FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

The Transient Tolerant Time-Triggered System-on-Chip (4TSoC)

DISSERTATION

zur Erlangung des akademischen Grades

Doktor/in der technischen Wissenschaften

eingereicht von

Mikel Azkarate-askasua Blazquez
Matrikelnummer 0928311

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Priv.Doz. Dipl.-Ing. Dr.techn. Roman Obermaisser

Diese Dissertation haben begutachtet:

(Priv.Doz. Dipl.-Ing. Dr.techn. Roman Obermaisser)	(Prof. Dr. Kees G.W. Goossens)

Wien, 15.10.2012

(Mikel Azkarate-askasua Blazquez)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

The Transient Tolerant Time-Triggered System-on-Chip (4TSoC)

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor/in der technischen Wissenschaften

by

Mikel Azkarate-askasua Blazquez
Registration Number 0928311

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Priv.Doz. Dipl.-Ing. Dr.techn. Roman Obermaisser

The dissertation has been reviewed by:

(Priv.Doz. Dipl.-Ing. Dr.techn. Roman Obermaisser)	(Prof. Dr. Kees G.W. Goossens)

Wien, 15.10.2012

(Mikel Azkarate-askasua Blazquez)

Erklärung zur Verfassung der Arbeit

Mikel Azkarate-askasua Blazquez
Arizmendiarrieta 2, 20500 Arrasate (Baskenland)

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

_____ _____

(Ort, Datum) (Unterschrift Verfasser)

The Transient Tolerant Time-Triggered System-on-Chip (4TSoC)

Embedded systems of different application domains (offshore windmills, railway, avionic, etc.) can benefit from integrated architectures. The functionality that required several chips in the past can now be integrated in a single chip due to the recent advances on silicon technology miniaturization. This approach carries interesting economical benefits due to the reduction on cost of electronic components and interconnection.

Most of the current integrated architectures have been implemented using a software approach (e.g., a hypervisor) in order to build the illusion of having several execution environments on monolithic processor chips. However, building integrated architectures using a hardware approach, upon a Multi-Processor System-on-Chip (MPSoC), the system not only achieves a better performance, but also increased power efficiency, and significantly higher reliability. In fact, high integration enables small transistor technologies, but causes more sensitivity w.r.t. energy variations which requires new fault tolerance measures to overcome the transient fault rates (e.g., soft-errors) that have significantly increased.

This dissertation presents a Transient Tolerant Time-Triggered System-on-Chip (4TSoC), an integrated architecture for safety-related embedded systems. The 4TSoC architecture introuduces fault-tolerance mechanisms for application components, communication interfaces and the Time-Triggered Network-on-Chip. As a prerequisite for replication we introduce on-chip fault-containment mechanisms along with design methods to address fault containment during synthesis.

A Fault Injection for System-on-Chip (FI4SoC) has been developed to test state-of-the-art integrated architectures (e.g., XtratuM, TTSoC) and validate 4TSoC hardening configurations. The experiments have provided experimental evidence for reliability of the 4TSoC architecture in the presence of soft-errors.

The Transient Tolerant Time-Triggered System-on-Chip (4TSoC)

Integrierte Architekturen stellen einen signifikanten Nutzen für eingebettete Systeme aus verschiedenen Anwendungsdomänen (z. B. Windenergie, Eisenbahn, Luftfahrtelektronik, etc.). Die Funktionalität, die bisher durch mehrere Einzelchips erreicht wurde, kann nun durch die Fortschritte der Silikonindustrie in einen einzigen Chip integriert werden. Dieser neue Ansatz bringt Kostenreduktion sowohl bezüglich der Anzahl der Komponenten als auch deren Verkabelung.

Viele gängige integrierten Systeme benutzen einen Softwareansatz (i.e. einen Hypervisor), um mehrere virtuelle Ausführungsumgebungen auf einem Chip zu emulieren. Im Gegensatz dazu basiert die vorliegende Arbeit auf einem Hardwareansatz, genauer einem "Multi-Processor System-on-Chip (MPSoC)". Diese Hardwarelösung bietet Vorteile hinsichtlich Performanz, Energieeffizienz und Zuverlässigkeit gegenüber dem Softwareansatz.

Diese Dissertation behandelt eine "Flüchtige Fehler tolerierende zeitgesteuerte System-on-Chip Architektur", die als integrierte Architektur für sicherheitskritische eingebettete Systeme konzipiert ist. Als Teil der Architektur wird eine Replikation der Applikationskomponenten und die dazugehörige Systemkomponente vorgestellt. Im Weiteren wird ein Fehlermodell für MPSoCs erarbeitet, das die Architektur durch Gliederung in Fehlerbegrenzungen ("Fault Containment Regions (FCR)") und deren Replikation beherrscht. Zum Testen der Architektur wurde ein Fehlereinstreuungssystem ("Fault Injection for System-on-Chip (FI4SoC)") entwickelt, um aktuelle integrierte Architekturen zu testen (z.B. XtratuM, TTSoC) und MaSSnahmen zur Härtung zu validieren.

Die Arbeit schlieSSt mit einer Betrachtung verschiedener MaSSnahmen zur Verbesserung der Fehlertoleranz in einem zeitgesteuerten System-on-Chip und deren Anwendbarkeit in verschiedenen Anwendungsdomänen.

The Transient Tolerant Time-Triggered System-on-Chip (4TSoC)

Aplikazio domeinu ezberdinetako sistema txertatuak (itsasoko haizerrotak, trengintza, hegazkingintza, etab.) txip bakarrean integratutako arkitekturez profita daitezke. Aurrez txip anitzetan inplementatutako funtzionalitateak txip bakar batean sar daitezke orain azken urteotan silizio teknologiak jazo duen izugarrizko miniaturizazioari esker. Honek ekonomikoki oso interesgarria den txip eta interkonexio murrizketa dakar.

Sistema txertatuentzako integratutako arkitekturek software mekanismoak erabili dituzte prozesadore bakarreko txipetan exekuzio ingurune ezberdinen ilusioak sortzeko (adb., hiperbisoreak). Aldiz, integratutako arkitektura berberak hardware mekanismoak erabiliz eraikiz gero, txipean txertatutako prozesadore anitzeko sistema (ingelesez MPSoC) bat erabilita, errendimendu hobea lortzeaz gain, energetikoki, eta batez ere, segurtasunari dagokioenean abantail ugari lor litezke. Izan ere, aipatutako integrazioa ahalbidetzen duen transistore teknologia nimiñoak, energia bariazioetara sentikorragoak diren txipak ekarri ditu, eta honek mekanismo berriak eskatzen ditu falta iragankorren maiztasun igoera nabarmenari (soft-error deitutakoak bereziki) aurre egiteko.

Tesi honek falta iragankorrekiko indartutako MPSoC bat (4TSoC) aurkezten du, segurtasunarekin lotutako sistema txertatuentzako integratutako arkitektura bat. Txip barnean erreplikatutako konponenteen erabilera proposatzen du horretarako, bai aplikazioaren menpe dauden konponenteak inplementatzeko, baita arkitekturako sistema-konponenteak berak indartzeko ere. Konfigurazio guztiekin MPSoCak fidagarriago egiteko aukeren modelo bat sortu da. Gainera, erreplikazio hauek baliagarri suertatzeko beharrezkoa den falta-kontentzioa aurkezten du eta berau nola bermatu hardwarea sintetizatzeko orduan.

Txipean txertatutako sistemak frogatzeko falta injekzio erraminta bat (FI4SoC) garatu da, eta akademiak aurkeztutako beste integratutako arkitektura batzuk (XtratuM, TTSoC) ikertzeaz gain proposatutako 4TSoC konfigurazioen liburutegia baliozkotzat eman da. Azkenik, konfigurazio interesgarrienak aplikazio domeinu ezberdinetan duten balioa neurtu da.

Acknowledgments

This dissertation has been developed in collaboration among the *Electronics* division of Ikerlan Research Center (Basque Country, Spain), the *Real-Time System* group of Vienna University of Technology (TU Wien, Austria) and the *Embedded Systems* group of the University of Siegen (Germany).

I would like to give very special thanks to six extraordinary professionals. First, to Prof. Roman Obermaisser for his full support in these 3 years, his great ideas and our passionate discussions in Vienna and Siegen. To Prof. Hermann Kopetz and Antonio Perez for giving me the opportunity to do this work between TU Wien and Ikerlan. To Prof. Kees Goossens for his reviews and contribution from Eindhoven as co-examiner of this thesis. And to Dr. Imanol Martinez and Dr. Jon Perez for their support from Ikerlan Research Center.

I would like to mention my colleges in Ikerlan (Iban, Niko and Xabi), TU Wien (Albrecht, Armin, Bekim, Benedikt, Christian E. S., Christian P., Ekarin, Harald, Michael, Roland, Oliver, Sven, Vaçlav and Wolfgang), Siegen (Rubaiyat and Zaher), all with whom I have discussed several points of my thesis and, of course, the rest of the friends and staff within the three institutions.

Finally, to all the people I crossed and enjoyed (I hope for long time) during these last years in Arrasate (Ander, Imanol, Jon and Vero), Vienna (Aaron, Ainhoa, Eirini, Iñaki, Irene, Jarek, Jonathan, Ju, Mikel G., Rocio, Sara and Virginia) and Siegen (Imad and Mohan).

Contents

List of Figures

List of Tables

List of Acronyms

4TSoC Transient Tolerant Time-Triggered System-on-Chip

ASIC Application Specific Integrated Circuit

AUTOSAR Automotive Open System Architecture

CF Compact Flash

COTS Commercial Off-The-Shelf

CPS Cyber-Physical System

CRCR Capture-Readback-Controlled Reset

DAS Distributed Application Subsystem

DMA Direct Memory Access

ECC Error Correcting Codes

EDC Error Detection Code

EMI Electromagnetic Interference

FCR Fault Containment Region

FI4SoC Fault Injection for System-on-Chip

FIT Failure in Time

FPGA Field Programmable Gate Array

GPIO General Purpose Input Output

HWIFI Hardware Implemented Fault Injection

IMA Integrated Modular Avionics

IP Intellectual Property

LI Local Interface

LIF Linking Interface

LRM Local Resource Manager

NoC Network-on-Chip

MBD Model Based Design

MBU Multiple Bit Upset

MFTF Mean Faults To Failure

MMU Memory Management Unit

MTTF Mean Time To Failure

MTTR Mean Time To Recover

MPSoC Multi-Processor System-on-Chip

NoTA Network on Terminal Architecture

NI Network Interface

OS Operating System

RTL Register Transfer Level

RTOS Real-Time Operating System

PLB Peripheral Local Bus

PIM Platform Independent Model

PSM Platform Specific Model

SEE Single Event Effect

SET Single Event Transient

SEU Single Event Upset

SOI Silicon On Insulator

SWIFI Software Implemented Fault Injection

TDI Technology-Dependant Interface

TDM Time Division Multiplexing

TID Total Ionizing Dose

TII Technology-Independent Interface

TISS Trusted Interface Subsystem

TMR Triple Modular Redundancy

TRM Trusted Resource Manager

TSS Trusted Subsystem

TTNoC Time-Triggered Network-on-Chip

TTSoC Time-Triggered System-on-Chip

URM Untrusted Resource Manager

VDSM Very Deep Sub-Micron

VMM Virtual Machine Monitor

List of Abbreviations

a.k.a. *also known as*
e.g. *exempli gratia* (for example)
i.e. *id est* (that is)
vs. *versus* (against)
w.r.t. *with respect to*

Ama, aitxa, arreba zein betiko lagunei
urrin ta hurbil dien honei
maitasunakin

The quotes at the beginning of each chapter are traditional Basque sayings, property of the Basque people for hundreds of years.

"Eroa da hasten düana
ürrent ez dirokean lana",
It is crazy the one who begins
the work that cannot finish

Chapter 1

Introduction

The tremendous advances of the semiconductor technology enables more powerful chips with more transistors using less silicon area. Current chips can integrate more than one billion 28 nanometer transistors switching faster than 1 GHz. Multi-core chips offer an interesting abstraction that eases the understanding of such complex and highly integrated chips, by partitioning the chip in several cores. They also offer a way to overcome the performance limits of monolithic processors [Gel01] and they are shown to be more energy efficient [PPB+07]. Furthermore, the parallel computation of multi-core chips offers means to deal with concurrency.

These interesting features have already penetrated the embedded market with the name of Multi-Processor System-on-Chips (MPSoCs) where, depending on the application, the energy efficiency or dealing with concurrency are usually mandatory requirements. According to predictions, by the year 2015 the use of MPSoCs in multimedia (e.g., smart-phones [K.11]) and mixed-criticality [Ern10] embedded systems will share between 30% and 90% of the embedded market.

Focusing on mixed-criticality embedded systems, where the safety of the implemented functions and the reliability of the integrated chips play a crucial role, MPSoCs can also provide interesting features [KOESH07]. The integration of several functions of different application systems and criticalities has typically been done using a software approach where a monolithic processor was partitioned by hypervisors and other virtualization mechanisms. However, a hardware approach using MPSoCs provides intrinsic design fault containment by dedicated processors with some containment coverage for physical faults that cannot be achieved by software approaches. Due to the spatial separation of the hardware resources within the MPSoC, physical faults can damage some processing elements but not all of them in some degree or coverage.

1.1 Problem Statement

Highly integrated silicon technologies show increasing rates of transient physical fault due to process variations, shrinking geometries, and lower power voltages [Con02]. These physical faults require rigorous function isolation (e.g., specified by the IEC-61508 safety standard [IEC09]) if a reliability increase is obtained by the replication of on-chip components. This rigorous on-chip replication cannot be achieved on any software approach upon monolithic processors due to the resource sharing of the software replicas. In contrast, the computation blocks of an MPSoC can provide certain fault containment coverage to host the replicas. Anyway, even in an MPSoC hardware approach, independent chips are necessary if safety-critical reliability is pursued by the replication of components and those component should communicate using distributed networks.

The reliability of the critical infrastructure is another technical problem of hardware-based integrated architectures at the chip-level compared to distributed systems. The shared resources and core services of the MPSoC (the communication service, the clock, etc.) could lead to single points of failures within the chip because the rest of the on-chip components rely on them.

1.2 Proposed Solution

This dissertation introduces the Transient Tolerant Time-Triggered System-on-Chip (4TSoC), an MPSoC based integrated architecture at the chip-level with system-level fault tolerance mechanism against soft-errors. These soft-errors are spontaneous bit-flips in flip-flops and memory elements and they are considered the most common type of transient fault. The MPSoC approach provides superior physical fault-containment coverage compared to software approaches and fault tolerance is provided not only for the application subsystem, but also for the critical infrastructure (e.g., NoC and core services) of the architecture.

In contrast to other low-level approaches such as redundancy [Gai06] [Xil06], Error Correcting Codes (ECCs) [PKCC06] or circuit level hardening [MSZ+05], reliability increase for chip against transient fault can benefit from system-level management of an MPSoC based integrated architecture. This architecture offers fault containment features that enable the incorporation of system-level fault tolerance mechanism from distributed systems (component TMR, communication channel duplication, etc.) by analogy. System-level mechanisms enable a higher abstraction level, higher resilience against proximity faults and

ease the chip development (e.g., ability to use standard libraries, reduce the complexity, early validation).

1.3 Contributions

The contribution or partial objectives of this dissertation consists of:

- **The 4TSoC model**: an MPSoC model with support for fault containment and fault masking based in a transient fault hypothesis for mixed-criticality systems, an extension of the TTSoC architecture. A synthesis model for such an MPSoC is also introduced inspired by the isolation requirements of the IEC-61508 standard.
- **Fault tolerance mechanisms library for MPSoCs**: an adaptation of fault tolerance mechanisms from distributed systems for on-chip multiprocessors, i.e., component TMR, network interface replication, the use of multiple NoCs and message-level ECCs.
- **The FI4SoC fault injection framework**: an FPGA based fault injection framework for integrated architectures supporting transient fault emulation at RTL level using dynamic partial reconfiguration.
- **Reliability Assessment of the TTSoC**: an evaluation of the reliability of the TTSoC components (network interfaces, NoC, etc.) and the effectiveness of its fault containment and on-chip replication mechanisms.
- **Comparison of fault tolerance mechanisms**: a comparison of the previously introduced fault tolerance mechanisms using a specificic soft-error (Single Event Transients, SETs) transient fault model.

1.4 Thesis Organization

This thesis is organized as described below:

- Chapter 2 describes the background and basic concepts on which the work of this thesis is based. It follows three main paths: the notion of time, integrated architectures and dependability.
- Chapter 3 analyzes state-of-the-art of integrated architectures with respect to timeliness, design fault containment and physical fault handling.

The scope of the analysis covers the XtratuM hypervisor, the Cell multiprocessor, the CoMPoC MPSoC template and the TTSoC architecture. The on-chip replication recommended by the IEC-61508 standard is also studied. The reasons for the selection are described in the chapter.

- Chapter 4 introduces the 4TSoC architecture, an MPSoC model to increase the reliability of single chips against the transient faults within the fault hypothesis. It offers a fault containment model and a collection fault tolerance mechanisms.
- Chapter 5 describes the evaluation platform for the validation of the 4TSoC approach.
- Chapter 6 describes the experiments assessing the MPSoC approach for on-chip fault-containment, the TTSoC architecture and the new features of the 4TSoC model.
- Chapter 7 reviews the evaluation results and provides a comparative overview of the different integrated architecture approaches and fault tolerance mechanisms.
- Finally, Chapter 8 shows the conclusion and future work.

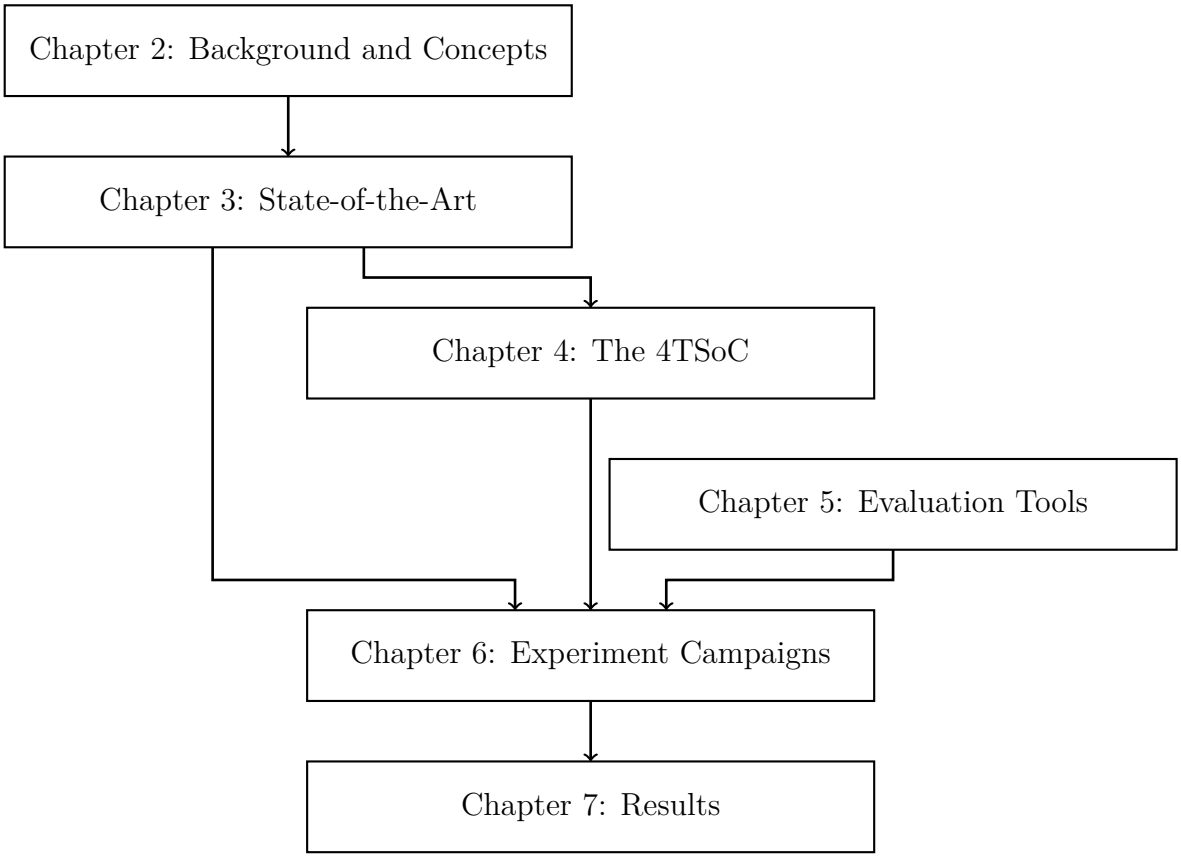


Figure 1.1: Thesis Organization

"Zaldirik seguruena, nork bere zangoak",
The safest horse, one's own legs

Chapter 2

Background and Basic Concepts

This chapter gives the background and the basic concepts on which this thesis is based.

2.1 The Notion of Time

Incorporating the notion of time into embedded systems is a key requirement [Per11]. In fact, embedded systems are also known as *Cyber-Physical Systems* (CPSs) to emphasize this integration with time and physical environment. The notion of time generically used in embedded systems is the Newtonian physics concept of time, dismissing relativistic effects.

2.1.1 Time Flow and Models

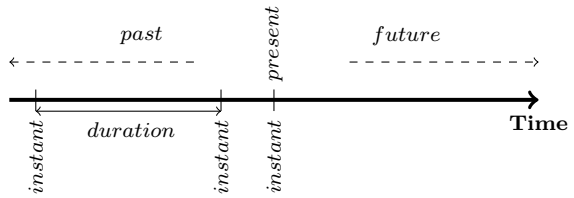
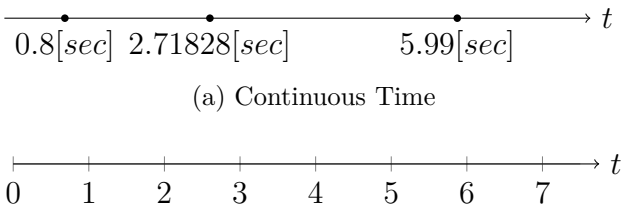


Figure 2.1: Time representation

The flow of time is represented as a straight line going from the past (on the left) to the future (on the right) as shown in Figure 2.1. A cut on this line is an instant and the present instant, now, decouples the past from the future. A relevant happening occurring on a particular instant is an event and the interval between two instants is named a duration [Kop06].

A suitable model for time in the real world is a continuous or dense time-line (Figure 2.2), whereas in digital systems a discretized model of time is used. In a dense model, time advances continuously with infinitesimal steps. A discrete model consists of time steps of a fixed duration adapting the dense model to the computational environment.



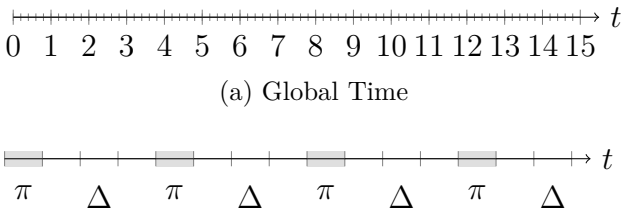
(a) Continuous Time

(b) Discrete Time

Figure 2.2: The dense and discrete models of time.

In distributed systems (Figure 2.3), a discrete global time (with a sparse time model) is established to provide a consistent temporal order of events based on their time-stamps. This global time is approximated by generating a macrotick clock using the local microtick clock of the distributed computers and a clock synchronization algorithm [Kop11]. Through this approach a maximal divergence of one tick is achieved among the local microtick clocks, which is known as the reasonableness condition.

A sparse time model [Kop07] restricts the occurrence of events (e.g., the sending of a message) that are in the sphere of control of the computer system to the activity intervals of a sparse-time base and the distributed parts share the same global time using a clock synchronization algorithm. Real-time is partitioned into a sequence of alternating intervals of activity of duration π and silence of duration Δ. All the events that happen within the same activity interval are considered simultaneous.



(a) Global Time

(b) Sparse Time

Figure 2.3: Distributed models of time.

The global time as well as the sparse time model can be applied at chip-level and distributed systems [KOESH07]. Focusing at chip level, the global time can be used to synchronize multiple clock domains and avoid undesired effects. These effects include:

- **Clock skew**: it is the maximum delay from the clock input of one flip-flop to the clock input of another flip-flop. It is also knows as phase noise. The skew is mainly caused by the differences among clock sources and the clock distribution network.

- **Clock jitter**: it refers to differences between actual output position and the ideal output position of the clock edge. There are two contributions, fixed and random, to the total clock jitter. The fixed jitter has more timing offset and it is caused by specific sources such as crosstalk, signal noise, etc. The random jitter is derived from environmental factors (temperature, radiation, etc.).

The time margin for the synthesis of a chip is too narrow for future on-chip distributed systems with a single clock source at a negligible skew [BDM02], therefore, the chip must use several clock domains. In these cases, a global time is an option to overcome skew effects due to the technology constrains and to provide synchronization of multiple clocks within the reasonableness condition. This on-chip global time can be implemented using a slow clock line distributed along the chip or using clock synchronization algorithms analogously to classic off-chip distributed networks (e.g., TTP).

Despite the linear representation of time models, embedded control systems typically exhibit a cyclic temporal structure with a set of steps that are repeated each cycle (Figure 2.4) [OSHK08]. This cyclic description fits better the human cognitive nature of time [Win01]. The time-diagrams of this thesis prefer this circular representation.

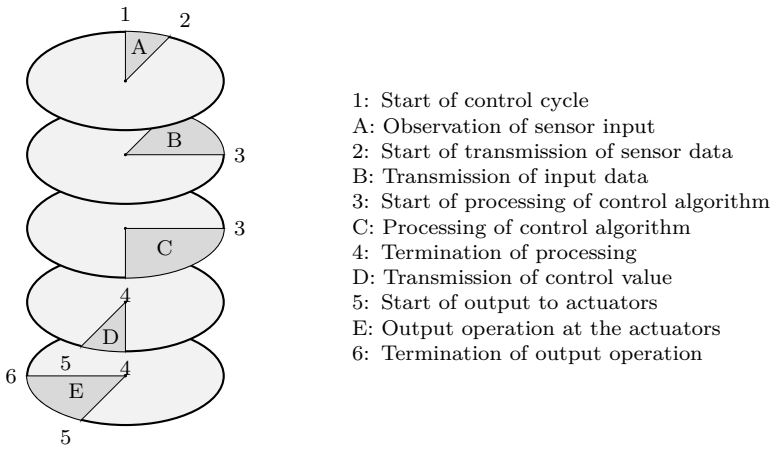


Figure 2.4: Cyclic representation of an embedded control system process.

2.1.2 State

Once a precise concept of time is introduced in a system a precise concept of state can be established. State decouples past from the future, in such a way that in a deterministic system, a future output can be determined by the future inputs and this system state [Mes89].

One can distinguish three types of system states, the initialization-state (i-state), the history-state (h-state) and the ground-state (g-state) [Kop11]:

- *i-state*: this is the state that can be loaded off-line, the static data structures (e.g., application program code, initialization data).
- *h-state*: it comprises the information required to start an "empty" composed function task at a given point in time. It is defined by the dynamic data structures at given instant that contain information about the current and past computations.
- *g-state*: it is the minimal h-state, when tasks are inactive and the channels are flushed. This g-state is ideally periodic.

A consistent notion of state is a prerequisite for fault masking through voting where the replicated subsystems must be replica determinate. This condition is only full-filled if each of the replicated nodes contains the same externally visible h-state at its ground state, and produces the same output messages at points in time at bounded time intervals [Pol95].

The instant when a system reaches the ground state is also called reintegration point. This minimal h-state is the optimal point in time for a subsystem to recover from a faulty-state by recreating it or copying [XR96] the g-state from a replicated node.

2.1.3 Determinism

A given item (e.g, property, output) is deterministic if it is completely predictable and does not depend on randomness for a given set of conditions [Kop08b]. For instance, an output is deterministic for a given set of relevant conditions, if given the same set of initial conditions then the system always generates the same outputs at the same time when given the same inputs at the same time.

As time is part of the definition of determinism, it must be consistent (w.r.t. an external observer) at the system level of the design. For instance, in the case of a distributed embedded system the notion of time should be based on the sparse time.

2.1.4 Behavior and Service

The behavior of a system can be defined as its activity (e.g., message sequence) during the progression of time. A system's service is the behavior according to the specification, whereas a failure is the opposite, a deviation of the system behavior from the specification.

2.2 Job, Partition and Component

In *Model Based Design* (MBD) the system services are first designed in a *Platform Independent Model* (PIM) and then mapped to a given platform using a *Platform Specific Model* (PSM). Job, partition and component concepts are introduced during this PIM to PSM refinement process.

2.2.1 Job

A job is the basic unit of work [KOESH07] which provides a service to other jobs across the linking interfaces and to the environment via the local interfaces into a PIM. A distributed application consist on more than one *Distributed Application Subsystem* (DAS). A DAS is a nearly autonomous application system that performs a composite of services. For example, a train has up to dozens of DASes, such as, traction control, signaling, infotainment, etc. On its behalf, each DAS comprises several jobs, the unit of distribution. Therefore, a DAS is a composition of jobs and services.

2.2.2 Partition

A partition is the physical execution environment for a job. Originally, the term partition referred to the allocation on time domain of a monolithic processor to multiple operating systems [Rus99], but in this dissertation it also refers to the spatial and physical decoupling of the execution environment. A partition can be a classical node (e.g., an ECU in the automotive domain), a virtual processor (e.g., in an hypervisor), a dedicated processor of a multi-core chip or a silicon fabric (e.g., an FPGA implementing a finite state-machine). The purpose of partitioning is fault containment: a failure in one partition must not propagate to cause failure in another partition. A partition should provide spatial and temporal partitioning with respect to other partitions [Rus99]:

- *Spatial Partitioning*: a partition ensures that the job in one partition cannot change the software or private data of another job, nor command the private devices of another partition.

- *Temporal Partitioning*: ensures that the service received from shared resources by a job in one partition cannot be affected by a job in another partition, including performance, rate, latency or jitter.

Partitioning is a prerequisite for the composability of jobs from different sources and criticalities.

2.2.3 Component

The mapping of a job into a partition results in a component. These hardware-software components, which are self contained subsystems, can be used as building blocks in the design of a larger system [KOESH07] [RE06] of a component-based design. A large proportion of complex systems in nature evolve from the hierarchical composition of simple components [Sim62]. The building of complex embedded systems can benefit from the easier understanding of hierarchical component-based design.

The definition of component interfaces is of utmost importance in order to avoid undesired interactions and facilitate the desired ones. Components interact using four basic types of interfaces (Figure 2.5) [OKP10]:

- *Local Interface* (LI): it establishes a connection between a component and its local environment.
- *Linking Interface* (LIF): the services of a component are offered to other components through this interface.
- *Technology-Independent Interface* (TII): this interface is used by the system to perform operations without the involvement of the application.
- *Technology-Dependant Interface* (TDI): it provides the means to look inside a component and to observe internal variables.

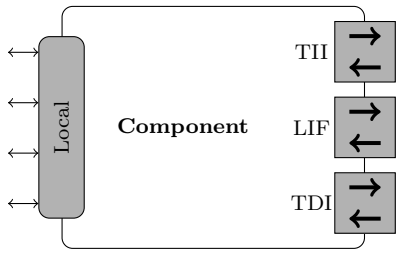


Figure 2.5: Interfaces of a component.

From the point of view of service provision, there are two types of components: system components and application components. System components provide an architectural service and they conform to the "style" of an architecture (the services and constraints that condition the implementation of a system). Application components implement the specified application functionality, the application service, and use the services of the available system components to reduce the effort required to implement the application functionality. Hence, an integrated architecture provides system components to integrate several application components that map jobs of multiple DASes on a single node.

2.3 Integrated Architectures

An architecture is a framework for the construction of systems for a chosen application domain that provides generic architectural services and imposes and architectural style for constraining an implementation in such a way that the ensuing system is understandable, maintainable, extensible, and can be built cost effectively [OK09]. The architectural style describes the principles, accepted statements about some fundamental insight in a domain, and structuring rules that characterize an architecture [Kop11].

An integrated architecture [OK09] provides multiple partitions for the mapping of jobs of different DASes into a single node. Whereas in federated architectures [KOPS04] each node provided a single partition for a single job, the high integration of transistors in semiconductors makes possible to integrate several jobs in a single chip. This integration enables the reduction of hardware and connection wires and the subsequent decrease of the amount of power used, weight, space and the number of computation chips. Several integrated architectures were created for specific application domains, for example: *Automotive Open System Architecture* (AUTOSAR) [GbR06], *Integrated Modular Avionics* (IMA) [ARI91] or *Network on Terminal Architecture* (NoTA) [KKOE07].

2.3.1 Implementation

The implementation of an integrated architecture can be done following two main approaches: a software approach or a hardware approach. On the first approach, the communication services are implemented in software (e.g., hypervisor queues) offering virtual processors as partitions for the execution of jobs. In the second option, the communication is implemented in hardware (e.g., a *Network-on-Chip* (NoC)) and dedicated processors or silicon fabric (e.g., FPGA) are provided as a partition for each job.

Software Approach

Several software architectures address temporal and spatial partitioning on a monolithic processor approach, from which the most populars are microkernels [Lie95] and hypervisors [CRM+09]. Whereas hypervisors provide partitions for operating systems (e.g., jobs), microkernels perform a context switch per thread where the system software itself acts as an operating system. Hence, the microkernel approach is not the most suitable for the integration of distributed jobs (with potentially heterogenous operating systems), but for non-distributed homogeneous threads. In this section the scope of the software approach will be put on embedded hypervisors.

Hypervisors, also known as *Virtual Machine Monitors* (VMMs), are programs that run on hardware or a host *Operating System* (OS) using the highest privilege level, governing and separating multiple partitions (Figure 2.6). They enable the software implementation of jobs that can be assisted by a general-purpose OS or a *Real-Time Operating System* (RTOS).

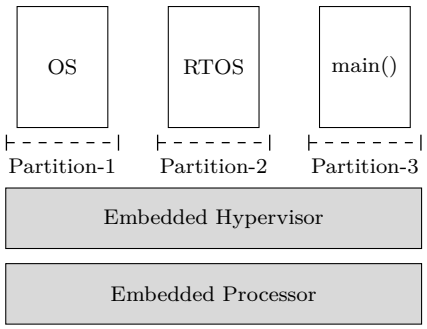


Figure 2.6: (Bare-metal) Embedded hypervisor architecture

Hypervisors are classified by the layer that executes them and the virtualization level. In the first category, type-1 hypervisors (also named bare-metal) run directly on the native hardware, whereas type-2 lay upon a host OS. The bare-metal approach reduces the virtualization overhead making it more efficient for embedded systems. Anyway, this approach can require specific hardware features of the processor, such as: additional privileged modes for each partition, direct interrupts or extended page table for the memories of each partition [Neu06]. In the second category, full-virtualized and para-virtualized architectures are distinguished. Full-virtualization offers a virtual image of all the resources and therefore support any kind of program or OS in the partitions. Para-virtualization replaces the critical or conflicting instructions that may break the isolation by hypervisor services or hypercalls. It minimizes hypervisor overhead, but it requires customization of the guest program or OS in order to support those hypercalls. The source code of the OS is frequently avail-

able in embedded system and, as it requires less overhead, the para-virtualized approach is often preferred.

The implementation of partitioning in hypervisors requires the following features: (1) spatial partitioning, achieved by a hardware mediation (e.g., a *Memory Management Unit* (MMU)) that prevents any job to write in the memory locations of other jobs or the hypervisor.(2) A fixed cyclic scheduling of the partitions, providing temporal partitioning for the integrated jobs.

Hardware Approach

The hardware approach aims at providing dedicated cores of a *Multi-Processor System-on-Chip* (MPSoC) as partitions for jobs. MPSoC architectures differ at the way the on-chip cores communicate among themselves, e.g., shared memories (e.g., caches [CS99], FIFOs [NTS+08]), buses, or through NoCs [HGBH09] [OSHK08]. The explicit timing of message based communication is preferred in order to enhance temporal partitioning for integrated architectures (e.g., using a deterministic schedule). Among message based communication options, NoCs scale better than buses when the number of cores increases [BDM02].

A NoC based integrated architecture approach provides message-based services to a set of hardware partitions (Figure 2.7) that constitute the MPSoC. These hardware partitions permit the heterogeneous (software or hardware) implementation of jobs. The software implementation of a job executes upon a dedicated core that can be assisted by an OS. The hardware implementation of a job upon a silicon fabric (e.g., FPGA) is also called an *Intellectual Property* (IP) core. Anyway, this term is controversial because many of these cores are not claimed for intelectual property and they are publicly available (e.g., opencores.org). This dissertation uses the term *core* to refer to the hardware implementation of jobs.

First of all, the NoC and *Network Interfaces* (NIs) are responsible for the information exchange among components, but also for the spatial and temporal partitioning of the chip. Spatial partitioning is ensured by limiting the interaction of components to the exchange of messages. The NIs of each partition (e.g., a time-triggered NoC) can contain the predefined points in time that each job can transmits messages and this information can not modified by the job. Hence, the temporal behavior of a job cannot be interfered (e.g, message collision) by other jobs.

Mixed Approach

There are multiple solutions using a software partitioning approach (e.g., hypervisor) upon a multi-core chip [RTS10]. Anyway, these works do not con-

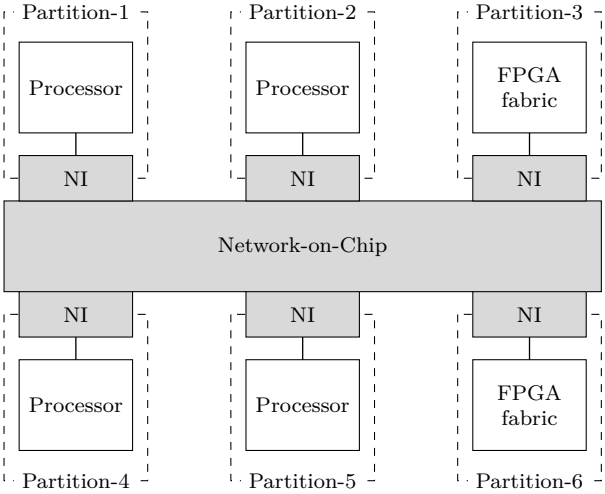


Figure 2.7: A NoC based MPSoC

sider both, the hardware and the software, in a single architectural style. The hardware is a predefined multiple-purpose multi-processor chip and the partitioning software must adapt to those limited resources that do not match all the requirements (e.g., memory independence) and constraints (e.g., physical fault containment coverage) of the integrated architecture style. Moreover, this mixed approach often address data-server applications more than embedded systems.

There are new projects such as the European FP-7 MULTI-PARTES project [Mul11] targeting a mixed integrated architecture approach addressing embedded systems and their use in safety-related applications.

2.3.2 Technologies

Integrated architectures are synthesized into a final end-device technology. Both the single processor behind the software approach and the MPSoC for the hardware approach can be implemented using ASIC or FPGA technology.

Application Specific Integrated Circuit (ASIC)

Application Specific Integrated Circuit (ASIC) devices are integrated circuits for a particular application that consist of a large number of primitive logic elements (e.g., NAND, NOR, AND), Flip Flops and interconnections. Current shrinking technology allows the integration of over 100 million gates on a single chip. Nowadays, it is also possible to mix analog primitives in digital ASIC fabric. Moreover, ASICs offer different levels of customization, like standard cell, structured, full custom, etc.

ASIC technologies are usually programmed using photo-lithographic masks which are too expensive (e.g., 1 billion dollars) to produce for low production volumes. Nevertheless, the largest number of embedded systems make use of generic processors chips built in this technology because they are produced in large volumes that make this approach economically profitable.

Field Programmable Gate Array (FPGA)

An FPGA is a device that can be reconfigured after manufacturing by storing a circuit diagram synthesized from a hardware description language (HDL) into its configuration memory. It is considered an alternative to ASICs for smaller design and lower production volumes. The configuration memory holds the setup of the FPGA configurable structures after a synthesis, placement and routing process.

Logic is implemented using Look Up Tables (LUTs) which store truth tables in the configuration memory. Sequential functions are implemented thanks to flip-flops and routing is performed using switching matrix structures and multiplexers. The LUTs, flip-flops and routing elements compose the programmable elements or Configurable Logic Blocks (CLBs). Custom clock frequencies and noise reduction are obtained with Digital Clock Managers (DCMs) and PLLs. The resulting clock signals are distributed by dedicated clock nets. Memories, buffers and registers can be implemented using the aforementioned LUT or flip-flop structures or with dedicated embedded block memories (e.g., Block RAMs). New application oriented FPGAs include DSP blocks or even silicon implemented processors (hard cores).

FPGAs can be classified by the memory technology used to store the circuit information: antifuze, flash (e.g., EPROM, EEPROM) and Static-RAM (SRAM) FPGAs.

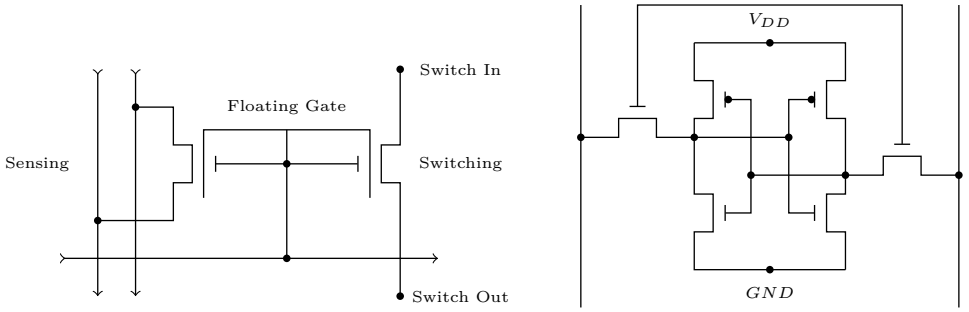
- *Antifuze FPGAs*: the antifuze technology is based in the following principle: in the open stage there is an insulator between two hardly doped semiconductors (Figure 2.8b) and the close stage is obtained by applying an important voltage (e.g., 16V) which causes the break of the insulator and makes the antifuze conducting (Figure 2.8a). This technology is programmable only once.

- *Flash FPGAs*: the flash FPGA configuration memory is based on a non volatile memory where two MOS transistors are used per bit cell (Figure 2.9a), similarly to other EPROM/EEPROM memories. It is the cheapest technology as it only uses two transistors per bit cell.

(a) Programmed (b) Unprogrammed

Figure 2.8: Actel antifuze technology cross section

- *SRAM FPGAs*: Nowadays, most of scommercial FPGAs are manufactured in SRAM technology. It is based on the well know six transistor (6T) SRAM technology (Figure 2.9b). It provides the highest integration capacity (e.g., $45nm$) compare with previous technologies and it is the most extended technology.



(a) Flash (b) SRAM

Figure 2.9: Memory cells technologies

2.4 Dependability Concepts

Figure 2.10: Fundamental chain of dependability threats

A fault is the first link of the fundamental chain of dependability threats (Figure 2.10) [ALR01]. A fault is defined as the hypothesized cause of an error, which consists on an incorrect state. Subsequently, an error can cause a failure, a deviation not only of the state but of the actual behavior with respect to the service specification.

The notion of dependability covers several meta-functional attributes of a computer system that relate to the quality of service (e.g., occurrence of failures) a

system delivers to its users during an interval of time [Kop11] [ALR01]. From those attributes we explain three that will be extensively used in this dissertation:

- *Availability*: it is a measure of the delivery of correct service with respect to the alternation of correct (*Mean Time To Failure* (MTTF)) and incorrect service (*Mean Time To Recover* (MTTR)). It is measured by the fraction of time that the system is ready to provide the service (Equation 2.1).

$$Availability = \frac{MTTF}{MTTF + MTTR} \tag{2.1}$$

- *Reliability*: the Reliability $R(t)$ of a system is the probability that a system will provide the specified service until time t, given that the system was operational at the beginning, i.e., $t = t_0$. When the failure rate (λ) of a system has an exponential distribution reliability at any mission time follows Equation 2.2. This reliability can be modeled within several configurations using mathematics [DS01a]. The standard measure for reliability is the *Failure in Time* (FIT) which provides the expected failures that a hardware component will suffer during 10^9 hours of operation (about 115000 years).

$$R(t) = e^{-\lambda t} \tag{2.2}$$

- *Safety*: it is the reliability regarding critical failure modes. In case of a detection of a fail safe system, it is possible to perform some corrective action, or to bring the system to a safe state, where the outputs of the local interfaces are set to a non dangerous value (e.g., the barriers are set down in a level crossing of a train).

2.4.1 Physical Faults in Semiconductors

There are several works on the taxonomy of faults [ALRL04] [ALR01]. In the scope of the physical faults experienced by integrated semiconductor devices, faults are basically classified as permanent or transient based on their persistence in the system [Con02].

Physical permanent faults are originated by underlying hardware irreversible damages [MSK+08] [GAM+02] (development imperfections, wear-out, etc.). For instance, cosmic rays can also cause permanent faults in few cases, i.e., with high ionizing doses, but they will be treated later as transient fault causes

due to the higher probability. The dependability means of the last decades have addressed permanent faults, but during the last years the research attention have switched to transient faults. Whereas permanent fault rates remain stable with technological improvements, the increasing sensitivity to voltage, frequency and energy variations due to transistor shrinking has resulted in higher transient fault rates [Con02]. Besides the well known *Electromagnetic Interferences* (EMIs) that remain in the system for limited periods (e.g., a maximal duration of 50 ms was tolerated in automotive applications [HT98]) and then disappear, new transient faults in the form of *Single Event Effects* (SEEs) have emerged.

These SEEs are spontaneous events of electronic systems caused when energized particles (cosmic neutrons, alpha particles, etc.) collide with integrated devices. Contrary to the aforementioned permanent faults and EMIs, SEEs do not affect nodes or integrated chips as a whole. The cross section of a single event is about hundreds of micrometers [CMFC+98] which is just a small fraction of current integrated chips that have silicon fabrics of some dozens of millimeters.

SEEs are classified based on the induced effect and severity as listed below:

- *Single Event Upset* (SEU): a bit-flip induced in a memory cell by a single energetic particle.
- *Multiple Bit Upset* (MBU): an event induced by a single energetic particle that causes multiple upsets (SEUs) during its passage through an electronic device. They are also called MCU (Multiple Cell Upset) when affecting bits of different memory words.
- *Single Event Transient* (SET): one or more voltage pulses (e.g., glitches) caused by a single event which propagate through the circuit.

The associated fault rate is expected to grow according to the International Technology Road-map for Semiconductors (ITRS) shown in Table 2.1. It must be also considered that the particle flux causing SEEs increases with altitude and is not equal at different longitudes and latitudes of the world.

Table 2.1: ITRS prediction for soft errors and MBUs

	2010	2013	2016	2019
Soft Error Rate (FIT/Mb)	1200	1250	1300	1350
Percentage of MBU	32%	64%	100%	100%

2.4.2 Fault Containment Regions (FCRs)

A *Fault Containment Region* (FCR) is a set of subsystems that share one or more common resources that one single fault may affect [Kop11]. To form a fault containment boundary around a collection of hardware elements, one must provide independent power and clock sources and additionally electrical isolation and spatial separation [LH94]. These requirements make it impractical to provide more than one FCR within a single semiconductor chip at a safety-critical rigor (at a probability of failure of 10^{-9} failures per hours).

If one distinguishes design and physical faults [Obe08], integrated architectures can provide FCRs for design faults. The spatial and temporal partitioning techniques [Rus99] for the components of an integrated architecture can provide design fault containment. For physical faults, the hardware approach can provide certain containment coverage by providing spatial separation of the partitions and multiple clock domains and pin-out (e.g., grounding) on the chip layout (e.g., for SEEs [CMFC+98]). These on-chip FCRs for physical faults work only at single chip failure probabilities (e.g., around 10^{-5} to 10^{-6} failures per hours [PMH98]).

Physical fault containment and design fault containment are orthogonal properties. Physical fault containment does not assure design fault containment and vice-versa. For instance, one may use two separated chip processors (two FCRs for physical faults) to implement a function but both can fail simultaneously due to a single design fault on the software. In the same way, a hypervisor can assure design fault containment for two independent operating systems within the same chip and a single physical fault can make both fail.

2.4.3 Fault Tolerance

Fault tolerance is a mechanism to deliver a correct service despite the occurrence of faults [Kop11]. The masking of a fault in order to hide it from the application, is always based on redundancy. One can distinguish temporal, information and hardware redundancies.

- *Temporal Redundancy*: it involves the repetition (rollback) of instructions, segments of jobs or entire jobs. The re-execution of these pieces of software must consider the duration of the fault and the handling of the state. For instance, temporal redundancy is not valid for permanent faults or transient faults of longer duration than the execution time.
- *Information Redundancy*: digital systems process, transmit and store data in the form of groups of bits and to avoid that any of these bits could

flip due to physical factors during processing, transmission or storage. Detection (*Error Detection Code* (EDC)) or correction (*Error Correcting Codes* (ECC)) codes are appended to the original binary bits by encoding part of the original information in order to detect or recover from those bit-flips.

- *Hardware Redundancy*: It is based on the replication of blocks (e.g., a component). These blocks perform the same job and faults can be detected (two replicas) or even tolerated (three or more replicas) by comparing the outputs of the replicas. Each of the replicated components must be an FCR in order to avoid common cause failures that would undermine the reliability increase, and they must be replica determinate to support exact voting [Pol95]. *Triple Modular Redundancy* (TMR) is one of the most used hardware redundancy mechanism where faults are masked based on a majority voting. The duplication of components is also an option when the replicated components show fail-silent behavior [Kop11].

2.4.4 Fault Injection

Fault injection can be defined as the artificial insertion of faults for the acceleration of their occurrence (normally) at the development phase of a system [Kop11]. This deliberate insertion of upsets (fault or errors) in computer systems is used for the evaluation of its behavior in the presence of faults or the validation of specific fault tolerance mechanisms [Ade03].

Injection techniques are classified by the mechanism used to insert the fault into hardware, software or simulator based fault injection.

- *Hardware Implemented Fault Injection* (HWIFI): is performed in a hardware model of the system (e.g., an early prototype of the final product) by inserting different physical perturbations, such as, electro-magnetic, thermal or radiation. One can identify two HWIFI categories: with contact (e.g., pin-level injection) and without contact (e.g., radiation beams).

- *Software Implemented Fault Injection* (SWIFI): it consists of reproducing at software level the errors that would have been produced upon faults occurring in the hardware or the software.

- *Simulator Based Fault Injection*: it allows to experimentally evaluate the dependability by using a model of the system before its final conception (e.g., design phase). Usually, this type of faults are performed by CAD tools at different abstraction layers (electrical level, gate level, RTL level, etc.) and using *Field Programmable Gate Array* (FPGA) emulation.

When emulating faults using FPGAs, two abstraction levels can be distinguished: (1) gate level (also called physical) and (2) *Register Transfer Level* (RTL) (also called logical).

- *At gate level*: the fault injection emulates a technology-dependent fault model. For instance, authors of [ACD+07][SATGM08] directly inject bit-flips (SEUs) at any position of the FPGA configuration memory. The results of such an injection campaign are only valid for a concrete FPGA technology, but an accurate measurement of the actual reliability is performed. One can correlate the SEU sensitivity of a specific FPGA technology (e.g., from vendor reliability data [Xil11]) with the fault injection results and give an approximate reliability in FIT of the tested FPGA design.
- *At RTL level*: one can raise the abstraction to a level where the end technology is still unknown. At this level, the previous configuration memory bit-flip fault model is not directly applicable. Therefore, one injects faults specifically in those flip-flops and registers of the FPGA that are also defined at the RTL level. Physical faults affecting other elements (e.g., configuration memory) are modeled indirectly through the failure rates and failure modes at the RTL level. For example, similarly to bit-flips in the configuration memory of an FPGA, transient pulses (e.g., SETs [ATM+07]) on the combinatory hardware of a VLSI chip can provoke register level changes (e.g., bit flips in registers or memory elements). One injects only those logical faults (i.e., bit-flip on actual RTL registers and memories) in the FPGA design. Although, the probability of a SET or a SEU to leak to RTL is unknown, one can compare fault tolerance mechanisms and give relative figures about which of them accomplishes its function better. In addition, the fault injection can already be performed earlier in the development process, because RTL models are available at an earlier development stage.

Fault injection at RTL using FPGAs has been mainly implemented using the simulation of modified HDL code. For instance, one can find fault simulation tools for system models designed in Verilog at RTL level addressing permanent (e.g, stuck-at) faults [MG96] or in VHDL addressing wider fault models (e.g., transient faults) [BGB+08]. Other approaches, like FT-UNSHADES [ATM+07], use other fault injection technologies, such as partial reconfiguration on Xilinx Virtex II FPGA technology to inject bit-flips in flip-flops and latches to emulate the effects of SETs at RTL level. The use of actual FPGAs for the emulation of faults significantly accelerates the injection compared to the use of simulation platforms. However, the above fault injection frameworks do not

target MPSoCs or integrated architectures. Aside from fault injection at RTL, there is only few work addressing NoCs, i.e., with a focus on on-chip routers and switches [FCCK06]. Other approaches include an additional wrapper to stimulate the IPs of the DUT at a higher abstraction level (i.e., the IEEE-1500 standard [IEE05]).

2.5 Cognitive Complexity

Integrated architectures must deal with the current complexity of embedded systems. Such architectures should not penalize the human comprehension, on the contrary, they should use representations and techniques fitting the human cognition and ease the work of integrated system engineers. In previous works, authors have identified three main simplification strategies tackling with this cognitive complexity that can be applied to embedded system architectures [Kop08b] [Per11]:

- *Abstraction*: it refers to the focusing on the relevant information to a particular purpose and ignoring the remaining information.
- *Separation of concerns*: the spatial decomposition of a problem (or a system) into smaller parts enables the isolated analysis of the parts.
- *Segmentation*: the temporal decomposition of a problem reduces the amount of parallel information to consider simultaneously.

Integrated architectures provide a powerful *abstraction* to cope with a chip consisting of more than a billion transistors running at more that 1GHz by only taking care of components and their interactions. Moreover, the partitioning into separate components fits the *separation of concerns* principle. The explicit notion of time into an integrated architecture permits the *segmentation* of sequential tasks and the easy understanding of component interaction through cognitively simple concepts, such as messages.

Introducing fault tolerance should not damage the aforementioned simplicity principles. In fact, the easy removal of redundancies is a claim for simplicity in many inter-disciplinary fields (e.g., cognitive linguistics [Cho11]). The use of component redundancy to implement fault tolerance enables the separation of concerns between fault tolerance and “normal” service. Therefore component replication could be preferred against more complicated ways of increasing the reliability that are merged with the application specific behavior of an embedded system (e.g., low level redundancy, state recovery strategies).

Specifically TMR is an example of an easily removable or transparent fault tolerance [BK00], and moreover, the number of interactions of a TMR subsystem is below the limit of 4 chunks of informations [Cow01] that an average human being can process at the same time due to the short-time memory.

"Beharra nola, izana hola",

Which needs you have, that way you are

Chapter 3

Analysis of the State-of-the-Art

This chapter analyzes integrated architectures and their suitability for embedded systems.

One of the challenges of the integrated architectures is reliability, for instance, the increasing rates of transient faults are more significant in highly integrated chips. The raise of mixed-criticality embedded systems, chips that share functionalities of different safety requirements, demands services enhancing on-chip fault-containment and fault tolerance.

In this section four software-hardware implementations of integrated architectures are surveyed. They are selected in order to cover a wide range of features. (1) XtratuM hypervisor, from the Universidad Politécnica de Valencia, is selected to analyze the current implementation of mixed-criticality architectures oriented to embedded and safety-critical systems market. (2) The Cell processor, an example of most commercial MPSoCs jointly developed by IBM, Sony and Toshiba, and (3) CoMPSoC MPSoC template, jointly implemented by the Eindhoven University of Technology and NXP semiconductors, besides being oriented to multimedia, this approaches show different aspects of MPSoCs (real-time, composability, etc.). Finally, (4) the TTSoC, created by the Vienna University of Technology, combines safety orientation and an MPSoC architecture with several key properties (e.g., fault and error containment, on-chip replication) that are detailed. Additionally, the on-chip replication proposed by the IEC-61508 international safety standard is analyzed as an additional contribution for integrated architectures.

3.1 XtratuM Hypervisor

XtratuM is a bare-metal and para-virtualized hypervisor designed for real-time embedded systems and oriented to safety-critical applications [CRM+09]

[MRCP10](Figure 3.1). XtratuM executes on top of an embedded processor and provides temporally and spatially partitioned virtual execution environments. The temporal partitioning is achieved by the use of a fixed cyclic scheduler to establish the execution time of each partition on the host processor. The spatial partitioning deals with the partition execution in processor user mode and the memory independence among the partitions. XtratuM distinguishes two types of environments: system and user partitions. System partitions are allowed to handle and monitor the state of other system and user partitions.

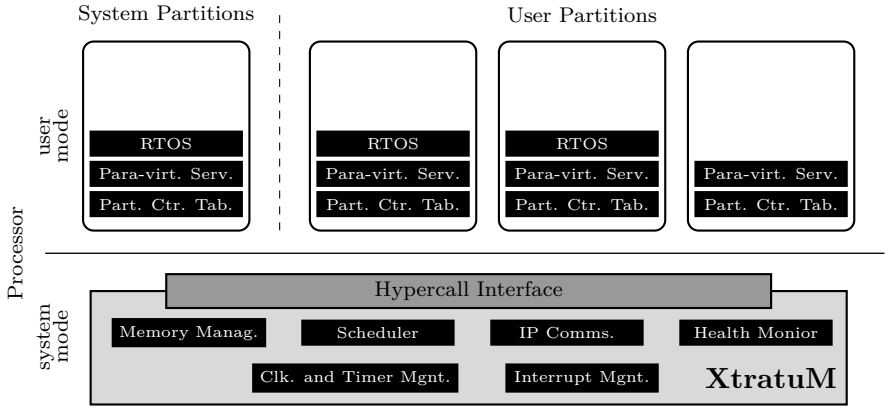


Figure 3.1: XtratuM architecture

The internal XtratuM architecture includes: memory management, scheduling, interrupt management, clock and timers management, interruption management, partition communication management and health monitoring. Moreover, like every para-virtualized hypervisor, conflicting instructions are replaced by explicit hypervisor instructions, namely the hypercalls. Xtratum claims to have deterministic and fast hypercalls. The resource allocation (partition duration, memories, peripherals, etc.) is done via an XML configuration file.

XtratuM supports x86 general purpose processors and the LEON (2 and 3) embedded processor based on the SPARC-v8 architectures. The supported operating systems include PartiKle and RTEMS.

3.1.1 Communications and Timeliness

The inter-partition communication implements the ARINC-653 avionic domain standard [Tok03]. The communication is based on shared memory sampling and queuing port communication model. The communication latency as in message based communication does not exist, but the minimal time from the send operation in one partition to the reception in another one is a function of the context time switch and the virtual partition execution window. The

partition context switch shows an average value of 110 microseconds and a maximum value of 116 microseconds (LEON3 version at 50 MHz [MRCP10]).

3.1.2 Fault Handling

Spatial separation among the partitions is achieved using a MMU (in the LEON3 version) that prevents any partition to write in any other partition's memory section. Fault management is the responsibility of the health monitor (HM) (also based in ARINC-653). It handles non-expected behaviors of the system, such as not considered design faults (e.g., the null return of a malloc allocation) namely HM_events. In case of HM_events, the health monitor can stop the faulty partition and log the event, or resume an alternate dormant partition.

3.2 Cell Multi-Processor

The cell multi-processor contains one PowerPC Element (PPE) microprocessor, 8 Synergistic Processor Elements (SPEs) and memory (MIC) and input-output interfaces (IOIFs) around a four channel ring shaped NoC (Figure 3.2) [KPP06]. Its initial target was the Play Station 3, but its capabilities makes it suitable for other visualization, signal processing or big workload task applications. The PPE runs the operating system and coordinates the SPEs. In fact, the goal of this multi-processor is to run a single application and parallelize it for best performance. The interconnect and DMA arbitration is optimized for average-case performance.

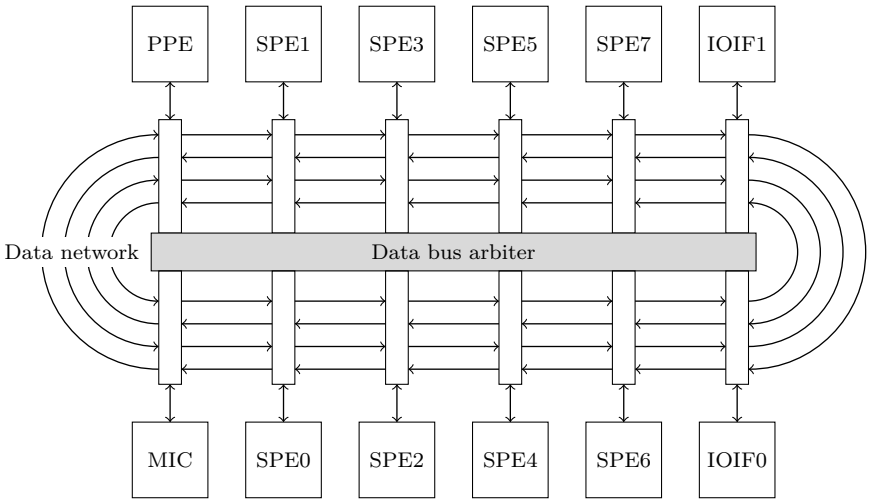


Figure 3.2: The Cell processor

3.2.1 Communications and Timeliness

Processing units cooperate through *Direct Memory Access* (DMA) coordinated by a central arbiter and an end-to-end command bus. The data arbiter implements round-robin bus arbitration.

The clock speed of the multi-processor, 3.2 GHz, allows a theoretical interconnect peak bandwidth of 204.8 GB/s for intra-chip communication. But, the command bus is severily limited in the number of bus requests it can concurrently process (one per bus cycle) and the coherent data rate is limited to 102.4 GB/s. This command bus limitation increases the packet latency by long sending and command phases [AP07]. Moreover, these latencies are not deterministic and vary depending on the load of the interconnection.

3.2.2 Fault Handling

The specification of the Cell processor [IBM07] shows that the architecture provides memory protection (e.g., segment fault, a mapping fault, or a protection violation) with a configurable range of interrupt actions. However, the works addressing fault handling using the Cell architecture do it at a software layer [TU08] and with a hardware supported flow controller. Nevertheless, the existing hardware primitives are not the optimal ones for fault handling: end-to-end commands ease the error propagation, there is not multicast for the NoC (e.g., to implemented TMR) and the central arbiter acts as a single point of failure.

3.3 CoMPSoC

CoMPSoC (Figure 3.3) [HGBH09] is a template MPSoC architecture that combines customizable processors and memory tiles around the Æthereal NoC. Originally designed for high computational performance demanded by consumer electronics, it emphasizes on performance guarantees, composability, verification and run-time reconfiguration for multiple application domains.

The processor and memory tiles are connected to the NoC by means of the NIs. The memory tiles include also a shell that connects the slave memory tile to the master (e.g., a processor tile) using a TDM arbitration to assure composability. The processor tiles are implemented using Silicon Hive VLIW processor cores.

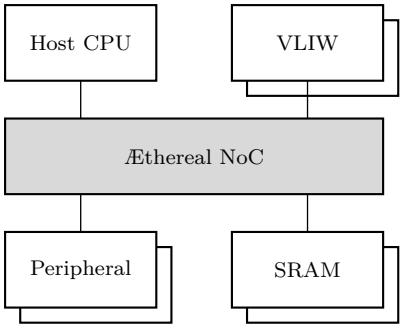


Figure 3.3: The CoMPSoC architecture

3.3.1 Communications and Timeliness

The Æthereal NoC [GH10] is composed by Network Interfaces (NIs), routers which are connected together by links. It supports guaranteed services (with bounded latency) which assures an amount of bandwidth for a given communication channel using Time Division Multiplexing (TDM) arbitration, and optional best effort services that take the remnant bandwidth. The type of routers changes for communication channels with different traffic types. The latency of the messages is equal to the addition of the duration of the guaranteed slot and the length of the path (measured in hops equal to a slot). There is also an additional flow control which complicates de calculation.

3.3.2 Fault Handling

A central host tile (e.g., an ARM processor) is responsible for all the administration of the processor tiles and the control registers of the communication infrastructure. Only this central host has the ability to configure the NIs and the memory arbiters. Hence, CoMPSoC claims to contain software errors occurring in the rest of the tiles, as they cannot propagate by faulty NIs or memory arbiter configurations. The configuration data coming from the central host is carried by the NoC using the concept of channel trees with the same isolation of the normal application traffic.

3.4 TTSoC

The Time-Triggered System-on-Chip architecture is a component based MPSoC platform [KOESH07]. It is the integrated execution environment for the TTA architecture. It distinguishes two subsystems: the *Trusted Subsystem* (TSS) and the application subsystem (Figure 3.4). The former consists of a

time-triggered NoC and other architectural cores that provide services to the application subsystem. The application subsystem integrates heterogeneous cores of diverse origin and criticalities.

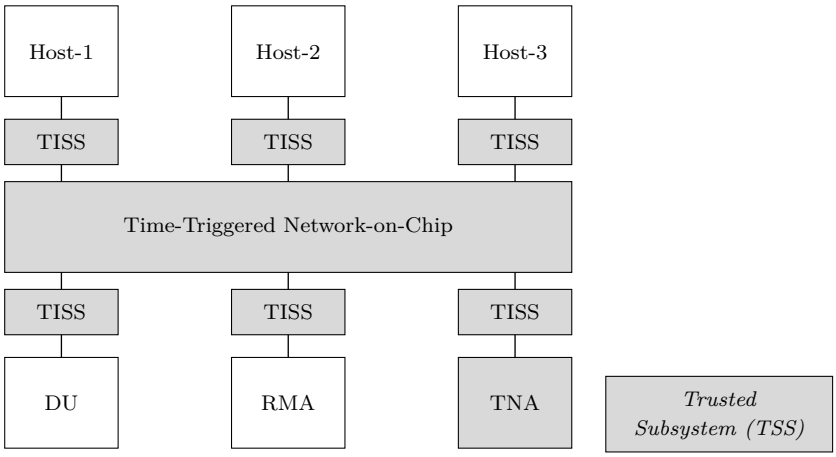


Figure 3.4: The TTSoC architecture. The gray area denotes the TSS

The TTSoC supports integrated resource management. This provides flexibility at the top level of the architecture which is interesting not only for many multimedia applications, but also for other domains where the application can benefit from a different resource allocation. Dedicated elements such as the *Trusted Resource Manager* (TRM) and the *Untrusted Resource Manager* (URM) process resource allocation requested by cores and reconfigure the SoC. Finally, diagnosis is part of the SoC by means of the Diagnostic Unit (DU). It monitors messages and stores diagnostic information for maintenance purposes using the TTSoC native multicast primitive.

The TSS of the TTSoC implements four main core services: time services, communication services, configuration services and execution services. The time services provide local clocks which are globally synchronized within the MPSoC. The communication service is message based and supports periodic and sporadic message exchange and multicast streaming primitive. The configuration service loads the software to the available hardware units. The execution control services are used to control the execution of a component.

3.4.1 Communications and Timeliness

The TTSoC follows a time-triggered schedule which configures off-line the transmission instants of the messages. It supports three message types: periodic exchange of messages, sporadic exchange of messages and primitive multicast streaming. The TTSoC provides temporal guarantees (bandwidth and latencies) to all classes of these classes of services.

The current implementation of the TTSoC architecture implemented on an FPGA [PK08] supports a throughput of 11.2 GB/s per encapsulated channel. The global time allows to minimize the end-to-end latencies through temporal alignment.

3.4.2 Fault Handling

Spatial and temporal partitioning for the cores is achieved using the TISSes as temporal firewalls [KN97] that prevent errors from propagating through faulty messages. Regarding fault tolerance the TTSoC supports on-chip TMR based on fault containment, replica determinism and predictability [OKS08]. The component replication of the TTSoC mainly addresses transient faults that affect specific locations of the chip (i.e., soft-errors).

3.5 IEC 61508 On-Chip Replication

The IEC-61508 international standard, which certifies electric, electronic and programmable electronic safety-related systems for multiple domains, introduced an on-chip replication section within its last edition [IEC09]. It defines the requirements, techniques and measures to increase the integrity level of a system by using on-chip replicated blocks. The implementation of these techniques can be found on FPGA [GHB10] and CPLD [GMSW09] even if they were originally intended for ASICs.

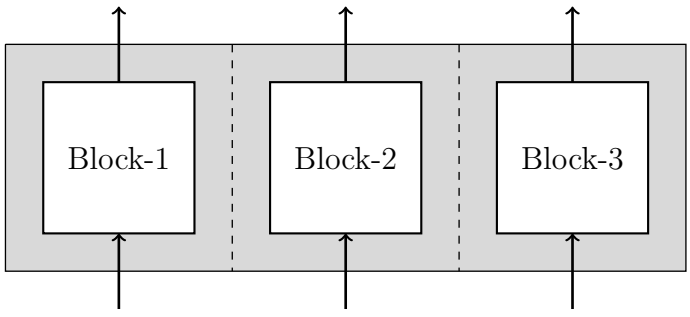


Figure 3.5: IEC 61508 on-chip block replication

This integration approach does not provide on-chip communication for the replicated component as a requirement for an increased fault containment (as shown in Figure 3.5) and the timeliness of the systems is out of its scope. Furthermore, there is no integration of multiple jobs, but just a single replicated job. Therefore, it cannot be considered an integrated architecture, because it does not provide system services further than fault-containment and there are

no jobs of different DASes. However, many of the requirements and techniques can be applied to existing integrated architectures.

3.5.1 Fault Handling

Due to the increasing use of Integrated Circuits (ICs) on current embedded systems the second edition of the IEC-61508 standard includes multiple references and two annexes to regulate their use into safety-related applications. For example, the new edition considers the increasing soft-error rates. However, the on-chip blocks of the IEC-61508 mainly address the isolation of permanent faults (shortcuts, hot-spots, etc.).

Annex-E, "Special architecture requirements for integrated circuits (ICs) with on-chip redundancy" [IEC09], supports that *"the highest safety integrity level that can be claimed for a safety function using an IC as described above is limited to SIL-3"*. Moreover, in this case up to 14 requirements must be accomplished in order to claim for SIL-3 integrity using on-chip redundancy. The most challenging requirements are listed bellow.

1. The effects of increasing temperature shall be considered to avoid common cause failures.

2. Separate physical blocks on substrate of the IC for each channel and each monitoring element shall be established.

3. The minimum distance between boundaries of different physical blocks shall be sufficient to avoid short circuit and crosstalk between these blocks.

4. The susceptibility of an IC with on-chip redundancy to common cause failures shall be estimated by determining a β-factor. The factor, starting with a basic 33%, must not exceed 25% after increasing and decreasing the Delta percentage contribution (in Table 3.1) of the implemented negative and positive techniques and measures regarding common-cause failure implication.

5. The minimum diagnostic coverage of each channel shall be at least 60%.

Annex-F, "Techniques and measures for ASICs: avoidance of systematic failures" [IEC09], shows the guidelines to follow during the development of safety-related embedded systems in order to use ICs. The most substantial measures are listed here:

Table 3.1: Techniques that increase and decrease the β-factor

Negative Measure (increases)	Delta	Positive Measure (decreases)	Delta
On-chip watchdog/monitoring	5-10	Diverse control in different channels	4
Internal connection between blocks	2-4	EMC testing	6
		Own power supply per block	5
		Structure that isolate and decouple	2-4
		Ground pins between pin-out	2
		Temperature sensor or high DC	2-9

- All tools, libraries and production procedures should be proven in use.
- The use of soft and hard cores is highly recommended.
- All activities and their results should be verified.
- Automation of the design implementation process should be used.

Finally, the IEC-61508 standard regarding ASIC development specifies that *"Only ICs with mature design and implementation processes should be used"*. The use of a generic integrated architecture for safety-critical applications should be preceded by its use in the non-safety-critical market in order to assure a maturity degree. Another option considers that chips using internal redundancy should be first combined with monolithic chips using off-chip redundancy until the required maturity is obtained.

3.6 Analysis

In this section, the integrated architectures are compared with respect to timeliness, design fault containment and physical fault handling.

3.6.1 Timeliness

For a variety of applications, ranging from safety-critical to multimedia applications passing through industrial control, real-time is a key requirement. The correctness of an embedded system does not only depend on the correct logical results of jobs but also on the bounded instants it produces these results and it exchanges this information with other jobs. Real-time must be considered in the execution of tasks but also in hardware implemented state machines, for instance, in the exchange of data among several components of an integrated architecture. Timeliness is present on the values transferred on a steer-by-wire application of a car, as it is on the encoding of a MPEG video or on managing

the IOs of a machine tool at precise instants. Furthermore, when addressing safety-critical applications, predictability becomes essential. It eases the detection and masking of faulty behaviors, as well as eases the certification process.

In the XtratuM hypervisor, event and communications latencies are a function of the context switch time and the number of partitions to switch before the handling partition executes. The Cell multiprocessor has end-to-end arbitration for the synchronization which increases the latencies. Moreover, the round-robin arbitration makes the throughput of a communication channel dependent of the overall communication traffic.

CoMPSoC and TTSoC architecture provide guaranteed throughput and latencies. CoMPSoC has a reserved bandwidth for each communication channel using a *Time Division Multiplexing* (TDM) communication access and bounded end-to-end latencies. In the TTSoC there is no arbitration due to the notion of global time which reduces the latencies for phase-aligned transactions (e.g., communication setup). Each message has a period and a phase that are established off-line which makes the architecture predictable. In contrast to the CoMPSoC architecture, the TTSoC is based on a sparse based time model, which provides determinism by handling simultaneity by design. Additionally, on the CoMPSoC not only the NoC is predictable, but also the processor and memory tiles.

3.6.2 On-Chip Design Fault Containment

An integrated architecture could host jobs of different criticality and suppliers. Partitioning among the jobs of the integrated architecture must be provided to assure composability and fault containment for design faults.

XtratuM has spatial and temporal partitioning which prevents design faults to affect several partitions at the same time. But, as the same hardware processor hosts all the partitions, it does not provide containment for physical faults. The cell multiprocessor has hardware flow control for temporal partitioning but it does not have any architectural means for spatial partitioning.

The CoMPSoC and the TTSoC architectures provide temporal and spatial partitioning by predictable communication channels. The communication schedule is stored in the network interfaces. This schedule can be only modified by a central resource manager (the central host and the TRM). Anyway, whereas the TTSoC minimizes the size of the trusted manager, CoMPSoC suggests a big embedded processor core to implement the central host (e.g., an ARM). The small size of the agent responsible for the resource management would reduce the probability of transient fault and the low complexity of the agent would ease the certification of the resource management service. In fact, CoMPSoC

focusses on composability, the mechanisms for the independent application design, verification and deployment, more that for fault tolerance.

3.6.3 On-Chip Physical Fault Containment and Fault Tolerance

Fault tolerance mechanisms need redundancy, of time, information or hardware, to provide a correct service in the presence of faults. The redundant artifacts should be fault containment regions in such a way that a single fault cannot affect multiple replicas at once, then causing a common failure.

XtratuM considers the use of a dormant partition to take over the functions of the faulty partitions. This approach only works for the containment of design faults due to the lack of fault containment for physical faults. Each of the partitions can contain diverse job implementations which provides design fault containment. The work using the Cell multiprocessor for fault tolerance implements this mechanisms at software level [TU08]. In fact, there are no physical fault containment or fault tolerance means on the Cell architecture. Moreover, the end-to-end arbitration is not desirable because it eases error propagation and the central arbiter constitutes a single point of failure. Anyway, the cell processor is presumably about 10000 times faster than the XtratuM approach due to the processor frequency and its flow controller could guarantee these communication.

The TTSoC supports on-chip TMR of application components. Fault containment, replica determinism and predictability are required to provide this on-chip replication that could be also theoretically implemented on the CoMPSoC chip. However, the trusted subsystem of the architecture shows a single point of failure [OKS08] and additional solutions should be taken in this subsystem to increase physical fault containment and reliability.

The patterns described in the IEC-61508 can be partially implemented on MPSoC architectures to increase the physical fault containment among the components. Another option to apply IEC-61508 in an integrated architecture is to implement a complete MPSoC or Hypervisor in each isolated block as shown in Figure 3.6.

3.7 Conclusion

The comparison (Table 3.2) among software and hardware implemented integrated architectures shows that the MPSoC approach with message based NoC

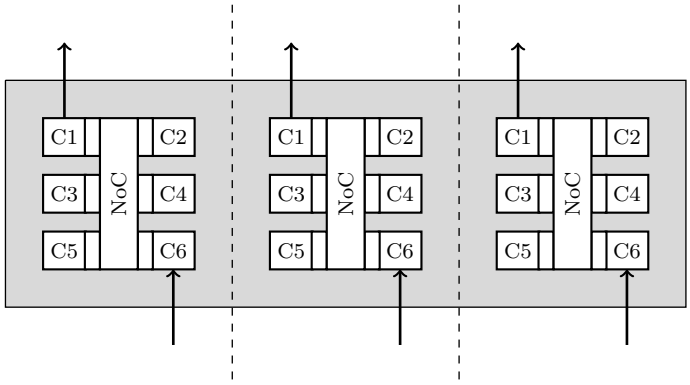


Figure 3.6: Applying IEC 61508 boundaries for integrated architecture

provides superior features regarding timeliness, fault containment and fault tolerance. If the analyzed architectures are mapped on a graph upon the axes of design and physical fault containment (Figure 3.7), one can notice that there is room for improvement on physical fault containment in order to approach the level of formerly federated (e.g., ECUs) architectures. From left to right, one can find approaches with high physical fault containment without design (software) fault containment (e.g., IEC-61508), MPSoC approaches that do not consider design fault containment and MPSoC approaches that provide mixed-criticality through a trusted NoC that avoids the propagation of design errors. The gap between a federated node and an MPSoC with a predictable communication infrastructure is explained by the use of independent chips for each component in the federated approach. Safety-critical physical fault containment requires components with independent chips for fault tolerance based on replication. The hardware flow controller of the cell processor enables temporal partitioning which can make the Cell processor valid for mixed-critically if implemented. Finally, the XtratuM hypervisor is directed to mixed-criticality by design, but as the whole set of partitions are executed in the same processor the architecture does not provide physical fault containment.

The notion of a global time of the partitions of an MPSoC guarantees, for periodic communication services, shorter latencies than end-to-end arbitration. Moreover, the explicit timing of the NoCs enhances fault containment against non-timely messages.

The handling of faults in integrated architectures, specially for physical faults, is still an active field of research where few implementations take reliability and fault tolerance in consideration (e.g., the TTSoC). Besides on-chip replication of application components, the fault tolerance of the system components constituting the integrated architecture is an issue that has not been addressed. The techniques recommended by the IEC-61508 standard for on-chip replication can be partially adopted to work in this direction.

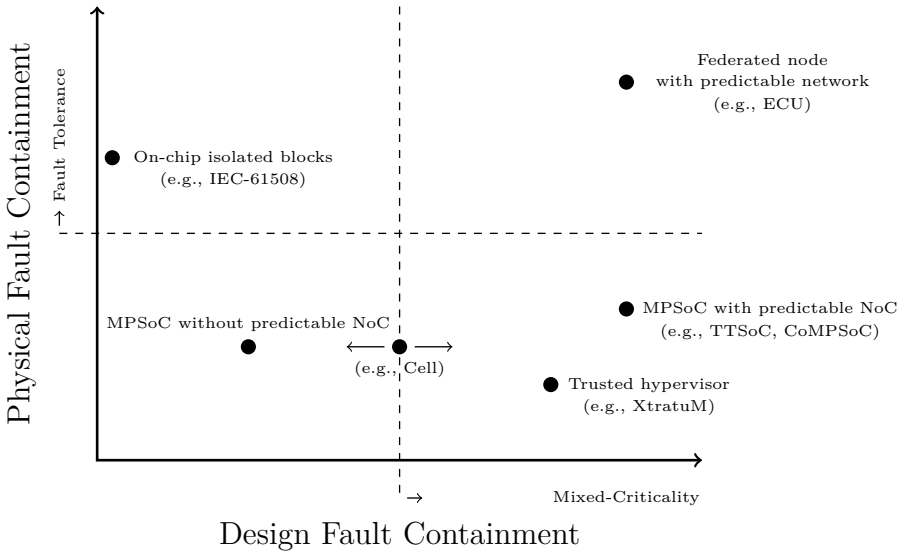


Figure 3.7: Solutions mapped on physical and design fault containment axes

Table 3.2: Comparison of integrated architecture implementation features

	XtratuM	**Cell processor**	**CoMPSoC**	**TTSoC**	**IEC 61508**
Partitions	Virtual processors	Processor elements	Processor and memory tiles	Heterogeneous components	Isolated blocks
Communication	ARINC-653 shared memories	DMA upon NoC	Shared memories upon NoC	Messages upon NoC	None
Arbitration	None	End-to-end	TDM	Time-triggered periodic control	None
Services	Fixed cyclic-schedule	Best performance arbitration	Mixed criticality	Time-triggered services (periodic and sporadic)	None
Design Fault Cont.	Memory (e.g., MMU) and Time separation	None	Separated memories and temporal firewalls on the NIs		None
Physical Fault Cont.	None	Separated processors			SIL3 fault containment
Fault tolerance	Dormant partition	None	None	On-chip TMR	On-chip replication and diagnosis

"Errota ongi da dabillen bitartiño,
ez geldirik dagolikan",
The mill is fine while working,
not when it remains stopped

Chapter 4

The 4TSoC

The *Transient Tolerant Time-Triggered System-on-Chip* (4TSoC) (pronounced /fɔːt - sɒk/, as "*fort-soc*") is the fault-tolerant implementation of the *Time-Triggered System-on-Chip* (TTSoC) architecture. It is an MPSoC architecture that interconnects multiple, possibly heterogeneous components from diverse origin and criticalities. The 4TSoC introduces a transient tolerant TSS, which offers core services to the application components (e.g., global time, message transportation, resource management, etc.) and ensures that a fault within a component cannot disrupt other components through spatial and temporal partitioning. In addition, it can operate despite the occurrence of transient SET faults within the fault hypothesis. Figure 4.1 shows the 4TSoC architecture supporting fault-tolerant core services (e.g., dual TISS) and a new naming convention compared to the TTSoC shown in Figure 3.4 (Chapter 3).

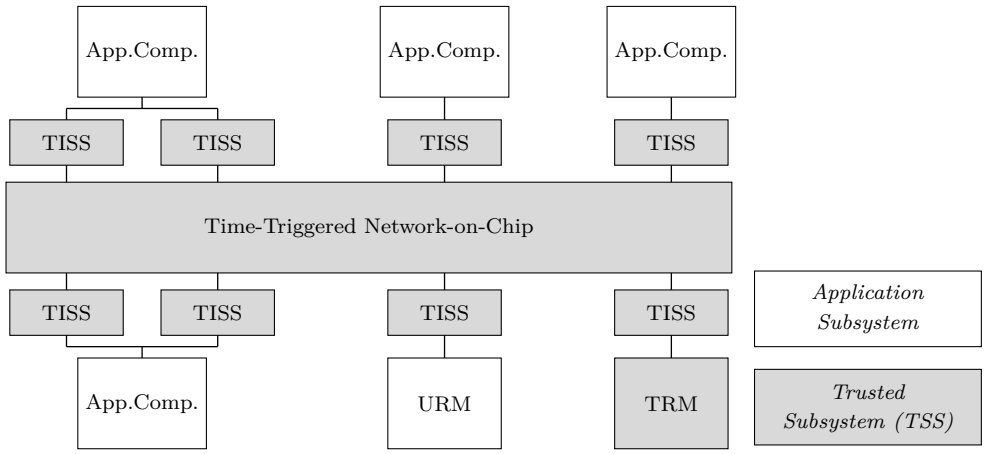


Figure 4.1: The 4TSoC architecture

Whereas the TTSoC supports fault tolerance for the application-specific subsystem by on-chip TMR of components [OKS08], the 4TSoC also supports fault

tolerance for the TSS. In fact, the TSS represents a single point of failure for the whole MPSoC. Although, the probability of such a failure, calculated by the consumed chip area, was estimated small, the simulation and fault injection results show that the TSS reliability plays a dominant role on the overall MPSoC reliability [OKS08]. The goal of the 4TSoC architecture is to increase the reliability of single chips that may later be part of a distributed safety-critical system. In fact, safety-critical levels of integrity (futher than SIL3 [IEC09]) cannot be achieved within a single chip.

This chapter describes the 4TSoC architecture and provides the fault hypothesis for the new core services and introduces the fault masking mechanisms. The details of the 4TSoC FPGA implementation patterns are also given.

4.1 Description

The 4TSoC distinguishes between the application-specific subsystem and the TSS. The first subsystem consist of application components and optional system components implemented according to the application specification. The second constitute the system architecture and provide the core services to the application-specific subsystem.

4.1.1 Application-Specific Subsystem

The components of the application-specific subsystem result from mapping the application jobs to the hardware partitions offered by the 4TSoC architecture. These components are potentially heterogeneous, they can execute software jobs through processor cores or the jobs can be directly implemented in hardware on silicon fabric (e.g., FPGA) by means of dedicated cores.

The application-specific subsystem supports mixed-criticality components. Safety-critical and non safety-critical application components can share the 4TSoC thanks to the spatial and temporal partitioning offered by TSS. Therefore, the TSS must be certified to the same integrity level of the most safety-critical application component.

Within this subsystem the 4TSoC distinguishes between application-specific components and system components. System components are shared among applications, but they are not mandatory as the elements of the trusted subsystem. An example of a system component is the URM for the resource management of the MPSoC [Hub08].

4.1.2 Trusted Subsystem

The TSS provides the core services and consists of three architectural elements, the *Time-Triggered Network-on-Chip* (TTNoC), the *Trusted Interface Subsystems* (TISSs) and the TRM. These three elements are trusted because the rest of the components relies upon them.

- *The Time-Triggered Network-on-Chip (TTNoC)*: the TTNoC supports message-exchange based communication with determinism and inherent fault isolation. It is formed by fragment switches that can be connected south, north, west or east through lanes to other fragment switches and TISSes. This exchange of messages is performed at predefined points in time according to a time-triggered communication schedule, which avoids the need for end-to-end arbitration. Therefore, the communication instants and message latencies are known a priori. It supports bus [Eng07], mesh [KOESH07] or ring [Sch07] implementations.
- *The Trusted Interface Subsystem (TISS)*: the TISS provides the interface between the TTNoC and each of the application components and acts as a guardian of the respective component. The TISS stores the knowledge about the permitted temporal behavior of the components and prevent temporal interferences. This knowledge can only be modified by the TRM and not by the component itself in order to preserve fault containment in the presence of a faulty component. The TISSes act as temporal firewalls [KN97].
- *The Trusted Resource Manager (TRM)*: the TRM besides configuring each TISS, verifies that any new communication schedule does not corrupt the static schedule of safety-critical components. The reliability of the TRM is based on its simplicity and rigorous design (including formal verification), as well as its small footprint that reduces the occurrence of physical faults [OKS08].

4.2 Fault Hypothesis

The fault hypothesis is a mandatory analysis of a system as a prerequisite for the design and validation of fault tolerance mechanisms. This section states what types of faults must be tolerated by the fault tolerance mechanism, the definition of FCRs and the failure types.

Beside other kinds of permanent and transient faults, the scope of the 4TSoC architecture focuses on SET faults of integrated chips. Whereas, other transient

faults (e.g., SEUs) have been extensively researched, SETs are still a matter of research for all integrated chip technologies. In fact, even rad-tolerant devices exhibit susceptibility to SET faults [BWL+06].

4.2.1 Fault Containment Regions

The 4TSoC distinguishes FCRs based on design and physical faults.

Each component of the architecture is a fault containment region for design faults. Strategies like the absence of a common piece of a design in multiple components and the potential diversification of each component's software and hardware due to different providers can avoid common design faults. In the time domain the TISS, which a priori knows the global times of message transmissions and receptions, prevents the component from sending untimely messages. In value domain, since the value failures are tightly related to the application, their detection relies on the application level (e.g. replication of application components).

Considering the multi-core nature of the 4TSoC architecture, each application component of the 4TSoC has a better fault containment coverage (e.g., physical separation, fewer shared resources) for physical faults than software-based integrated architecture approaches (e.g., hypervisors). Therefore, each components in the 4TSoC architecture is an FCR for physical faults at single chip containment coverage (e.g., the probability of a correlated failure for a chip is around 10^{-5} to 10^{-6} failures per hour [PMH98]). Anyway, fault containment for physical faults is not achievable within a single chip at a safety-critical rigor (at a probability for correlated failures of 10^{-9} per hour), shared resources (e.g., substrate, manufacture process) and spatial proximity make it statistically unfeasible.

The TSS itself is considered an FCR for design faults, but in a deeper refinement each elements of the TSS can be considered an independent on-chip FCR for single event faults. In fact, the technology mapping (e.g., FPGA type, ASIC) of the 4TSoC is of utmost importance for the validity of these on-chip FCRs for physical faults. In Section 4.5 (Page 55) these technology constraints are further discussed.

4.2.2 Failure Modes and Rates Assumptions

The 4TSoC model distinguishes two types of failures at NoC message level depending on the message arrival to the TISS (simplified from the DSoS model [Jon02][Cri91]):

- *Message omission failure*: no message has arrived at the network interface within the specified receive window.
- *Value-incorrect message*: a message has arrived at the network interface within the specified time windows but the data is corrupted.

Regarding failure rates, the occurrence of transient has been shown to be 100 to 1000 times more likely than permanent faults in integrated architectures [OP06]. For instance, a raw 65nm FPGA (smallest Virtex-5 LX30) has a FIT (failures per 10^9 hours of operation) of 1192.8 for transient faults and only 12 for permanent faults per device [Xil11]. However, the leakage of a transient fault to a message level failure must overcome several technology barriers. For instance, in the case of SETs, logical, electrical and temporal masking can prevent a SET from being captured by a sequential element [SKK+02]. And then, the probability of an erroneous flip-flop to propagate to message level provides another barrier to propagation. Anyway, the effect of these technological masking effects on flip-flops is smaller with technology shrinking and it is believed that SETs in combinatory logic and SEUs in memory elements are reaching the same failure-rate [SKK+02]. Finally, the soft-error failure rate varies with altitude as it is proportional to the neutron flux in the atmosphere. Experimental data shows that the neutron flux increases an average of 2.2 times every 1000 meters (1.3 times every 1000 feets) [ST11].

4.3 4T Core Services

The 4TSoC core services are based on the basic services of the GENESYS MP-SoC [OKP10]. These four core services are important for many cross-domains application and they are extended with a fault tolerance orientation.

4.3.1 4T Time Services

In the 4TSoC each component and the NoC can work on their own (local) clock domain. In the time-triggered NoC multiple clock domains can co-exist upon a common time notion. Common time is the base for the provision of a deterministic communication infrastructure for distributed components. For instance, the time-triggered NoC requires a common notion of time among all the components, and this reduces end-to-end latencies through temporal alignment of components for the communication activities at the NoC.

When establishing a global time in an on-chip distributed system, there are two approaches depending on the means to distribute this global notion of

time: (1) It can be obtained through the message arrival instants using clock synchronization algorithms or (2) by routing a physical clock line with the global time macrotick frequency to all the components.

Clock Synchronization Approach

Clock synchronization for the establishment of a global time among distributed computers has been extensively used for off-chip networks. In time-triggered systems, this clock synchronization is processed by all the components of the distributed systems by comparing the actual arrival instant against the expected arrival time. There are many techniques that implement this clock synchronization.

The Central Master algorithm is one option [Kop11]. The central master node periodically sends synchronization messages with the value of its internal counter to the rest of participants. The slaves then adjust the local clock counter according to the difference between the master's time, contained in the synchronization message, and the time-stamp of message arrival is recorded by the slave, corrected by the known latency of the message transport.

Fault Tolerant Synchronization algorithms [PS03] [RSB90] can protect the global time from byzantine faults. These algorithms consists of a control system with a feedback loop in order to regulate the global clock with three phases that differ with the application and the specific algorithm. (1) Monitoring the exchange of messages in the system, each node acquires knowledge about the state of the global time counters in the rest of the nodes. (2) The nodes locally analyze these messages and compute a convergence function that provides the correction for the local clock. And (3) the local clock is adjusted.

Clock synchronization can be processed in different clock domains at the TISSes of the components that cohabit the same NoC based MPSoC. Nevertheless, in contrast to the buses used in distributed systems, not all the messages of the system arrive at all the components. A significant number of messages must arrive at all the components of the MPSoC in order to establish a consistent global time.

Global Clock Line

Even if a single high frequency clock cannot be provided in current integrated chips due to technological limits (e.g., clock skew), a slower global time clock can be physically routed though all the components of an MPSoC. In this way, components can be mapped into multiple clock domains that are synchronized to that global time. In contrast to the clock synchronization approach where

the reasonableness condition was obtained by approximating the global time macrotick, in this approach this requirement is fulfilled off-line by temporal constraints at chip synthesis phase. Alternatively, a faster centralized synchronous clock can be implemented with combinations of techniques such as waterfall clock distribution and synchronous latency insensitive design [ES04].

Regarding fault tolerance, this physical global line represents a single point-of failure for the whole chip. Additional clock distribution lines or topologies should be provided in order to harden this shared resource.

4.3.2 4T Communication Services

The fault tolerance implementation of the TTSoC also provides a message-passing based communication service among components with: periodic messages, sporadic messages and streaming, all of them with multi-cast support. The innovation of the 4TSoC architecture is the support of replicated channels.

First of all, the message-passing paradigm implies several superior properties:

- *Explicit timing*: the timing of message exchanges is explicitly defined (e.g., period and phase) and this avoids the use of separate synchronization such as in shared memory communication.
- *Fault Containment*: the uni-directional nature of messages (one sender, one or multiple receivers), provides the identification of the sender (e.g., if it is faulty). The explicit timing provides the basis for the containment of temporal faults in the form of messages sent outside the specification.
- *Universality*: message-passing is an universal and basic model. It permits the implementation of other communication paradigms (e.g., shared memory) on top of it. For instance, on an Internet mail-list, the users can create a shared table (cf. virtual shared memory) by updating the contents and broadcasting them.

As in the TTSoC, the 4TSoC provides three basic message primitives:

- *Periodic Exchange of Messages*: the architecture supports the exchange of state messages within a predefined period and phase from one sending component to one or more receiving components.
- *Sporadic Exchange of Messages*: the 4TSoC can send event information at arbitrary instants only constrained by a minimum interarrival time of events. Otherwise, out of these pre-reserved slots, collision could occur.

- *Primitive Multi-cast Streaming*: this primitive serves the transmission of a sequence of variable size data elements (e.g., multimedia streaming) with corresponding temporal properties.

The 4TSoC supports replicated channels through the same or through replicated NoCs. The TISS has been found specially vulnerable to transients compared to the rest of the components of the TTSoC. Therefore, it is useful to use multiple TISSes for each of the application components and compare their outputs in order to provide a more reliabile communication service. The 4TSoC implementation prefers the use of dual TISSes upon a fail-silent hypothesis, because it reduces the amount of hardware resources compared to other approaches (e.g., TMR with majority voting). These duplicated TISSes can be connected even to the same NoC, or to replicated NoCs. Despite the goal of these duplicated TISS architectures for the increase of reliability, they can also be used to duplicated the throughput of a given component. The 4TSoC also support the option to attach ECCs to the message in order to detect and correct possible bit-flips.

Network Interface Replication

This approach provides two network interfaces or TISSes to every component of the MPSoC that requires an enhanced reliable communication. The whole TISS is replicated. Each of them has its own input and output buffers and the component needs to read and write twice every message at their respective memory addresses. The replicas connect to different lanes of the NoC (Figure 4.2) and they can need different routing information according to the route of the messages within the NoC.

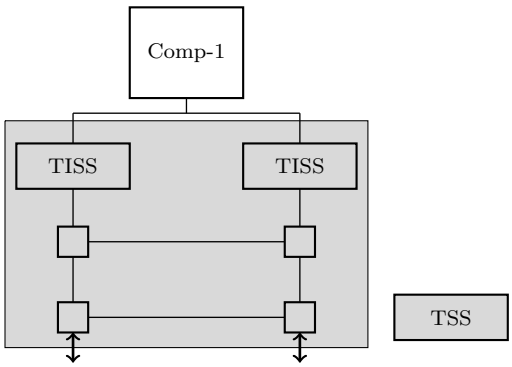


Figure 4.2: TISS Replication for 4TSoC

NoC Replication

Replicated NoCs can also be provided to the components of the 4TSoC in addition to TISS replication. The whole NoC replication can be understood as a special case of the NI replication mechanism. In this case, there is no communication between the switches of both NoCs and the component is the unique connection among them. Figure 4.3 shows mesh-based symmetric NoCs, but asymmetric NoCs are equally valid. Rings are also an alternative layout to the mesh formation in the figure, e.g., an internal and external ring to the cores can avoid the crossing of lanes increasing the spatial separation of the replicated NoCs when the number of cores scales.

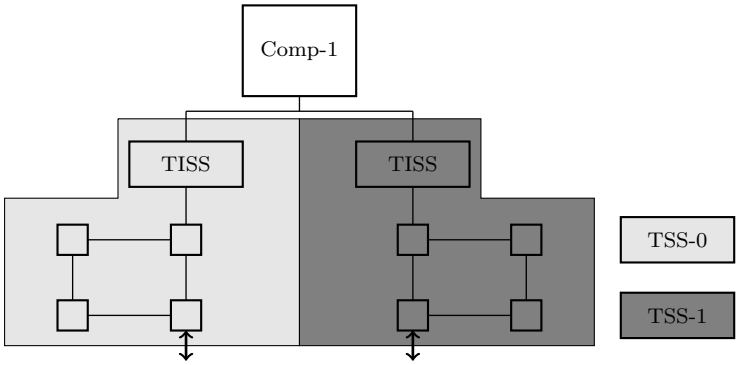


Figure 4.3: NoC Replication for 4TSoC

Message-Level Error Correcting Codes (ECCs)

An ECC can be computed upon a data-flit before NoC transmission and attached to the flit of the 4TSoC architecture. It permits at reception the detection and even correction of possible errors (bit-flips). For instance, a code with a hamming distance of 3 can detect and correct any single bit-flip on the data-stream (including the ECC code itself). The 4TSoC architecture (Figure 4.4) computes ECCs twice, first in the source core from the original data-bits to generate them, and then in the destination core from the received data-bits. A comparison between the received and locally computed ECC bits denotes the position of the inverted bit (if any).

4.3.3 4T Configuration Services

The configuration services of the 4TSoC implementation provide the means to load the software jobs to the partitions implemented by processor cores. The configuration of purely hardware-implemented jobs (e.g., state-machine

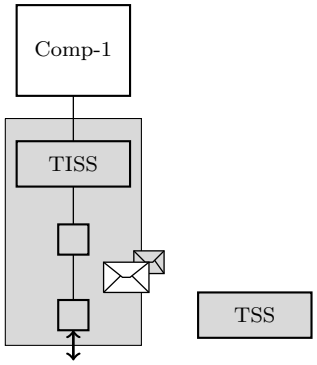


Figure 4.4: Error Correcting Codes for 4TSoC

implementations) is out of the scope of this service. The configuration services are needed for a dynamic reconfiguration of the resources (e.g. reassigning a software-implemented job to another hardware partition). In mixed-criticality systems where safety-critical and non safety-critical components cohabit the same integrated chip, the safety-critical part is often restricted by safety standards (e.g., IEC 61508, DO-178B) to static configurations.

Hence, configuration services must be established, which do not preclude certification of safety-critical components, and permit to build more complex configuration services for non safety-critical components upon these basic services. The 4TSoC implementation identifies three core services:

- *The boot service* : this primitive assigns a software job to a hardware partition of the 4TSoC.
- *The identification service*: it provides a single identifier to each partition within the MPSoC.
- *The inter-component channel configurator*: it supports the allocation, modification and removal of communication channels.

There are two dedicated system components to provide subsequently the generation of resource allocations and the verification/adoption of these resource allocations. As safety-critical components require support for static resource guarantees, the TRM verifies that any new allocation of the non-safety critical subsystem has no adverse effects on the behavior of the safety-critical component. The flexibility demands from non safety-critical components are supported by the URM that computes these resource allocations that later can be acepted or rejected by the TRM. The current research on flexible and trusted resource management on avionic domain is an application example for this technology [And08].

Thanks to the splitting of trusted and untrusted resource manager agents, the complexity and the size of the TRM is reduced, thus lowering the probability of design and physical faults that could affect this critical part. The separation also eases the certification process which is only required for the TRM.

The 4TSoC presents fault-tolerant versions of the configuration services using hardware and temporal replication or the merging of the TRM notion into the TISSes of the architecture.

Replicated Resource Managers

An option to harden the configuration service against physical faults is to replicate the untrusted and trusted resource managers. For instance, triple modular redundancy can be applied (Figure 4.5). In such a configuration, the schedule generation is generated at the same time by three URMs and verified independently by the three TRMs. Finally, each component votes on the received information locally.

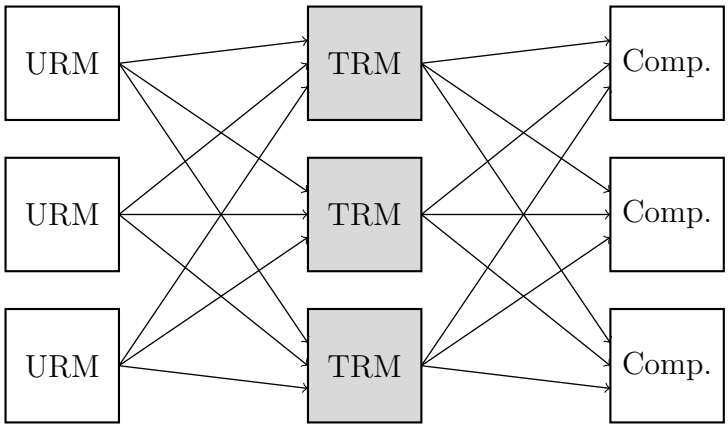


Figure 4.5: TMRed configuration services

Re-execution Based Resource Managers

The hardware overhead introduced by the replication of the resource managers can be avoided by a temporal redundancy approach where the new schedule computation and its verification is re-executed several times in the URM and TRM respectively. To apply this approach both resource managers, the URM and the TRM, should have a ground state that is visited after every execution. This requirement is very restrictive and application-dependent. This approach is valid to mask transient faults affecting the resource manager state, that can be recovered by returning to the ground state (e.g., a reset). Hence, the

transient faults affecting the technological layer (e.g., the configuration memory of a SRAM FPGA) are not tolerated, and the URM and the TRM should be implemented in rad-tolerant technology (e.g., ASIC or specific FPGA).

Distributed TRM

The functionality of the TRM can be merged into the TISS in the cases that the safety-critical part of the MPSoC does not change at all during the mission time of the system due to application requirements. In such a static configuration, the memory contents of the TISSes hosting the message routing and instants can be hard wired in order to intrinsically avoid the possibility of non-safety critical components to interfere with safety critical ones. For instance, the non-safety critical traffic is not allowed by design during the communication time of safety critical components. This approach does not support the change to pre-compiled and certified schedules during mission time, which can be a limitation for some kind of applications.

4.3.4 4T Execution Services

These services control the execution of a component through its TII interface. The basic execution control considers three commands: execution request, termination request and reset request. These commands are computed by the *Local Resource Manager* (LRM) assumed to be implemented in every component.

The trust on the messages carrying the execution services (as well as the configuration messages) can be provided by an error detecting code and a time stamp only computable at the origin. Hence, fragment switches of the TTNoC are not able to modify these features.

Fault Tolerance on Execution Services

This execution services can be demanded by a system component (e.g., the RCU [OKS08]) that can be replicated in hardware or time domains and the TRM should agree on the execution of the services upon the possible fault-tolerant architectures for the resource manager previously introduced.

4.4 4TSoC Fault Tolerance Model

The 4TSoC architecture includes fault tolerance mechanisms addressing transient faults at system level by using on-chip replication of components and ele-

ments of the TSS. The physical fault removal strategies are not a choice for soft-error transients: shielding would require approximately 3 meters of concrete and technology solutions such as hardened cells and *Silicon On Insulator* (SOI) introduce a big penalty or are not feasible on current technology sizes [MER05]. Other approaches tried to overcome the impact of transients faults by the application of low-level Error Correcting Codes (ECCs) [PKCC06], low-level redundancy [Gai06] [Xil06] or circuit level hardening [MSZ+05]. Component level redundancy provides superior time and energy efficiency, the ability to use standard libraries, higher resilience against spatial proximity faults, foundation for design diversity and heterogeneity, simplicity and early validation.

- *Reduced voting and recovery hardware overhead*: when one pursues fault tolerance, the use of component level redundancy and recovery services, such as voting or state recovery, are implemented once for all the cores reducing the overhead that such approaches require at lower abstraction layers (e.g., at flip-flop level).
- *Ability to use standard libraries*: the modification of standard and proven core libraries is not needed.
- *Resilience against spatial proximity*: this provides superior resilience against correlated faults (i.e., MBUs) and the components can be separated without proximity-timing constraints.
- *Diversity and heterogeneity*: replicated components can be implemented in different technologies or using software implemented by several programming teams.
- *Simplicity*: component replication enables the easy removal of the redundancy for a better understanding of the system.
- *Early validation*: hardening at component level permits the reliability assessment of the architecture and the design space exploration of different solutions at an earlier development stage.

4.4.1 On-chip TMR

The replication of cores (Figure 4.6) increases the chip reliability in the presence of transient faults. Even though the probability (cross-section) of a transient fault to affect more than one replica at the same time is small, the single point of failure at the trusted element could undermine the overall reliability of the SoC. A voting system on the end-component provides a majority based decision about the correctness of the messages.

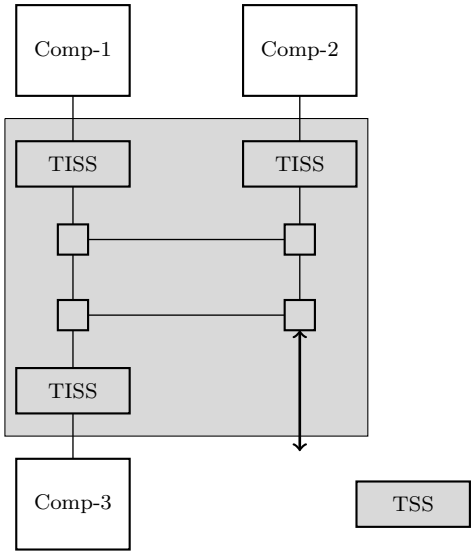


Figure 4.6: Application component TMR

4.4.2 On-chip TMR Upon Replicated Channels

The combination of the replication of TISSes and application components significantly decreases the occurrence of failures due to single faults (Figure 4.7). Upon a dual channel and fail silent communication network, this approach triplicates application components. A dual channel configuration with fail silent behavior carries less overhead than other replication approaches (e.g., TMR). While it is possible to have a fail silent TISS, an application component itself cannot be considered fail silent. Therefore, TMR is necessary to avoid application component failures.

4.4.3 Recovery upon TMR

On-chip TMR provides a limited reliability for long mission-times or when the use-case scenario is fault-prone [DS01b]. Therefore, if the behavior of one replica deviates from the other two replicas (a failure occurs), one can perform a repairing action (e.g., a restart-request message). After the restart, the replica must be able to recover from transient faults because this failure-mode does not damage the hardware. The recovery process goes through obtaining a correct state from one of the other replicas. This approach has been previously presented for multiprocessor systems [GSVP03].

The 4TSoC approach considers a restart only for application components and this restart is commanded by a system component part of the TSS (i.e., the RCU [OKS08]). The repair-rate and the common-cause mode failures can undermine the reliability increase of the recovery approach.

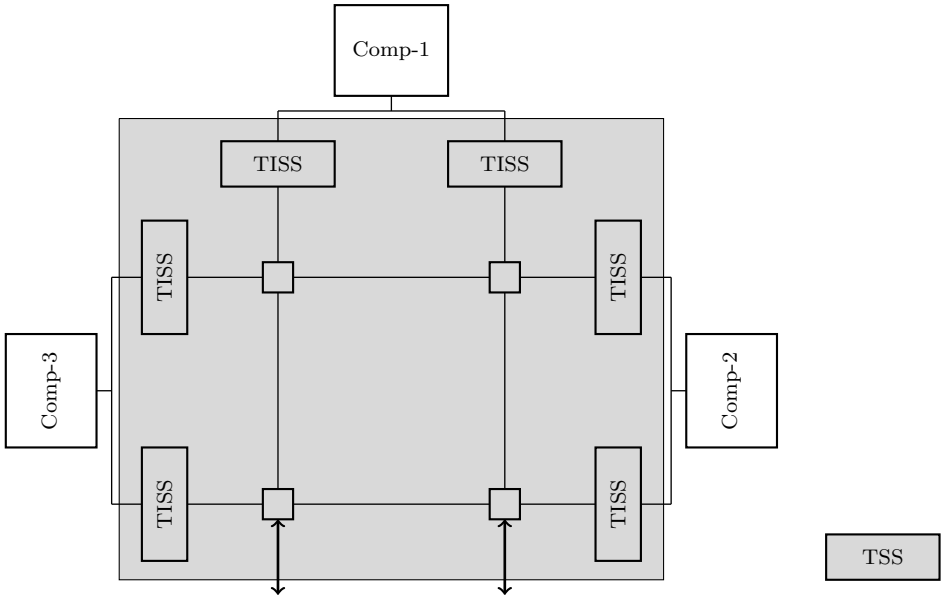


Figure 4.7: TMR configuration with two TISSes

The use of recovery is not currently recommended by safety-critical certification standards because it introduces uncertainty to the system. In fact, the validation of recovery mechanisms is a tedious work that requires the analysis of a large number of states and the interaction among all the state combinations.

4.5 4TSoC Synthesis Model

The technological synthesis of the 4TSoC is of utmost importance for the reliability of the final chip. Fault containment regions and failure rate assumptions are determined by the technology below its RTL or VHDL models. In this section, the suitability of ASIC and FPGA technologies is discussed and the example of the prototype 4TSoC is described with optional synthesis patterns for a higher reliability on a Xilinx FPGA.

4.5.1 ASIC and FPGA End-Devices

The same VHDL code (with few modifications) can be used to implement 4TSoC on different FPGA technologies (e.g., flash-based, antifuze) and even as an ASIC (these technologies are introduced in Chapter 2, Page 16). The fault containment capabilities and the radiation immunity degree of each technology is analyzed against different development constraints and commercial options.

Fault Containment

A specific ASIC device is the best choice to accurately position the cores on an integrated MPSoC as spatially separated regions and provide dedicated routing lines for each one. For instance, the IEC-61508 standard revision for on-chip developments highly recommends the use of dedicated power lines and input-output routing [IEC09]. Also substrate engineering techniques (e.g., triple-well, doping level) could be used on ASICs. For example, guard-rings for radio-frequencies are used in [BLY02] where an important noise containment is obtained adding physical separation.

FPGAs cannot provide these specific measures. Programmable technology arrives to allocate logic resources on specific areas through synthesis constraints, but allocating routing is not possible for all manufacturers (e.g., Xilinx). Moreover, configuration logic is normally shared by all the resources of the FPGA chip which can lead to the whole chip failing like a single unit [GHB10]. Nevertheless, it is possible to work on spatial separation by providing guard bands on the FPGA layout between the elements. For instance, areas where no logic resources are used could be constrained before synthesis in order to minimize the common effects of physical faults (e.g., wide cross-sectioned MBUs).

Radiation Immunity

SRAM memory based FPGAs are highly susceptible to SEU like faults, with fatal consequences. A bit-flip in this configuration memory could change the implemented circuit and can lead to a functional failure or Single Event Functional Interrupt (SEFI). According to the Xilinx Reliability Report they present a Failure In Time (failures per 10^9 hours) of 158 FIT per megabit on a $45nm$ technology. Vendors include fabric information redundancy codes for these memories: Altera includes a CRC per frame [Alt07] and Xilinx a 12 bit ECC for each frame or configuration word and a CRC for the complete memory for their Virtex-5 devices [Xil08]. Specific for FPGAs supporting dynamic partial reconfiguration (e.g. Xilinx), the configuration memory could be modified online which opens the doors to new and interesting recovery and fault injection strategies. For instance, Xilinx supports the periodic re-writing of the static values into the configuration memory from a golden hardened memory [BPP+08].

For Flash FPGAs, the sensitivity to radiation, is far lower from volatile memory sensitiveness (e.g., SRAM) [SNJ97]. Antifuse FPGAs and ASIC technologies are programmed only once using high energy techniques, therefore, they are the most robust technologies. Both exhibit similar failure rates if the *Total Ionizing Dose* (TID) parameter, only interesting for space applications and which is not

considered. Anyway, all the technologies are equally sensitive to pulse-like SET faults on routing and combinatory logic.

Flexibility

FPGA technologies permit to specifically define the resources of a given application during development phase, while ASIC technology cannot. For example, if a given application requires n cores, FPGA technology permits to synthesize these n cores once the application is analyzed. ASIC technology need to work upon a known amount of resources, which makes it harder to optimally fit to the actual application requirements, except if you create a new ASIC which carries long development times and big amounts of money. Within the FPGA family, re-programmable technologies (SRAM and Flash) allows to redefine the functionality of the chip as many times as necessary.

Time-to-Market and Price

The development of ASIC chips requires a very long process and it requires a big amount of money (e.g., few million dollars per design, hundreds of million dollars for large ASICs) that can only be afforded by big production quantities and large markets. Therefore, without a standard integrated embedded system architecture ASICs are not feasible for generic embedded applications. FPGAs offer a good trade-off between time-to-market and price. Furthermore, the designer could make use of existing dependable VHDL defined cores (e.g., soft-core) that could shorten even more the development time.

Conclusion

Certainly, optimized ASIC chips are the best solution to build a safe integrated embedded system. ASICs show the best fault containment and radiation immunity (and other desirable properties, such as power and energy efficiency). However, if we consider other parameters like flexibility or time-to-market the FPGA technology is preferred. Inside the different configuration memory technologies of FPGAs, antifuse and flash technologies show the best trade-off between safety and feasibility. Table 4.1 sums up the comparison.

More considerations could be taken by ASIC (e.g., processor designers) and FPGA manufacturers to make possible the development of, respectively, more competitive (e.g., time-to-market) and more reliable (e.g., fault containment) solutions. ASIC manufacturers could provide a safety optimized standard architecture with a range of possible resources (e.g., number of cores). FPGA

Table 4.1: Comparison for end-device candidates

	ASIC	**FPGA**		
		Antifuse	**Flash**	**SRAM**
Fault Containment	+	-		
Radiation Immunity	++	+	+	-
Flexibility	-	+	++	
Time-to-market and Price	-	+		

manufacturers could provide better isolated programmable regions within the same FPGA die.

4.5.2 Xilinx Implementation Patterns

This section shows the implementation patterns of the 4TSoC model on a Xilinx FPGA. They are inspired by the IEC 61508 specification for on-chip replication and related work on FPGA certification.

Local Clocks and Global Time

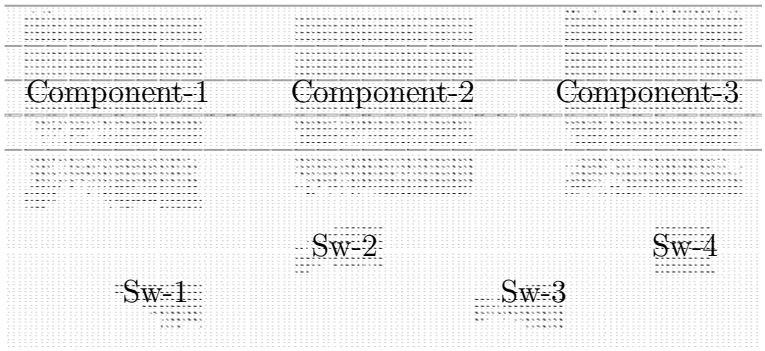
Xilinx FPGAs have multiple independent clock regions that can host the components of the 4TSoC. It also permits both implementations of global time through clock synchronization or using a slower global clock signal. This second option was adopted for the prototype implementation, and the global time signal makes use of the global clocking resources that arrive to all the clock regions of the FPGA. In fact, current FPGA limitations do not enable higher frequencies than some hundreds of megahertz (e.g., <500MHz), and forces to distribute the global time line across the whole chip with a low frequency global time.

Spatial Separation and Common-Failures

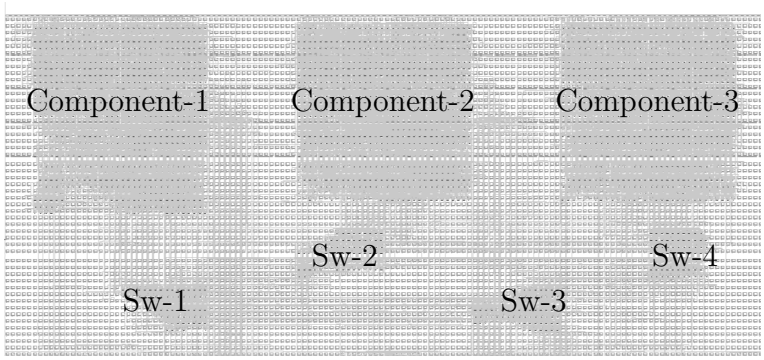
Spatial separation and other decoupling techniques prevent the chip to fail due to common physical faults. This fault containment is interesting when higher reliability chips are obtained using on-chip redundancy, or when one requires higher safety, thus separating the diagnostic units from the supervised logic.

The components of the 4TSoC architecture can be separated in order to avoid common cause failures due to spatial proximity. This separation can avoid multiple resources to fail due to permanent or transient faults. On the one hand, in [GHB10] a minimal distance of 7 CLBs is proposed to avoid the connection of two separated blocks and prevent the propagation of high voltage

or high current caused failures. On the other hand, protection against multiple single events can be achieved with further separation (e.g., 300μm) that avoids correlated errors due to single energized particle grazing [CMFC+98]. Figure 4.8 shows an example of a layout with three spatially separated microblaze processors with TISSes and four fragment switches. The placing constraints are highly simplified using the Planahead tool of Xilinx. The figure shows how in Xilinx the logic primitives (LUTs, flip-flops, BRAMs, etc) can be constrained to specific areas, but the routes are out of the control of the designer and may cross paths.



(a) No routes considered.



(b) With routes.

Figure 4.8: 4TSoC layout on a Virtex-4 LX160 FPGA (FPGA Editor)

Other techniques, such as separated pinout (e.g., ground and power supply) are recomended by the IEC-61508 safety standard [IEC09]. In the case of Xilinx FPGA the configuration lines (GCAPTURE, GRESTORE, etc.) constitute single points of failure [GHB10].

Scrubbing and ECC Memories

In the specific case of Xilinx technology, it is possible to rewrite in runtime the configuration memory of the FPGA in order to avoid undesired bit-flips. This technique is called scrubbing [GT04]. It can be done using an external or an internal configuration port [BPP+08]. The original version of the configuration memory (the bitstream) is stored on a rad-hardened memory (e.g., flash memory) and this information is periodically rewritten. Scrubbing is limited to the static configuration information of the memory, and Xilinx tools provides masking files to avoid the alteration of dynamic bits. Furthermore, this mechanism is transparent for the on-chip architecture, enabling scrubbing does not increase the synthesis constraints (e.g., timinig effort) as it is a mechanism out of the HDL design. Anyway, the certification of scrubbing and partial reconfiguration techniques is still controversial and an open field of research [SAS+04] [RMG+07].

Additionally Xilinx Virtex configuration memories contain ECC protections for each frame, which can be used to trigger a sporadic scrubbing. These ECC codes can also be found in block RAM primitives to protect not only static configuration memory, but also user dynamic information.

Temperature Sensor

The IEC-61508 standard highly recommends the use of temperature sensors in conjunction with on-chip TMR. The rise of temperature is often a sign of short-circuits and other problems that can propagate through the shared substrate of components replicated within the same chip.

Virtex-4 technology contains the interface for a temperature sensing diode (included on the System Monitor in newer Xilinx technologies) that can control an interrupt to turn off the clock, turn on a fan, or perform another operation to reduce heat [Xil]. Another option is to implement temperature sensors on configurable logic using ring-oscillators [BLbG01]. These oscillators are built upon FPGA resources and can be distributed along the chip which enables more specific actions. Anyway, the low precision and temperature sensitivity of ring-oscillators may be not enough for safety related use.

4.5.3 An IEC-61508 Compliant FPGA

The ideal technology implementation for the 4TSoC chip would be an IEC-61508 compliant FPGA. It could provide the trade-off between the superior

flexibility-price feature of FPGAs and the physical fault containment requirements demanded by safety applications (specified in the IEC-61508 standard [IEC09]).

Such an IEC-61508 FPGA, would provide several FPGA blocks or islands within a single chip, that do not share power supply, grounding, clock source or configuration port (e.g., JTAG) and would be spatially separated to avoid the common-cause failure of inner faults (e.g., hotspots) or external physical threats (e.g., soft-errors). It would consist of independent FPGAs within a single chip. Additional containment measures among the FPGA islands such as guard-rings [BLY02] or other microelectronics techniques could be implemented by design. Temperature sensing could also be merged into the chip. The unique connection among the on-chip FPGA blocks would be a hard-wired data bus that would implement the lanes to the components of the 4TSoC architecture.

The IEC-61508 compliant FPGA would not only target the 4TSoC architecture and other MPSoC architectures, there are many applications where the fact of distinguishing several IEC-61508 blocks in a single chip can reduce the number of chips used for an embedded system and reduce the complexity of the PCB.

"Egiteak egiten derakuske",

The done is shown by doing

Chapter 5

Evaluation Tools

This chapter introduces the framework and methods for the evaluation of the 4TSoC implementation.

The FI4SoC evaluation framework provides the means to assess the reliability of integrated architectures (e.g., MPSoCs) against single event transient faults. Faults are injected on an FPGA prototype by means of dynamic partial reconfiguration. As mentioned in Chapter 4 the main focus is put on SET like faults. Into a SET or soft-error fault model, if faults are injected at RTL, the end-device (FPGA or ASIC) can be abstracted, in such a way that the fault-injection results are valid for any RTL implementing technology. Additionally, the framework supports the modeling of the assessed architecture on the Möbius simulation tool.

Finally, the details of the FPGA implementations of the XtratuM hypervisor, the TTSoC architecture and 4TSoC model are given to complete the tools used for the evaluation.

5.1 FI4SoC: Fault Injection Framework

The *Fault Injection for System-on-Chip* (FI4SoC) is a framework for the assessment of integrated architectures in the presence of transient faults. In the following sections, first the requirements for the framework are given, and then the resulting FI4SoC implementation is described.

5.1.1 Fault Injector Requirements

The requirements for a fault injection framework for integrated architectures are:

Req. A: *The set of covered faults must be representative for MPSoCs and support most frequent fault types at an early development stage*

In *Very Deep Sub-Micron* (VDSM) technology, transient faults represent a major concern compared to permanent faults. This fact motivates the focus on the effects of radiation particles, cross-talk and electromagnetic noise [Con02] in the covered fault set. The impact of the injected faults must be relevant on higher abstraction levels. The injection of faults that do not have an effect on the behavior of the MPSoC would not be interesting to evaluate fault tolerance mechanisms. For instance, [SV07] considers two fault kinds, critical and non-critical, are distinguished depending on the severity of the functional effects of a bit-flip into the configuration memory of the FPGA. On a platform independent fashion, one can distinguish RTL-relevant and non-RTL-relevant faults. If one considers a transient faulty pulse (a SET) at gate level, caused e.g., by a radiation particle, this fault becomes interesting only when it is captured by a register (overcoming logical, electrical or latching-windows masking [SKK+02]) and rises to RTL. Thus, for example, faults can be directly injected bit-flips in registers assuming the effects of SETs.

One can distinguish different abstraction levels ranging from RTL level to on-chip message exchange level. The lower abstraction bound (RTL) is generic enough not to be technologically dependent. Thus, it is possible to perform fault injection without attaching the results to a concrete technology (a specific FPGA or ASIC). The injection results can still be valid at different VLSI end-devices if the synthesis process guarantees netlist equivalence to the RTL source code (e.g., using formal equivalence checking). Fault injection at higher abstraction levels (e.g., message-level) can be executed using system level languages such as SystemC [Per09] without using FPGAs. Nevertheless, the accuracy of the results of such a simulation depends on the level of detail of the system level model. Therefore, emulating such faults of high abstraction level in FPGAs still makes sense for detailed results. For instance, the refinement of a model of a processor in SystemC is lower than the one in an FPGA.

Req. B: *The failure detection and classification of MPSoCs must process the data and cope with limited resources (e.g., higher abstraction monitoring)*

The failure detection at gate output level of an MPSoC comprising a billion transistors switching at 1 GHz frequency leads to complexity. It implies an overhead to enable fault injection and monitoring for every gate and the subsequent long time to execute and analyze such a platform. Moreover, the resulting accuracy is not required for the assessment of the fault tolerance mechanisms, because these measures are applied at a higher abstraction level of MPSoCs. Therefore, by raising the abstraction level of the failure detection one would

benefit from the reduction of the amount of data to be analyzed and the simplification of the observation logic. An option is to consider an MPSoC where cores communicate through a message-based NoC. A monitor connected to the NoC can detect the failures of the cores at the NoC interfaces. The classification of failures can be adapted from the classification of messages in distributed systems (e.g., [Jon02, Cri91]).

The performance of an FPGA emulated fault injector is orders of magnitude higher than simulation based HDL approaches. In the case of MPSoCs, the performance advantage of using FPGAs is evident, because processors working in parallel are more efficiently emulated in a natively parallel device like an FPGA than on a sequential simulator. But, when using FPGA technology to emulate faults on an MPSoC, the instrumentation overhead must be minimal and should not scale with the number of cores. Otherwise, the hardware overhead and frequency penalties can undermine the performance of the injector.

Req. C: *The injection of faults must be controllable in space and time domains*

The framework should support the fault injection in specific components of the MPSoC. For example, in order to validate on-chip Triple Modular Redundancy (TMR) the injector should be able to inject faults into one of the replicas of the TMR configuration. In the same way, the instants of the fault injection should be under control. The shared resources of the MPSoC (the NoC, a gateway to an off-chip network, etc.) are used by different cores at different times. By controlling the injection instant, the injector will also know which is the processor affected by a specific fault at that precise moment.

Req. D: *The fault injection logic must not afflict itself*

The framework should assure that the effects of an injected fault in the design under test do not propagate to the injector. In other words, fault containment must be assured between the fault injector and the design under test. This point has particular importance when the injector and the tested design share the same FPGA chip. When performing the place-and-route of both blocks into the same chip, some routes belonging to the injector could cross the design under test (even if the logic resources are placed separately). This possible hidden channel of error propagation must be considered.

5.1.2 FI4SoC Description

FI4SoC aims to accomplish the aforementioned requirements by introducing an FPGA-implemented and MPSoC-oriented fault injection framework. It uses

partial reconfiguration to inject faults w.r.t. an RTL model into memories and flip-flops of the MPSoC design. A single FPGA integrates the necessary injection logic (the monitor, the injector, etc.) into the same FPGA chip of the MPSoC design (Figure 5.1). This technology is implemented in Xilinx Virtex-4 FPGAs.

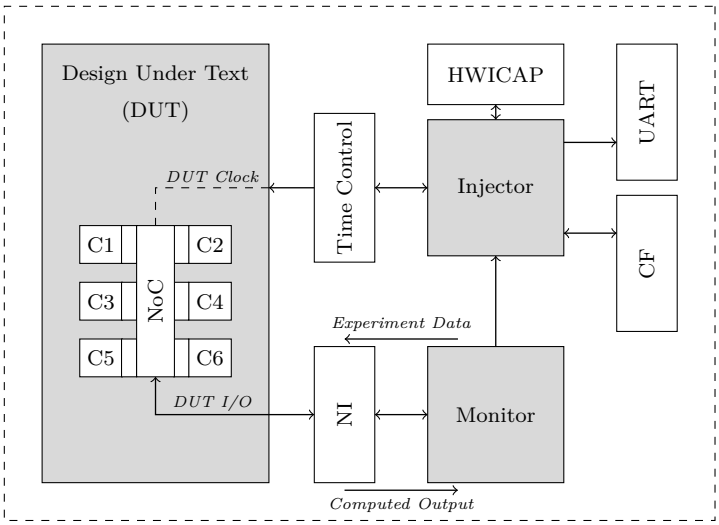


Figure 5.1: The FI4SoC architecture

The implementation of the FI4SoC framework consist of these basic blocks:

- *Design Under Test (DUT)*: this block contains the target of the fault injection, namely the MPSoC or a part of the MPSoC (e.g., a specific core or NoC element). Its extension limits the area where the framework injects faults. The input-output nets, which connect the DUT to the monitor through a network interface, are the only logical connection to the rest of the framework.

- *Network Interface and Monitor*: the network interface provides to the monitor a message-based communication with the MPSoC. The monitor (in this work a Xilinx Microblaze processor) serves as the emitter and recipient for the messages of the MPSoC. It provides the functional experiment data to the DUT and it receives the computed outputs. It also executes the error detection mechanism and alerts the injector in case of a failure in the DUT.

- *Fault Injector*: it is the key component of the framework (also implemented by a Microblaze processor). It inserts faults in the FPGA's RTL-mapped memories through the internal configuration port according to the positions and instants obtained from the fault vectors stored in a

Compact Flash (CF) memory. The Fault Injector controls the stop time of the DUT by programming the counter registers of the Time Control unit. The injector is also responsible for storing in the CF the failures detected by the monitor. The UART port is used to print status messages for the user.

- *ICAP*: it is the internal configuration port of Xilinx FPGAs which gives read and write access to the configuration memory of the FPGA and enables injecting faults modifying it. Xilinx offers hardware (HWICAP) and software drivers to manage this port from the cores. There are other ways to access the FPGA configuration memory (e.g., JTAG, SelectMap) but those ways require and external access to the chip what would increase the development time and cost of the platform.

5.1.3 Injection of Supported Faults

FI4SoC emulates two main physical faults at logical level in Xilinx Virtex-4 FPGAs: (1) SEUs in memory elements and (2) SETs in combinatory hardware. These two specific fault types have been selected these because they are frequently mentioned as the major concerns regarding transient faults [ME02][Kop08a]. The SEUs in memory elements are directly mapped into bit-flip injections into FPGA memories (distributed or Block-RAMs) at RTL level. SETs are glitches or transient pulses in the combinatory logic, which cannot be directly modeled in FPGAs or RTL. Therefore, FI4SoC emulates their effects when they are captured by flip-flops as spontaneous bit-flips.

Injection in Distributed RAM and Block-RAM

Even if there is no difference at RTL level, the injection of bit-flips in distributed RAM (LUT implemented memory) and Block-RAM (dedicated RAM memories) of Xilinx devices shows some differences. One can select which of both types of memories needs to instantiate for use in the FPGA at place-and-route time. After this step, the positions of used distributed and block-RAM positions are listed in an automatically generated file (a.k.a logic allocation or ".LL" file). While it is straightforward to modify a LUT implemented memory, Block-RAMs are protected and one needs to take care of writing the proper write-enable bits (save-data bits) every time the injector wants to inject a fault. Those save-data bits must be written to '0' every time a new modification is wanted to be stored in a Block-RAM [LBN10]. The last difference resides on the API that Xilinx provides to access both technologies: distributed RAM is accessed by slice coordinates, while access to Block-RAMs uses the frame offset

in the configuration memory. The slice coordinate and frame offset information of each instantiated memory is listed in the ".LL file".

Injection in Flip-Flops

Whereas some FPGA resources are directly mapped in the configuration memory (e.g., the aforementioned RAM contents), the values of the flip-flops, as sequential elements with dedicated set and reset signals, are not. One needs to capture them in the configuration memory and restore their values using tricky options of Xilinx Virtex technology. The values of the flip-flops can be captured at any moment and then they can be readback through ICAP. Once one knows flip-flop contents, one can change the reset value consequently and provoke a reset to inject the fault. The reset type and the reset signal are shared by every two FFs (see Figure 5.2). This process has been named *Capture-Readback-Controlled Reset* (CRCR). CRCR requires the following steps for every injection:

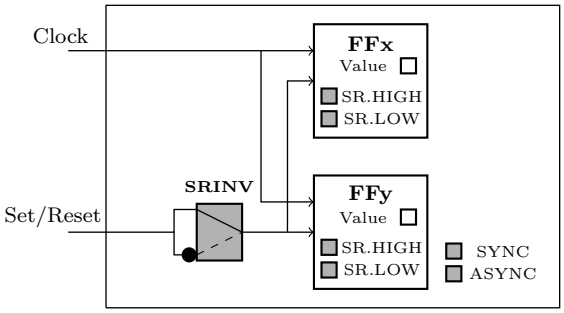


Figure 5.2: Simplified representation of a Virtex-4 CLB slice

1. Stop the clock of the design part that is under test.
2. Assert a "Capture" signal to copy the values of all the FFs of the FPGA into the correspondent places of the configuration memory.
3. Readback the current value of the specific FF and its pair from the updated configuration memory.
4. Set the reset type of the slice to asynchronous.
5. Set the reset value of the target FF to its opposite and its FF pair to the same value.
6. Provoke a local reset by toggling "SRINV" multiplexer.
7. Rewrite original reset values and reset type of both FFs.

8. Resume the clock.

For both cases (SEU and SET), the FI4SoC also considers multiple upsets caused by a single event. By simultaneously flipping several bits into contiguous RAM memory cells or related flip-flops one can emulate Multiple Bit Upsets (MBUs), while the clock is stopped. The coordinates of the used flip-flops of a given design are also provided by the aforementioned ".LL file".

5.1.4 Injection Process

The injector insert faults at instants and positions according to the information contained in the CF, until a message-level failure is detected by the monitor. Then, it stores the failure and reconfigures itself. The injector repeats the process with a new fault vector set.

Table 5.1 shows the information of each fault vector. It provides the fault *type* (SETs in flip-flops or SEUs in memories) and the amount of *locations*, if *multiple* upsets are desired, and the subsequent area coordinates. In the time domain, the *period* of the next fault and the *phase* of the fault injection for this period are provided. The clock of the DUT is stopped during the reading of the vector and the fault injection. As illustrated in Figure 5.3, during a *waiting time* the monitor checks the possible propagation of the injected faults to the message-level.

Table 5.1: Information of a fault vector

Parameter	Description
Type:	SET or SEU fault
Multiple:	Single or multiple fault locations
Location:	Slice or Frame coordinates of the fault
Phase:	Offset of the fault in the time domain
Period:	Period to next fault vector read

The failure detection and classification is performed by the monitor checking the message arrival at its network interface. An input assertion classifies the messages using a-priori knowledge of the application running on the MPSoC. The application can be a synthetic application designed for the fault injection (e.g., a counter) or an actual embedded application. In the same way, the monitor can get the messages serving as recipient or using multi-casting to diagnose all the messages among the cores. The evaluation of input assertions serves for the identification of two types of failures:

- *Message omission failure*: no message has arrived at the network interface within the specified receive window (e.g., a give number of microticks).

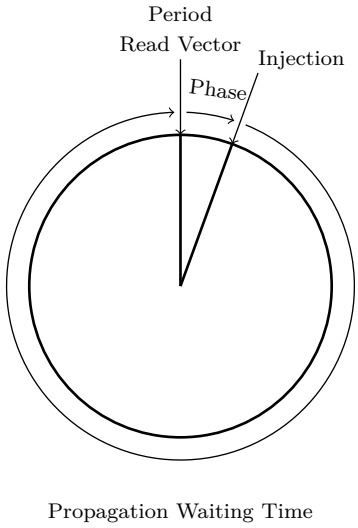


Figure 5.3: Fault injection period

This failure mode includes fail-silent failures and untimely messages (such as babbling idiot, delayed, etc.).

- *Value-incorrect message*: a message has arrived at the network interface within the specified time windows but the data is corrupted.

Finally, after the failure is detected, the injector stores in the CF the number of the fault causing the failure and its type (no message, incorrect, etc.). Then, it provokes a general FPGA reset and the whole chip is automatically reconfigured to the same initial fault free state using the bit-stream stored on a flash memory.

5.1.5 Framework Tools

Several tools assist the steps to perform a fault injection experiment using the framework (Figure 5.4). Planahead[1] simplifies the assignment of specific regions of the FPGA to the hardware instances. In this way, it is possible to generate fault patterns filtered by the occupied areas (or coordinates). The generation of fault vectors for statistical fault injection parses the ".LL" file and uses a pseudo-random generator to choose positions and time instants (e.g., with Matlab). The interaction of the input vectors and the output results is done using a CF. The fault injector reads the fault vectors and writes the observed results to the CF. At the end, a parser or a spreedsheet analyzes the outputs and gives the average numbers of faults to cause a failure and the failure mode statistics (e.g., percentage of no message failures).

[1]Planahead: http://www.xilinx.com/tools/planahead.htm

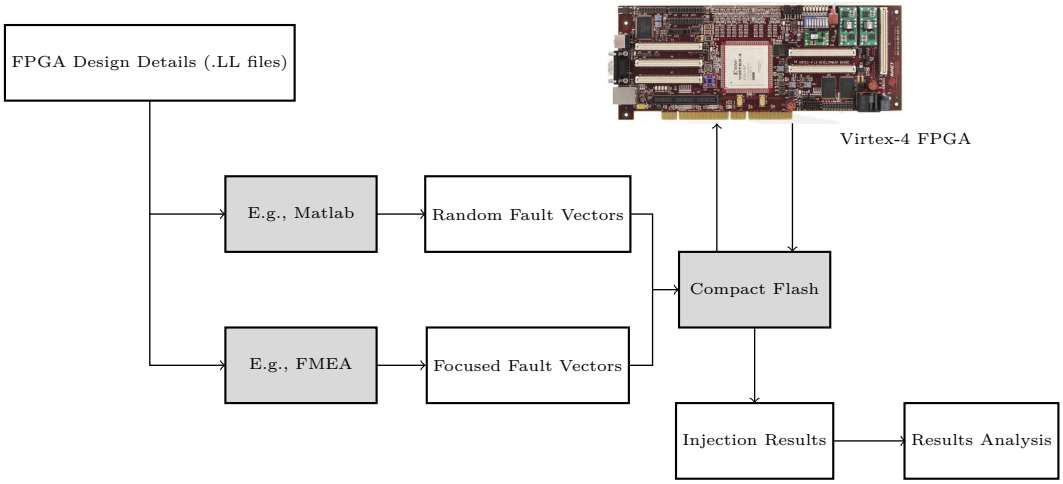


Figure 5.4: Tool work-flow in the framework

5.1.6 Discussion

This section discusses the relationship between the FI4SoC framework and the requirements described in Section 5.1.1 (Page 63).

Covered Faults

FI4SoC has modeled two generic VLSI fault types at RTL level: SEUs and SETs. Both faults are transients and can be caused by several sources: radiation particles, cross-talk, electromagnetic noise, etc. The use of an RTL abstraction level permits the early evaluation of the architectural FT-mechanisms of an MPSoC without having to consider the technological implementation and its related specific faults.

Failure Detection and Classification

The fault detection and classification have been performed by monitoring the messages of the NoC at the MPSoC. This approach reduces the data to be analyzed, but requires a waiting time to permit the propagation of faults from RTL to message-level. The FI4SoC monitor distinguishes between message omission and incorrect-value message failures using assertions that classify the messages according to a priori knowledge of the application running on the MPSoC. Beyond the quantity of failures per injected fault, the identification of incorrect-value message failures is very helpful to determine the type of FT-mechanism (e.g., double or triple modular redundancy) required by the specific MPSoC application.

Fault Controllability

The 4TSoC approach obtains controllability in the spatial domain. On the one hand, one can associate a particular logical resource with its functionality using the ".LL" file. On the other hand, the validation engineer can place specific blocks of the DUT in specific areas of the FPGA (e.g., the Planahead tool), and these positions can be directly associated with those blocks. In time domain, the injector can be synchronized with the NoC by means of the Time Control block. This Time Block contains two registers that are programmed with the microtick and macrotick values on which fault must be injected. When the instant is reached the Time Block stops the DUT clock. This is particularly interesting when the NoC is driven by the global time: faults can be injected in synchrony with the global time and, for instance, failures can be correlated with the load of the network at any time.

Independence of observation and fault injection logic

In Xilinx technology, one can assure the allocation of logic resources (LUTs, flip-flops, etc.) into specific positions of the FPGA. Thus, the framework can separate the injector and the DUT along the FPGA layout. However, the connection routes among those logic resources cannot be constrained to any area, and consequently, routes of the injector could cross the DUT. One can solve this problem by raising the abstraction level to RTL. Faults are only injected on logic elements which are defined at RTL. For instance, the works where faults were injected in any position of the configuration memory [ACD+07][SATGM08] also have to deal with the corruption of routing resources. For this reason, that works use a second FPGA for the injector to assure fault containment with respect to the DUT.

Overhead and Performance

The partial reconfiguration approach does not use any extra logic for every flip-flop to perform fault injection. Low level primitives accessed for capture, readback and local-reset are hard-wired resources on the FPGA. Only the access logic to the configuration memory and the failure checking blocks require additional logic in our approach and this overhead maintains stable even if the MPSoC becomes bigger (e.g., more cores).

The monitor and the injector need slices and block RAMs and they consume 6% of the combinatory resources and 30% of the structured memories of a Virtex-4 LX160 FPGA. On-chip RAM, mostly used to store the program code

of the injector can be loaded to an external RAM. Safety standards (e.g., DO-254) recommend to inject faults under the same constraints that will be used in the final end device. As our approach follows a dependability assessment at logical level, it is independent of area and frequency constraints. This approach proposes the use of a (big) dedicated FPGA just for fault injection.

Regarding the performance, the injection process is short compared to the time the monitor waits for possible failures after every flip-flop fault. It takes 201500 clock cycles to execute the flip-flop value inversion process. At 125MHz this requires 1.612ms. The initialization of the framework, done once per set, takes 69072779 clock cycles (0.522s at 125MHz). In one of our experiments [AaOMI11], the framework waits 0.49152s per fault. One can choose this time (i.e., 150 communication cycles[2]) in order to let the fault in the flip-flop propagate to the outputs. In the experiment with 1024 sets of 100 faults (102400 faults), if all the faults were injected, the total time spent in injection would be 165.068 seconds (2.7 minutes), the initialization times 565.844s (9.4 minutes) and the accumulated waiting time 50331 seconds (14 hours). The waiting time to let the possible errors propagate dominates the performance of the experiments.

Upgrade to Newer Xilinx Technologies

As mentioned before, the FI4SoC have been implemented using Xilinx Virtex-4 technology. Whereas the injection in Distributed RAM and Block-RAM memories is directly portable to new Xilinx technologies, the fault injection in flip-flops using the previously introduced CRCR strategy is no more implementable beyond Xilinx Virtex-4 series. From Virtex-5 onwards the FPGA slice has been redesigned and the inverter-multiplexer (SRINV) enabling the flip-flop reset through ICAP has disappeared.

An alternative to the CRCR is the use of the *"Capture and Restore"* primitives of Xilinx FPGAs [Xil]. The *Capture* command updates specific reserved memory bits with the current state of all the flip-flops of the FPGA at the instant the command is submitted. In the contrary, the *Restore* command atomically writes into all the actual flip-flops of the FPGA the values stored in those reserved memory bits. Therefore, it is possible to inject faults by intentionally modifying these intermediate memory bits after *Capture* and before *Restore*. It is important to highlight that both commands are applied to all the flip-flops of the FPGA at the same time and it is not possible to do it locally, to a specific flip-flop or group of flip-flops, only globally to the entire set of the chip. The limitation of the *"Capture and Restore"* approach is that it requires two

[2]each of the TDMA rounds of a time-triggered network

FPGAs. If FI4SoC is implemented on a single FPGA, both, the DUT and the injection logic (the Fault Injector in the FI4SoC framework), share the same configuration logic, therefore, the *Capture* for the whole FPGA occurs while the DUT remains stopped but the Fault Injector continues working, and later, the state of this injection logic is "corrupted" when the previously captured bits are restored for the entire FPGA.

It would be possible to build a kind of "state-less" injector that, for instance, is reset just after the restore, which would enable the *"Capture and Restore"* approach on a single chip. Anyway, during the development of this thesis, and working close to Xilinx experts, many issues have been found in order to implement this approach (e.g., capture set-times, readback corruptions), and it was finally discarded.

5.2 The Möbius Tool

Möbius is a software tool for modeling performance and dependability of complex systems. It uses an integrated multi-formalism, multi-solution approach [DCC+02]. The first step in the model construction process is to generate a model using a suitable formalism (e.g., stochastic petri-nets), which is also called an *atomic model*. Then, the measures of interest are specified by generating a *reward model*. If the model is complex, a *composed model* can be built up by replicating or associating sub-models at different abstraction levels. Finally, a solution is computed by generating the *solved model* where the calculation method could be exact, approximate, or statistical.

The atomic model enables to model failure and repair rates by different probability distributions which avoids the use of purely mathematical and complex models [DS01b].

5.2.1 The Möbius Tool and FI4SoC Framework

In this work the Möbius tool complements the FI4SoC framework for the evaluation of recovery services. FI4SoC injects faults periodically and the distribution of faults is simplified. In that scenario, it is not possible to assess the actual influence of recovery services upon the overall reliability. For instance, if one takes the example of a fault injection campaign into a TMR system where faults are injected periodically, one can determine the average number of faults to provoke the TMR system fail. But, if one adds an ideal recovery mechanism that resets a block faster than the arrival of a new fault, the system would never fail.

By using Möbius realistic distributions of faults (e.g., exponential) can be applied, as well as the recovery rates. In this work fault injection results are used to feed the Möbius simulation model with actual failure-rates per component. For instance, a TMR Möbius model is elaborated and failure transitions are fired according to the failure probabilities obtained through FI4SoC fault injection experiments.

5.3 Evaluated Architectures

The experiments on this chapter use the implementation of three integrated architecture on a Xilinx Virtex-4 LX160 FPGA. The details of these implementations are given in the following subsections.

5.3.1 XtratuM LEON3 Implementation

The Xtratum hypevisor (introduced in Chapter 3, Page 27) has been implemented upon a LEON3 processor on a Xilinx Virtex-4 FPGA. The LEON3 soft-core has been downloaded from Gaisler page from which the ML403 version can be executed on the LX160 board with almost no modification. The XtratuM hypervisor and the code for the partitions is compiled in Ubuntu and the uploaded from a PC through Ethernet to the FPGA implemented LEON3 processor. The code is stored on the external DDR RAM memory because it is too big for the FPGA internal RAM memory.

The LEON3 processor and the surrounding logic required to implement the XtratuM hypervisor make use of the FPGA resources described on Table 5.2.

Table 5.2: LEON3 resources on Xilinx Virtex-4

	LUTs	FFs	Block RAMs
Core	7707	2561	22
APB0 bus	624	89	0
AHB0 bus	371	46	0
DDR interface	965	536	4
GPTTimer	879	162	0
MCTRL	247	244	0

5.3.2 TTSoC Implementation

The TTSoC implentation used for evaluation has been ported from the Altera TTSoC-NG implementation described in [Pau08]. The porting to Xilinx only

required few changes:

- *Reserved words*: some of the signal names of the Altera TTSoC used names that were reserved for the Xilinx XST synthesizer and they have been re-named.
- *Memory primitives*: the dual-RAM memories of the Altera TISS has been adapted to Xilinx primitives. For instance, the off-line initialization of these memories is done from raw binary data files in Xilinx, whereas Altera allows the use of hexadecimal files.
- *TISS and Component interfacing*: the Altera Avalon bus has been changed to the *Peripheral Local Bus* (PLB) of Xilinx to interface the Microblaze processor.

Table 5.3 shows the resource occupation of each basic component of the Xilinx TTSoC.

Table 5.3: 4TSoC IP resource on Xilinx Virtex-4

	LUTs	FFs	Block RAMs
Microblaze Component	2528	1700	8
TISS	1088	1170	14
Switch	347	438	0

5.3.3 4TSoC Implementation

The 4TSoC architecture takes the Xilinx implementation of the TTSoC [Pau08] as a baseline for the fault tolerance improvements. The 4TSoC integrates the implementation model introduced in Chapter 4 (Page 16):

- *Global Time*: a global clock line 100 times slower than the local microtick is distributed along the whole chip.
- *Spatial Separation*: the application components (e.g., the microblazes) and the elements of the TSS have been located on different configuration regions of the FPGA.
- *Scrubbing and ECC memories*: the static part of the configuration memory of the Xilinx FPGAs can be protected with a continuous rewriting of the contents from a rad-hardened (e.g., flash) memory, what is known as scrubbing. The dynamic memories (e.g., BRAM) can be implemented using ECC protected RAMs which are available on the Xilinx Virtex-4 device.

Nevertheless, none of these technology patterns can be evaluated at RTL level. The evaluation deals with the use of multiples TISSes, NoCs and components. The software driver (e.g., configuration and message handling API) has been modified to support multiple TISSes.

"Saiatzen danak bilatzen du",
The one who tries finds

Chapter 6

Experiment Campaigns

This Chapter describes the experiments for the validation of the 4TSoC implementation. It is divided in three basic campaigns, two of them focus on reference architectures: the evaluation of a software approach (XtratuM) and the TTSoC architecture; and the third validating the 4TSoC extension (Figure 6.1).

1. *MPSoC approach evaluation*: the experiment evaluating the MPSoC hardware approach will assess the partitioning and the fault containment property through a TTSoC implementation comparing it to a hypervisor based software approach of an integrated architecture (XtratuM).

2. *TTSoC architecture evaluation*: the 4TSoC is an extension of the TTSoC architecture, therefore, its original reliability will be evaluated. The failure rates of component, TISS and switch entities of the TTSoC, as well as the component TMR are assessed. The TISS analysis is refined due to its critical impact on the overall TTSoC architecture reliability.

3. *4TSoC implementation evaluation*: finally, the new features of the 4TSoC are evaluated. The limits of ECCs, TISS and NoC duplication, and the application component TMR upon duplicated TISSes are analyzed. Recovery upon application component TMR will be also analyzed.

All these experiments (except recovery on the 4TSoC evaluation that is simulated in Möbius) are executed by the FI4SoC framework using a statistical approach. Based on the equation on [BKH01], an error of 3% at a confidence level of 95% using 1024 injection sets is established. The results are almost independent from the application running in the components and the network load. The injection sets are generated off-line using Matlab and a pseudo-random algorithm. The back-annotation file of Xilinx development tools denotes the

exact positions of the flip-flops. Each injection set comprises the random position and macrotick instants of injections of single bit-flip faults (100 or 200 fault vectors per injection set, depending on the size of the DUT). The injector sequentially injects faults (every 150 communication cycles) in a way that one can determine the mean faults needed to make a particular design fail or the DUT reliability despite the occurrence of faults. While FI4SoC is injecting faults, the system runs a synthetic application for all experiments in order to detect errors. This synthetic application is a mirror application that returns a counter value generated on the FI4SoC monitor.

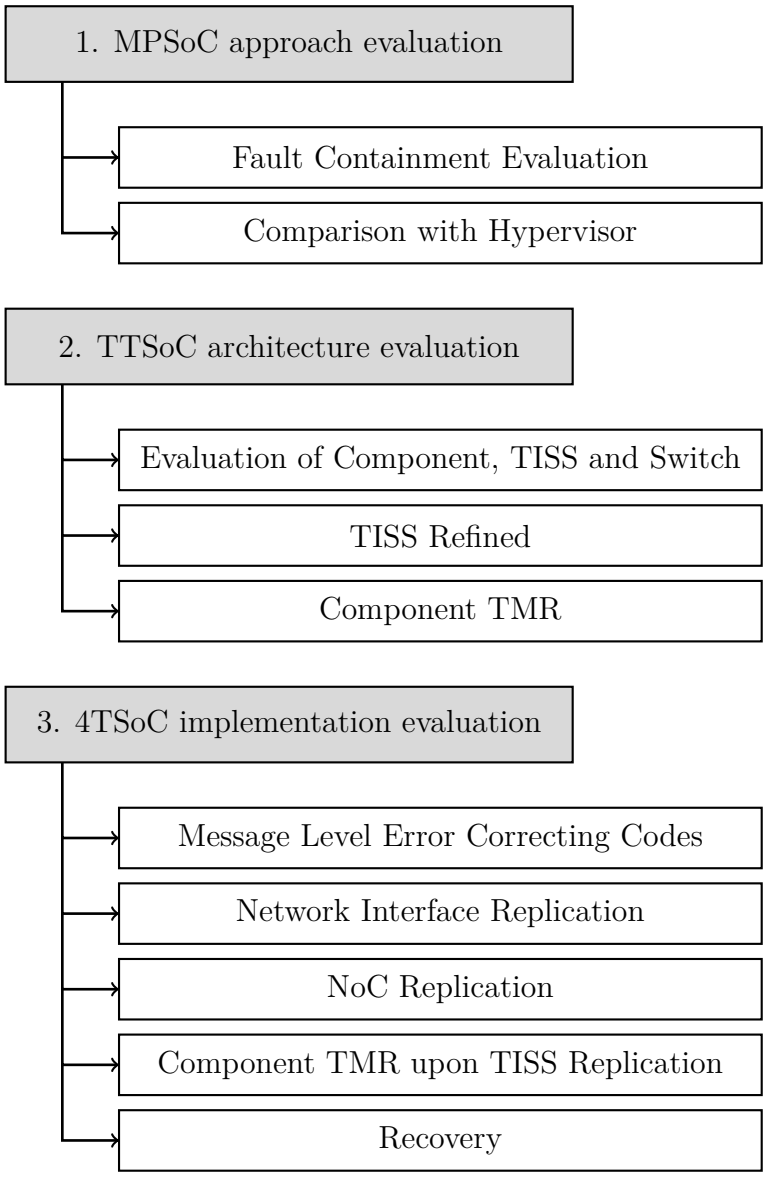


Figure 6.1: Experiment campaings

6.1 MPSoC approach evaluation

This experiment campaign wants to compare the intrinsic fault containment of the hardware (MPSoC) and software (hypervisor) approaches to implement an integrated architecture. The TTSoC architecture is used as an example of an MPSoC approach and the Xtratum hypervisor as the software partitioned approach.

6.1.1 Fault Containment Evaluation

This experiment aims at assessing the fault containment among the components of an MPSoC thanks to the mediation of the TISSes as a temporal firewall. The MPSoC is implemented using the TTSoC architecture. Two components communicate independently with the monitor (Figure 6.2), but only the second component is subject to faults. Any error of the first component due to the communication of the second component would go against the fault containment statement.

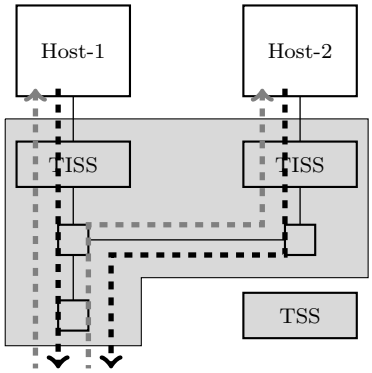


Figure 6.2: A two component MPSoC for fault containment assessment

Hypothesis for Experimental Validation

Hypothesis 1 *"The TISSes avoid the propagation of any error between two independently encapsulated components"*

The TISS limits the sending of messages by the component to specific instants of time that the component cannot change at runtime. Therefore, the TISS acts as a guardian for the temporal behavior, and avoids the presence of untimely messages in the NoC.

Experiment Settings

Figure 6.3 shows the network message configuration for the evaluation of this hypothesis, the sending instant of each message w.r.t the global time (macrotick) and the nature (periodic or sporadic) and size of the message. Each macrotick is divided in 100 microticks and the microtick frequency is equal to the frequency of the hardware clock (100 MHz in all the experiments).

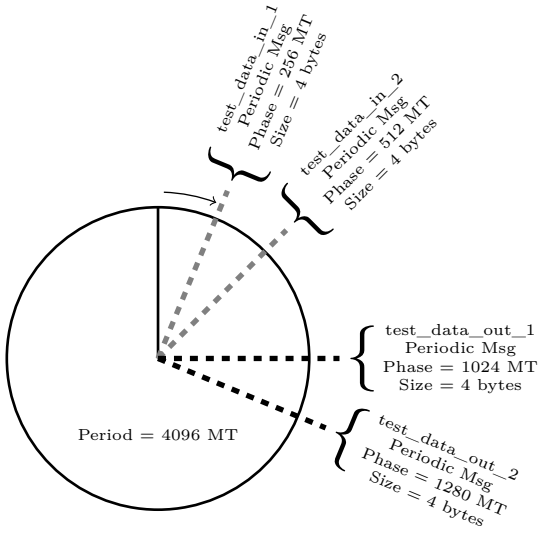


Figure 6.3: NoC scheduling for fault containment evaluation

6.1.2 Comparison with Hypervisor

The NoC based MPSoCs exhibit better fault containment features than embedded monolithic hypervisors. Using the Xtratum upon a LEON3 processor this experiment aims at showing the vulnerability of this software approach against common-cause failures.

The LEON3 processor is implemented on the same Virtex-4 board that hosts the FI4SoC framework. Instead of connecting both using a TISS, the FI4SoC monitor and the LEON processor are connected using *General Purpose Input Outputs* (GPIOs). Specific pins of the GPIOs are controlled by each of the partitions implemented on the XtratuM hypervisor executing on the LEON3 processor (see Figure 6.4). The failure detection is not done by message arrival, but periodically checking the state of the GPIOs by the FI4SoC monitor. These GPIOs show the state of an internal software counter implemented in each of the hypervisor partitions.

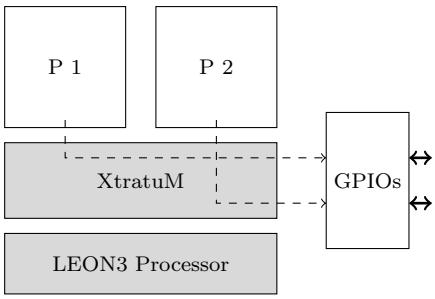


Figure 6.4: Hypervisor assessment setup

Hypothesis for Experimental Validation

> **Hypothesis 2** *"The number of common failures will be significantly higher than the independent partition failures"*

The partitions of the XtratuM hypervisor share the main hardware resource, the processor, of the integrated system. In any case, certain containment coverage is expected because the context switch between partitions can act as a state recovery service for the next partition.

Experiment Setting

XtratuM hypervisor settings are configured offline using an XML file (Listing 1), with two partitions, their respective memory space and the GPIOs. For these experiments two setups will be used. In the first one, the durations of the partitions will be the same (50% of the processor time). In the second, partition-1 will use three times more processor time than partition-2 (75% and 25%) as shown in Figure 6.5.

After every injection set, the FPGA is reconfigured, and the Xtratum code is reloaded into the LEON3 from a PC using ethernet. The PC acknowledges by UART when the LEON3 is ready to receive the code after reconfiguration.

6.2 TTSoC architecture evaluation

The reliability assessment of the TTSoC architecture is done by means of three experiments. The first experiment measures the reliability of the application components, the TISSes and the switches are measured by independently injecting fault in each component. The second experiment analyzes the TISS in

```
...
<Plan id="0" majorFrame="500ms">
<Slot id="0" start="0ms" duration="375ms" partitionId="0"/>
<Slot id="1" start="375ms" duration="125ms" partitionId="1"/>
</Plan>
...
<Devices>
<MemoryBlock name="gpios" start="0x80000000" size="64KB"/>
</Devices>
...
```

Listing 1: Excerpt of a XtratuM configuration file

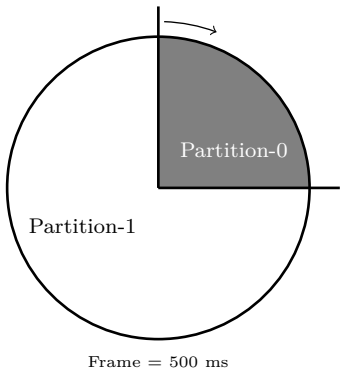


Figure 6.5: Uneven partition durations on the processor frame

depth as the most sensitive component. The third experiment measures the reliability increase by TMR of application components in the TTSoC architecture.

6.2.1 Evaluation of Component, TISS and Switch Reliability

This experiment evaluates the reliability of each of the blocks of the TTSoC architecture using a single channel configuration (Figure 6.6). The component is implemented using a Xilinx Microblaze soft core, hence, the reliability results on the component are dependent on this specific soft-core. The component receives and transmits messages that cross the TISS and the two switches. The fault injection framework injects faults in each element of the TTSoC (e.g., the component, the TISS and the NoC switches) and any failure is detected by the monitor.

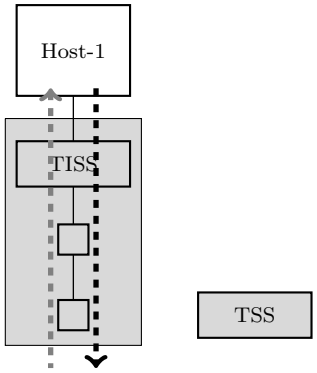


Figure 6.6: Experiment configuration for the TTSoC assessment

Hypothesis for Experimental Validation

> **Hypothesis 3** *"The elements of the TSS (the TISSes and the TTNoC) can exhibit value-incorrect failures behavior in the presence of physical faults"*

The TSS of the current version of the TTSoC architecture does not provide fault tolerance mechanisms that would offer protection against faults affecting the TSS itself. The assumption of the low impact of TSS failures on the overall chip reliability is based on the small size of the TSS [OKS08]. Therefore, one expects to find value-incorrect messages at the TISSes or the switches forming the TTNoC under fault scenarios.

Experiment Setting

Figure 6.7 illustrates the communication instants of the component and the parameters of the messages. The period denotes the time in macroticks (global time ticks) between two consecutive transmissions of a message. In this experiment, it is equal to the length of the communication cycle (each message has a single slot in the cycle). The phase shows the offset of the message from the beginning of the communication cycle.

6.2.2 TISS Refined

This particular evaluation aims at refining the behavior of the TISS in the presence of faults. For that, two experiments are described: (1) for the behavior of the TISS under several network load conditions and (2) for measuring

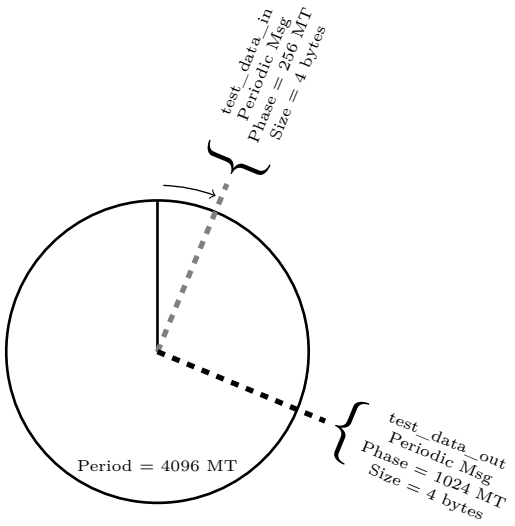


Figure 6.7: Experiment schedule for the TTSoC assessment

the contribution of each of the entities forming the TISS to the overall TISS reliability. An independent experiment is performed injecting 1000 faults into each of the entities.

Hypothesis for Experimental Validation

Hypothesis 4 *"The communication load of the TTNoC does not have a significant influence on the TISS reliability against SET faults"*

The communication load should affect the sensitivity of the TISS to transient faults. A more intensive use of the hardware resources of the TTSoC should result on a higher probability of a SET affecting a functional resource. Anyway, the message exchange subsystem of the TTSoC is only a small part of the TISS and the biggest fraction of the subsystem load is working at the same pace independently of the communication load.

Hypothesis 5 *"A low number of internal entities can dominate the reliability of the overall TISS"*

The TISS contains about a dozen internal blocks, accurately described in [Pau08], from which a few ones could play a dominant role among the rest of the entities. In fact, a fault in some registers (e.g., the current microtick register) can directly induce a failure due to an erroneous scheduling instant and a violation of the time specification, whereas other faults can remain dormant without affecting the normal operation.

Experiment Setting

First, to increase the load of the TTNoC three communication schedules with different cycle periods are proposed. The previous experiments have a period of 4096 macroticks, now shorter periods of 256 (2^8) and 16 (2^4) macroticks are also evaluated. This means that with two messages per cycle the original load of 0.04% is increased subsequently to a bandwidth of 0.78% and 12.5%.

Then, for the evaluation of the entities forming the TISS, the schedule is the same with a period of 4096 macroticks, but these entities are mapped separately into the DUT of the FI4SoC framework. Figure 6.8 illustrates the layout in PlanAhead.

6.2.3 Component TMR

The TMR implementation of three components is performed by the triplication of a Xilinx Microblaze processor with the same application code (no hardware-software diversity is applied). The TISSes and the NoC are not hardened in this configuration (see Figure 6.9).

Hypothesis for Experimental Validation

> **Hypothesis 6** *"Simple TMR improves the reliability of the channel, but TTSoC failures due to a single fault on the TSS can still occur"*

The replication of components should increase the chip reliability under transient faults. In fact, the usual probability (e.g., cross-section) of a transient fault is small to affect more than one replica at the same time. Anyway, the single point of failure at the TSS could undermine the overall reliability of the SoC. The current TSS implementation does not include any fault tolerance measure, hence, it should fail non-silently as stated in the hypothesis.

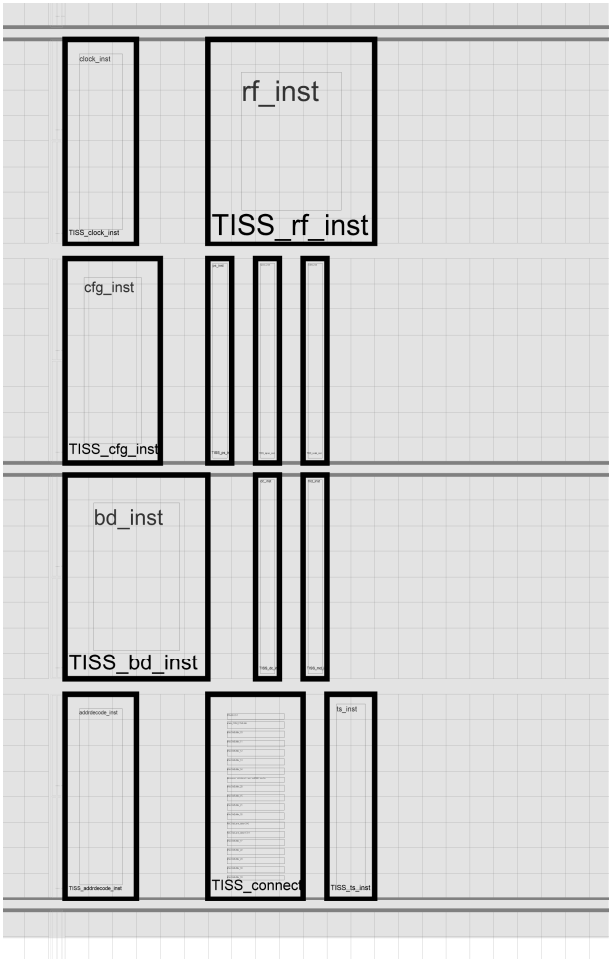


Figure 6.8: Mapping of the TISS entities on the FPGA layout

Experiment Setting

This experiment aims at measuring the native TTSoC reliability using application component TMR, while faults are injected in all the blocks of the TTSoC (components, TISSes and switches). The scheduling of the NoC is shown in Figure 6.10.

6.3 4TSoC implementation evaluation

This campaign evaluates the fault tolerance mechanism for MPSoC introduced by the 4TSoC model. The message level ECCs and the experiments assessing the dual TISS and dual NoC approaches for the hardening of the 4TSoC communication service are introduced. Then, the component TMR upon the dual TISS channel experiment is described.

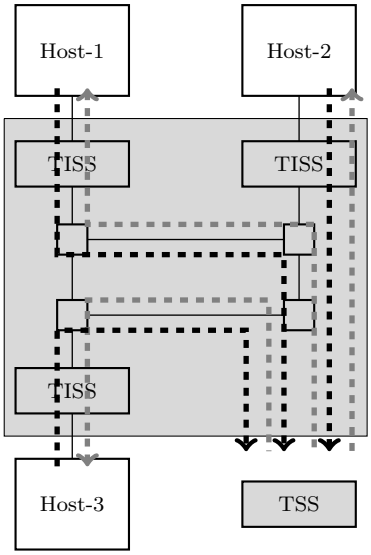


Figure 6.9: Experiment configuration for the TMR configuration assessment

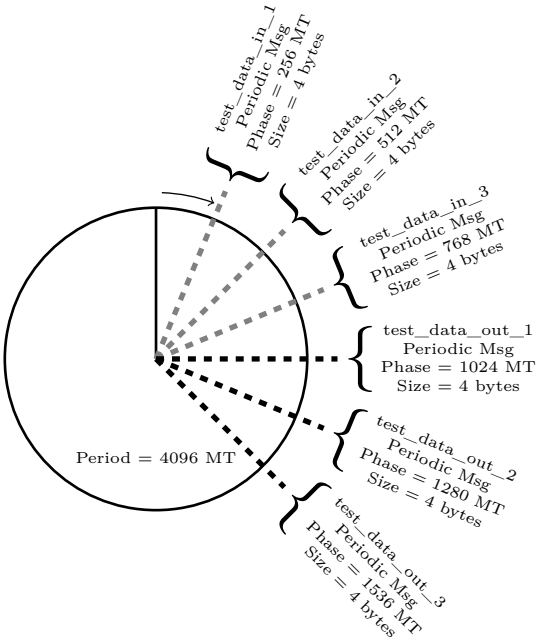


Figure 6.10: Experiment schedule for TMR assessment

6.3.1 Message Level Error Correcting Codes

Through this experiment the hardening of ECC protected messages for the 4TSoC is evaluated. As the 4TSoC has flit positions of 32 data bits, an ECC of 6 parity bits is used. They are sent in the last position of the flit (grouped

if necessary). Figure 6.11 shows the configuration of the experiment.

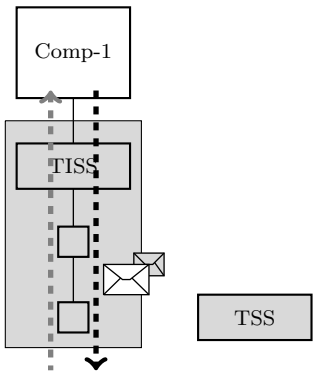


Figure 6.11: Experiment configuration for the message-level ECC assessment

Hypothesis for Experimental Validation

Hypothesis 7 *"ECCs increase the reliability of the channel"*

The ECC only protects the data-path of the communication infrastructure, this hardening mechanism does not cover other failure types, such as failures on the TISS that lead to message omission behaviors.

Experiment Settings

The experiment consists of injecting faults into the 4TSoC blocks: a component, a TISS and two fragment switches (together and separately). They are connected to the FI4SoC framework, which provides computation data and receives the output in order to supervise the behavior of the DUT. Figure 6.12 shows the message scheduling of the NoC.

As mentioned before, each flit has two parts: the first with the field data and the second with the ECC code. In this experiment the ECCs are computed in software. The messages, *test_data_in* and *test_data_out*, are named from the perspective of the component under test and their offset or phase into the communication cycle is given in global time ticks or macroticks.

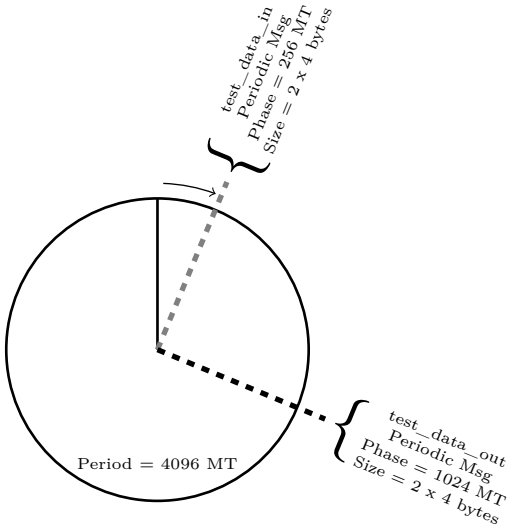


Figure 6.12: Experiment schedule for ECC assessment

6.3.2 Network Interface Replication

This section describes the evaluation procedure of the replicated TISSes of 4TSoC approach (Figure 6.13 and Figure 6.14). Faults are injected all over the component, both TISSes and the four switches.

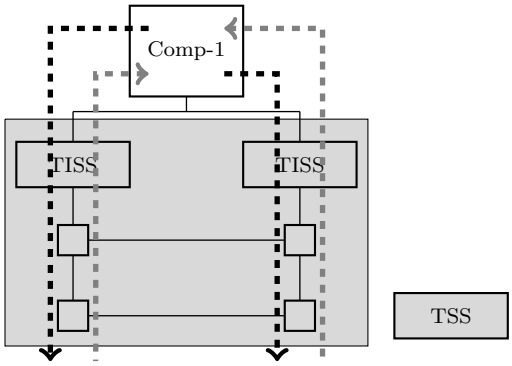


Figure 6.13: Experiment configuration for the TISS replication assessment

Hypothesis for Experimental Validation

Hypothesis 8 *"A single fault in one of the replicated TISSes cannot corrupt the communication of the component"*

At the beginning of the transmission process, the component writes the message to the output buffer of each interface at a different instant, minimizing the probability of a transient corrupting both channels at this first stage even if the component is affected by a transient fault. A crucial requirement for this configuration is the fail silent behavior of the channels: it either produces correct results or no results at all, i.e., it is silent in case it cannot deliver the correct service [Kop11]. From previous work, it is known that the the switches fail in a silent manner, but the TISSes fail mainly non-silently [AaOMI11]. Therefore, both assumptions will be evaluated: the first, considering the non-silent behavior of the TISSes, and the second, emulating an ideal fail silent behavior of the NIs.

Experiment Settings

The FI4SoC framework is used to inject faults in the component, the two TISSes and the two pairs of fragment switches of the NoC. All the messages through both channels are included in a single communication cycle. In Figure 6.14, one can see in light gray the messages associated with the first channel, and in dark gray the time allocated to the second communication link.

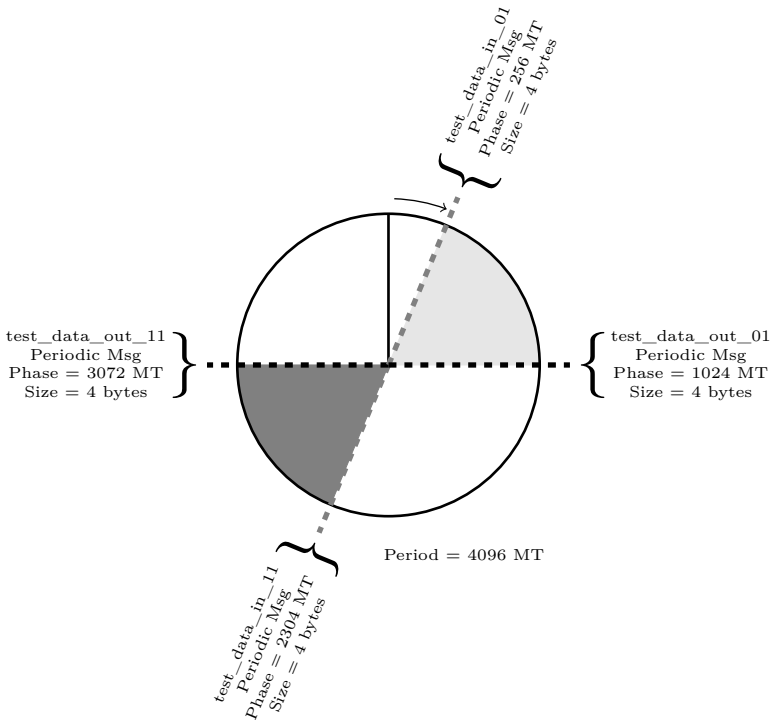


Figure 6.14: Experiment schedule for dual TISS

6.3.3 NoC Replication

This experiment evaluates the validity of NoC replication for the 4TSoC hardening (Figure 6.15). For the assessment, both NoCs have the same topology and communication schedule.

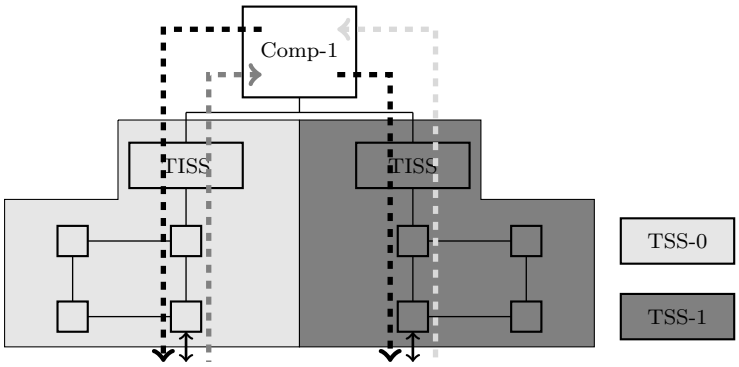


Figure 6.15: Experiment configuration for the NoC replication assessment

Hypothesis for Experimental Validation

Hypothesis 9 *"A single fault in one of the replicated TSSes (TISSes or NoCs) cannot corrupt the communication of the component"*

The flits use independent NoC paths (switches and lanes) which avoids the corruption of the redundant messages due to a single switch fault. The destination component reads the input buffers of each channel and makes a comparison among the received values.

Experiment Setting

Figure 6.16 illustrates the experiment settings for the NoC replication architecture, there is an identical network schedule for each NoC. The injection area includes the component, two TISSes and two instances of the NoC.

6.3.4 Component TMR upon TISS Replication

In the same way as the application component TMR experiment for the TTSoC evaluation, a Xilinx Microblaze processor is TMRed without diversity, but upon replicated TISSes (Figure 6.17).

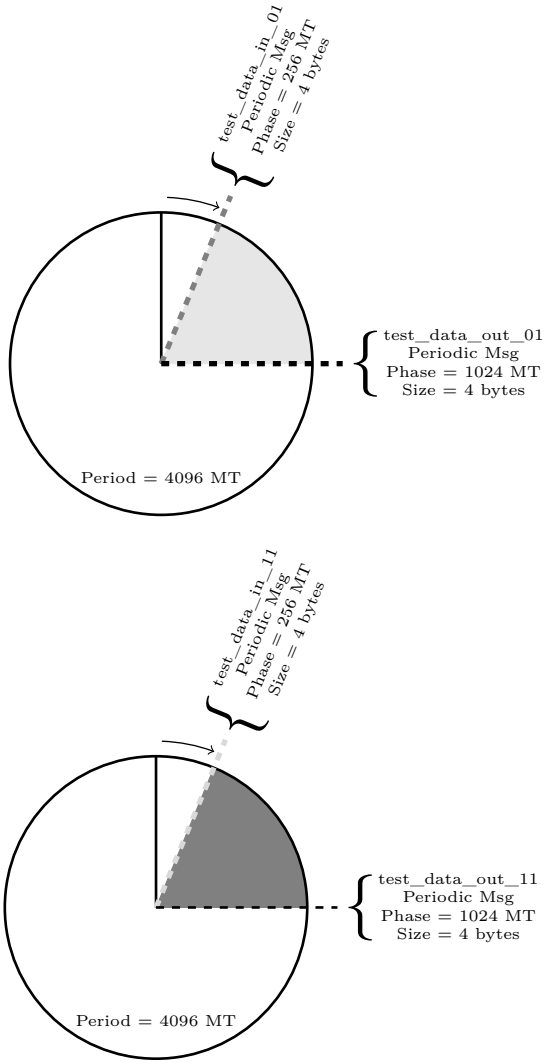


Figure 6.16: Experiment schedule for dual NoC assessment

Hypothesis for Experimental Validation

Hypothesis 10 *"Dual TMR improves the reliability of the dual channel, the probability of failures due to a single fault is decreased"*

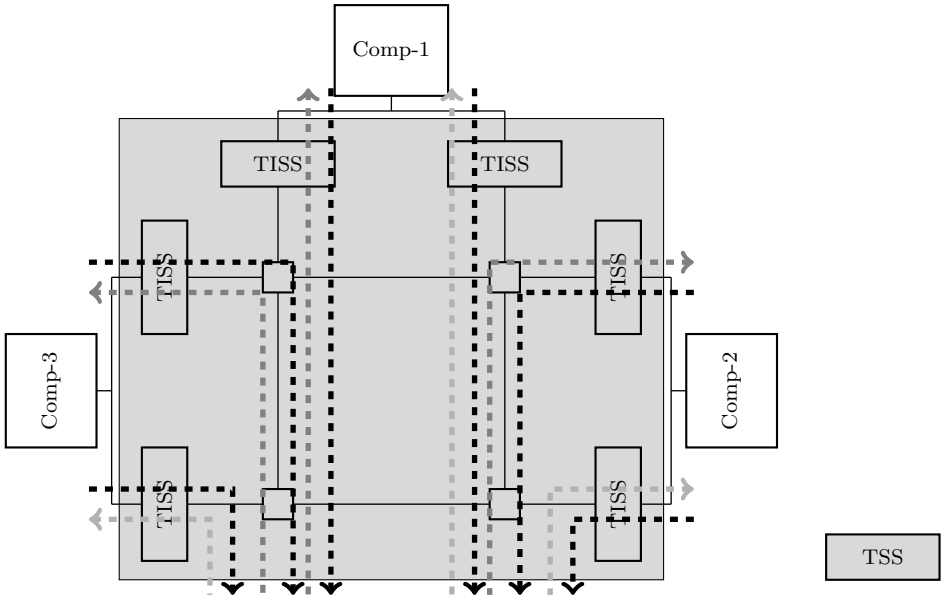


Figure 6.17: Experiment configuration for the TMR-dual TISS assessment

Experiment Setting

Faults are injected in all the elements of the dual-TMR architecture. The diagram of Figure 6.18 shows the scheduling used by the TTNoC for the fault injection experiment.

6.3.5 Recovery

On-line repair of faulty replicas is needed to increase the availability of replicated components on the 4TSoC. These effects are shown using Möbius with the data obtained from the fault injection campaigns. The development of the equations considering recovery becomes a complex and tedious work [DS01b]. For that, one can use the Möbius tool which substantially simplifies the analysis by automatically solving the petri net representation.

Hypothesis for Experimental Validation

Hypothesis 11 *"Recovery applied to an on-chip TMRed MPSoC increase the availability of the system with the appropriate repair-rate and common-mode failure-rate"*

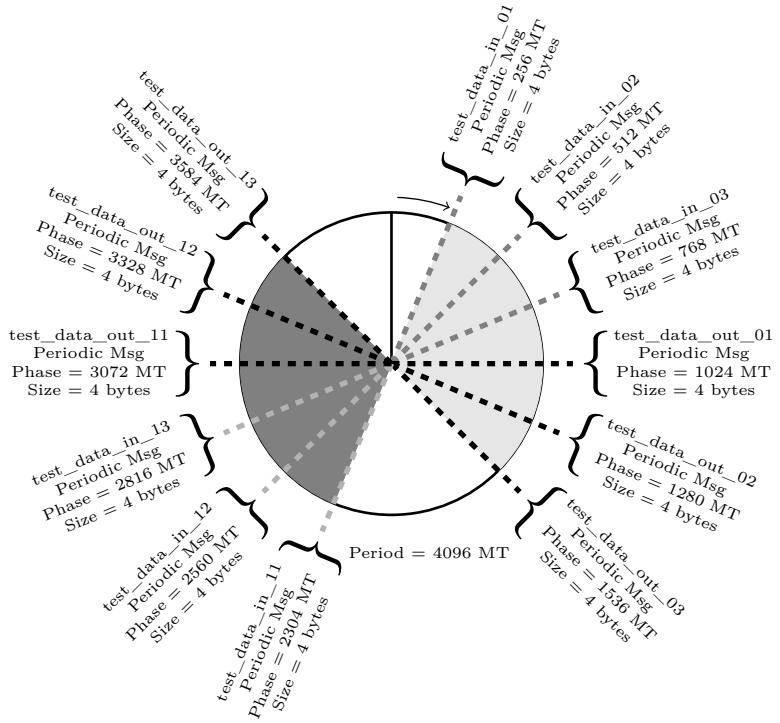


Figure 6.18: Experiment schedule for Dual TMR assessment

Experiment Setting

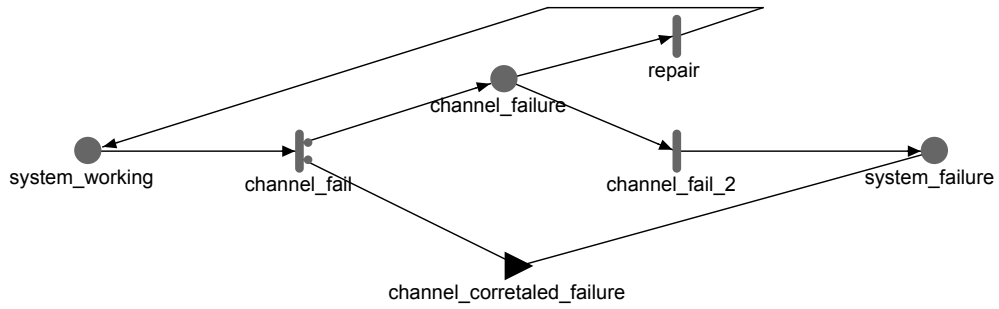


Figure 6.19: Möbius model of component TMR

Figure 6.19 shows the petri-net based Stochastic Activity Network (SAN) designed in Möbius to emulate a TMR system with recovery considering common-cause failures and recovery. The failure rate is applied by channel, and not by component, thus, simplifying the model. The circles in the figure (e.g., *system_working*) represent places that contain tokens and these tokens pass to the next place (e.g., *channel_failure*) when the vertical bars (e.g., *channel_fail*), timed activities, match a certain probability distribution. The triangles (e.g.,

channel_correlated_failure) are output gates that, under a condition, can apply an equation upon the tokens of a place (e.g., they can remove two tokens when a common cause failure occurs). More details about Möbius SAN language can be found in [DCC+02].

In our example, *system_working* starts with 3 tokens, each for channel, and the system fails when 2 tokens arrive to the *system_failure* place. In every *channel_fail* the tokens are subtracted from *system_working* and distributed to *channel_fail* and *system_failure* according to the percentage of common mode failures. If only one channel fails the *repair* activity can bring back a token to *system_working* and repair a channel.

The fault injection results are given on technology agnostic parameters (e.g., fault rate per 1000 flip-flop), therefore, to give a taste of the actual reliability one can provide an arbitrary (and pessimistic) fault rate, 100FIT per channel, which will be used to evaluate the effects of different repair-rates and common-cause percentages on our TMR architectures. The simulations are launched at a confidence level of 99% and a confidence interval of 0.01%.

"Besteren eskuekin ezta uztarik biltzen",
Cannot harvest with the hands of others

Chapter 7

Results

The results of the fault injection experiments and Möbius simulations on the XtratuM, TTSoC and 4TSoC FPGA implementation are given in this chapter.

7.1 MPSoC vs. Hypervisor

This section provides the results that sustain that a hardware implementation (e.g., TTSoC) of an integrated architecture is better w.r.t. fault containment than purely software approaches (e.g., XtratuM).

7.1.1 Fault Containment Evaluation

From the injected 1024 sets of 200 faults each (204800 faults), there was no propagation of an error originated within one host of the 4TSoC to another. This fact provides arguments for the assessment of the fault containment at logical level, without considering physical substrate related faults. The effectiveness of the logical fault containment at physical level depends to a large extent on the layout, the implementation technology of the chip and the TSS contribution. The layout should provide spatial separation among the components in order to avoid correlated events (e.g., MBUs). The implementation technology of the components could introduce additional error propagation channels (e.g., JTAG, clock). To avoid this, a custom ASIC implementation can provide the desired containment measures (e.g., guard bands, independent pin-outs). Finally, an error in the TSS can leak to an overall failure of the chip, therefore, the number of errors in the TSS causing common-failure must be reduced (e.g., using fault tolerance mechanisms).

7.1.2 Comparison to a Hypervisor Approach

These results show how the XtratuM hypervisor approach suffers from more common failures than independent failure of each partition. A hypervisor provides temporal separation, but not spatial and hardware separation of the partitions. According to the experiment results for each partition scheduling (50-50% and 75-25%) shown in Table 7.1 only the 30.57-33.01% of the failures occurred independently on one of the partitions, and they are distributed according to scheduled intervals of each partition. The rest of the failures, 66.99-69.43% (2 out of 3), are making the entire integrated system fail.

Table 7.1: Common-independent failures in XtratuM Hypervisor

	Common	Indep.	Partition-1	Partition-2
P1-P2 50-50%	686 (66.99%)	338 (33.01%)	171 (50.59%)	167 (49.41%)
P1-P2 75-25%	711 (69.43%)	313 (30.57%)	240 (76.68%)	73 (23.32%)

Hypervisors show a positive side-effect when the number of partitions scales. Upon an increase of the number of partitions brings a reduction of the common failures due to the recovery effect introduced by every context switch. Table 7.2 illustrates this effect for the Xtratum hypervisor with 1, 2 and 4 partitions. An analogy to MPSoCs can be found: when the number of processor cores increases the percentage represented by the shared resources (e.g., NoC) decreases, which leads to the same tendency. More partitions lead to fewer common mode failures. However, this tendency shows an asymptotic limit, beyond which the benefits may not be substantial.

Table 7.2: Mean Fault to Fail and number of partitions

Partitions	1	2	4
Mean Fault to Common Fail	8.47	10.15	11.08

7.2 TTSoC Evaluation

In this section, the reliability results of the TTSoC architecture are given from the FI4SoC SET like fault injection campaigns. Additional results on the reliability of the TTSoC network interface, the TISS, are given as it has been found to be the most fault-sensitive block of the architecture. The reliability increase by on-chip replication of components on the TTSoC is also assessed.

7.2.1 Component, TISS and Switch Evaluation

A fault injection campaign has been performed in each of the elements of the TTSoC architecture. First, only in the application component, then in the TISS, and third and last, in one of the switches. Table 7.3 illustrates the sensitiveness to SET like faults of the different elements of the TTSoC architecture. A conservative scenario have been selected where the component is more sensitive to faults than the TSS. In fact, the TISS fails in average every 9.09 faults compared to the injected 20.70 faults to make an application component fail, where both, the TISS and the component have a similar number of flip-flops. It has been confirmed that the elements of the TSS fail in other ways than message-omission and one can see that the TISS only exhibits message-omission behaviors in 10.16% of the cases. The switch is found to be more robust to transient faults, probably due to the state recovery with every new message. Contrary to other approaches, the reliability concerns of the TTSoC architecture should be directed to the TISSes more than to the switches.

Table 7.3: Results for the TTSoCA blocks

	Component	TISS	Switch
Mean Faults to Failure	20.70	9.09	>100
Message-omission Failures	-	10.16%	100%
No Failure	95.22%	89.05%	99.98%
Number of FFs	1377	1190	290

7.2.2 TISS Refinement

The results of focused fault injection on the TISS block are divided into two aspects. With respect to the network load (see Table 7.4), the TISS is not significantly influenced by the load of the TTNoC. Only a decrease from 9.09 to 8.00 fault per failure (1.47%) can be observed when the network load increases 256 times.

Table 7.4: TISS reliability for different cycle length

Cycle Length	4096	256	16
Comms. Load	0.04%	0.78%	12.5%
Mean Faults to Failure	9.09	8.38	8.00
No Failure (%)	89.05	88.08	87.42
Failures with 1 Fault	98	108	124
Failures with 2 Faults	101	100	100
Failures with 3 Faults	94	113	103

This low impact of network load can be explained by the most sensitive entities of the TISS to SET like faults. These entities seems to follow the Pareto principle where the 20% of the resources are causing the 80% of the failures. In fact, as shown in Table 7.5, three of the entities (the Register File, the Clock Module and the Address Decode) out of twelve (25%), are causing the 84.70% of the single fault failures (736 out of 869 failures). If the focus is restricted to these three entities one can see that they are not directly related with network load and this explains the previous results.

Table 7.5: TISS reliability per building instance

	No Failure (%)	1 F.	2 F.	3 F.	FFs
Register File	61.768	390	237	141	227
Clock Module	76.226	232	180	143	69
Address Decode	90.679	114	124	92	129
Subtotal		*736*	*541*	*376*	*425*
Rx. Window Det.	92.414	61	88	68	14
Port Sync.	99.949	27	25	23	15
Memory Digger	99.980	21	11	14	39
Burst Dispat.	99.989	11	10	16	206
Configurator	99.991	5	7	3	190
Connectivity	99.998	8	3	5	175
Dissem. Control	99.999	0	1	0	19
Routing Proc.	100	0	0	0	8
Time Stamper	100	0	0	0	79
Total		*869*	*686*	*505*	*1170*

At first, it seems that by hardening only these three most sensitive entities the reliability increase must be substantial. Hence, two prototypes have been tested applying entity-level TMR respectively to the most two and four sensitive entities. Table 7.6 illustrates that the reliability increase, the decrease of the mean fault to failure, is substantial, but the size (e.g., in flip-flops) and this benefit are similar to duplicating the whole TISS.

Table 7.6: Normal TISS, internally TMRed TISS, and dual TISS reliability

	TISS	2xTMRed	4xTMRed	2xTISS
Mean Faults to Failure	9.09	11.48	16.19	16.53
No Failure (%)	89.05	91.33	93.71	93.94
Failures with 1 fault	98	14	3	44
Failures with 2 faults	101	38	13	46
Failures with 3 faults	94	45	21	43
Number of FFs	1190	1861	2309	2380

7.2.3 Component TMR

This experiment explores the reliability increase of application component triplication. Table 7.7 shows the comparison between the TMR configuration of components and the single component reliability of the TTSoC evaluation. The table shows in its last column that the *Mean Faults To Failure* (MFTF) figure does not improve significantly (1.98 times), and the observation that the fault probability increases linearly with the size of the required hardware (6.44 times bigger), bringing to the conclusion that the TSS contribution undermines chip reliability in fault prone scenarios as expected in [OKS08]. Anyway, the TMR reduces the number of failures with few faults (e.g., the absence of single-fault failures with TMR compared to 51 failures without TMR), on the basis of fault containment. This fact combined with scheduled maintenance actions can increase the reliability of the redundant system. Alternatively, non repairable redundant on-chip systems can be deployed if the mission time is short.

Table 7.7: Results for the TMR experiment

	1 Component	TMR	TMR/Component
Mean Faults to Fail	20.70	41.05	x1.98
No Failure	95.22%	97.57%	
Failures with 1 fault	51 out of 1024	0 out of 1024	-
Failures with 2 faults	29 out of 1024	1 out of 1024	x29
Failures with 3 faults	40 out of 1024	2 out of 1024	x20
Number of FFs	1377	8861	x6.44

7.3 4TSoC Evaluation

This last section reviews the results for the evaluation of the hardening techniques of the 4TSoC transient-tolerant model. The effectiveness of ECCs, dual TISS, dual NoC and recovery mechanisms is given according to the FI4SoC SET-like fault injections. A summary of the results is given in Table 7.10.

7.3.1 Message Level Error Correcting Codes

The ECC coding only protects the data-path of the 4TSoC channel as predicted by the hypothesis, but not other critical entities of the architecture. Table 7.8 shows that it is much more effective for application component hardening (an improvement of 25%) than to protect the trusted system components.

Table 7.8: ECC contribution in the different blocks

Block	MFTF	MFTF with ECC	Enhancement
Component	20.70	25.88	x1.250
TISS	8.69	8.73	x1.005
Switch	195.09	195.09	x1.000
Channel	15.36	16.68	x1.086

7.3.2 4T Communication Services

The 4T communication services based on dual TISS and dual NoC topologies is explored through the fault injection results. These results, out of the 1024 repetitions, are shown in Figure 7.1. The vertical axis gives the number of experiments without failure for the number of faults indicated in the x axes. The number of injected faults is normalized per 1000 flip flops to consider also the size of the architecture as a penalty. It shows that the use of dual TISS is much more effective for channel hardening than ECC codes. The improvement of the ECC approach is minimal compared to a non-protected version. The actual dual TISS has a limited fail silent behavior, but there is margin for improvement.

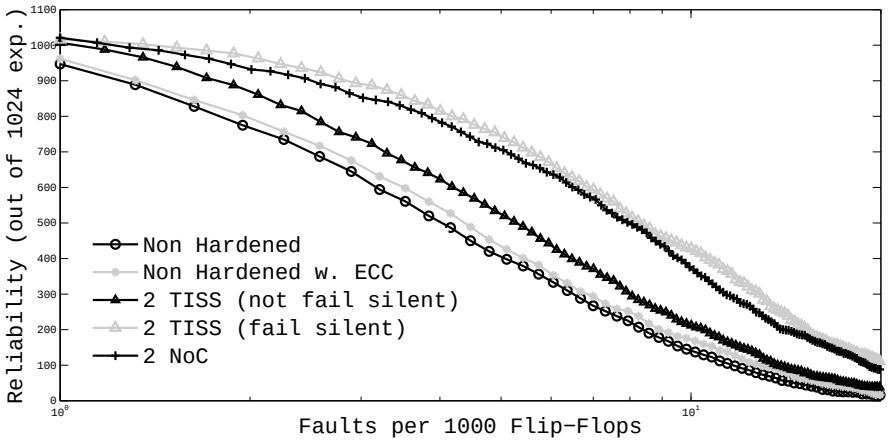


Figure 7.1: Trusted system component hardening results

Figure 7.1 illustrates that the evaluation hypotheses are confirmed and the dual TISS architecture increases channel reliability by an important factor. The actual dual TISS configuration does not fully fail silently and this reduces the reliability improvement compared to a fully fail-silent channel implementation. The interrupt service of the TISS has been found as the main source for non-silent failures: the TISS interrupt service fails and the application component

does not update the out buffers continuously sending the same message. This problem can be solved by a redesign of the hardware requiring an update of a bit of the buffers by the component to send any message, or by software with the use of time-stamps in the sender and discarding consecutive messages with the same time-stamp on the receiver.

According to the fault injection experiments, the replication of the whole NoC does not provide any reliability benefit compared to a single NoC with replicated TISSes. This is due to the low failure rate of the switches that do not disrupt the function of the NoC from single failures and the bigger size of the whole NoC replication approach. Moreover, no correlated faults (e.g., MBUs) have been injected that would affect the switches of the same NoC with more probability than in the version with replicated NoCs.

7.3.3 Component Replication

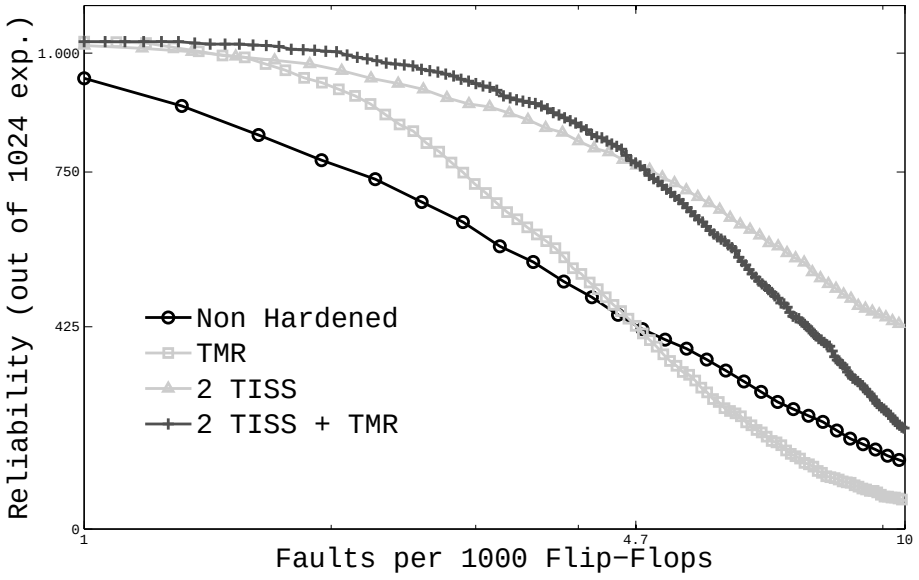


Figure 7.2: Application components hardening results

Regarding the application subsystem of the MPSoC architecture, the fault injection results ploted in Figure 7.2 show the dual TISS approach as the most reliable in terms of MFTF. In fact, TMR upon a simple or dual channel provides limited reliability improvement in extremely fault-prone scenarios or long mission times. Moreover, the resource occupation of TMRed solutions is substantially bigger than the dual TISS approach, for instance, simple TMR

requires 1.48 times more flip flops and 1.16 times more LUTs than the dual TISS architecture.

Regarding the dual TISS architecture, the number of failures with few faults is reduced, but due to its big size, the reliability improvement in MTTF is not that significant when the number of faults increases.

Mathematical Model

From the results on Table 7.10 a mathematical model of the 4TSoC can be completed with the assumption of no failure correlation among the replicated blocks, inspired in [DS01a]. It is possible to deduce the failure rate (λ) of a component in function of the MFTF from the fault injection results and the occupied area. This relation is shown in Equations 7.1 and 7.2.

$$MTTF = \frac{10^9 \cdot MFTF}{FIT \cdot A} \tag{7.1}$$

$$\lambda = 1/MTTF \tag{7.2}$$

SET caused failures show a negative exponential distribution on reliability ($R_{Comp}(t)$). Using the previous failure-rate this reliability is described with Equation 7.3.

$$R_{Comp}(t) = e^{-\lambda t} \tag{7.3}$$

With the reliability of a channel (one component, one TISS and two switches) a TMR mathematical model can be described (Equation 7.4 and Equation 7.5).

$$R_{TMR}(t) = 3 \cdot R_{Ch}^2 - 2 \cdot R_{Ch}^3 \tag{7.4}$$

$$R_{TMR}(t) = 3 \cdot e^{-2\lambda_{Ch}t} - 2e^{-3\lambda_{Ch}t} \tag{7.5}$$

The same can be done for a dual TISS mathematical model using one component reliability and NoC reliability (one TISS and two switches), described in Equation 7.6 and Equation 7.7.

$$R_{Dual}(t) = R_{Comp} \cdot (2 \cdot R_{NoC} - R_{NoC}^2) \tag{7.6}$$

$$R_{Dual}(t) = e^{-\lambda_{Comp}t} \cdot (2 \cdot e^{-\lambda_{NoC}t} - e^{-2\lambda_{NoC}t}) \tag{7.7}$$

Finally, the previous two approaches can be merged into a dual TISS TMR mathematical model (Equation 7.8).

$$R_{DualTMR}(t) = 3{\cdot}(R_{Comp}{\cdot}(2{\cdot}R_{NoC}{-}R_{NoC}^2))^2{-}2(R_{Comp}{\cdot}(2{\cdot}R_{NoC}{-}R_{NoC}^2))^3 \quad (7.8)$$

From these results, the MTTF of each configuration can be obtained backwards by the integration of the reliability, as shown in Equation 7.9.

$$MTTF = \int_0^{MT} R(t)\,\mathrm{d}t \quad (7.9)$$

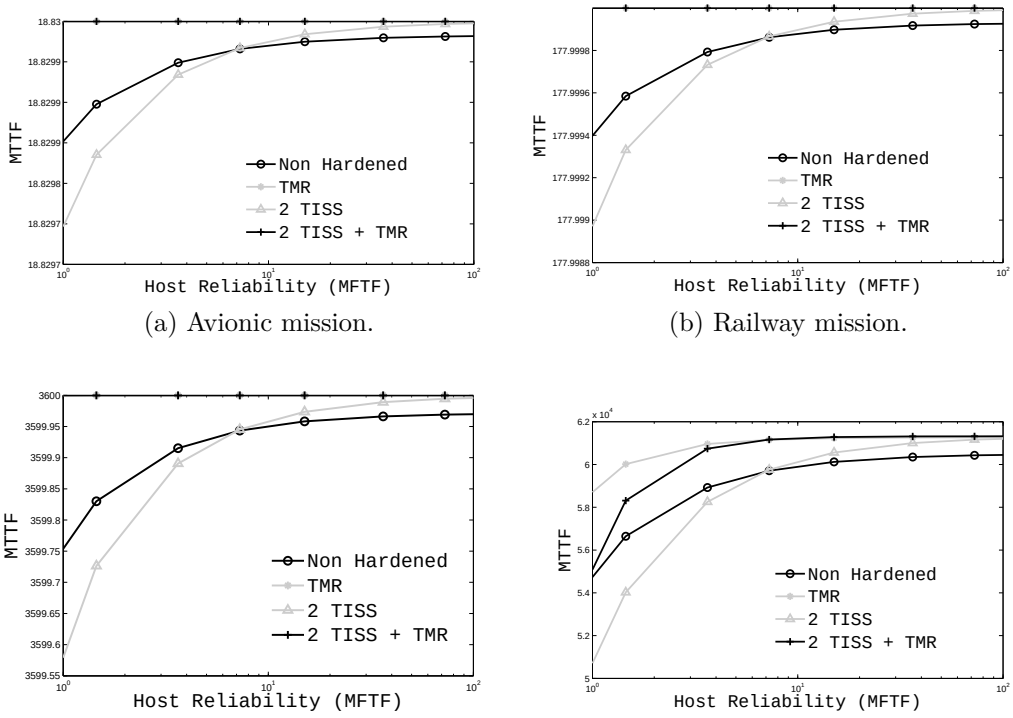
Actual Missions

In order to provide field results for different application domain, mission times and failure-rates of avionics, railway, offshore windmills and spatial domains are shown in Table 7.9. From these data the MTTF of each mission can be plotted.

For that, a SET error-rate must be estimated which requires several assumptions due to the lack of actual data. It is estimated that the SEU-rate and SET-rate may be currently comparable [SKK+02]. Therefore, one can do the assumption stating that the number of SEUs that an FPGA would suffer due to soft-errors is comparable to the number of SETs of a chip. From the Xilinx documentation we can estimate that the slices to implement 1000 flip-flops require around 400Kb of configuration memory in Virtex-4 technology (each CLB column contains 64 slices that are defined within 22 frame of 1312 bits each on the configuration memory). According to the Xilinx Reliability Report the failure-rate of the Virtex-4 configuration memory due to soft-errors is around 250 FIT/Mb [Xil11]. Therefore we can estimate, following the previous assumptions, that the SET error-rate is about 100 FIT per 1000 flip-flop on sea level.

The soft-error rates are also altitude dependent which affect the application domains at high altitude (e.g., avionics) or outside the atmosphere (e.g., spatial). An average incremental factor of 2.2 will be used to every 1000 meters of additional altitude [ST11].

The results on the four mission times under evaluation, the non hardened version and the dual TISS architectures are the ones with shorter MTTF and the dual TISS and the TMR and the dual TISS TMR are subsequently the next more reliable solutions for the Microblaze (Comp. MFTF = 20.70). It is also shown that the dual TMR configuration does not provide any significant



(a) Avionic mission.

(b) Railway mission.

(c) Offshore windmill mission.

(d) Spatial mission.

Figure 7.3: 4TSoC Reliability into different application domains

Table 7.9: Mission times by application domain

Application Domain	Mission Time	Estimated FIT per 1000 FF
Avionics [1]	18 h 50 min	2200
Railway [2]	178 h (aprox. 7.5 days)	100
Offshore Windmill [3]	3600 h (5 months)	100
Spatial [4]	61320 h (aprox. 7 years)	10000

improvements and the dual TISS option can be used on the cases that the application component is reliable by itself.

In order to find a mission where the dual TISS configurations are clearly more interesting, a very faulty environment must be envisioned with a long mission

[1]Current longest scheduled flight: Newark Liberty International Airport (New Jersey, USA) to Singapore Changi Airport, operated by an Airbus A340-500

[2]Longest non-stop train service: Moscow - Vladivostok(Russia)

[3]Inaccessible winter period in the North See wind farms

[4]Mercury's MESSENGER space probe estimated mission time: 6.5 years to arrive to Mercury and several months of capturing data

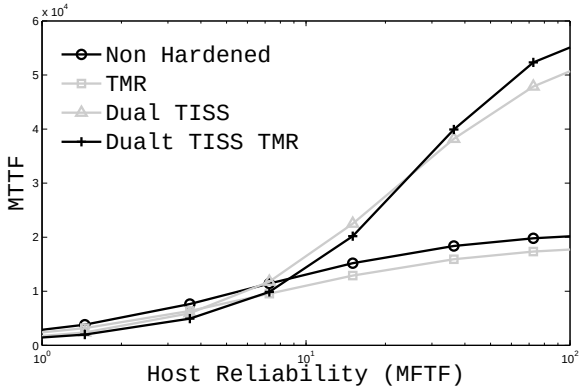
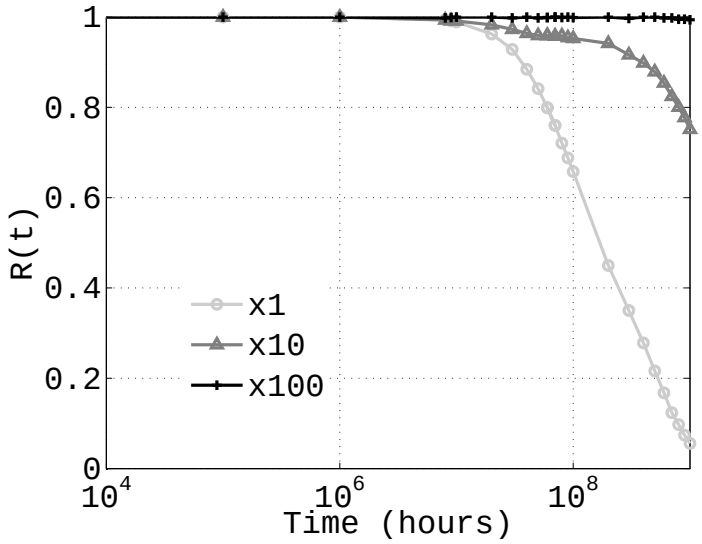


Figure 7.4: Mission where dual TISS configurations are clearly more reliable

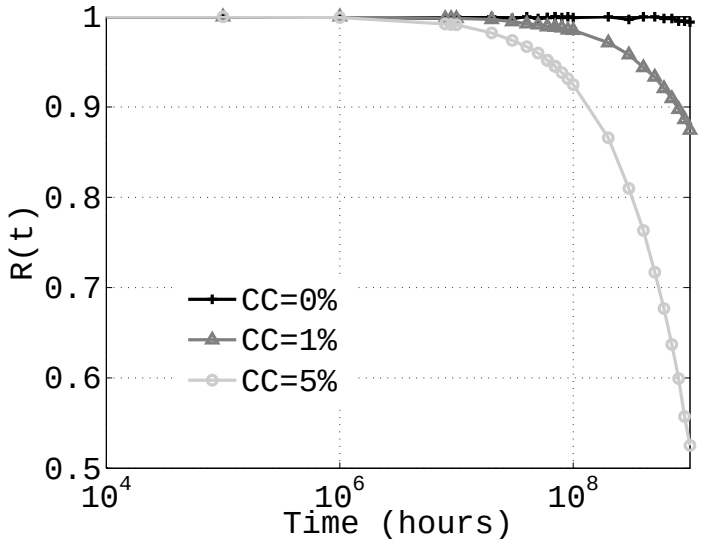
time. Figure 7.4 shows the MTTF for the four fault-tolerant configurations for a mission time of 7 years and a FIT of 1000000 failures per 10^9 hours. In this case the dual TISS configuration has a longer MTTF than the other configurations for non very sensitive components. These failure rate would be only possible on very faulty environment using extremely sensitive technology.

7.3.4 Recovery

The Möbius tool supports the reliability model with exponential distribution of faults over time, using the introduced 4TSoC model and a failure rate of $\lambda = 6.51 \cdot 10^{-9}$ per channel which corresponds to the non-hardened channel Mean Faults To Failures (MFTF) of 15.36 from fault injection results (from Table 7.10) with a hypothetical raw SET probability of 100FIT per channel. From that, it can be concluded that common-mode failures dominate the performance of the on-line recovery approach (Figure 7.5b). The improvement provided by the increase of the repair-rate becomes negligible, as it is 100 times higher than the failure-rate (Figure 7.5a), an easily achievable rate. Therefore, the performance of the on-line recovery resides on the common-mode failure of our design and the probability of failure of the recovery mechanism itself that is out of the scope of this study.



(a) W.r.t. repair-rates.



(b) W.r.t. common-cause.

Figure 7.5: TMR and recovery reliability

Table 7.10: Results for 4TSoC hardening mechanism

	Sing. Chan.	ECC	2TISS (not FS)	2TISS (FS)	2NoC	TMR	2NI+TMR
Flip-Flops	3123	3123	5490	5490	6666	9912	15326
LUTs	3920	3920	9164	9124	10552	12236	16316
BRAMs	352Kbit	352Kbit	792Kbits	792Kbits	792Kbits	1188Kbit	1944Kbits
MFTF	15.36	16.68	33.07	52.10	58.62	41.05	102.80
Normalized MFTF, 1000 FFs	**4.92**	**5.34**	**6.02**	**9.49**	**8.79**	**4.14**	**6.71**
Failures with 1 fault	77	62	17	8	3	0	0
Failures with 2 faults	58	60	19	6	14	1	0
Failures with 3 faults	61	56	22	7	14	2	0
Failures with 4 faults	53	43	27	10	7	9	0

7.4 Summary of the Results

This section reviews the hypotheses formulated with each experiment. Table 7.11, Table 7.12 and Table 7.13 give the hypotheses of the experiments and the results of the subsequent MPSoC approach, TTSoC and 4TSoC evaluations. The tables extract the acceptance criteria (*Pass/Fail* or a *Continous Value*) from the hypothesis.

Table 7.11: Summary of MPSoC Apporach Evaluation

Experiment	Hypothesis	Criteria	Result
Fault Containment Section 6.1.1	**Hyp.1** "The TISSes avoid the propagation of any error between two independently encapsulated components"	*Pass/Fail*	**Pass**: No error propagation for 204800 faults within the 4TSoC
Hypervisor Section 6.1.2	**Hyp.2** "The number of common failures will be significantly higher than the independent partition failures"	*Cont. Value*	The **67-70%** of the XtratuM hypervisor failures are common to all the partitions

Table 7.12: Summary of the TTSoC Evaluation

Experiment	Hypothesis	Criteria	Result
TTSoC Section 6.2.1	**Hyp.3** "The elements of the TSS (the TISSes and the TTNoC) can exhibit value-incorrect failures behavior in the presence of physical faults"	*Pass/Fail*	**Pass**: The TISS exhibits value-incorrect failures in **89.84%** of the failures
TISS Refined Section 6.2.2	**Hyp.4** "The communication load of the TTNoC does not have a significant influence on the TISS reliability against SET faults"	*Cont. Value*	**1.47%** more failures when the load increases 256 times
	Hyp.5 "A low number of internal entities can dominate the reliability of the overall TISS"	*Cont. Value*	The **25%** of the entities causes the **84.7%** of the TISS failures
TMR Section 6.2.3	**Hyp.6** "Simple TMR improves the reliability of the channel, but TTSoC failures due to a single fault on the TSS can still occur"	*Cont. Value*	Out of 1024 experiments: no single fault failures, 1 failure with one fault, 2 failures with three faults. Normalized MTTF: **4.14**

Table 7.13: Summary of the 4TSoC Evaluation

Experiment	Hypothesis	Criteria	Result
ECCs Section 6.3.1	**Hyp.7** "ECCs increase the reliability of the channel"	*Cont. Value*	The ECC protected channel needs **8.6%** more faults to fail. Normalized MTTF: **5.34**
TISS Replication Section 6.3.2	**Hyp.8** "A single fault in one of the replicated TISSes cannot corrupt the communication of the component"	*Pass/Fail*	**Pass**: No single fault failures. Normalized MTTF: **9.49**
NoC Replication Section 6.3.3	**Hyp.9** "A single fault in one of the replicated TSSes (TISSes or NoCs) cannot corrupt the communication of the component"	*Pass/Fail*	**Pass**: No single fault failures. Normalized MTTF: **8.79**
TMR and dual TISS Section 6.3.4	**Hyp.10** "Dual TMR improves the reliability of the dual channel, the probability of failures due to a single fault is decreased"	*Cont. Value*	Normalized MTTF: **6.71**
Recovery Section 6.3.5	**Hyp.11** "Recovery applied to an on-chip TMRed MPSoC increase the availability of the system with the appropriate repair-rate and common-mode failure-rate"	*Pass/Fail*	**Pass**: Reliability increased. The common-cause failure-rate dominates the recovery reliability

"Begia noraino, gogoa haraino",

The ambition goes as far as the eyes do

Chapter 8

Conclusion

This chapter reviews the Transient Tolerant Time-Triggered System-on-Chip (4TSoC) model, discusses the resutls and suggests further areas of research.

8.1 Summary

This thesis presents the Transient Tolerant Time-Triggered System-on-Chip (4TSoC) model, an MPSoC approach to increase the reliability of integrated architectures by a system-level approach.

This MPSoC model is described in Chapter 4 by means of a description of the architecture and the 4T core services, a set of hardened services that support the integration of jobs with an enhanced reliability. Fault tolerance mechanisms based on the replication of components and message level Error Correcting Codes (ECCs) are also introduced upon the 4TSoC model. The actual effectiveness of the fault tolerance mechanisms strongly depends on the technological synthesis of the model. Synthesis patterns of the 4TSoC model are discussed on the basis of spatial separation and rigorous sharing of critical resources (clock, pinout, routes, etc.).

Chapter 5 presents the evaluation tools to assess the features introduced by the 4TSoC model. The Fault Injection for System-on-Chip (FI4SoC) framework is described as the means to perform the reliability assessment though FPGA prototypes at RTL model. A SET-fault model is selected as a generic and interesting fault-model for integrated architecture. The experiments are described on chapter 6. First, analyzing the MPSoC approach versus software based integrated architectures (e.g., hypervisors). Second, evaluating the TTSoC reliability as a reference for the evaluation on 4TSoC, that is also an extension of this architecture. And third, the fault tolerance mechanisms, and indirectly

the 4T communication service, are evaluated to measure the improvement of the 4TSoC model.

Finally, Chapter 7 shows the results on the fault injection campaigns using the FI4SoC framework. These results are classified in the three campaigns:

1. A purely software based integrated architecture (XtratuM hypervisor) approach shows more common mode failures that a hardware MPSoC approach (the TTSoC).

2. The dependability assessment of the TTSoC identifies the TISS, the network interface of the architecture, as the most sensitive element. Further refinements pinpoint the register file, the clock module and the address decode as the critical entities of the TISSes. Regarding the application component TMR, the results show that the central and trusted subsystem of the TTSoC undermine the potential reliability increase by providing a single point of failure.

3. The 4TSoC addresses the reliability increase of MPSoC systems. The use of a fault-tolerant communication service (with duplicated network interface or NoC) achieves this goal by reducing the mean faults needed to make the MPSoC chip fail (i.e., reduce the failure-rate) and the failures with few accumulated faults are reduced. Among the evaluated fault-tolerant configurations for the critical parts of the architecture, the network interface replication shows the most promising results. It shows superior normalized results (considering mean fault to failure and flip-flop size) when applied upon a single channel or a TMR configuration. Nevertheless, the TMR approach with a single TISS is the best option for most of the analyzed application domains due to their short mission time.

8.2 Future Research

This dissertation leaves some open paths for research, in brief:

- *The validation of the 4T resource management and recovery services by fault injection.* So far, only the fault tolerance mechanisms and the communication service of the 4TSoC model have been assessed. The validation of the resource management will require a synthetic application with a change of scheduling for the TTNoC and the submission of the executable program of a component through the network while faults are injected. The recovery services could be tested, for instance, by continuous requests.

- *The on-chip implementation of clock synchronization services, such as, FT clock synchronization or Central Master mechanisms.* FT clock synchronization implementation carries challenges like the provision of enough synchronization messages to every component of the MPSoC. Otherwise, the MPSoC can host several masters and the components could locally synchronize to them.

- *The assessment of the technology synthesis model of the 4TSoC by hardware transient fault injection (e.g., radiation beamer).* Measure the reliability benefits of the spatial separation of components, the hardening of the clock resources and the impact of other shared resources (e.g., pinout, configuration memory) of the MPSoC that can make the whole integrated system fail as a single unit. They are not evaluable through RTL models.

- *The update of the FI4SoC Xilinx Virtex-4 for technology to other Xilinx families (e.g., Spartan 6).* The way to implement the SET fault model flipping the state of the flip-flops is not portable to newer Xilinx series due to the FPGA slice simplification.

- *The study of the combination of virtualization approaches and MPSoCs.* In this dissertation hardware and software approaches to implement integrated architectures have been treated as ends of non mixable technologies. However, virtualization has been already applied to multi-core technologies in order to partition *Commercial Off-The-Shelf* (COTS) chips. For instance the European FP7 project MULTI-PARTES explores this combination of hypervisors and MPSoC on embedded systems.

Bibliography

[AaOMI11] M. Azkarate-askasua, R. Obermaisser, I. Martinez, and X. Iturbe. Dependability assessment of the time-triggered SoC prototype using FPGA fault injection. *Proc. of the 37th Annual Conference of the IEEE Industrial Electronics Society, IECON*, 2011.

[ACD+07] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, and G. R. Sechi. Evaluation of single event upset mitigation schemes for SRAM based FPGAs using the FLIPPER fault injection platform. In *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, DFT*, pages 105–113, 2007.

[Ade03] A. Ademaj. Assessment of the error detection mechanisms of the time-triggered architecture using fault injection. *PhD Disertation. TU Wien*, 2003.

[ALR01] A. Avizienis, J. C. Laprie, and B. Randell. Fundamental concepts of dependability. *Research Report*, 2001.

[ALRL04] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[Alt07] Altera. Robust SEU mitigation with stratix III FPGAs. *White Paper*, 2007.

[And08] B. Andrillon. Contribution of integrated modular avionics of second generation for business aviation. *SCARLETT PROJECT*, 2008.

[AP07] T.W. Ainsworth and T.M. Pinkston. On characterizing performance of the Cell broadband engine element interconnect bus. In *First International Symposium on Networks-on-Chip, NOCS*, pages 18 –29, 2007.

[ARI91] ARINC. ARINC specification 651: Design guide for integrated modular avionics. *Aeronautical Radio, Inc*, 1991.

[ATM+07] M. A. Aguirre, J. N. Tombs, F. MuÃśoz, V. Baena, H. Guzman, J. Napoles, A. Torralba, A. Fernandez-Leon, F. Tortosa-Lopez, and D. Merodio. Selective protection analysis using a SEU emulator: Testing protocol and case study over the Leon2 processor. *IEEE Transactions on Nuclear Science*, 54(4):951–956, 2007.

[BDM02] L. Benini and G. De Micheli. Networks on chips: A new SoC paradigm. *Computer*, 35(1):70–78, 2002.

[BGB+08] J. C. Baraza, J. Gracia, S. Blanc, D. Gil, and P. J. Gil. Enhancement of fault injection techniques based on the modification of VHDL code. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(6):693–706, 2008.

[BK00] G. Bauer and H. Kopetz. Transparent redundancy in the time-triggered architecture. *International Conference on Dependable Systems and Networks, DSN*, pages 5–13, 2000.

[BKH01] J. E. Bartlett, J. W. Kotrlik, and C. C. Higgins. Organizational research: Determining appropriate sample size in survey research. *Information Technology, Learning, and Performance Journal*, 19(1):43–50, 2001.

[BLbG01] E. Boemo, S. Lopez-buedo, and J. Garrido. Measurement of FPGA die temperature using run-time reconfiguration. *in Proceedings of the 7th International Workshop on Thermal Investigations of ICs and Systems*, 2001.

[BLY02] T. Blalack, Y. Leclercq, and C. P. Yue. On-chip RF isolation techniques. In *Proceedings of the IEEE Bipolar/BiCMOS Circuits and Technology Meeting*, pages 205–211, 2002.

[BPP+08] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K.A. LaBel, M. Friendlich, H. Kim, and A. Phan. Effectiveness of internal versus external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis. *IEEE Transactions on Nuclear Science*, 55(4):2259–2266, 2008.

[BWL+06] M. Berg, J.-J. Wang, R. Ladbury, S. Buchner, H. Kim, J. Howard, K. LaBel, A. Phan, T. Irwin, and M. Friendlich. An analysis of single event upset dependencies on high frequency and architectural implementations within Actel RTAX-S family field programmable

gate arrays. *Nuclear Science, IEEE Transactions on*, 53(6):3569 –3574, dec. 2006.

[Cho11] N. Chomsky. Language and other cognitive systems: What is special about language. *Talk in University of Cologne*, 2011.

[CMFC+98] A. B. Campbell, O. Musseau, V. Ferlet-Cavrois, W. J. Stapor, and P. T. McDonald. Analysis of single event effects at grazing angle. *IEEE Transactions on Nuclear Science*, 45:1603–1611, 1998.

[Con02] C. Constantinescu. Impact of deep submicron technology on dependability of VLSI circuits. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 205–209, Washington, DC, 2002.

[Cow01] N. Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, pages 87–114, 2001.

[Cri91] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, 1991.

[CRM+09] A. Crespo, I. Ripoll, M. Masmano, P. Arberet, and J.J. Metge. XtratuM: An open source hypervisor for TSP embedded systems in aerospace. *European Space Agency, (Special Publication) ESA SP*, 669 SP, 2009.

[CS99] C. H. Chen and A. K. Somani. Fault-containment in cache memories for tmr redundant processor systems. *IEEE Transactions on Computers*, 48(4):386–397, 1999.

[DCC+02] D.D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J.M. Doyle, W.H. Sanders, and P.G. Webster. The mobius framework and its implementation. *IEEE Transactions on Software Engineering*, 28(10):956–969, 2002.

[DS01a] B. S. Dhillon and P. Subramanian. Reliability analysis of triple modular computer systems with redundant voters and restricted maintenance. *Journal of Quality in Maintenance Engineering.*, 2001.

[DS01b] B.S. Dhillon and P. Subramanian. Reliability analysis of triple modular computer systems with redundant voters and restricted maintenance. *Journal of Quality in Maintenance Engineering*, 7(2):151–164, 2001.

[Eng07] G. Engleder. Time-triggered network-on-a-chip. *Master thesis, Vienna University of Technology, Faculty of Computer Science, Real-Time Systems Group*, 2007.

[Ern10] R. Ernst. Certification of trusted MPSoC platforms. *MPSoC*, 2010.

[ES04] A. Edman and C. Svensson. Timing closure through a globally synchronous, timing partitioned design methodology. In *Proceedings of the 41st annual Design Automation Conference*, pages 71–74, 2004.

[FCCK06] A.P. Frantz, L. Carro, ÃL. Cota, and F.L. Kastensmidt. Evaluating SEU and crosstalk effects in network-on-chip routers. *Proceedings - 12th IEEE International On-Line Testing Symposium, IOLTS*, 2006:191–192, 2006.

[Gai06] J. Gaisler. The Leon3FT-RTAX processor family and SEU test results. *Proc. of the 9th Annual Military and Aerospace Programmable Logic Devices International Conference*, 2006.

[GAM+02] P. Gil, J. Arlat, H. Madeira, Y. Crouzet, T. Jarboui, K. Kanoun, T. Marteau, J. Duraes, M. Vieira, D. Gil, J. C. Baraza, and J. Gracia. Fault representativeness. *Deliverable (ETIE2) of the European Project Dependability Benchmarking DBench (IST-2000-25425)*, 2002.

[GbR06] AUTOSAR GbR. Autosar. *Technical Overview V2.0.1*, 2006.

[Gel01] P. Gelsinger. Microprocessors for the new millenium, challenges. opportunities, and new frontiers. *Proc. of the Solid State Circuit Conference*, 2001.

[GH10] K. Goossens and Hansson. The aEthereal Network-on-Chip after ten years, evolution, lessons, and future. *Proc. Design Automation Conference, DAC*, 2010.

[GHB10] R. Girardey, M. Hubner, and J. Becker. Safety aware place and route for on-chip redundancy in safety critical applications. *Proceedings - IEEE Annual Symposium on VLSI, ISVLSI*, pages 74–79, 2010.

[GMSW09] G. Griessnig, R. Mader, C. Steger, and R. Weiß. Fault insertion testing of a novel CPLD-based fail-safe system. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 214–219, 2009.

[GSVP03] M. Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz. Transient-fault recovery for chip multiprocessors. *Conference Proceedings - Annual International Symposium on Computer Architecture, ISCA*, pages 98–109, 2003.

[GT04] M. Garvie and A. Thompson. Scrubbing away transients and jiggling around the permanent: Long survival of FPGA systems through evolutionary self-repair. In C. Metra, R. Leveugle, M. Nicolaidis, and J. P. Teixeira, editors, *Proceedings - 10th IEEE International On-Line Testing Symposium, IOLTS*, pages 155–160, 2004.

[HGBH09] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken. CoMPSoC: A template for composable and predictable multi-processor system on chips. *ACM Transactions on Design Automation of Electronic Systems*, 14(1), 2009.

[HT98] G. Heiner and T. Thurner. Time-triggered architecture for safety-related distributed real-time systems in transportation systems. *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing*, pages 402 –407, 1998.

[Hub08] B. Huber. Resource management in an integrated time-triggered architecture. *PhD Thesis, Vienna University of Technology, Faculty of Computer Science, Real-Time Systems Group*, 2008.

[IBM07] IBM. Cell broadband engine architecture. *Version 1.02*, 2007.

[IEC09] IEC. Special architecture requirements for integrated circuits (ICs) with on-chip redundancy. *IEC-61508-2, Annex E*, 2009.

[IEE05] IEEE. IEEE standard testability method for embedded core-based integrated circuits. *IEEE Std 1500-2005*, 2005.

[Jon02] C. et al. Jones. Final version of the DSoS conceptual model. *DSoS Project (IST-1999-11585)*, 2002.

[K.11] Sravan K. Multi-core processor penetration in smartphones will hit 15 percent in 2011. *White Paper, Strategy Analytics*, 2011.

[KKOE07] K. KronlÃűf, S. Kontinen, I. Oliver, and T. Eriksson. A method for mobile terminal platform architecture development. *Advances in Design and Specification Languages for Embedded Systems*, pages 285–300, 2007.

[KN97] H. Kopetz and R. Nossal. Temporal firewalls in large distributed real-time systems. In *Distributed Computing Systems, 1997., Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of*, pages 310–315, 1997.

[KOESH07] H. Kopetz, R. Obermaisser, C. El Salloum, and B. Huber. Automotive software development for a multi-core system-on-a-chip. In *Fourth International Workshop on Software Engineering for Automotive Systems, SEAS*, 2007.

[Kop06] H. Kopetz. Mitigation of transient faults at the system level-the TTA approach. *Proc. 2nd Workshop on System Effects of Logic Soft Errors*, 2006.

[Kop07] H. Kopetz. Why do we need a sparse global time-base in dependable real-time systems? In *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, ISPCS*, 2007.

[Kop08a] H. Kopetz. The ARTEMIS technology platform. *IST Conference*, 2008.

[Kop08b] H. Kopetz. The complexity challenge in embedded system design. Invited paper. In *Proceedings - 11th IEEE Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC*, pages 3–12, 2008.

[Kop11] H. Kopetz. Real-time systems, design principles for distributed embedded applications. *Springer Book, 2nd Edition*, 2011.

[KOPS04] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri. From a federated to an integrated architecture for dependable real-time embedded systems. *Research Report*, 2004.

[KPP06] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *IEEE Micro*, 26(3):10–23, 2006.

[LBN10] U. Legat, A. Biasizzo, and F. Novak. Automated SEU fault emulation using partial FPGA reconfiguration. *IEEE 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS*, pages 24 –27, 2010.

[LH94] J. H. Lala and R. E. Harper. Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82(1):25–40, 1994.

[Lie95] J. Liedtke. On microkernel construction. *Proceedings of the 15th ACM Symposium on Operating System Principles, SOSP*, 1995.

[ME02] D.G. Mavis and P.H. Eaton. Soft error rate mitigation techniques for modern microcircuits. *Annual Proceedings - Reliability Physics Symposium*, pages 216–225, 2002.

[MER05] S. S. Mukherjee, J. Emer, and S. K. Reinhardt. The soft error problem: An architectural perspective. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 243–247, 2005.

[Mes89] M.D. Mesarovic. Abstract system theory. *Lecture Notes in Control and Information Science. Springer Verlag*, 1989.

[MG96] Weiwei Mao and Ravi K. Gulati. Improving gate level fault coverage by RTL fault grading. *IEEE International Test Conference (TC)*, pages 150–159, 1996.

[MRCP10] M. Masmano, I. Ripoll, A. Crespo, and S. Peiro. XtratuM for LEON3 : an open source hypervisor for high integrity systems. *ERTS*, 2010.

[MSK+08] P. Mangalagiri, Bae Sungmin, R. Krishnan, Xie Yuan, and V. Narayanan. Thermal-aware reliability analysis for platform FPGAs. In *IEEE/ACM International Conference on Computer-Aided Design, ICCAD*, pages 722–727, 2008.

[MSZ+05] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim. Robust system design with built-in soft-error resilience. *Computer*, 38(2):43–52, 2005.

[Mul11] MultiPARTES. Multicores partitioning for trusted embedded systems. *Factsheeet*, 2011.

[Neu06] D. Neumann. Intel virtualization technology in embedded and communications infrastructure applications. *Intel Technology Journal*, 10(3):217–226, 2006.

[NTS+08] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. Deprettere. Daedalus: toward composable multimedia MP-SoC design. In *Proceedings of the 45th annual Design Automation Conference*, pages 574–579, 2008.

[Obe08] R. Obermaisser. Temporal partitioning of communication resources in an integrated architecture. *IEEE Transactions on Dependable and Secure Computing*, 5(2):99–114, 2008.

[OK09] R. Obermaisser and H. Kopetz. A candidate for an ARTEMIS cross-domain reference architecture for embedded systems. *Sudwestdeutscher Verlag*, 2009.

[OKP10] R. Obermaisser, H. Kopetz, and C. Paukovits. A cross-domain multiprocessor system-on-a-chip for embedded real-time systems. *IEEE Transactions on Industrial Informatics*, 2010.

[OKS08] R. Obermaisser, H. Kraut, and C. Salloum. A transient-resilient system-on-a-chip architecture with support for on-chip and off-chip tmr. In *Proceedings - 7th European Dependable Computing Conference, EDCC*, pages 120–134, Kaunas, 2008.

[OP06] R. Obermaisser and P. Piti. A fault hypothesis for integrated architectures. In *Proceedings of the Fourth Workshop on Intelligent Solutions in Embedded Systems, WISES*, pages 47–64, 2006.

[OSHK08] R. Obermaisser, C. E. Salloum, B. Huber, and H. Kopetz. The time-triggered system-on-a-chip architecture. In *IEEE International Symposium on Industrial Electronics*, pages 1941–1947, 2008.

[Pau08] C. Paukovits. The time-triggered system-on-chip architecture. *PhD Dissertation*, 2008.

[Per09] J. Perez. Codesign and simulated fault injection of safety-critical embedded systems using systemc. *European Dependable Computing Conference, EDCC*, 2009.

[Per11] J. Perez. Executable Time-Triggered Model (E-TTM) for the development of safety-critical embedded systems. *PhD Dissertation. TU Wien*, 2011.

[PK08] C. Paukovits and H. Kopetz. Concepts of switching in the time-triggered network-on-chip. In *Proceedings - 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA*, pages 120–129, Kaohsiung, 2008.

[PKCC06] F. A Pereira, F. L. Kastensmidt, L. Carro, and E. Cota. Dependable network-on-chip router able to simultaneously tolerate soft errors and crosstalk. *IEEE International Test Conference.*, pages 1–9, 2006.

[PMH98] B. Pauli, A. Meyna, and P. Heitmann. Reliability of electronic components and control units in motor vehicle applications. *VDI Berichte*, pages 1009–1024, 1998.

[Pol95] S. Poledna. Fault-tolerant real-time systems, the problem of replica determinism. *Springer Verlag*, 1995.

[PPB+07] F. Poletti, A. Poggiali, D. Bertozzi, L. Benini, P. Marchal, M. Loghi, and M. Poncino. Energy-efficient multiprocessor systems-on-chip for embedded computing: Exploring programming models and their architectural support. *IEEE Transactions on Computers*, 56(5):606–621, 2007.

[PS03] M. Paulitsch and W. Steiner. Fault-tolerant clock synchronization for embedded distributed multi-cluster systems. *15th Euromicro Conference on Real-Time Systems*, 2003.

[RE06] B. Rumpler and W. Elmenreich. Considerations on the complexity of embedded real-time system design tasks. In *IEEE International Conference on Computational Cybernetics, ICCC*, 2006.

[RMG+07] B. Rousseau, Ph Manet, D. Galerin, D. Merkenbreack, J. D. Legat, F. Dedeken, and Y. Gabriel. Enabling certification for dynamic partial reconfiguration using a minimal flow. In *Proceedings - Design, Automation and Test in Europe, DATE*, pages 983–988, Nice Acropolis, 2007.

[RSB90] P. Ramanathan, K. G Shin, and R. W Butler. Fault-tolerant clock synchronization in distributed systems. *Computer*, pages 1106–1112, 1990.

[RTS10] Real-Time-Systems. RTS real-time embedded hypervisor. *http://www.real-time-systems.com/*, 2010.

[Rus99] J. Rushby. Partitioning in avionics architectures: Requirements, mechanisms, and assurance. *NASA Langley Technical Report Server*, 1999.

[SAS+04] E. Schoitsch, E. Althammer, G. Sonneck, H. Eriksson, and J. Vinter. Support for modular certification of safety-critical embedded systems in DECOS - the generic safety case. *IEEE International Conference on Industrial Informatics, INDIN*, 2004.

[SATGM08] L. Sterpone, M. Aguirre, J. Tombs, and H. Guzman-Miranda. On the design of tunable fault tolerant circuits on sram-based fpgas

for safety critical applications. In *Proceedings -Design, Automation and Test in Europe, DATE*, Design, Automation and Test in Europe, DATE, pages 336–341, Munich, 2008.

[Sch07] M. Schoeberl. A time-triggered network-on-chip. In *Proceedings - International Conference on Field Programmable Logic and Applications, FPL*, pages 377–382, Amsterdam, 2007.

[Sim62] H. A. Simon. The architecture of complexity. In *Proceedings of the American Philosophical Society*, pages 467–482, 1962.

[SKK+02] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 389–398, 2002.

[SNJ97] H. R. Schwartz, D. K. Nichols, and A. H. Johnston. Single-event upset in flash memories. *IEEE Transactions on Nuclear Science*, 44(6 PART 1):2315–2324, 1997.

[ST11] SEU-TEST. Soft-error testing resources @ONLINE. *http://www.seutest.com/*, 2011.

[SV07] L. Sterpone and M. Violante. A new partial reconfiguration-based fault-injection system to evaluate SEU effects in SRAM-based FPGAs. *IEEE Transactions on Nuclear Science*, 54(4):965–970, 2007.

[Tok03] J. L. Tokar. Space & time partitioning with ARINC 653 and pragma profile. *Ada Lett.*, XXIII:52–54, 2003.

[TU08] J.S. Teller and The Ohio State University. *Scheduling tasks on heterogeneous chip multiprocessors with reconfigurable hardware.* The Ohio State University, 2008.

[Win01] A. T. Winfree. The geometry of biological time. *Springer Verlag.*, 2001.

[Xil] Xilinx. *Virtex-4 User Guide.*

[Xil06] Xilinx. Xilinx tmrtool user guide. *TMRTool Software Version 8.2i*, 2006.

[Xil08] Xilinx. Virtex-5 FPGA. *User Guide*, 2008.

[Xil11] Xilinx. Device reliability report. *Third Quarter 2011*, 2011.

[XR96] J. Xu and B. Randell. Roll-forward error recovery in embedded real-time systems. *Parallel and Distributed Systems, 1996. Proceedings., 1996 International Conference on*, pages 414 –421, 1996.

Selected Publications

- Mikel Azkarate-askasua, Roman Obermaisser, Imanol Martinez, Xabier Iturbe **Dependability Assessment of the Time-Triggered SoC Prototype using FPGA Fault Injection**. *The 37th Annual Conference of the IEEE Industrial Electronics Society, IECON, 2011, Australia.*

- Mikel Azkarate-askasua, Roman Obermaisser, Xabier Iturbe, Imanol Martinez **FI4SoC: A Fault Injection Framework for Transient Fault Effects in Embedded MPSoCs**. *9th IEEE Workshop on Intelligent Solutions in Embedded Systems, WISES, 2011, Germany.*

- Mikel Azkarate-askasua, Roman Obermaisser, Imanol Martinez, Xabier Iturbe **Suitability of hypervisor and MPSoC architectures for the execution environment of an integrated embedded system**. *14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, ISORCW, 2011, USA.*

- Jon Perez, Mikel Azkarare-askasua, Antonio Perez. **Codesign and simulated fault injection of safety-critical embedded systems using SystemC**. *8th European Dependable Computing Conference, EDCC, 2010, Spain.*

- Xabier Iturbe, Mikel Azkarate-askasua, Imanol Martinez, Jon Perez, and Armando Astarloa. **A novel SEU, MBU and SHE handling strategy for Xilinx Virtex-4 FPGAs**. *19th International Conference on Field Programmable Logic and applications, FPL, 2009, Czech Republic.*

Curriculum Vitae

Details

Name: Mikel, Azkarate-askasua (*azkarate - gmail - com*)
Birth: 1984-06-24

Education

2009 - present
Teschnische Universität Wien (TU Wien, Austria)
Institute of Computer Engineering, Real-Time Systems Group
PhD: The Transient Tolerant Time-Triggered System-on-Chip (4TSoC)

2011 - 2011
Stage in Universität Siegen (Uni-Siegen, Germany)
Embedded Systems Group

2006 - 2008
École Nationale Superieur of Bordeaux (ENSEIRB, France)
M.Sc. in Embedded Systems

2002 - 2006
Mondragon Unibertsitatea (MU, Spain)
B.Eng. in Industrial Electronics

Professional Experience

2008 - present
Ikerlan Research Center (Mondragon, Spain)
Research focus on safety-critical embedded systems (IEC-61508, etc.), MPSoCs and FPGAs

2008 - 2008
Master Thesis in the Technische Universiteit Delft (The Netherlands)
JPEG2000 image compression in MPSoC

2004 - 2006
Ikerlan Research Center (Mondragon, Spain)
Bachelor Thesis: Time-Triggered Embedded Systems
Student Collaborator with FAGOR S.COOP