facility placed at 30° angle from the beam, much like the original ICE House facility. Both of these facilities provide a white neutron source that emulates the energy spectrum of the atmospheric neutron flux. The available neutron flux was

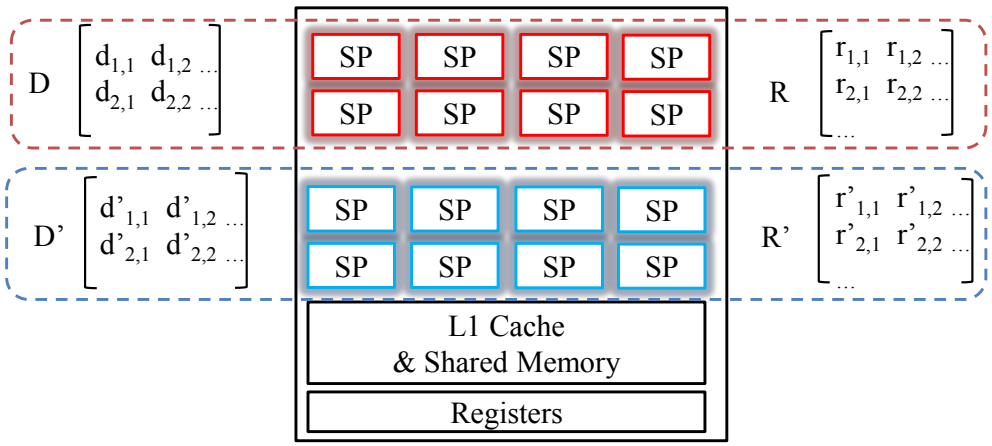



Figure 4.18. A simplified representation of the thread redundancy approach with different input data.

approximately $1 * 10^6 n/(cm^2 * s)$ for energies above 10 MeV. The beam was focused on a spot with a diameter of 2 cm plus 1 cm of penumbra, which provided uniform irradiation of the GPU chip without directly affecting nearby board power control circuitry, ARM processor, and critical peripherals. This is essential for preventing neutron-induced errors on power switches, which may compromise the experiment, and on the PCI express protocol that manages the data communications between the CPU and the GPGPU. The CARMA DevKit board has been located inside the radiation room and placed at a distance of 70 inches from the beam source; the ARM processor, running a Linux Ubuntu embedded operative system, has been linked through an Ethernet cable to a host computer located in the control room. Moreover, leaving the ARM core out of the beam allowed to safely supervise the experiments from the control room without any error occurrences on the experimental data transmission.

### Radiation testing analysis and results

The purpose of this section is the evaluation of (1) the soft error detection capability of the proposed methods, and (2) the measure of the performance degradation, i.e., temporal overhead, due to the insertion of the replica computation.

The benchmark application used to evaluate the proposed method is the matrix multiplication algorithm, since this algorithm allows to fully exploit the parallel computational units (i.e., the SPs) embedded in a GPGPU. Nevertheless, the proposed methods are algorithm independent and are suitable for any GPGPU algorithm. The implemented matrix multiplication algorithm operates with 2 matrices of 1KB

size (16x16 integer values) as input data and generates as a result a matrix of 1KB size. For the experiments with the time redundancy technique16 thread blocks, each one with 16 threads, have been instantiated, where each thread is in charge of calculating just one element of the result matrix. For the thread redundancy technique, instead,16 thread blocks, each one with 32 threads, have been instantiated, where half of the threads in a block were devoted to compute the results and the other half to compute the replica.

The experimental campaigns have been performed exposing only the GPUPU device to the neutron flux; moreover, the experiments have been executed of about $1.5x10^6$ iterations for each of the proposed techniques.

In Table 4.2, the main results obtained during the radiation experiments are reported, with particular emphasis on the execution times and on the number of soft error detected. In the second column of Table 4.2 the total time required for a complete iteration of the matrix multiplication algorithm (for each of the proposed methods) is reported. In general, when a kernel function is executed into a GPU, the main steps performed by the CPU and the GPU are three:

- the CPU initializes the GPU memory with the input data;
- the GPU executes the kernel algorithm;
- the results are copied from the device memory into the host memory.

In the third column of the Table 4.2 (i.e., CPU-GPU MemoryCopy Time) the percent values of the sum of the time required by the step 1 and 3, with respect to the total iteration time, are reported; instead, the percentage time required to compute the kernel function is reported in the fourth column (i.e., GPU Kernel Time). Since the goal of the techniques presented in this section is the evaluation of the soft error detection techniques applied at the algorithm execution, it is essential that the time required by the GPU to manage the memory transfers (from and to the CPU memory) is much less than the time spent to execute the algorithm with the detection technique. In this way, checking the results one can state that all the detected errors have been generated during the algorithm execution, and not during the memory management. Considering the data reported in the fourth column of Table 4.2, in a single iteration the time devoted to the algorithm execution is greater than 90% in all the considered cases. To achieve this result, in the experiments the matrix multiplication algorithm has been modified: the steps needed to compute each value of the result matrix have been repeated $10^5$ times, adding into the result matrix the obtained values.

If the execution times are considered, the temporal overhead introduced by the time redundancy technique is about 100% for both the cases (single data instance and double data instance), as common in the traditional duplication with comparison

approaches based on time redundancy [115]. In the thread redundancy approaches, instead, the temporal overhead is almost negligible, since the threads devoted to the replica computation are executed in parallel with the threads aimed at the regular computation.

Considering the detection capability, in the last column of Table 4.2 ( i.e., Detected Errors column), for each technique, the percentage of the detected errors with respect to the observed errors are reported. The detected errors have been generated by the host device by comparing the two result matrices obtained through the execution of the redundancy approaches in the GPU. The observed errors, instead, have been computed by comparing the result matrices with the expected values contained in the CPU memory. As the reader can notice, the error detection capability of the techniques operating with two instances of the input data is always higher if compared with the same technique executed with only one instance of the input data. Moreover, the time redundancy technique with two data instances has the higher soft error detection capability.

Table 4.2. The results obtained applying the proposed fault detection techniques to the matrix multiplication algorithm.

| Method | 1 Iteration Time evaluation | | | |
|---|---|---|---|---|
| | Total Execution Time [ms] | CPU-GPU MemoryCopy Time [%] | GPU Kernel Time [%] | Detected Errors [%] |
| Original | 12.40 | 6.89% | 98.15% | - |
| Time Redundancy (2 input instances) | 24.50 (+97.58%) | 5.71% | 94.29% | 87.50% |
| Time Redundancy (1 input instance) | 24.27 (+95.73%) | 4.82% | 96.18% | 68.75% |
| Thread Redundancy (2 input instances) | 12.91 (+4.11%) | 5.75% | 94.25% | 75.00% |
| Thread Redundancy (1 input instance) | 12.74 (+2.74%) | 4.75% | 95.25% | 62.50% |

As a conclusion of this research activity, in this section a couple of methods aimed to soft error detection for algorithms running on a GPGPU are presented. The proposed methods have been applied to a benchmark application and have been validated through neutron-based radiation test experiments. The collected results

provide some first interesting data about the capability of the traditional soft error detection techniques if applied to applications running in GPGPU devices.

# Chapter 5

# An industrial demonstration: test and validation of an automotive timing multicore co-processor module

Over the past decade, the complexity of electronic devices in the automotive systems is increased significantly. The today high level vehicles include more than 70 Electronic Control Units (ECUs) aimed at manage the powertrain of the vehicle, and improve passengers comfort and safety. ECU microcontrollers aimed at the control of the fuel injection system have a key role. In this section a new FPGA-based platform is introduced; it is able to supervise and validate Commercial-Off-The-Shelf timer modules used in today state-of-the-art software applications for automotive fuel injection system with an accuracy improvement of more than 20% with respect to traditional approach. The proposed approach allows an effective and accurate validation of timing signals and it has two main advantages: can be customized with the exact timing module configurations to meet the constraints of new tests and allows effective modularity design test. As case study two industrial Time Modules manufactured by Freescale and Bosch have been used. The experimental analysis demonstrate the capability of the proposed approach providing a timing and angular precision of 10 ns and $10^{-5}$ degrees respectively.

This work has been done in collaboration with General Motors Powertrain Europe (site of Torino, Italy).

## 5.1 Motivation and introduction

The today automotive development processes are characterized by an increasing complexity in mechanic and electronic. However, electronic devices have been the major innovation driver for the automotive systems in the last decade [129][130]. In this context, the requirements in terms of comfort and safety lead to an increasing number of on-vehicle embedded systems, with more and more software-dependent solutions using several distributed Electronic Control Units (ECUs).

Sophisticate engine control algorithms require performance enhancement of microprocessors to satisfy real-time constraints [131]. Moreover, the code generation, the verification, and the validation of the code itself, become key part in the automotive domain: the software component development processes have to be as efficient and effective as possible. Moreover, without a reliable validation procedure, the automotive embedded software can lead to a lot of errors and bugs, and decrease the quality and reliability of application software components.

Electronic devices managing the fuel injection in today engines have a key role, in order to guarantee efficient and powerful vehicles [132]. The recent research works in the area are aimed at reducing the fuel consumption, while maximizing the power conversion and reducing air pollution emissions [133][134]. As reported in [135], a major challenge being faced in diesel technology is meeting current and future emission requirements without compromising fuel economy. Clearly, these goals could be reached through improvements in the engine electronic management. In this context, efficient fuel injection control is required [136].

Given this behavior, microcontrollers devoted to the engine management contain specific timer modules aimed at generating, among the others, the signals used by the mechanical parts controlling the fuel injection in the cylinders[137]. The scope of these timers is to provide the real- time generation of the signals, ensuring an efficient engine behavior. In order to achieve the correct level of synchronization between the engine position and the generated fuel injection signals, the automotive timer modules typically receive a set of reference signals from the engine; the most important are the *crankshaft* and the *camshaft* [138]. These signals are used to detect the current engine position, i.e., the angular position of the cylinders within the engine [139]. A precise detection of these information items represents a key point for all the electronic engine management [140].

The correct programming of the timer modules is a key aspect in the automotive domain, due to the complexity of its programming code, and of the applications they have to manage. Consequently, efficient and precise validation methods and platforms are required. The current main methods to validate automotive engine applications are based on models [136][141][142], or on ad-hoc special purpose test equipment available on the market [143]. As timer module become more advanced, it raises the cost associated with validating these new modules, since extremely

complex and expensive equipment must be adopted and traditional equipment are no longer able to keep up with constantly changing requirements of these systems.

In this section a new FPGA-based validation platform aimed to the validation of the applications running in the real-time timer modules used in the today vehicles is presented. The purpose of this platform is to provide the developers of automotive applications with a flexible and efficient architecture able to effectively validate the code running in the most popular timer modules. More in particular, the proposed platform has the capability of generating the engine reference signals (i.e., the crankshaft and camshaft reference signals) that are typically used by the automotive microcontrollers to generate the fuel injection signals, and acquiring the signals generated by the timer module under test, verifying the synchronization between these signals and the provided engine reference signals. The proposed platform is useful to validate the functioning of several timer modules running in different engine configurations (e.g., with different profile of the crankshaft and camshaft signals). The platform can be customized with the exact instrument modules to meet the exigency of new test, it has flexible functionalities for diverse test bench purposes. Finally, it allows effective testing of modular designs and if compared with traditional approaches used to test state-of-the-art timer modules, it has an accuracy improvement of more than 20% [143] providing a timing and angular precision of 10 ns and $10^{-5}$ degrees respectively.

As case study, two important timer modules used today in the automotive domain have been used: the *Enhanced Time Processor Unit (eTPU)* developed by Freescale [144][145], and the *Generic Timer Module (GTM)* developed by Bosch [146]. These two modules have been selected since they are used, among the others, for the generation of the fuel injection signals in several engines; moreover, eTPU and GTM represent a good set of benchmarks, since the way in which they manage the automotive applications are different: in fact, in the eTPU several software routines share the same processing unit, while in the GTM several tasks can be directly managed by hardware parallel processing units.

The acquired results demonstrate the validity of the proposed approach, since using the proposed platform, it is possible to verify the synchronization between the inputs and the generated fuel injection signals with a very high degree of precision. Moreover, the proposed platform is able to verify the signals synchronization both in static engine conditions (i.e., constant engine speed), and in dynamic conditions (i.e., variable engine speed).

## 5.2 Related Works

With the development of electronic technology and the application of control theory in the automotive control [147], many research works have been developed with

the purpose of improve the control of the fuel injection. The motivation of these research works is that, nowadays, the fuel injection system is the most important part of diesel engines, and its working state directly influences the performance, the consumption and the air pollution, as documented in [132][136][140].

In [132] the authors present a new fuel injection intelligent control system, designed to improve the testing accuracy. The proposed system can automatically test the state of the injection pump, and it obtains all the parameters of the fuel injection system without human intervention by the use of PC and AT89C52 single chip microcomputer. Such system is designed and realized on the SYT240 fuel injection system test platform, which can automatically fetch and display the main parameters. Although the approach presented seems to be promising, it is strongly based on the usage of a dedicated test platform.

In [133] the authors face the problem of improving the accuracy of the engine control electronic, and they affirm that one potential way to do this is by use real-time in-cylinder pressure measurements. Consequently, the authors propose an approach that derives the pressure information from the measurement of the ordinary spark plug discharge current. The motivation of this work is that, by monitoring the pressure of each cylinder, the electronic engine control can be optimized in terms of fast response and accuracy, thus enabling online diagnosis and overall efficiency improvement.

Another research work addressing the usage of cylinder pressure-based combustion control is presented in [135], where the authors explain that in case of multiple fuel injections, the timing and the width of the fuel injection pulses need to be optimized. More in particular, this paper presents several methods in which the cylinder pressure signal is used for multiple-pulse fuel injection management for a diesel engine capable of running in low-temperature combustion modes. In [134] the authors explain that it is important to avoid discrepancies between the fuel amounts injected into the individual cylinders, in order to avoid excessive torsional vibrations of the crankshaft. Consequently, the authors present a general adaptive cylinder balancing method for internal combustion engines; the proposed algorithm uses online engine speed measurements. The motivation of this work is that, due to varying dynamics and ageing of components of the fuel-injection systems, there may be a significant dispersion of the injected fuel amounts.

In order to implement all the fuel injection optimization methods proposed above, in the real engine behavior, the engine angular position has to be identified as precisely as possible. In [144] the authors present an example of engine position identification by using the eTPU module embedded in the MPC5554 microcontroller.

Another work addressing the problem of a precise angular engine position detection is reported in [140]; more in particular, the authors explain that, due to mounting and packaging tolerances, the magnetic field at the sensors position varies, resulting in angular measurement. Mounting and packaging tolerances cannot be

avoided; consequently, the authors propose a compensation method based on a new filter structure.

Summarizing, several research works have been developed in the last decade in order to optimize the fuel injection systems of the today vehicles. In order to achieve this goal, the usage of specific timer modules, i.e., the eTPU and the GTM, is today required; the main tasks typically managed by these modules are acquire in a very precise way the engine angular position, and generate, among the others, the signals aimed to control the cylinders fuel injection. To do this, these modules have to be configured, using their specific programming code; this task is a difficult and an important part of the fuel injection control systems development, since a small error can cause relevant problems in the engine behavior, thus compromising the efficiency of the entire system.

In the solution proposed in this PhD thesis, an FPGA-based validation platform able to emulate the engine behavior (i.e., generate the crankshaft and the camshaft signals) is presented; the main goal of this solution is to acquire the fuel injection signals generated by the timer module under test; more in particular using this platform, it is possible to validate the timer module in a very precise way since the precision of the data acquisition is about $10^{-5}$ engine angular degrees, which is an improvement of more than 25% with respect to traditional methods [142][143]. The effective synchronization validation between the input and the output signals is performed by the developed software framework which compares the data achieved during the experimental analysis with the expected ideal values.

The main contribution of this work is to provide at the developers of the automotive applications a precise and flexible validation platform, useful to check the correctness of the developed software routines, thus ensuring an efficient system development. Moreover, using this platform it is possible to check the functioning of the real microcontroller, avoiding the unexpected misbehaviors due to the model-based validation of the developed applications.

## 5.3 Timer modules in Automotive applications

The today automotive microcontrollers contain specific timer modules to manage the engine signals. In Figure 5.1, the most important signals related to the cylinders fuel injection are shown. All the main tasks performed by the automotive microcontrollers are based on a precise detection of the engine angular position (i.e., the precise position of the cylinders with respect to the crankshaft). This is done using the two reference signals coming from the engine, i.e., the crankshaft and the camshaft. The crankshaft, typically, is a square wave signal, where each falling edge transition represents a partial rotation of the crankshaft. For example, if the crankshaft phonic wheel [148] is composed of 60 teeth, each falling edge transition of

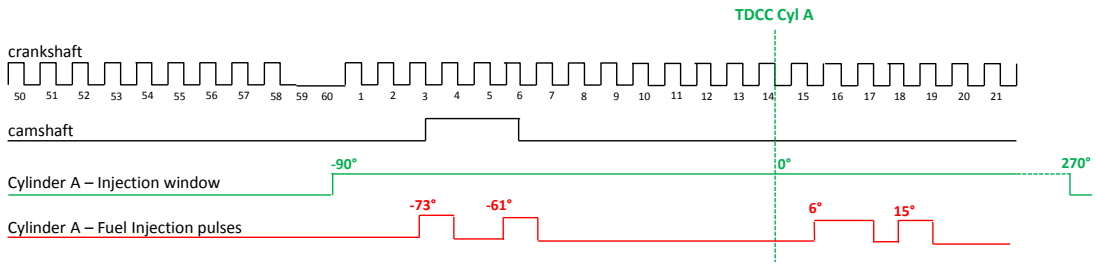



Figure 5.1. Example of the main signals received and managed by the automotive timer modules, in order to efficiently supervise the engine behavior.

the crankshaft signal indicates a rotation of 6°. Moreover, in a determinate position of the crankshaft signal, a *gap* (i.e., a missing tooth) is present: this gap is used as reference point to understand the correct engine angular position [153]. On the other side, the camshaft is a signal composed of few pulses synchronized with the crankshaft. Since the engines addressed in the context of this research work are 4-stroke engines, the complete 4-stroke sequence (i.e., intake, compression, power, and exhaust) takes two full rotations of the crankshaft. By only looking at the crankshaft signal, there is no way to understand if the crank is on its intake-compression rotation or on its power-exhaust rotation. To get this information, the camshaft signal is required; moreover, due to the 4-stroke configuration, the camshaft rotates at half the crankshaft speed (a rotation of 360° of the camshaft implies a rotation of 720° of the crankshaft); consequently, a signal generated once per rotation of the camshaft is sufficient to supply the required information. According to the features of these signals, the *Top Dead Cylinder Center (TDCC)* for each considered cylinder is identified [149]. The fuel injection pulses are electronic pulses that act on the fuel injector of each cylinder. These pulses have to be generated in a very precise angular position, where the reference point is the TDCC. The range in which these pulses can be generated is called *Injection Window (IW)*; typically, the width of the IW is 360°. In the context of this research work, the maximum number of injection pulses is equal to 16. The angular position of the beginning of an injection pulse is called *Start Of Injection (SOI), or Start Angle*; moreover, the injection pulses can be programmed using a temporal displacement between them (Dwell). Finally, in the typical automotive applications, timer modules generate, also, a sequence of high frequency *pulse width modulation (PWM) pulses*, in order to trigger other engine sensors. The generation of the fuel injection pulses is not the main goal of this research work: the method proposed in the next section, in fact, is focused on the verification of the precision and of the synchronization of the generation of the fuel injection pulses.

## 5.4 Proposed validation platform

The main purposes of the proposed platform are generate the engine reference signals, and acquire the fuel injection pulses, generated by the microcontroller under test, correlated with the provided engine reference signals themselves. In Figure 5.2 the overview of the proposed validation behavior is shown. It is composed of the FPGA-based validation platform and of the external controlling computer.

The validation platform is composed of the 32-bit Xilinx MicroBlaze processor core [150] and of a set of special purpose DSP peripherals designed to support the validation of the signals generated by the timer module under test. Both the MicroBlaze processor and the DSPs are implemented in a FPGA device. The communication between the processor and the DSP are managed using a Processor Local Bus (PLB) interface. An external controlling computer is directly connected to the processor, in order to provide the input parameters required to the generation of the engine signals. Moreover, a RS232 interface is used to save the acquired data into a database contained in the external computer itself.

Two are the most important DSP peripherals developed in this context: the crank/cam generator which consists on the module used to generate the crankshaft and the camshaft signals according to the user parameters, and the measurement module which samples and stores the signals provided by the timing module under test.

### 5.4.1 Crankshaft and Camshaft DSP peripheral

In Figure 5.3 the composition of the crankshaft and camshaft DSP peripheral is shown. It is composed of three main sub-modules: *clock_div*, *selector_crank*, and *selector_cam*. The goal of this peripheral is to generate the crankshaft and camshaft signals in two different ways, according to the user requirements: in the former, the signals have to emulate the signals of an engine running with constant *rounds per minutes (rpm)*, while in the second case these signals have to emulate the dynamic behavior of an engine, i.e., non-constant rpm.

The clock_div sub-module receives in input from the MicroBlaze processor (through the use of slave registers) the 32-bit values *Delta_period* and *Num_cycles*; the former is a timeout value (expressed in number of clock cycles) in which the period of the crankshaft teeth has to remain constant. When this time is elapsed, the *request signal* is raised in order to communicate at the MicroBlaze processor (through the use of an interrupt) that the new speed parameters can be sent; this mechanism allows to generate dynamic or static crankshaft and camshaft signals. The second value received by the peripheral (i.e., Num_cycles) represents, instead, the actual speed of the engine: in particular, it is the duration, expressed in number of clock cycles, of half period of the crankshaft signal. The internal circuitry (i.e., the selector crank

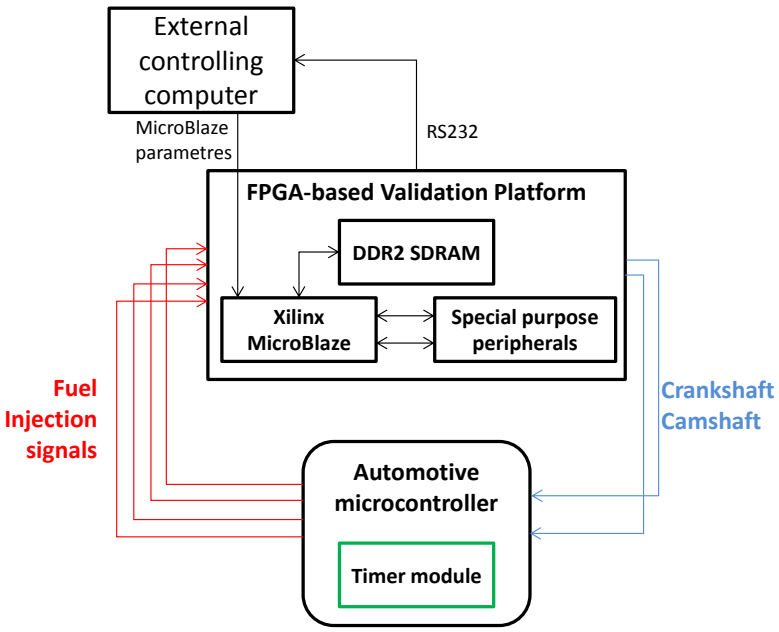



Figure 5.2. The overview of the proposed validation platform.

sub-module) causes a toggle of the output crankshaft signal every Num_cycles clock cycles. A similar approach is used to generate the output camshaft signal: the signal *Cam Enable 2*, generated by the selector crank sub-module, reports at the selector cam sub-module when toggle the output camshaft signal. This has been done to ensure a very precise synchronization between the crankshaft and the camshaft signals generated by this peripheral

### 5.4.2 Measure DSP peripheral

The measure peripheral receives in input the injection pulses and the PWM pulses signals, both generated by the timer module under test; using as absolute reference the crankshaft and the camshaft signals (generated by the specific peripheral explained in the previous section) it performs the required measurements of the input signals. This peripheral is able to measure the signals of one cylinder at a time.

In Figure 5.4, the main sub-modules composing the measure DSP peripheral are shown. The *sampling window sub-module* triggers the other two sub-modules: it receives in input (*TDC_ang signal*) from the MicroBlaze processor (through the use of a slave register) the number of the crankshaft tooth falling edge corresponding to the angular value of the TDCC referred to the cylinder to be monitored. It also receives in input the crankshaft signal generated by the other peripheral (*crank_cont*

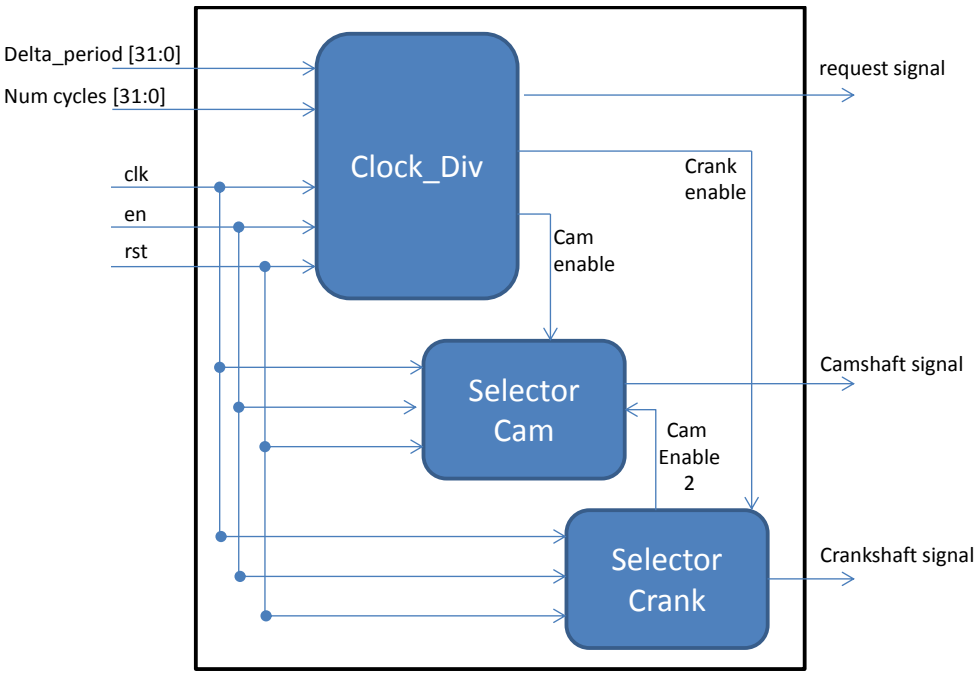



Figure 5.3. The crankshaft and camshaft generator peripheral.

*signal*). By counting the falling edges of the crankshaft signal, the sampling window sub-module is able to produce in output the sampling signal, that remains active (i.e., logic value equal to 1) for 360° according to the programmed cylinder. This signal is connected to the *pulse measure* and to the *PWM measure sub-modules*, in which it is used as "clear" signal for the internal counters.

The pulses measure sub-module counts the number of incoming pulses (*injection pulses signal*) and exploiting a couple of latches, it stores in a set of internal signals the time stamps at which the rising and falling edges occur. To do this, an internal counter is used, where the frequency of the counter itself is the same of the frequency of the clk signal; moreover, this counter is reset at the beginning of each injection window. As soon as the sampling signal becomes 0, all the stored results are transferred to the respective 512-bit outputs. The length of the output signals is due to the fact that inside a programming window, there should be at most 16 injection pulses; consequently, 16 parameters have to stored, each one represented with 32 bit. The output data of this sub-module are: (1) *angle measure*, which contains the measured start angle time stamps of the injection pulses; (2) *dwell* which contains the measured time between the current pulse and the previous one; (3) *num pulses*, which indicates the total number of detected pulses inside the injection window; (4) *width*, which contains the measured time duration of each injection pulse; and

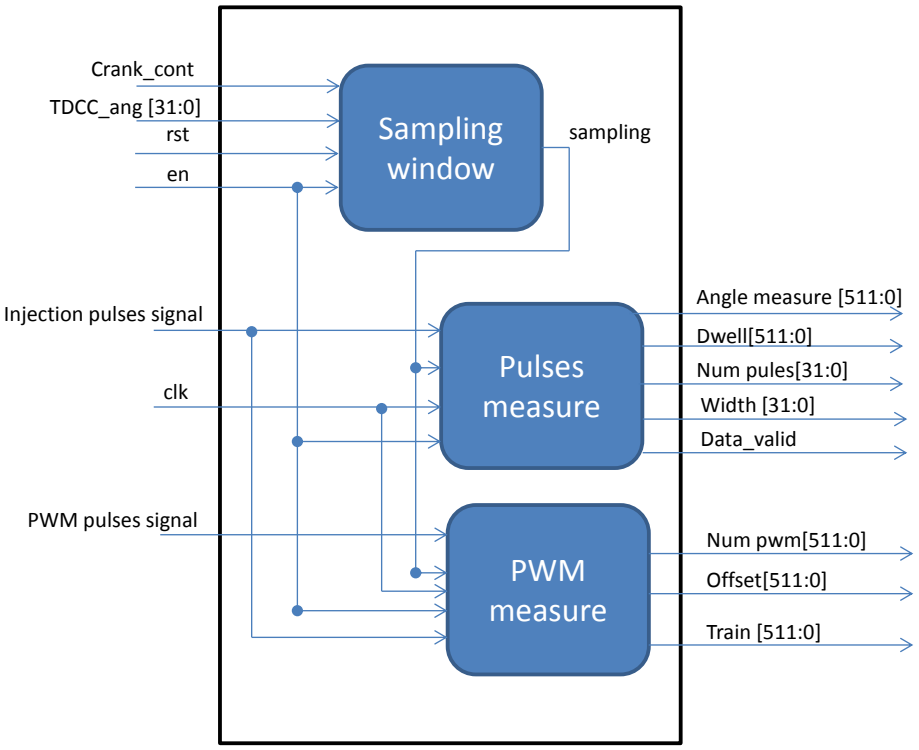



Figure 5.4. The measure peripheral.

(5) *data valid*, that signals when all the measured values have been copied in the output signals; this value is used to signal at the MicroBlaze processor (through an interrupt) that the current data could be transferred in the SDRAM.

A similar approach has been used for the PWM measure module, in which the features of the PWM signal generated by the timer module under test are measured. This module is activated in a particular point of the IW, depending on the automotive application under analysis. The output data of this module are: (1) *offset*, which contains the measured offset between a determinate point within the IW (e.g., the TDC) and the first PWM rising edge; (2) *train*, which contains the measured PWM train duration (in terms of number of clock cycles), from the first rising edge to the last falling edge; and (3) *num_pwm*, which contains the number of counted PWM pulses.

### 5.4.3 Xilinx MicroBlaze processor tasks

Considering the special purpose peripherals described in the previous sections, the MicroBlaze processor has three main tasks. The first is to provide the crankshaft

and camshaft generator peripheral with the data about the features of the camshaft and crankshaft signals that have to be generated, i.e., the crankshaft period and the dynamic of the crankshaft signal itself. The second performed task is, whenever the data valid signal is received from the measure peripheral, transfer the acquired data (contained in the output signals of the measure peripheral) into the SDRAM; this allows to acquire the same data (about the fuel injection and the PWM signals) in different time instants, in order to characterize in a very precise way the measured signals produced by the microcontroller under test. Finally, when the required number of measurements have been acquired, the MicroBlaze processor takes care of sending (through a RS232 interface) the data contained in the SDRAM at the external controlling computer.

### 5.4.4 Output data format

All the measurements acquired by the peripherals described in the above sections are based on temporal intervals and are contained in a file called data_raw.txt.

Since the main purpose of the proposed validation platform is to verify the synchronization of the fuel injection pulses with respect to the engine reference signals, the acquired data (about the pulses start angle) have to be translated from the FPGA time domain (i.e., the number of clock cycles measured by the peripheral) to the engine angle domain. To do this, the following Formula 5.1 has been used, where 6 are the degrees of each crank falling edge transition, and RPM are the current rounds-per-minute of the engine. Clearly, this formula can be used in the case of constant engine rpm; in case of dynamic engine behavior, the formula has to take in consideration the different RPM values during the measurements interval.

$$StartAngle[deg] = \frac{(6 * StartAngle[ClockCycleNum] * RPM}{ClockCyclePeriod[s]} \tag{5.1}$$

## 5.5 Experimental results

In Figure 5.5 the experimental flow is shown. A Matlab parser translates the data acquired by the proposed validation platform. Then, these data are compared with the file containing the expected values (called ideal_values.txt); this file is generated by an ideal data generator written in C language. As a result, several report files are obtained; these files contain the details about the measures, including the average error, the maximum error and the standard deviation of each feature of each injection pulse generated by the timer module under test. As case study, two timer modules have been used: the eTPU and the GTM; in the following section the main features of these modules and the obtained measurements results are shown.

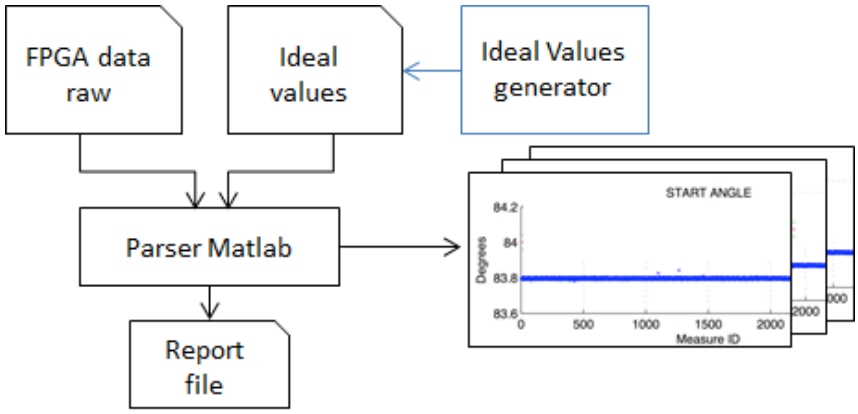



Figure 5.5. The experimental flow.

### 5.5.1 Use Case: the Enhanced Time Processor Unit (eTPU)

The Enhanced Time Processor Unit (eTPU) [144][145] by Freescale, is an effective timing co-processor available in the automotive domain; it is used to efficiently manage I/O processing in advanced microcontroller units. From a high level point of view, the eTPU has the characteristics of both a peripheral and a processor, tightly integrated between each other [151]; essentially, it is an independent microcontroller designed for timing control, I/O handling, serial communications, and engine control applications [145]. More in particular, the eTPU is mainly used to decode the engine angular position, and, consequently, to control actuators such as the fuel injectors and the spark plugs, thanks to the high flexibility of the dedicated programmable hardware.

In the context of this research work, the eTPU module embedded in the microcontroller SPC5644AM has been used; moreover, the automotive functions set available in [152] has been used.

### 5.5.2 Use Case: Generic Timer Module (GTM)

The Generic Timer Module (GTM) [146] is a recent hardware module provided by Bosch. It is composed of many sub-modules with different functionalities. These sub-modules can be interconnected together in a configurable manner in order to obtain a flexible timer module for different application domains. The scalability and configurability is reached by means of the architectural structure of the module itself: a set of dedicated sub-modules is placed around a central routing unit, which is able to interconnect the sub-modules according to the programmed configuration specified in the running software [146]. The GTM is designed to run with a minimal

CPU interaction and to unload the CPU itself from handling frequent interrupts service requests.

In the context of this research work, the GTM module embedded in the microcontroller SPC574K72 has been used; moreover, a set of automotive functions useful to generate the fuel injection pulses using only the GTM module has been implemented for this scope.

### 5.5.3 The used Xilinx FPGA board

In order to implement the proposed validation platform, the Xilinx Virtex-5 XC5VL X50T FPGA has been used. This FPGA is embedded in a Digilent Genesys board. The working frequency of the MicroBlaze processor implemented in the FPGA itself in 100MHz; also the clock frequency provided at the special purpose peripherals is 100MHz. This allows us to obtain time measurements with a precision of 10ns, and angular measurements with a precision of $10^{-5}$ degrees.

## 5.6 Main obtained results

The main purpose of this section is to give the reader an idea of the effective features of the proposed validation platform, highlighting its capability of making measures with a very high degree of precision.

In Figure 5.6 two graphs are shown: the first (a) reports the measurements of the width of an injection pulse, while the second (b) shows the measurements of the PWM offset; in this case the module under test is the GTM.

In Figure 5.7 a graph reporting the measurements of the start angle of an injection pulse generated by the eTPU module is shown. By looking at this graph, it is possible to understand if the injection pulse is generated in the correct position, i.e., in a determinate angle position, according to the crankshaft and the camshaft signals. In this case, the precision of the measurements is $10^{-5}$ degrees.

As it is possible to notice by the graphs reported in this section, using the proposed validation platform it is possible to understand if the injection pulses are correctly generated. This allows to understand if the software applications running in the considered timer modules are correct (ensuring a real-time behavior) or contain software bugs. Using this platform, thus, the developers of automotive applications can verify if the applications that they are writing efficiently manage the fuel injectors.

As a conclusion of this research activity, in this research work a new platform for the validation of timer modules used in automotive applications has been proposed. The high flexibility, combined with the capability of extreme precise measurements,

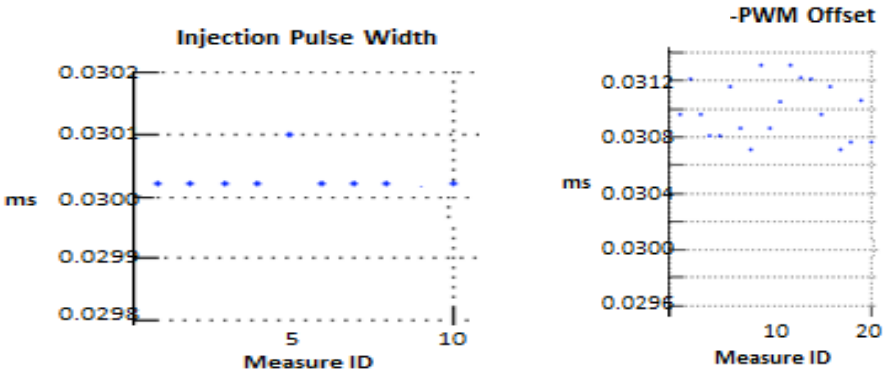



Figure 5.6. Measures of injection pulse width (a), and of the PWM offset (b); both the signals are generated by the GTM.

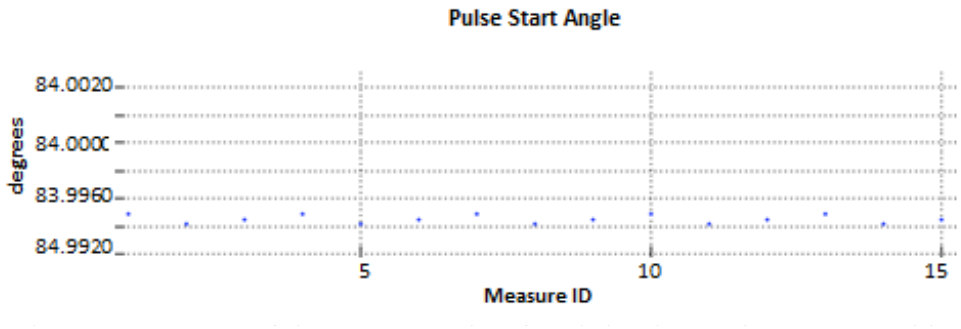



Figure 5.7. Measures of the Start Angle of an injection pulse generated by the eTPU module.

make the platform very suitable to be used by the developers of automotive applications during the software development. As case study, two important timer modules employed in the today vehicles have been used.

# Chapter 6

# Conclusions

Nowadays, integrated electronic systems are more and more used in a wide number of applications and environments, ranging from low cost to safety critical products. This wide distribution is mainly due to the miniaturization surrounded by an increasing computing power of semiconductor devices. Among the different challenges associated to this phenomena, the reliability of electronic systems is becoming more and more relevant.

In this PhD thesis, several reliability techniques have been proposed. The common thread of these techniques is that they have been developed addressing multicore processor units. In particular, new methods have been generated addressing VLIW processors, GPGPUs, and the Generic Timer Module (GTM) by Bosch.

Considering VLIWs processors, new test and diagnostic methods have been studied and implemented in order to detect and localize permanent faults; they are mainly based on the Software-Based Self-Test (SBST) technique. The obtained results show that with the proposed methods it is possible to decrease the time required to perform the test of a generic VLIW processor, and to efficiently localize the faulty module.

In the GPGPUs context, instead, the effects introduced by soft errors have been analyzed; this works have been done through the execution of three different neutron radiation tests. The gathered data provide several interesting suggestion about the configuration of the applications running in the GPGPU embedded in safety critical environments.

As industrial case, the test and the validation of a timing multicore co-processor module used in the today Electronic Control Units (ECUs) have been designed and implemented. More in particular, an FPGA-based validation platform has been developed. The main feature of this low cost device is the ability to efficiently verify the functioning of the timing module under test, thus ensuring a correct implementation of the software routines running on it. This work has been done in collaboration with General Motors Powertrain Europe (Turin).

Concluding, novel algorithms for reliability characterization of multicore processing units have been developed; moreover, several results, never appeared in the literature before, have been proposed as a proof of the goodness of the proposed methods.

# Appendix A

# List of published papers

In this PhD thesis, several reliability challenges have been addressed. The most important results are described in several papers published in conference proceedings, book chapters, and international journals. In this section the complete list of that papers is presented.

- **Journal Paper**

  1. Sabena D., Sonza Reorda M., Sterpone L., *On the Automatic Generation of Optimized Software-Based Self-Test Programs for VLIW Processors*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, ISSN 1063-8210, vol. 22, n. 4, pp. 813-823, April 2013.
  2. Sabena D., Sonza Reorda M., Sterpone L., Rech P., Carro L., *Evaluating the radiation sensitivity of GPGPU caches: New algorithms and experimental results*, Microelectronics Reliability, ISSN 0026-2714, vol. 54, n. 11, pp. 2621 - 2628, November 2014.
  3. Sabena D., Sterpone L., Carro L., Rech P., *Reliability Evaluation of Embedded GPGPUs for Safety Critical Applications*, IEEE Transactions on Nuclear Science, ISSN 0018-9499, vol. 61, n. 6, pp. 3123 - 3129, December 2014.

- **Book Chapter**

  1. Sabena D., Sterpone L., Sonza Reorda M., *On the Automatic Generation of Software-Based Self-Test Programs for Functional Test and Diagnosis of VLIW Processors*, VLSI-SoC: From Algorithms to Circuits and System-on-Chip Design, Springer Berlin Heidelberg, pp. 162-180, 2013.

- **Conference Proceedings Paper**

1. De Carvalho M., Sabena D., Sonza Reorda M., Sterpone L., Rech P., Carro L., *Fault Injection in GPGPU Cores to Validate and Debug Robust Parallel Applications*, IEEE 20th International On-Line Testing Symposium (IOLTS), Platja d'Aro, pp. 210-211, July 2014,
2. Sabena D., Sterpone L., Schölzel M., Koal T., Vierhaus H.T., Wong S., Glein R., Rittner F., Stender C., Porrmann M., Hagemeyer J., *Reconfigurable High Performance Architectures: How much are they ready for safety-critical applications*, 19th IEEE European Test Symposium (ETS), Paderborn, pp. 175-182, May 2014.
3. Sabena D., Sonza Reorda M., Sterpone L., *Soft Error Effects Analysis and Mitigation in VLIW Safety-Critical Applications*, IFIP/IEEE 22nd International Conference on Very Large Scale Integration (VLSI-SoC), pp. 135-140, October 2014.
4. Sterpone L., Sabena D., Ullah A., Porrmann M., Hagemeyer J., Ilstad J., *Dynamic Neutron Testing of Dynamically Reconfigurable Processing Modules Architecture*, IEEE NASA/ESA Conference on Adaptive Hardware and System (AHS), pp. 184-188, June 2013.
5. Sabena D., Sonza Reorda M., Sterpone L., *On the development of diagnostic test programs for VLIW processors*, 21st IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pp. 87-92, October 2013.
6. Sabena D., Sonza Reorda M., Sterpone L., Rech P., Carro L., *On the evaluation of soft-errors detection techniques for GPGPUs*, 8th IEEE International Design and Test Symposium (IDT), pp.16-18, December 2013.
7. L. Sterpone, D. Sabena, M. Sonza Reorda, *A New Fault Injection Approach for Testing Network-on-Chips*, International Conference on Parallel, Distributed and network-based Processing (PDP), pp. 530-535, February 2012.
8. L. Sterpone, D. Sabena, M. Sonza Reorda, *A New SBST Algorithm for Testing the Register File of VLIW Processors*, IEEE Design, Automation and Test in Europe (DATE), pp. 412 -417, March 2012.
9. Sabena D., Sonza Reorda M., Sterpone L., *On the development of Software-Based Self-Test methods for VLIW processors*, IEEE International Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT) Symposium, pp. 25-30, October 2012.
10. Sabena D., Sonza Reorda M., Sterpone L., *On the optimized generation of Software-Based Self-Test programs for VLIW processors*, IEEE/IFIP

20th International Conference on VLSI and System-on-Chip (VLSI-SoC), pp. 129-134, October 2012.

11. L. Sterpone, D. Sabena, S. Campagna, M. Sonza Reorda, *Fault Injection Analysis of Transient Faults in Clustered VLIW Processors*, 14th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), pp. 207-212, April 2011.

# Appendix B

# Abbreviations

The following list describes the significance of various abbreviations and acronyms used throughout this PhD thesis.

**ADAS** Advance Driver Assistance System

**ATE** Automatic Test Equipment

**BIST** Built-In Self-Test

**CD** Computational Domain

**COB** Cost of Build

**CPU** Central Processing Unit

**DfT** Design for Test

**DSP** Digital Signal Processing

**DUT** Device Under Test

**ECU** Electronic Control Unit

**FPGA** Field Programmable Gate Array

**FU** Functional Unit

**GPGPU** General Purpose Graphic Processing Unit

**GPU** Graphic Processing Unit

**GTM** Generic Timer Module

**HPC** High Performance Computing

**ILP** Instruction Level Parallelism

**JTAG** Joint Test Action Group

**ISA** Instruction Set Architecture

**POST** Power-On Self-Test

**SIMD** Single Instruction Multiple Data

**SoC** System on Chip

**SBST** Software Based Self Test

**SRAM** Static Random Access Memories

**VLIW** Very Long Instruction Word

**VLSI** Very Large Scale Integration

# Bibliography


[1] C. A. Mack, "Keynote: Moore's Law 3.0," in *Proceedings of the IEEE Workshop on Microelectronics and Electron Devices (WMED)*, pp. 8-9, April 2013.

[2] F. Chaochao, L. Zhonghai, A. Jantsch, Z. Minxuan, X. Zuocheng, "Addressing Transient and Permanent Faults in NoC With Efficient Fault-Tolerant Deflection Router," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 5, pp. 1053 -1066, May 2013.

[3] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," in *IEEE Micro* , vol. 23, no. 4, pp. 14 - 19, September 2013.

[4] R. Jeyapaul, F. Hong, A. Rhisheekesan, A. Shrivastava, L. Kyoungwoo, "UnSync-CMP: Multicore CMP Architecture for Energy-Efficient Soft-Error Reliability," in *IEEE Transactions on Parallel and Distributed Systems* , vol. 25, no. 1, pp. 254 - 263, November 2013.

[5] D. Kim, V.W. Lee, C. Yen-Kuang, "Image Processing on Multicore x86 Architectures," in *IEEE Signal Processing Magazine* , vol. 27, no. 2, pp. 97 - 107, March 2010.

[6] J.A. Fisher, P. Faraboschi, and C. Young, "Embedded computing: a VLIW approach to architecture, compilers and tools," Morgan Kaufmann, 2004.

[7] D. Sabena, M. Sonza Reorda, L. Sterpone, "On the Automatic Generation of Optimized Software-Based Self-Test Programs for VLIW Processors," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 4, pp. 813 - 823, March 2014.

[8] D. Sabena, M. Sonza Reorda, L. Sterpone, P. Rech, L. Carro, "Evaluating the radiation sensitivity of GPGPU caches: new algorithms and experimental results," in *Elsevier Microelectronics Reliability*, to be published.

[9] NVIDIA. (2013, Feb.). NVIDIA's Next Generation CUDA Computer Architecture: Kepler GK110, Santa Clara, CA, USA [Online]. Available: http://www.nvidia.com/content/PDF/kepler

[10] S. Mu, Y. Deng, Y. Chen, H. Li, J. Pan, W. Zhang, Z. Wang, "Orchestrating Cache Management and Memory Scheduling for GPGPU Applications," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 8, pp. 1803 - 1814, July 2014.

[11] Generic Timer Module (GTM) Product Information. Available [Online]: http://www.boschsemiconductors.de/media/en/pdf_1/ipmodules_1/timer/bosch_product_info_gtm_ip_v1_1.pdf.

[12] B. Bostein, T. Estlin, B. Clement, P. Springer, "Using a Multicore Processor for Rover Autonomous Science," in *Proceedings of IEEE Aerospace Conference*, pp. 1-9, 2011.

[13] Tilera Corporation, "Multicore Development Environment User Guide," Doc#UG201 Release 1.2, February 2008.

[14] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. Sonza Reorda, "Microprocessor software-basedself-testing," in *IEEE Design & Test of Computers*, vol. 2, no. 3, pp. 4 - 19, May-June 2010.

[15] S. Di Carlo, G. Gambardella, M. Indaco, I. Martella, P. Prinetto, D. Rolfo, P. Trotta, "A software-based self test of CUDA Fermi GPUs," in *Proceedings of 18th IEEE European Test Symposium (ETS)* , pp. 1 - 6, May 2013.

[16] H. J. Wunderlich, C. Braun, S. Halder, "Efficacy and Efficiency of Algorithm-Based Fault-Tolerance on GPUs," in *Proceedings of IEEE 19th International On-Line Testing Symposium (IOLTS)*, pp. 240 - 243, July 2013.

[17] D. Sabena, M. Sonza Reorda, L. Sterpone, P. Rech, L. Carro, "On the evaluation of soft-errors detection techniques for GPGPUs", in *Proceedings of 8th IEEE International Design and Test Symposium (IDT)*, pp. 1 - 6, December 2013.

[18] E. Armengaud, A. Steininger, M. Horauer, "Towards a Systematic Test for Embedded Automotive Communication Systems," in *IEEE Transactions on Industrial Informatics*, vol. 4, n. 3, pp. 146 - 155, August 2008.

[19] Y. Kanehagi, D. Umeda, A. Hayashi, K. Kimura, H. Kasahara, "Parallelization of automotive engine control software on embedded multi-core processor using OSCAR compiler," in *IEEE Cool Chips XVI*, p. 1 - 3, April 2013.

[20] A. Avizienis, J. C. Laprie, B. Randell, "Fundamental Concepts of Dependability," in *Proceedings of the 3rd IEEE Information Survivability Workshop (ISW-2000)*, pp. 7 - 12, October 2000.

[21] A. Avizienis, J. C. Laprie, B. Randell, C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," in *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11 - 33, January 2004.

[22] L. Ciganda, "New Techniques for Reliability Characterization of Electronic Circuits," PhD Thesis, February 2013. Available [Online]: www.phd-dauin.polito.it/pdfs/Lyl%20CIGANDA_thesis.pdf

[23] IEEE, "610-1991 - IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries," IEEE Computer Society Standard, ISBN 1-55937-079-3, January 1991.

[24] R. W: Smith, L: Dietrich Duane, "The Bathtub Curve: An Alternative Explanation," in *Proceedings of Annual Reliability and Maintainability Symposium*,

pp. 241 - 247, January 1994.

[25] Availability definition and comments. Available [Online]: http://en.wikipedia.org/wiki/Availability

[26] P. Maxwell, I. Hantanto, L. Bentz, "Comparing functional and structural tests," in *Proceedings of International test Conference*, pp. 400 - 407, October 2000.

[27] H. A. Toku, "Developing New Automatic Test Equipments (ATE) using Systematic Design Approaches," in *Proceedings of IEEE AUTOTESTCON*, pp. 1 - 7, September 2013.

[28] E. B. Eichelberger, T. W. Williams, "A logic Design Structure for LSI testability," in *Journal of Design Automation & Fault-tolerant Computing*, vol. 2, pp. 165 - 178, May 1978.

[29] J. P. Hayes and A. D. Friedman, "Test point placement to simplify fault detection," in *IEEE Tansactions on Computers*, vol. 23, no. 7, pp. 727-735, July 1974.

[30] R. Ubar, V. Indus, O. Kalmend, T. Evartson, E. Orasson, "Functional Built-In Self-Test for processor cores in SoC," in *Proceedings of NORCHIP*, pp. 1 - 4, November 2012.

[31] E. Sanchez, M. Sonza Reorda, G. Squillero, "On the Transformation of Manufacturing Test Sets into On-Line Test Sets for Microprocessors," in *Proceedings of 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pp. 494 - 502, October 2005.

[32] S.Tammali, "Industrial practices of test cost reduction: Perspective, current design practices," in *Proceedings of 28th VLSI Test Symposium (VTS)*, pp. 124 - 125, April 2010.

[33] K. Chakrabarty, "Low-cost modular testing and test resource partitioning for SoCs," in *IEE Proceedings Computer & Digital Techniques*, vol. 152, no. 3, pp. 427 - 441, May 2005.

[34] M. Beck, O. Barondeau, F. Poehl, X. Lin, and R. Press, "Measures to improve delay fault testing on low-cost testers"A case study," in *Proceedings of the VLSI Test Symposium*, pp. 223 - 228, May 2005.

[35] Y. Zorian, "Guest editor's introduction: What is infrastructure IP," in *IEEE Design & Test of Computers*, vol. 19, no. 3, pp. 3 - 5, June 2002.

[36] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. Sonza Reorda, "Microprocessor software-based self-testing," in *IEEE Design & Test of Computers* , vol. 2, no. 3, pp. 4 - 19, May 2010.

[37] ISO, "26262 - Road vehicles - Functional safety standard," International Organization for Standarization Standard, 2011.

[38] G. Hetherington, et al., "Logic BIST for large industrial designs: real issues and case studies," in *Proceedings of the International Test Conference*, pp. 358 - 367, September 1999.

[39] G. Theodorou, N. Kranitis, A. Paschalis, D. Gizopoulos, "Software-Based Self Test Methodology for On-Line Testing of L1 Caches in Multithreaded Multicore Architectures," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 4, pp. 786 - 790, April 2013.

[40] Z. Al-Ars, S. Hamdioui, G. Gaydadjiev, and S. Vassiliadis, "Test set development for cache memory in modern microprocessors," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 6, pp. 725 - 732, June 2008.

[41] A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, "Advanced Test Methods for SRAMs," in *Springer Book*.

[42] A.J. Van De Goor, I.B.S. Tlili, "March tests for word-oriented memories," in *Proceedings of IEEE Design, Automation and Test in Europe (DATE)*, pp. 501 - 508, February 1998.

[43] N. Kranitis, A. Paschalis, D. Gizopoulos, and G. Xenoulis, "Software-Based Self-Testing of embedded processors," in *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 461-475, April 2005.

[44] P. Bernardi, M. Grosso, E. Sanchez, and O. Ballan, "Fault grading of Software-Based Self-Test procedures for dependable automotive applications," in *Proceedings of IEEE Design, Automation and Test in Europe (DATE)*, pp. 1 -2, March 2011.

[45] P. K. Parvathala, K. Maneparambil, W. C. Lindsay, "Functional random instruction testing (FRITS) method for complex devices such as microprocessors," *U.S. Patent 6948096*, September 2005.

[46] K. Batcher and C. Papachristou, "Instruction randomization self test for processor cores," in *Proceedings of the VLSI Test Symposium*, pp. 34 - 40, April 1999.

[47] N. Foutris, M. Psarakis, D. Gizopoulos, A. Apostolakis, X. Vera, A. Gonzalez, "MT-SBST: Self-test optimization in multithreaded multicore architectures," in *Proceedings of International Test Conference*, pp. 1 - 10, November 2010.

[48] C. Li, S. Dey, "Software-based diagnosis for processors," in *Proceedings of 39th Design Automation Conference*, pp. 259 - 262, 2002.

[49] J. Lagos-Benites, D. Appello, P. Bernardi, M. Grosso, "An Effective Approach for the Diagnosis of Transition-Delay Faults in SoCs, based on SBST and Scan Chains," in *Proceedings of 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, pp. 291 - 302, 2007.

[50] D. Sabena, M. Sonza Reorda, L. Sterpone, "On the development of diagnostic test programs for VLIW processors," in *Proceedings of IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 84 - 89, October 2013.

[51] K. P. Prasenjit, S. Harshpreet, S. Gurpreet, "Fault Tolerance Techniques and Comparative Implementation in Cloud Computing," in *International Journal*

*of Computer Applications*, vol. 64, no. 14, pp. 37-41, February 2013.

[52] J. von Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliable components," Automata Studies, pp. 43 - 98, 1956.

[53] A. Avizienis, J.P.J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," in *IEEE Computer*, vol. 17, no. 8, pp. 67 - 80, 1984.

[54] J. M. Smith, "A Survey of Software Fault Tolerance Techniques," Columbia University Academic Commons [Online]. Available: http://academiccommons.columbia.edu/catalog/ac%3A142350

[55] B. Randell, "System Structure for Software Fault Tolerance," in *IEEE Transactions on Software Engineering*, vol. 1, no. 2, pp. 220 - 232, June 1975.

[56] A. Avizienis, J.P.J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," in *Computer*, vol. 17, no. 8, pp. 67 - 80, August 1984.

[57] Wikipedia ATE definition [Online]. Available: http://en.wikipedia.org/wiki/Automatic_test_equipment

[58] Radio-Electronics.com ATE types summary [Online]. Available: http://www.radio-electronics.com/info/t_and_m/ate/automatic-test-equipment-basics.php

[59] IEEE Standards association - IEEE 1149.1-2013 - IEEE Standard for Test Access Port and Boundary-Scan Architecture [Online]. Available: http://standards.ieee.org/findstds/standard/1149.1-2013.html

[60] G. Garry Herzberg, C. Robert Reinolds, "Automated test apparatus for aircraft flight controls," Google patents, [Online]. Available: http://www.google.com/patents/US5023791

[61] The VEX toolchain [Online]. Available: http://www.hpl.hp.com/downloads/vex/

[62] J.A. Fisher, P. Faraboschi, and C. Young, "Embedded computing: a VLIW approach to architecture, compilers and tools," Morgan Kaufmann, 2004.

[63] M. Beardo, F. Bruschi, F. Ferrandi, and D.Sciuto, "An approach to functional testing of VLIW architectures," in *IEEE High-Level Design Validation and Test Workshop*, pp. 29 - 33, 2000.

[64] D. Sabena, M. Sonza Reorda, and L. Sterpone, "A new SBST algorithm for testing the register file of VLIW processors," in *IEEE International Conference on Design, Automation & Test in Europe (DATE)*,pp. 412 - 417, March 2012.

[65] S. Wong, F. Anjam, and F. Nadeem, "Dynamically reconfigurable register file for a softcore VLIW processor," in *IEEE International Conference on Design, Automation and Test in Europe (DATE)*, pp. 962 - 972, March 2010.

[66] S. Wong, T. Van As, and G. Brown, "$\rho$-VEX: a reconfigurable and extensible softcore VLIW processor," in *International Conference on ICECE Technology*, pp. 369 - 372, December 2010.

[67] J. Fischer, "Very long instruction work architectures and the ELI-512," in *IEEE Solid-State Circuits Magazine*, vol. 1, no. 2, pp. 23 - 33, 2009.

[68] J.A. Fischer, "Trace Scheduling: a technique for global microcode compaction," in *IEEE Transactions on Computers*, vol. C-30, no. 7, pp. 478 - 490, July 1981.

[69] T. Koal, H.T. Vierhaus, "A software-based self-test and hardware reconfiguration solution for VLIW processors," in *IEEE Symposium on Design and Diagnostic of Electronic Circuits and Systems*, pp. 40 - 43, April 2010.

[70] M. Ulbricht, M. Scholzel, T. Koal, and H.T. Vierhaus, "A new hierarchical built-in self-test with on-chip diagnosis for VLIW processors," in *IEEE Symposium on Design and Diagnostic of Electronic Circuits and Systems*, pp. 143 - 146, April 2011.

[71] A. Pillai, W. Zhang, and D. Kagaris, "Detecting VLIW hard errors cost-effectively through a software-based approach," in *Advanced Information Networking and Applications Workshops*, pp. 811 - 815, 2007.

[72] C. Bolchini, "A software methodology for detecting hardware faults in VLIW data paths," in *IEEE Transaction on Reliability*, vol. 52, no. 4, pp. 458 - 468, December 2003.

[73] D. Sabena, M. Sonza Reorda, and L. Sterpone, "On the development of software-based self-test methods for VLIW processors," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 25 - 30, October 2012.

[74] D. Sabena, M. Sonza Reorda, and L. Sterpone, "On the optimized generation of software-based self-test programs for VLIW processors," in *IFIP/IEEE International Conference on Very Large Integration (VLSI-SoC)*, pp. 129 - 134, October 2012.

[75] D. Gizopoulos, M. Psarakis, M. Hatzimihail, M. Maniatakos, A. Paschalis, A. Raghunathan, and S. Ravi, "Systematic software-based self-test for pipelined processors," in *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, vol. 16, No. 11, pp. 1441 - 1453, November 2008.

[76] A. Paschalis, D. Gizopoulos, N. Kranitis, M. Psarakis, and Y. Zorian, "Deterministic software-based self-testing of embedded processor cores," in *IEEE International Conference on Design, Automation and Test in Europe (DATE)*, pp. 92 - 96, 2001.

[77] N. Kranitis, D. Gizopoulos, A. Paschalis, and M. Psarakis, "Instruction-based self-testing of processor cores," in *IEEE VLSI Test Symposium*, pp. 223 - 228, 2002.

[78] E. Sanchez, M. Sonza Reorda, and G. Squillero, "On the transformation of manufacturing test sets into on-line test sets for microprocessor," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 494 - 502, October 2005.

[79] J.M.P. Cardoso, M. Hübner (Eds.), "Reconfigurable Computing: From FPGAs to Hardware/Software Codesign", Springer, 2011.

[80] P.S. Kabiri, Z. Navabi, "Effective RT-level software-based self-testing of embedded processor cores," in *IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 209 - 212, 2012.

[81] P. Bernardi, E. Sanchez, M. Schillaci, G. Squillero, M. Sonza Reorda, "An Effective technique for the Automatic Generation of Diagnosis-oriented Programs for Processor Cores," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 3, pp. 570 - 574, March 2008.

[82] M. Scholzel, T. Koal, H.T. Vierhaus, "An adaptive self-test routine for in-field diagnosis of permanent faults in simple RISC cores," in *IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 312 - 317, 2012.

[83] J. Abramson, P. C. Diniz, "Resiliency-aware Scheduling for Reconfigurable VLIW Processors," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pp. 1 - 7, 2012.

[84] S. Holst, H.-J. Wunderlich, "Adaptive debug and diagnosis without fault dictionaries," in *IEEE European Test Symposium*, pp. 7 - 12, 2007.

[85] P.G.Ryan et al., "Fault dictionary compression and equivalence class computation for sequential circuits," in *IEEE International Conference on Computer-Aided Design*, pp. 508 - 511, 1993.

[86] The Synopsys TetraMAX ATPG tool features [Online]. Available: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Test/Pages/TetraMAXAT

[87] S. Di Carlo et al., "Increasing the robustness of CUDA Fermi GPU-based systems," in *IEEE 19th International On-Line Testing Symposium (IOLTS)*, pp. 234 - 235, July 2013.

[88] S. Tselonis et al., "The Functional and Performance Tolerance of GPUs to Permanent Faults in Registers," in *IEEE 19th International On-Line Testing Symposium (IOLTS)*, pp. 236 - 239, July 2013.

[89] T. Tang, X. Yang, and Y. Lin, "Cache Miss Analysis for GPU Programs Based on Stack Distance Profile," in *31st International Conference on Distributed Computing Systems (ICDCS)*, pp. 623 - 634, June 2011.

[90] S. Wang, J. Hu, S.G. Ziavras, "Replicating Tag Entries for Reliability Enhancement in Cache Tag Arrays", in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 4, pp. 643 - 654, April 2012.

[91] S. Di Carlo, P. Prinetto, A. Savino, "Software-Based Self-Test of Set-Associative Cache Memories", in *IEEE Transactions on Computers*, vol. 60, n. 7, pp. 1030 - 1044, 2011.

[92] J. Tan, N. Goswami, T. Li, X. Fu, "Analyzing Soft-Error Vulnerability on GPGPU Microarchitecture", in *IEEE International Symposium on Workload Characterization (IISWC)*, pp. 226 - 235, November 2011.

[93] P. Rech, C. Aguiar, R. Ferreira, C. Frost, and L. Carro, "Neutron Radiation Test of Graphic Processing Units," in *IEEE 18th International On-Line Testing*

*Symposium (IOLTS)*, pp. 55 - 60, 2012.

[94] P. Rech, L. Pilla, F. Silvestri, P. Navaux, L. Carro, "Neutron sensitivity and software hardening strategies for matrix multiplication and FFT on graphics processing units", in *3rd ACM Workshop on Fault-tolerance for HPC at extreme scale*, pp. 13 - 20, 2013.

[95] P. Rech, C. Aguiar, C. Frost and L. Carro, "Experimental Evaluation of Thread Distribution Effects on Multiple Output Errors in GPUs", in *IEEE European Test Symposium (ETS)*, pp. 27 - 32, May 2013.

[96] P. Rech, C. Aguiar, C. Frost, and L. Carro, "An Efficient and Experimentally Tuned Software-Based Hardening Strategy for Matrix Multiplication on GPUs," in *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 2797 - 2804, 2013.

[97] Fermi architecture documentation. [On-Line]. Available at www.nvidia.com/object/fermi-architecture.html.

[98] CUDA C Programming Guide. [On-Line]. Available at docs.nvidia.com/cuda/cuda-c-programming-guide/

[99] CUTA PTX ISA v2.1. [On-Line]. Available at https://code.google.com/p/libptx/downloads/detail?name=ptx_isa_2.1.pdf

[100] NVIDIA BENCH: Tesla C2050 Performance Benchmarks. [On-Line]. Available at: http://www.siliconmechanics.com/files/C2050Benchmarks.pdf.

[101] DEVKIT features. [On-Line]. Available at: shop.seco.com/carma-devkit.html.

[102] Nvidia Quadro features. [On-Line]. Available at http://www.nvidia.com/object/quadro-mobile-features-benefits.html.

[103] M. Violante et al., "A New Hardware/Software Platform and a New 1/E Neutron Source for Soft Error Studies: Testing FPGAs at the ISIS Facility", in *IEEE Transactions on Nuclear Science*, Vol. 54, Issue 4, Part 2, pp. 1184 - 1189, 2009.

[104] A. Manuzzato, S. Gerardin, A. Paccagnella, L. Sterpone, M. Violante, "Effectiveness of TMR-Based Techniques to Mitigate Alpha-Induced SEU Accumulation in Commercial SRAM-Based FPGAs," in *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 1968 - 1973, 2008.

[105] JESD89A: Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft-errors in semiconductor devices, October 2006.

[106] H. R. Ghasemi, S. C. Draper, N. S. Kim, "Low-voltage on-chip cache architecture using heterogeneous cell sizes for high-performance processors", in *IEEE 17th International Symposium on High Performance Computer Architecture*, pp. 38 - 49, 2011.

[107] Charles Slayman, "Soft Error Trends and Mitigation Techniques in Memory Devices", presented at OPS A La Carte LLC 2010 [On-Line]. Available at http://www.opsalacarte.com/pdfs/Tech_Papers

[108] R. Baumann, "Radiation-Induced Soft Errors in Advanced Semiconductor Technologies", in *IEEE Transactions on Devices and Materials Reliability*, Vol.

5, No. 3, pp. 305 - 316, 2005.

[109] J. F. Zigler and H. Pucher, "SER - History, Trends and Challenges", Cypress press, 2010.

[110] M. White, "Scaled CMOS Technology Reliability User Guide", JPL Publication 09-33, 2010.

[111] B. Fang, J. Wei, K. Pattabiraman, M. Ripeanu, "Evaluating the Error Resilience of GPGPU Applications", in *SC Companion: High Performance Computing, Networking Storage and Analysis*, pp. 1504, 2012

[112] B. Ranft, T. Schoenwald, B. Kitt, "Parallel Matching-based Estimation: a Case Study on Three Different Hardware Architectures", in *IEEE Intelligent Vehicles Symposium (IV)*, pp. 1060 - 1067, 2011.

[113] Wilson W. L. Fung Ivan Sham George Yuan Tor M. Aamodt, "Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow", in *40th IEEE/ACM International Symposium on Microarchitecture*, 2007.

[114] M. Dimitrov, M. Mantor, and H. Zhou, "Understanding Software Approaches for GPGPU Reliability,". [On-line]. Available at http://www.eecs.ucf.edu/ zhou/GPGPU_v1.pdf

[115] Chong Ding, Christer Karlsson, Hui Liu, Teresa Davies, and Zizhong Chen, "Matrix Multiplication on GPUs with On-line Fault Tolerance," in *IEEE 9th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pp. 311 - 317, May 2011.

[116] ESA corot mission documentation. Available [Online]: www.esa.int/Our_Activities/Space_Science/COROT

[117] O. Bender, HiPEAC 2014, Available [Online]: http://www.across-project.eu/workshop2013/121108_ARAMIS_Introduction_HiPEAC_WS_V3.pdf

[118] P. Rech, L. L. Pilla, P. O. A. Navaux, and L. Carro, "Impact of GPUs Parallelism Management on Safety-Critical and HPC Applications Reliability," in *IEEE Dependable Systems and Networks (DSN)*, June 2014.

[119] L. Battista Gomez, F. Capello, L. Carro, N. DeBardeleben, B. Fang, S. Gurumurthi, K. Pattabiraman, P. Rech, and M. Sonza Reorda, "GPGPUs: How to Combine High Computational Power with High Reliability," in *IEEE Design, Automation and Test in Europe (DATE)*, March 2014.

[120] H. Jeon, M. Annavaram, "Warped-DMR: Light-weight Error Detection for GPGPU," in *45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 37 - 47, 2012.

[121] N. Seifert, N. Tam, "Timing vulnerability factors of sequentials," in *IEEE Transactions on Device and Materials*, Vol. 4, No. 3, pp. 516 - 522, September 2004.

[122] J.W. Cooley, and J.W. Tukey, "An algorithm for the machine calculation of complex Fourier series," in *Math. Comp.*, vol. 19, pp. 297 - 301, 1965.

[123] Y. Lee, et al, "An efficient FFT algorithm based on building on-line butterfly sub-structure," in *Fourth International Conference on Signal Processing Proceedings (ICSP)*, pp. 97 - 100, 1998.

[124] M. Fallahpour, C. Lin, M. Lin, C Chang, "Parallel One- and Two-Dimensional FFTs on GPGPUs," in *International Conference on Anti-Counterfeiting, Security and Identification (ASID)*, pp. 1 - 5, August 2012.

[125] P. Rech, T.D. Fairbanks, H.M. Quinn, L. Carro, "Threads Distribution Effects on Graphics Processing Units Neutron Sensitivity," in *IEEE Transactions on Nuclear Science*, vol. 60, no. 6., pp. 4220 - 4225, 2013.

[126] T. Santini, P. Rech, G. L. Nazar, L. Carro, and F. R. Wagner, "Reducing Embedded Software Radiation-Induced Failures Through Cache Memories", in *proc. IEEE European Test Symposium*, pp. 1 - 6, 2014.

[127] Titan - Oak Ridge Leadership Computing Facility [Online]. Available at: https://www.olcf.ornl.gov/titan/

[128] Moonlight - HPC at LANL - Los Alamos National Laboratory [Online]. Available at: http://hpc.lanl.gov

[129] E. Armengaud, A. Steininger, M. Horauer, "Towards a Systematic Test for Embedded Automotive Communication Systems, IEEE Transactions on Industrial Informatics, vol. 4, n.3, 2008.

[130] M. Steger, C. Tischer, B. Boss, A. Muller, O. Pertler, W. Stolz, S. Feber, "Introducing PLA at Bosch Gasoline Systems: Experiences and Practices, Springer Software Procuct Lines, Lecture Notes in Computer Science, vol. 3154, pp. 34 - 50, 2004.

[131] Y. Kanehagi, D. Umeda, A. Hayashi, K. Kimura, H. Kasahara, "Parallelization of automotive engine control software on embedded multi-core processor using OSCAR compiler, IEEE Cool Chips XVI, p. 1 - 3, 2013.

[132] F. Juan, M. Xian-Min, "Research on fuel injection intelligent control system, IEEE Conference on Industrial Electronics and Applications (ICEA), pp. 2782 - 2785, May 2009.

[133] A.D. Grasso, S. Pennisi, M. Paparo, D. Patti, "Estimation of in-cylinder pressure using spark plug discharge current measurements, European Conference on Circuit Theory and Design (ECCTD), pp. 1 - 4, 2013.

[134] F. Ostman, H.T. Toivonen, "Adaptive Cylinder Balancing of Internal Combustion Engines, IEEE Transacrions on Control System Technology, vol. 19, n. 4, pp. 782 - 791, 2011.

[135] I. Haskara, W. Yue-Yun, "Cylinder Pressure-Based Combustion Controls for Advanced Diesel Combustion With Multiple-Pulse Fuel Injection, IEEE Transactions on Control Systems Technology, vol. 21, n. 6, pp. 2143 - 2155, 2013.

[136] Q. Lui, H. Chen, Y. Hu, P. Sun, J. Li, "Modeling and Control of the Fuel Injection System for Rail Pressure Regulation in GDI Engine, IEEE/ASME Transactions on Mechatronics, vol. 19, n.5, pp. 1501 - 1513, 2014.

[137] J. Larimore, E. Hellstrom, S. Jade, J. Li, "Controlling Combustion Phasing Variability with Fuel Injection Timing In a Multicylinder HCCI Engine, American Control Conference (ACC), pp. 4435 - 4440, 2013.

[138] T. A. Johansen, O. Egeland, E. A. Johannessen, R. Kvamsdal, "Free- piston diesel engine timing and control - toward electronic cam- and crankshaft, IEEE Transactions on Control System Technology, vol. 10, n. 2, pp. 177 - 190, 2002.

[139] S. Hainz, E. Ofner, D. Hammerschmidt, T. Werth, "Position Detection in Automotive Application by Adaptive Inter Symbol Interference Removal, IEEE 5th conference on Sensors, pp. 1103 - 1106, 2006.

[140] S. Hainz, D. Hammerschmidt, "Compensation of Angular Errors Using Decision Feedback Equalizer Approach, IEEE Sensor Journal, vol. 8, n. 9, pp. 1548 - 1556, 2008.

[141] F. Li, T. Shen, X. Jiao, "Model-based design approach for gasoline engine control Part I: Modeling and validation," 32nd Chinese Control Conference (CCC), pp. 7774 - 7779, 2013.

[142] E. Alabastri, L. Magni, S. Ozioso, R. Scattolini, C. Siviero, and A. Zambelli, "Modeling, analysis and simulation of a gasoline direct injection system, in Proc. 1st IFAC Symp. Adv. Automot. Contr., 2004, pp. 273 - 278, 2004.

[143] National Instrument Data-Sheet, "Counter/Timer Overivew, pp. 386 - 393, 2013.

[144] W. Dafang, L. Shiqiang, J. Yi, Z. Guifan, "Decoding the Engine Crank Signal Referring to AUTOSAR, Intelligent Computation Technology and Automation (ICICTA), 2011 International Conference on, pp. 616 - 618, March 2011.

[145] Enhanced Time processor Unit (eTPU). Available [Online]: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=eTPU

[146] Generic Timer Module (GTM) Product Information. Available [Online]: http://www.bosch-semiconductors.de/media/en/pdf_1/ipmodules_1/timer/bosch_product_info_gtm_ip_v1_1.pdf.

[147] A. Ohata and K. R. Butts, "Improving model-based design for automotive control systems development, in Proc. 17th World Congr., ,pp. 1062 - 1065, 2008.

[148] X. Ying, Y. Qiangqiaang, L. Fuyuan, "Research of Crankshaft Grinding Wheel Dresser Based FANUC NC System, 3rd International Symposium on Information Processing (ISIP), pp. 189 - 192, 2010.

[149] T. Yamanaka, M. Esaki, M. Kinoshita, "Measurement of TDC in Engine by Microwave Technique, IEEE Transaction on Microwave Theory and Techniques, vol. 33, n. 12, pp. 1489 - 1494, 1985.

[150] Xilinx MicroBlaze details. Available [Online]: http://www.xilinx.com/tools/microblaze.htm

[151] C. Rodrigues, "A case study for Formal Verification of a timing co- processor, 10th Latin American Test Workshop (LATW), pp. 1-6, 2009.

[152] Freescale automotive functions set. Available [online]: http://www.freescale.com/webapp/etpu/

[153] Freescale Application Note. Available [Online]: http://cache.freescale.com/files/32bit/doc/app_note/AN3769.pdf