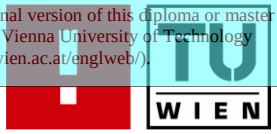

FAKULTÄT FÜR !NFORMATIK

Evaluation of Temporal and Spatial Partitioning in the Time-Triggered System-on-a-Chip Architecture

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Oliver Höftberger
Matrikelnummer 0325723

an der
Fakultät für Informatik der Technischen Universität Wien

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

Wien, 28.01.2010

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Oliver Höftberger
Hildebrandgasse 30/1/2
1180 Wien

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.

Wien, 28. Jänner 2010

_____
(Unterschrift Verfasser)

Evaluation of Temporal and Spatial Partitioning in the Time-Triggered System-on-a-Chip Architecture

Abstract

Temporal and spatial partitioning ensure that one component cannot interfere with the correct behavior of other components in the value and time domain. The time-triggered system-on-chip (TTSoC) architecture provides a framework for the design and implementation of systems-on-chip (SoCs) with inherent temporal and spatial partitioning. Multiple heterogenous IP-cores are interconnected by a time-triggered network-on-chip (TTNoC), which uses a precise interface specification to encapsulate the communication activities of components. To dynamically adjust the system to changing communication and power requirements, integrated resource management is provided. Within this thesis, the effectiveness of temporal and spatial partitioning in the TTSoC architecture is investigated. Therefore, an experimental FPGA-based setup is designed using the TTSoC execution platform. This setup allows the injection of faults in the system, while monitoring the behavior of components. The system is exposed to different load scenarios, bit flips that simulate transient and permanent faults, and reconfiguration scenarios, to observe the system behavior in the presence of faults. The results of the experiments provide evidence for the correctness of temporal and spatial partitioning and demonstrate the suitability of the TTSoC architecture as an execution platform for component-based design.

Evaluierung der Zeitlichen und Räumlichen Abgrenzung in der Time-Triggered System-on-a-Chip Architektur

Kurzfassung

Zeitliche und räumliche Abgrenzung stellen sicher, dass eine Komponente das korrekte Verhalten anderer Komponenten im Werte- und Zeitbereich nicht stören kann. Die Time-Triggered System-on-Chip (TTSoC) Architektur bietet die Grundlage für die Entwicklung und Realisierung von Systems-on-Chip (SoC) mit inhärenter zeitlicher und räumlicher Abgrenzung. Mehrere verschiedenartige IP-cores werden mittels eines Time-Triggered Network-on-Chip (TTNoC) verbunden, welches eine präzise Schnittstellenspezifikation nutzt um die Kommunikation von Komponenten abzukapseln. Um dynamisch auf sich verändernde Kommunikations- und Leistungsbedingungen reagieren zu können, wird integriertes Ressourcenmanagement bereitgestellt. In dieser Arbeit wird die Effektivität der zeitlichen und räumlichen Abgrenzung in der TTSoC Architektur untersucht. Deshalb ist eine experimentelle, FPGA basierte SoC Struktur entwickelt worden, die die TTSoC Architektur als Plattform verwendet. Dieser Aufbau ermöglicht das absichtliche Einstreuen von Fehlern in das System, während das Verhalten von Komponenten beobachtet wird. Das System wird verschiedenen Belastunsszenarien, Bit Flips, welche transiente und permanente Fehler simulieren, und Rekonfigurationsszenarien ausgesetzt, um zu beobachten wie sich das System in der Gegenwart von Fehlern verhält. Die Ergebnisse der Experimente zeigen die Korrektheit zeitlicher und räumlicher Abgrenzung, und demonstrieren die Eignung der TTSoC Architektur als Plattform für die Entwicklung von zuverlässigen Systemen bestehend aus Einzelkomponenten.

Acknowledgments

At the beginning, I want to thank all of those who contributed in one or the other way to the realization of this thesis. It's been hard work with many ups and downs, but at the end I was able to finish this work with the support and help of many people.

First of all, I have to express my gratitude to the advisor of my thesis, Priv. Doz. Dipl.-Ing. Dr. Roman Obermaisser, who gave me the opportunity to accomplish my experiments and finally write this document. He patiently provided me with valuable comments and positive criticism, and thereby, raised my interest to continue my academic career.

Thanks to my colleagues and stuff at the Department of Computer Engineering at the Vienna University of Technology, and particularly to Christian Paukovits, who supported me within countless discussions.

I'm especially grateful to my girlfriend Martina, as her love encouraged me every day more to make progress with this thesis. Also thanks for proof-reading the document during the past days.

Furthermore, I would like to thank my brother Daniel for proof-reading my thesis, and thus, contributing to improve my work.

Special thanks to my parents, not only for making my educational and academic career possible, but also for their affection, support and valuable advices during my whole lifetime.

– Oliver Höftberger

Vienna, January 2010

Danksagung

Zu Beginn möchte ich allen danken, die auf die eine oder andere Weise zur Durchführung dieser Diplomarbeit beigetragen haben. Es war viel Arbeit, mit ebenso vielen Höhen und Tiefen, aber am Ende konnte ich diese Arbeit mit der Unterstützung und der Hilfe vieler Leute erfolgreich abschließen.

Zuerst muss ich dem Betreuer meiner Diplomarbeit, Priv. Doz. Dipl.-Ing. Dr. Roman Obermaisser, meinen Dank aussprechen, der mir die Möglichkeit gegeben hat, meine Experimente durchzuführen und schließlich dieses Dokument zu verfassen. Geduldig unterstützte er mich mit wertvollen Kommentaren und konstruktiver Kritik, und steigerte dadurch mein Interesse für eine weitere akademische Laufbahn.

Dank auch an meine Kollegen und die Mitarbeiter des Instituts für Technische Informatik der Technischen Universität Wien, und vor allem an Christian Paukovits, für die Unterstützung in unzähligen Diskussionen.

Ganz besonders dankbar bin ich meiner Freundin Martina, da ihre Liebe mich jeden Tag aufs Neue dazu ermutigt, an meiner Diplomarbeit weiter zu arbeiten. Ich danke ihr außerdem, dass sie in den letzten Tagen dieses Dokument für mich durchgelesen und korrigiert hat.

Weiters möchte ich meinem Bruder Daniel dafür danken, dass er meine Diplomarbeit korrekturgelesen, und so zur Verbesserung meiner Arbeit beigetragen hat.

Speziell möchte ich noch meinen Eltern danken, nicht nur dafür, dass sie mir meine schulische und akademische Laufbahn ermöglicht haben, sondern auch für deren Zuneigung, Unterstützung und wertvolle Ratschläge in meinem bisherigen Leben.

– Oliver Höftberger

Wien, Jänner 2010

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the past decades the semiconductor industry developed chips with ever increasing complexity and number of transistors per chip, while the management of complexity becomes more and more challenging. To reduce the *cognitive complexity* [Kop08] of building embedded computer systems, component-based design methodologies have been introduced, which allow prevalidated components to be combined. Multiple components are integrated into a single chip to build a system-on-chip (SoC), which additionally helps to stop the increase in the number of node computers [Obe07]. An embedded execution platform is required for the integration of multiple heterogenous components, which are interconnected by a network-on-chip (NoC). For this purpose the time-triggered system-on-chip (TTSoC) architecture was developed at the Institute of Computer Engineering at the Vienna University of Technology. One of the key properties of this architecture is to support *composability* [KO02], i.e., the composition of component services without side-effects. Several independent functional units can be integrated into a single chip where the communication amongst these components is based on a time-division-multiple-access (TDMA) scheme. As a consequence, self-contained functional units in the system cannot be influenced by other malfunctioning units. The architecture presents a predictable and deterministic interconnection with inherent error containment, which makes the development of complex systems easier [OSHK08]. To satisfy changing communication and power demands of individual components, the TTSoC architecture provides support for integrated resource management.

1.1 Motivation

It is the objective of this thesis to provide evidence that one component cannot interfere with the correct behavior of other components in the value and time do-

main. Temporal partitioning guarantees the independence of temporal properties of messages exchanged by distinct subsystems, while spatial partitioning assures that no data is invalidated in the communication subsystem. Therefore, the focus of this work is to evaluate those mechanisms, that are designed to ensure the independence of individually designed and validated components.

To obtain adequate results, an experimental framework is developed, upon which distinct fault injection experiments are executed. Using fault injection, faults are intentionally introduced to simulate faulty behavior of subsystems. The behavior of distinct components in the framework, in the presence of such adverse conditions, gives information about the correctness of hypotheses concerning temporal and spatial partitioning.

1.2 Structure of the Thesis

Chapter 2 explains the basic concepts of the thesis. The reader is introduced to the terminology which will be used in subsequent chapters.

Thereafter, in chapter 3 the architectural concept and fundamental properties of the TTSoC architecture are presented. Especially, the partitioning in the TTSoC architecture is detailed.

An overview of related work is given in chapter 4. Different existing solutions for SoC architectures are described and finally compared to the TTSoC architecture.

The actual hypotheses, which have to be evaluated by this thesis, are declared in chapter 5. Three different experimental scenarios are explained that are used to verify the correctness of these hypotheses. Additionally, a detailed description of the experimental framework gives an insight into the functionality of the evaluation process.

Chapter 6 presents the results of the experiments. For each experimental scenario the outcome is illustrated by multiple diagrams and tables.

In chapter 7 the results are interpreted and discussed with respect to the validity of hypotheses. At last, chapter 8 brings a conclusion of the presented work.

Chapter 2

Basic Concepts

In this chapter background information is presented to introduce different terms and expressions used in this thesis. The first section introduces different terms of dependability. Then, in the second section, the purpose and required attributes of fault hypotheses are explained. The next section gives an overview about distinct viewpoints of the partitioning of systems. In section 2.4 the purpose and the process of fault injection will be discussed.

2.1 Dependability

Dependability is defined as the ability to trust a computer system in the way that its user can rely on the correct delivery of services. The service of a system is its behavior as the user can observe it. A user may either be a human or another computer system [Lap92].

In [Lap95] different aspects of dependability were elaborated, which are then grouped into the three classes: *impairments, means, attributes.*

2.1.1 Impairments

Impairments to dependability – *faults, errors, failures* – are circumstances which cause or come from the deviation of the delivered service from its specification. These circumstances are undesired but not necessarily unexpected.

When the actual delivered service deviates from the specified or intended system function, a system *failure* occurred. The unintended internal state of the system, which is responsible for the system failure, is called *error*. A *fault* is the cause of an error and thus, indirectly liable for the occurrence of subsequent failures [ALR00]. In a system faults, may reside (e.g., software faults), that have not been

activated yet. Thus the fault does not yet have any effect on the system behavior. Before such a fault results in an error, it needs to be activated. In case of a system composed of multiple components, the failure of one component may propagate to cause a fault in another component, and thus, the whole system may fail. Figure 2.1 illustrates the fault-error-failure chain in a system with multiple components. To prevent this propagation of failures, a well designed system contains distinct fault containment regions (see subsection 2.2.1).

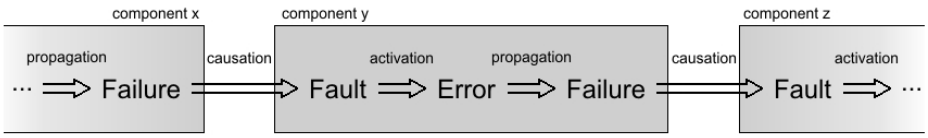

Figure 2.1: Pathology of faults.

According to the phase in which a fault is created, there is a distinction between *development faults* and *operational faults* [Lap92]. Development faults can further be divided into *design faults* – faults introduced during the design phase – and *implementation faults* – faults introduced during the system implementation. Operational faults appear while the system is in operation and may have physical reasons or may also be caused by user interactions [Ade03].

The persistence of faults is another important classification. Faults that occur only once and remain in the system for some time before they disappear without repair are called *transient faults*. These faults often have physical reasons like radiation, lightning strokes or electromagnetic interferences. If a transient fault affects the state of a system it can also result in a permanent error [Kop97]. In contrast, a *permanent fault* appears and remains in the system until an explicit repair action is applied. Examples are damaged connections, functional design errors or hardware defects due to manufacturing problems. Between these classes, the group of *intermittent faults* can be identified as permanent faults that appear like transient faults. A cause of intermittent faults can be, e.g., loose connections, critical timing or aging of components [Ade03].

For a more detailed description of faults the interested reader is referred to [Lap92].

2.1.2 Means

The means for dependability are a set of methods and techniques to improve dependability. In [Lap92] four different classes of these methods and techniques are introduced:

> **Fault prevention:** is concerned with the prevention of the occurrence and introduction of faults.

Fault tolerance: tries to guarantee a correct system service despite the presence of faults.

Fault removal: means to reduce the number and seriousness of present faults in the system.

Fault forecasting: comprises methods and techniques to estimate the present number, future incidence and the consequences of faults.

2.1.3 Attributes

There are five important attributes of dependability which were defined in [Lap92]: *reliability, safety, maintainability, availability* and *security.*

Reliability: This attribute is defined by the continuity of the service. It specifies the probability that a system provides its intended service up to time t when the system was in operation at $t = t_0$. With a constant failure rate of λ failures/hour, the reliability is given by [Kop97]:

$$R(t) = \frac{1}{e^{\lambda(t-t_0)}} \tag{2.1}$$

The mean-time-to-failure (MTTF) is given in hours and is the inverse of the failure rate: $1/\lambda$.

Safety: The safety of a system regards the prevention of critical failure modes in a system. Especially the avoidance of catastrophic consequences on the environment is of concern. The cost of a critical failure can be orders of magnitude higher than the expected utility when the system is operational.

Maintainability: A system that can be repaired after the occurrence of a failure is said to be maintainable. The probability $M(d)$ that the system will be restored within the duration d after a failure is a measure of maintainability. Similarly to reliability, the mean-time-to-repair (MTTR) is defined as the inverse of the constant repair rate μ (repairs/hour).

Availability: The correct operation of a system is sometimes disrupted by durations where the system does not provide its intended function. As a measure of the alternation of correct and incorrect delivery of system service, the attribute of availability A is introduced. It defines the fraction of time that the system is ready for use.

If there are constant failure and repair rates in the system, the availability is defined as:

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2.2}$$

The sum in the denominator of the equation ($MTTF + MTTR$) is also called the mean-time-between-failures (MTBF).

Security: The ability of a system to prevent unauthorized disclosure of information and improper alterations of data is referred to as security.

2.2 Fault hypothesis

During the design and validation of fault-tolerant systems the *fault hypothesis* is of high importance. It specifies assumptions about the behavior of a system in the presence of faults [Ade03]. Assumptions on the types of faults, the rate at which components fail and how components fail are the core of the fault hypothesis [OP06]. The *assumption coverage* [Pow92] defines at which degree these assumptions hold in reality. If the assumptions made in the fault hypothesis are not correct, the whole system may fail as the fault-tolerance mechanisms of a safety-critical system are built upon these assumptions.

The next subsection defines the *unit of failure* – the fault containment regions – of a system. Afterwards, the different *failure modes* are specified. The last three subsections describe assumptions about the *failure rate*, *maximum number of failures* and the *recovery interval*.

2.2.1 Fault containment region (FCR)

A fault containment region (FCR) is a set of components that may either fail as an atomic unit or works correctly regardless of any arbitrary logical or electrical fault outside the region [KPJ+01][LH94]. In contrast, no other component outside the FCR is influenced by a fault inside the FCR. By the usage of shared physical resources (e.g., power supply, timing source), the occurrence of external faults (e.g., electromagnetic interference, spatial proximity) or an improper system design can compromise the independence of FCRs [OP06].

A fault inside one FCR that results in the failure of that FCR may become a fault at system-level (compare the fault-error-failure chain in subsection 2.1.1). The fault-tolerance mechanisms employed in the system need to be able to tolerate such system-level faults. Otherwise, the system-level fault may lead to the failure

of the whole system. A typical approach to cope with the failure of components is to use N-modular redundancy [Avi71]. The service request is processed by N replicated components that all provide the equal service. A voting mechanism either decides (e.g., based on majority) which of the outcomes of the replicas to use or it combines the results to one singe output (e.g., average value). Triple-modular redundancy (TMR) is the most frequently used variant of N-modular redundancy.

2.2.2 Failure mode assumption

When an FCR fails, different effects of this failure can be observed by the user of the FCR's services. These effects can be categorized to define distinct *failure modes* [OP06][Cri91]. Based on the assumed failure modes, the degree of redundancy necessary to guarantee correct error processing, can be identified. The following hierarchy of failure modes is based on the rigidity of assumptions:

Fail-stop failures: When an FCR omits to produce any output to subsequent inputs a fail-stop failure occurred. The FCR needs to be restarted before it returns to its intended behavior. A fail-stop failure is detected by all correct operating FCRs.

Crash failures: A crash failure is very similar to fail-stop failures: the FCR does not produce any outputs. The difference is that crash failures possibly are not detected by correct FCRs.

Omission failures: In case of an omission failure, the sending FCR fails to send a message, or the receiving FCR does not receive a message which was previously sent. Consequently, the receiver cannot respond to an input. Similarly to crash failures, omission failures may remain undetected.

Timing failures: An FCR that suffers a timing failure produces its output too early or too late relating to the temporal specification.

Byzantine or Arbitrary failures: For the byzantine failure no restrictions can be applied. Even forging of messages and "two-faced" FCR behavior must be considered. This behavior is described as the problem of Byzantine Generals [LSP82].

Additionally, the following different types of failure modes can be distinguished [Kop03][OP06]:

Babbling Idiot: When a babbling idiot failure occurs, the FCR sends untimely messages. Thus, the communication channel may be monopolized by the FCR.

Slightly-off-Specification (SoS): This is a special kind of byzantine failure which can be differentiated into temporal and value SoS failures. An example for a value SoS failure is, when the electrical voltage is close to the threshold between logical 0 and logical 1. The value can be judged to be 0 by one observer and to be 1 by the other recipient. A temporal SoS failure may arise when a message is slightly outside its specified interval of the receive instant. Due to the fact, that clocks cannot be perfectly synchronized, one of two different receivers accepts the message while the other receiver detects a message timing failure.

Masquerading: This is a failure where one component uses the identity of another component without authority to send or receive messages.

2.2.3 Failure rate assumption

The specification of the arrival rate of failures of FCRs is another part of the fault hypothesis. It defines the temporal distance between the occurrence of two successive failures, without compromising the correct operation of the system [Ade03]. For the failure rate assumption, both must be considered, the different failure modes and the persistence of failures. Transient failures disappear by themselves while permanent failures require an explicit repair action.

2.2.4 Maximum number of failures

The fault hypothesis must also include the maximum number of failures the system can handle. The number of tolerated failures determines the level of redundancy in the system. It depends on the failure rate and the recovery interval of FCRs. Typically, the occurrence of one single failure is assumed in present day safety-critical systems [OP06].

2.2.5 Recovery interval of an FCR

After the occurrence of a failure in an FCR, it must recover to provide the specified functionality again. The maximum duration between the failure of the FCR and the instant where the FCR operates correctly again is called the recovery interval. For permanent failures, the recovery interval either equals the mission time or the duration between maintenance actions. After a transient failure, the sum of the failure detection latency, the FCR restart duration and the state restoration duration build the recovery interval [OP06].

2.3 Partitioning

The partitioning of a system into different fault containment regions (FCRs) is done to ensure fault containment. These FCRs need to be independent from each other, so that a failure in one FCR does not propagate to cause a failure in another FCR [Obe07]. In an inadequately partitioned system a fault in one FCR could corrupt code, control signals or data owned by another FCR. On the other hand, it could also have an impact on the ability of another FCR to access shared resources (e.g., processor, communication subsystem).

When a system is partitioned, two dimensions of partitioning can be distinguished, according to the two types of hazards which may arise [Rus99]:

Spatial partitioning: Spatial partitioning prevents that one FCR changes private data of another FCR (in memory or in transit). Additionally, the FCR is not allowed to command private devices nor actuators owned by different FCRs.

Temporal partitioning: With temporal partitioning it is ensured that one FCR cannot affect the access of other FCRs to shared resources (e.g., shared CPU, communication network). Also the temporal behavior of the services of these resources may not be altered (e.g., rate of access, latency, jitter).

In a well partitioned system the effort of system integration, verification and validation can be reduced [WH09]. Due to the separation of functionality to distinct FCRs, where faults are contained and isolated, the components can be verified independently and the system integration can be carried out with previously tested components. This loose coupling of FCRs also allows the coexistence of components with different criticality levels.

2.4 Fault Injection

Dependability evaluation of fault-tolerant computer systems, based on the analysis of failures and errors in operational systems, is quite difficult. The destructiveness of severe failures and a long error latency complicate the detection of reasons of failures in operational systems. *Fault injection* is a widely accepted technique for dependability and robustness evaluation of computer systems, that can eliminate these drawbacks [Koo02]. It can be used to determine the behavior of the system in the presence of faults by deliberately introducing faults into the system, and thus, accelerating the occurrence of faults [AAA+90].

Using fault injection techniques it is possible to [HTI97]:

- identify dependability bottlenecks
- analyze the system behavior in the presence of faults
- find out the error detection coverage and effectiveness of recovery capabilities
- determine the effectiveness of fault tolerance mechanisms

Typically, a *fault injection environment* consist of the target system (whose dependability is to be evaluated), a *fault injector*, *fault library*, *workload generator with workload library*, a *controller*, *monitor*, *data collector* and a *data analyzer*. While the target system executes commands provided by the workload generator, the fault injector introduces faults into the system (e.g., by corrupting memory or interfering with computational logic). A monitor keeps track of the execution of commands and eventually triggers the data collector to sample information. The collected data can either be processed online or offline by the data analyzer. It is the controllers duty to guide the evaluation process.

Depending on the design phase in which fault injection is applied and the techniques used, different types of fault injection can be distinguished: *simulation-based fault injection*, *hardware fault injection* and *software implemented fault injection* [GS95].

2.4.1 Simulation-based fault injection

The simulation-based fault injection is a low-cost, non-intrusive dependability evaluation concept applied during the design phase. Faults are injected into a simulation model of the system under test. It can be used to evaluate the effectiveness of fault-tolerance mechanisms and dependability measures of the system model.

As the system in early design phases often only consists of high-level abstractions, this fault injection technique does not provide feedback about temporal behavior. The early detection of design faults, on the other hand, reduces the cost of correcting conceptual errors. Simulation-based fault injection provides a high degree of controllability and observability, as faults can even be injected to regions which are inaccessible to fault injection methods based on physical devices [FSK98]. The overhead of time introduced by hardware and software simulations limit the practical application of simulation-based fault injection.

Depending on the level of abstraction, simulation-based fault injection can be classified [LN09][Ade03]:

Electrical level simulation: For this class of fault injection the currents and voltages in a circuit model are changed to simulate realistic hardware faults like transient faults. This allows the observation of physical reasons (e.g., metastability or electromagnetic interference) for higher level faults. Due to high simulation effort fault injection at the electrical level is appropriate to evaluate the behavior of vital components of the circuit (e.g., the interface from analog to digital components) rather than the whole system.

Gate level simulation: Simulation-based fault injection at gate level is applied to simulate physical faults where the cost of fault injection at the electrical level is to high. At this level faults are injected to logic gates. Typical fault models are stuck-at faults and inversion of logic levels. Gate level fault injection can be used to obtain information about the effect of faults at gate level to higher levels. For large computer systems this fault injection approach is infeasible.

Register transfer level (RTL) simulation: System models implemented using hardware description languages such as VHDL or Verilog are used to simulate RTL faults. Read and write operations at registers and memory locations are modified to inject faults. This type of fault injection can be employed to determine the effectiveness of error detection mechanisms aimed at the detection of faults in registers and memories. The complexity of modern multicore SoCs makes it difficult to evaluate complete systems.

Another grouping of simulation-based fault injection is according to the applied mechanisms [LN09]:

Simulator command: Built-in commands of the simulator are used to inject faults to variables and signals of the model of the target system. The quantity of commands offered by the simulator determine

the appropriateness of such fault injection techniques. An advantage of simulator commands is that the system model need not be changed for fault injection.

Simulation code modification: Two types of simulation code modification can be differentiated: *saboteur techniques* and *mutant techniques*. In the first case a saboteur component is added to the system model to inject faults. This module changes values or timing characteristics of distinct signals. The mutant technique is based on the modification or corruption of existing modules of the target system. The drawback of both techniques is, that the system model needs to be changed for fault injection.

Simulator modification: This type of simulation-based fault injection rather changes the simulator than the simulation model. For evaluation, the target system is considered either as black- or white-box. Various kinds of faults can be injected at the boundary of the system model. Thus, no modification of the system model is required.

2.4.2 Hardware fault injection

During hardware fault injection experiments the target system is exposed to physical disturbances (e.g., radiation, electromagnetic interference, extreme thermal conditions, pin forcing), which even might lead to the destruction of prototype hardware. Additional hardware facilities are required to perform hardware fault injection. As these techniques are executed on real hardware prototypes, no timing assumptions are necessary and a high time-resolution of hardware triggering and monitoring is possible. Distinct types of hardware fault injection allow the introduction of faults to locations that are inaccessible to other fault injection techniques. Depending on the location to which the faults are injected and the types of faults, two different categories are defined in [HTI97]:

Hardware fault injection with contact (pin-level injection): Injectors with physical contact to the target system are used to produce voltage and current changes at the desired locations of the tested system. So the duration and the location of faults can be easily controlled. Two methods of pin-level injection are commonly used:

Active probes: Probes are attached to selected pins of the integrated circuit to force determined voltages or currents to flow. The technique is especially applicable for injection of stuck-at or bridging faults. When additional current is carelessly forced to a device it may result in the damage of the target hardware.

Socket insertion: A socket is placed between the target hardware and its circuit board. Stuck-at, open and even more complex logic faults can be injected by the socket. The analog value of signals can be forced to a desired level or the signals of pins can be logically combined.

Hardware fault injection without contact: The target system is exposed to radiation, electromagnetic fields or extraordinary thermal conditions. This generates currents in the device or alters its electrical characteristics, which is similar to natural physical phenomena that either affect the device transiently or permanently. The problem of contactless hardware fault injection is, that it is difficult to control the faults in the time and space domain. This is because the precise moment and location of the discharge of heavy-iones and electromagnetic pulses cannot be exactly controlled.

2.4.3 Software implemented fault injection

Software implemented fault injection is a technique for the insertion of faults to a target system that does not require expensive hardware. With software fault injection, both, software and hardware faults in the target system, can be emulated. Software faults include the target application as well as faults in the operating system. This can hardly be done with hardware fault injection techniques.

For software implemented fault injection the original code is extended with mechanisms – implemented with more code – that alter the existing syntax or forces a state when the software is operational. Thus, the software or its behavior, respectively, is modified for fault injection [Voa97]. Faults can only be introduced to locations that are accessible to software (memory locations and registers). In case faults need to be injected to inaccessible regions, other fault injection techniques have to be used [Ade03].

Software fault injection techniques are in the time and spatial domain high controllable and repeatable. On the other hand, the additional code may influence the workload on the target system and even may introduce behavioral changes of the original software. Due to poor time-resolution, software implemented fault injection is not appropriate to emulate short latency faults of, for example, a data bus or the CPU.

In [HTI97] software fault injection methods are distinguished, based on when faults are injected:

Compile-time injection: To emulate effects of hardware, software and transient faults, errors are introduced to the source code of the target application by modification or insertion of instructions. This

equals an erroneous software image that is loaded to the target system. When the target program is executed and the modified code reached, the fault is activated.

No additional software is needed during the execution of the experiments. As the modified code is only used for fault injection experiments, no perturbations are introduced to the final target system. The hard-coding of faults allows the introduction of permanent faults, but no injection of faults during runtime is possible.

Runtime injection: For runtime injection of faults, additional commands are placed in the target application. The fault injection code is executed along with the original target program. Transient and intermittent faults in memory and registers can be injected when this code is triggered. Common triggering mechanisms for runtime fault injection can be differentiated into:

Time-out: A hardware or software timer interrupt is used to trigger the injection of faults at predetermined instants in time. The code to inject faults is linked to the interrupt service routine. This technique does not require substantial modification of the target application or workload program. As the faults are injected at arbitrary chosen instants in time, realistic emulation of unpredictable fault effects, like transient or intermittent faults, can be achieved.

Exception/trap: Hardware exceptions or software traps transfer the control to the fault injector. Faults are injected on the occurrence of certain events or conditions (e.g., a fault is injected before a specified instruction is executed).

Code insertion: Additional instructions in the target program enable the injection of faults before particular instructions of the original application. In contrast to compile-time code modification methods, this technique rather inserts code than modifying existing instructions, and fault injection is performed during runtime. Compared to the trap method, the fault injector is part of the target program instead of being executed as interrupt service routine in system mode.

Chapter 3

Partitioning in the TTSoC Architecture

This chapter summarizes the properties of the TTSoC architecture and gives an architectural overview. Afterwards, the partitioning in the TTSoC architecture is presented. Especially, the architectural components needed to implement temporal and spatial partitioning are detailed.

3.1 The TTSoC architecture

The development of the TTSoC architecture is driven by the desire to gain an architectural framework with support of composability [KO02], that provides a component-based design methodology and that clearly decouples computational components from the communication infrastructure. It incorporates many features which are elaborated in [OSHK08] and [Pau08]. The result is a predictable and deterministic time-triggered network-on-chip (NoC) with inherent fault isolation to interconnect the computational components. In the following subsection the key properties of the architecture can be found. Afterwards, the architectural elements of the TTSoC architecture are presented in subsection 3.1.2.

3.1.1 Properties of the TTSoC architecture

The following properties are essential for the TTSoC architecture and help to improve the design and understanding of a system-on-chip (SoC).

Elevation of the level of design abstractions

The level of abstraction is essential for the management of complexity [Kop08] of evolving designs. A conceptual model of components is necessary to create stable intermediate forms for the assembling to systems-of-systems. The component's properties need to be detailed by an adequate interface specification. Hence, the internal structure and interactions within a component need not be known and can be disregarded. The additional advantage is, that in case of a change or enhancement of a component's implementation due to technological improvements, no redesign of the whole system is required.

In the TTSoC architecture a *micro component* builds such a unit of abstraction. Its functionality is provided to other interacting micro components at a well-defined message-based network interface [GIJ+03]. As there is a strict division into the processing within micro components and their interactions among each other, the TTSoC architecture represents a communication-centric model [BM02], [WG02] which can be applied in a multitude of applications.

Predictability and determinism

The predictability of the on-chip interconnection is a pivotal property of the TTSoC architecture. A time-triggered communication schedule is used to assign different slots of time to each micro component. It is also specified whether the component is supposed to use the communication system for sending or receiving during this dedicated time slots. The communication system with the time-triggered schedule protects the micro components from interfering one another.

This leads to a deterministic behavior of the communication subsystem, i.e. the ability to reason about the future behavior and derive the future state of a system, when an initial state and future inputs are known. With deterministic systems it is possible to implement triple-modular redundancy (TMR), which allows the transparent masking of hardware errors within micro components [Pol94].

Error containment through encapsulation

Also elementary for the TTSoC architecture is the integrated error containment, as it makes modular certification easier and helps to increase robustness and composability [OKSH07]. Due to a strict separation of the SoC into independent micro components which may only interact by exchanging messages via the NoC, the property of error containment can be achieved. As the time-triggered approach makes it possible to protect the NoC from unauthorized access by the use of guardians at each component, misbehavior of any micro component (e.g., message timing failures), may not affect the operation of other components.

Encapsulated communication channels [Pau08], that are represented in the time-triggered communication schedule, build the basis for the encapsulation paradigm. The communication of a micro component is only visible to other components if they share the same communication channel. With such an encapsulated communication channel, a micro component obtains a guaranteed bandwidth and a bounded latency of communication is assured. Additionally, *temporal ordering* and a *consistent delivery order* of messages at all micro components can be guaranteed, i.e., each micro component receives its messages in the same order as they were sent and all components obtain their incoming messages in the same order. Message ordering is vital for distributed consensus protocols and agreement algorithms. It is also necessary to achieve *replica determinism* [Pol94], i.e., replicated components are in the same state at about the same time [Kop97, p. 40].

The mechanism of encapsulation also helps to reduce the cognitive complexity, as interfering subsystems show a more complicate behavior than clearly encapsulated systems do. This in turn keeps the effort of testing and validation smaller than for systems with interfering components.

Global time base and clock domains

Providing a single clock domain for a whole SoC brings many troubles with it. To overcome these problems, multiple clock domains are introduced in the TTSoC architecture with the advantage of a simplified handling of clock skew, clocking down of individual IP blocks for power management or the support of heterogeneous IP blocks with different speeds [OSHK08]. The TTSoC architecture provides a system-wide global time base beside the existence of multiple clock domains. The global time base is a sparse time base – as described in [Kop97, p. 55] – generated by internal clock synchronization, that also supports synchronization to a SoC-external reference time.

In the sparse-time model, the dense timeline of the real world is partitioned into durations where events can take place – duration of activity – and intervals where no events are allowed – duration of silence. Events inside the same interval of activity are said to happen simultaneous, while events, that are timestamped at different durations of activity and that are the required interval of silence apart from each other, can be consistently temporally ordered in the whole distributed system without the usage of an agreement protocol [Kop92]. All events in the sphere of control of the system are assured to happen within an interval of activity. On the other hand, an agreement protocol is necessary to assign events to a particular duration of activity when they are outside the sphere of control of the system.

With the global time base, a temporal coordination of actions of micro components

in a SoC or of an ensemble of different SoCs is possible. When distinct timestamps are assigned at different micro components, they can be related to each other. Hence, timestamps that have been assigned to events at one micro component are also meaningful outside this micro component.

According to [Pau08] the TTSoC architecture supports the following clock domains:

- The coordination of actions in the system is based on the global time base. For example, the start of communication is triggered by the global time at each micro component simultaneously. Its granularity determines how tight events can be synchronized. As the global time base may be relatively slow compared to other clock domains, it can span the whole system, whereas clock skew and driver capacity remain sufficiently small to be easily handled by system designers.

- Despite the synchronization of communication activities at all micro components (e.g., the start instant of communication) by the global time base, the communication subsystem has its own clock domain. The frequency of this system-wide clock is higher than that of the global time base. Each elementary data package – called *flit* – which a message consists of, is transmitted from one network element to the neighboring element within one cycle of the communication system clock. Hence, data can be transported faster between micro components than with the granularity of the global time base.

  As each micro component may run on its own clock frequency, that is typically faster than the global time base, the communication subsystem needs to be able to deal with different clock domains. Therefore an asynchronous handshake protocol is applied at the boundaries of the different clock domains.

- A micro component itself can be composed of modules like processor cores, memories and IP blocks that all may have a distinct local clock domain which is transparent outside the micro component.

Due to this structure of clock domains, local modifications of clock domains, either of micro components or the communication subsystem (e.g., changing the frequency), do not require the redesign of the complete system.

Integrated resource management

The integrated resource management feature of the TTSoC architecture concerns the adaptation of resources according to the actual demand of the application.

Changing requirements can address communication resources (e.g., bandwidth, latency, latency jitter), computational resources (e.g., dynamic allocation of micro components to application subsystems), and power (e.g., power limiter) [OSHK08].

In case a micro component suffered a permanent fault, a spare micro component can take on the application functionality of the failed micro component. Hence, the SoC is able to carry out its specified services despite the failure of a micro component. Furthermore, the individual switching of micro components into defined application modes helps to implement power-aware systems [UK03].

3.1.2 Architectural Elements

In figure 3.1 the general structure of the TTSoC architecture is illustrated. Multiple, possibly heterogenous IP blocks – so called *micro components* – are interconnected by the central time-triggered NoC [OSHK08]. The architecture distinguishes between trusted and non-trusted elements. The *trusted subsystem* prevents a fault (e.g., a design fault) inside the host of a micro component from violating the temporal interface specification of the micro component. As the trusted subsystem inhibits the untimely access to the NoC, the communication between other micro components cannot be disrupted by a faulty micro component.

The TTSoC architecture introduces two architectural elements – the *trusted network authority (*TNA*)* and the *resource management authority (*RMA*)* – that support dynamic resource allocation. The RMA receives requests for resource relocation and calculates the new resource requirements, which must be granted by the TNA. Afterwards, the SoC is reconfigured according to the newly approved resource allocation. This can be done, for example, by dynamically updating the time-triggered communication schedule or by switching between power modes.

Micro component

The TTSoC architecture allows the integration of distinct application subsystems – which probably have different criticality levels – to one overall system. A micro component is a nearly autonomous, possibly heterogenous IP block, that belongs to a particular application subsystem. These self-contained micro components are built by a *host* and a *trusted interface subsystem (*TISS*)*. Application services are performed by the host, while the TISS protects the time-triggered NoC from unauthorized access. For this purpose, the TISS contains a table with a priori knowledge – a *communication schedule* – about sending and receiving instants of its corresponding micro component. Hence, the micro component cannot access the NoC outside the allowed intervals. As the micro component has no access to the schedule, no fault within the host can disrupt the exchange of messages of

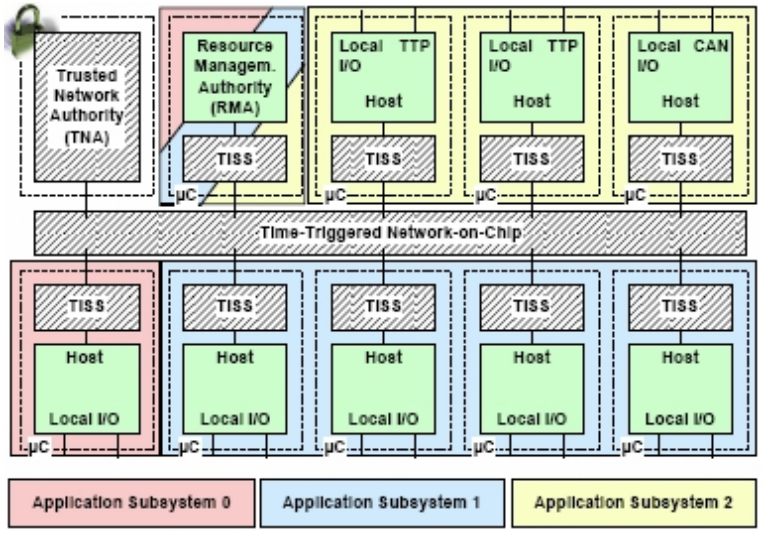

Figure 3.1: Structure of the TTSoC architecture: *shaded area represents the trusted subsystem, hosts of micro components are not trusted.* From [OSHK08].

other micro components.

Only by the exchange of messages on the time-triggered NoC the micro components of an application subsystem can interact with each other. Thus, each micro component is encapsulated. Temporal and spatial interference with other micro components is prevented by design. A faulty micro component that delivers faulty input is the only way of affecting other micro components.

The paradigm of encapsulation supports *replica determinism*[Pol94], *composability*[KO02] and the integration of application subsystems with different criticality level into a single SoC.

Time-triggered NoC

The objective of the time-triggered NoC, beside the interconnection of micro components, is *clock synchronization* and the *predictable transport of messages.* A global time base is provided by synchronization of multiple clock domains used in the system. This enables that actions in the SoC or in an ensemble of SoCs are temporally coordinated.

The time-triggered NoC consists of several interconnected *fragment switches* that transport individual flits (fragments of a message), one per clock cycle. The TISS of one micro component is connected to exactly one of these fragment switches. A route from one sending micro component to the receiving micro components is

determined by the fragment switches that need to be traversed by a message. To move the message through the NoC the message contains a header that tells each fragment switch to which of its output ports the message must be forwarded.

For the predictable transport of messages a TDMA scheme is used to partition the existing bandwidth into periodic conflict-free sending slots. These slots can either be used for the *periodic transport of messages* or *sporadic transport of messages*. Sporadic transmission means, that the sending slot is only used if there actually is a message to be sent. Otherwise, the slot remains unused. *Pulsed data streams* [Kop06] are used to allocate the sending slots to micro components. These are *time-triggered periodic unidirectional data streams* which transmit fixed length data pulses according to a priori knowledge from *one* sender to n receivers. The transmission is carried out at a specified phase of each cycle of a periodic control system [OSHK08]. This allows an efficient transport of large data of applications requiring temporal alignment of sender and receiver.

The allocation of TDMA slots to micro components is accomplished by the 3-tuple $< pulse\ period, pulse\ phase, duration >$ which determines a pulsed data stream. Such a pulsed data stream comprises *periodic pulses* of specified *pulse period* and *pulse phase*. The pulse phase gives the offset to the start instant of the pulse relative to the start of the pulse's period. Pulses are composed of at least one *fragment*, which also might be interleaved by other pulse's fragments. The *duration* is defined as the time between the first and the last fragment of a pulse. Furthermore, a fragment is a set of *flits*, that are atomic entities which occupy exactly one TDMA slot.

Elements for resource management

Integrated resource management is intended to dynamically allocate computational resources (i.e., micro components) to application subsystems, communication resources and power modes to micro components. Two architectural elements are used for this purpose: the TNA and the RMA.

Upon a request for resource relocation, the RMA calculates new schedules and the allocation of computational resources. Application specific knowledge (e.g., communication topology) is necessary for these tasks. As the RMA is not part of the trusted subsystem, it may not change the configuration of the SoC directly. The actual reconfiguration must be granted and executed by the TNA. This separation of duties results from the fact, that the actual calculation of resource allocation is much more demanding that the sole check of validity. Thus, in contrast to the TNA, the RMA does not require a stringent certification process.

The TNA finds potential violation of resource allocation (e.g., a collision on the NoC). In case the new schedule cannot be permitted, it will be rejected and the

SoC is not reconfigured. Otherwise, the TISSes of the affected micro components are updated and the new allocation of resources is activated.

Encapsulated communication channels

The physical separation of micro components entails the encapsulation on the level of micro components. For achieving encapsulation with respect to the communication infrastructure, *encapsulated communication channels* are introduced to the TTSoC architecture. These are unidirectional channels for the transmission of messages from one source to multiple destinations at a priori known points in time [OSHK08]. Communication actions are only visible to participants connected to the same encapsulated communication channel. *Ports* – which can be distinguished between output and input ports – denote the endpoints of the encapsulated communication channels. Each micro component can be attached to multiple communication channels. As there can be multiple destinations of one encapsulated communication channels, *single-cast, multi-cast* and *broad-cast* topologies are possible.

To each encapsulated communication channel a specific pulsed data stream is assigned, that reserves a set of TDMA slots for this channel. The TISS of each micro component accesses the time-triggered NoC according to the TDMA slots assigned to a communication channel. A time-triggered schedule that holds the points in time when to access the NoC – the so-called message descriptor list (MEDL) – is part of the TISS. This MEDL cannot be accessed by the host of a micro component. It can only be altered during the reconfiguration process by the TNA. Hence, a faulty host cannot introduce interferences to the communication subsystem.

3.2 Partitioning in the TTSoC

To detect errors which propagate from one application to another one, distinct fault-containment regions (FCRs) are needed. A FCR prevents the propagation of errors across its interfaces so that no failure in one FCR may cause a fault in another FCR [Obe07]. If there is only one FCR in a system, an error can affect both, the messages sent by an application and the proper function of error detection mechanisms.

To preserve the independence of FCRs and to avoid common mode failures, a system must be effectively partitioned. The TTSoC architecture allows the coexistence of components with different criticality levels within a single hardware platform. Hence, it is of paramount importance that no failure is able to propagate from any component with lower criticality level to components with higher

criticality level.

The existence of physical faults in a computer chip – like a particle strike or a permanent hardware defect – can never be ruled out. Thus, for the design of ultra-dependable systems the entire SoC needs to be considered as one single FCR [OKSH07]. As, for example, the breakdown of the power supply or the shared oscillator may disrupt the operation of the entire SoC. In case of an ultra-dependable system different FCRs can only exist if distinct computer chips are interconnected to build a larger system-of-systems.

On the other hand, in the TTSoC architecture distinct FCRs are integrated that prevent the propagation of design faults and permanent or transient physical faults – that are restricted to an individual component – from one FCR to other FCRs.

The TTSoC architecture distinguishes between two different types of FCRs [OKSH07]:

Trusted Subsystem: This FCR includes the actual time-triggered NoC and a trusted interface subsystem (TISS) for each of the micro components. By assumption the trusted subsystem is free of design faults. Its small and simple design simplifies the formal analysis and minimizes the probability of transient physical faults.

The communication subsystem uses a time-division-multiple-access (TDMA) scheme. The available communication bandwidth is split into periodic sending slots, that are free of conflicts. Each of these slots can contain periodic or sporadic messages. In contrast to periodic messages, a sporadic message is only sent if a new event has to be transmitted. The usage of predefined conflict-free sending slots guarantees the predictable transport of messages.

Due to the time-triggered design the possibility of, so-called, Heisenbugs is largely eliminated, as logical and temporal controls are unambiguously separated and no race conditions can exist. Heisenbugs are design errors in the software which result in quasi-random failures. In contrast to these errors, Bohrbugs are software design errors that create reproducible failures, and hence, are easier to correct.

The sparse global time base, which is provided by the trusted subsystem, builds one of the core mechanisms for the time-triggered design, as all control signals are derived from this distributed notion of time. All TISSes are guaranteed to be in the same state within each interval of silence of the sparse time base.

Micro Components: The host of a micro component provides the services of the actual application. The hardware and software design

> of the micro components need not be free of design errors. Especially, in the case of non safety-critical components the effort of formal proves and certification is out of scale. But even though, a malicious fault inside a non safety-critical host must not affect the correct function of other – potentially safety-critical – micro components. The TISSes connected to the individual micro components act as guardians to prevent the unauthorized access to the NoC. Hence, the trusted subsystem ensures the encapsulation of each of the micro components.

The only interaction between the micro components is based on the exchange of messages on the time-triggered NoC. Any failure in the sending component must propagate through the time-triggered NoC to cause a fault in another micro component.

Generally, there is a distinction between *message timing failures* and *message value failures* as it was defined in [CA85] and adopted for the TTSoC architecture in [OKSH07]. Essentially, a message timing failure occurs when the send instant of a message or its moment of reception is outside of the specification. A message value failure means that a message is not valid (e.g., the CRC does not match) or a valid message contains data structures that are not correct.

The concept of time-triggered communication prevents the existence of race conditions where distinct hosts try to send at the same time. Each TISS contains a priori knowledge about the sending and receiving instants of its corresponding micro component. It acts as a guardian to ensure the predictable transport of messages. Thus, no message timing failures may arise.

Chapter 4

Related Work

Different SoC architectures using NoCs for component interconnection have been proposed until now. This chapter gives an overview of these architectures and related work. After the presentation of each architecture, in section 4.5 these are compared with the TTSoC architecture. Especially, characteristics relating to temporal and spatial partitioning as well as reconfiguration are outlined.

4.1 Æthereal

Æthereal [GDR05] is an architecture for NoCs that provides both *guaranteed services* (GS) and *best-effort services* (BES).

> **Guaranteed services:** These services comprise uncorrupted, lossless, ordered data delivery as well as guaranteed throughput and bounded latency. For guaranteed services resources need to be reserved to be able to provide the services also during worst case scenarios.
>
> **Best-effort services:** A best-effort service is free to use the bandwidth that is not reserved for guaranteed services. Also for best-effort services data integrity, lossless and ordered data delivery are assured.

An Æthereal NoC consists of *routers* and *network interfaces*. Routers transport data from one network interface to another one and are further divided into GS routers and BES routers. GS routers transmit only data for guaranteed services while BES routers transport all other messages. Each of the network interfaces (NIs) is on one side attached to an IP-core and on the side connected with one of the routers. NIs convert the local protocol used by the IP-core to the router view on communication – to packets – and provide end-to-end services to the IP modules.

The NoC provides a *shared-memory abstraction* [RDG+04] to IP modules. A transaction-based protocol is used for communication. *Request messages* (e.g., read or write commands to addresses, possibly with data) issued by the master IP module initiate a *transaction*. The request message is then processed by slave IP modules addressed in the request. Slave modules may reply with *response messages* (e.g., status of the command executed, requested data). This concept is used for compatibility to existing protocols like AXI, OCP or DTL.

The decoupling of computation and communication together with guaranteed services reduces the complexity of system development and facilitates the composition of independently designed and validated IP modules.

4.1.1 Connections in Æthereal

Æthereal communication is based on *connections* that can be point-to-point, multicast (multiple slaves; all executing each transaction) or narrowcast (multiple slaves; only one slave executes the transaction). A connection is divided into unidirectional point-to-point *channels* reaching from a single master to a single slave. Connections contain request channels (from master to slave) and response channels (from slave to master).

Each channel comprises a buffer at the NI of the sender and a buffer at the receiver's NI. Guaranteed message delivery is ensured by the usage of *credit-based flow control* which prevents router and NI buffer overflows [RDP+05]. Figure 4.1 illustrates the implementation of Æthereal connections.

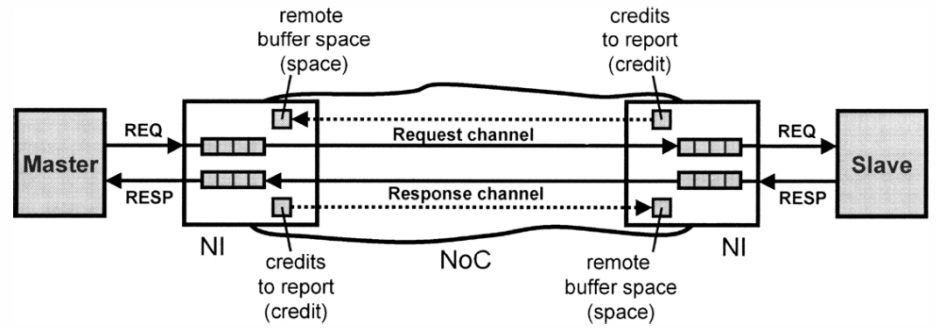

Figure 4.1: Implementation of Æthereal connections. From [RDP+05].

A counter a the sender side of a channel keeps track of the available space in the receiver's buffer. Initially the value of the counter equals the size of the receiving buffer. Whenever data is transmitted from the sender buffer to the receiver, the counter is decremented. After the receiver consumed data from its receiving buffer, a *credit* is generated to indicate that additional space at the receiver is free. The

credits are transported – either in a dedicated message or piggypacked in the header of another data packet – to the previous sender. There the credits are added to the counter.

As mentioned earlier, Æthereal provides guaranteed services (GS) and best-effort services (BES). Dedicated GS routers and BES routers are placed in parallel, where GS routers have higher priority. The guarantee of throughput and latency for GS is accomplished by implementing connections as *pipelined time-division-multiplexed circuits* [RDP+05]. For this purpose the routers and NIs in the network require a common notion of synchronicity. Reservation of bandwidth is realized with tables in the NIs that contain entries for reserved time slots. These time slots tell the NI when to send or receive data packets and the route on which a packet is transported. The size S of the tables determines the granularity of bandwidth B_i that is reserved by one entry. A slot equals $\frac{1}{S}$ of the maximum bandwidth. Thus, when N slots are reserved for a connection, its total bandwidth is $N \cdot B_i$.

Unreserved time slots in the tables can be used by BES routers for best-effort transmission of packets. The header of these packets contains routing information which is used by BES routers to determine the desired output port of the router. Link-level flow control is used to prevent the overflow of the input buffer of the subsequent router or NI.

4.1.2 Configuration of Æthereal

The configuration in Æthereal is done to adapt communication demands to specific use-cases [HCG07], e.g., when the mp3 player of a mobile phone is switched on or off. Connections need to be established or broken down and bandwidth must be reserved or released. Thus, the new configuration is loaded to the NI of each IP-core that has changed communication requirements in the new configuration.

The configuration data comprises the *path* through the network of routers, a *queue identifier* to specify the target queue at the receiver's NI, *end-to-end flow control credits* and the *time slots* to determine when the channel uses the network link [HG07]. This information can either be calculated off-line or on-line. A dedicated *configuration master* executes the configuration actions.

The existing infrastructure – i.e., routers and wires – is used to transport configuration data to the NIs. Thus, no additional control network is needed. *Configuration connections* are established from the configuration master to those NIs which actual configuration deviates from the new one. These connections comprise a *configuration request channel* and a *configuration response channel*. Three types of operations of the configuration master to manage channels and connections are distinguished:

- open a new connection
- modify an existing connection
- close an existing connection

When the configuration connection is finally set up, the configuration master sends the new configuration data to the corresponding NI. *Configuration registers* are read and written through a memory-mapped *configuration port* which is a logical port in the NI. These configuration registers hold control information about the path through the network (using source routing), the destination queue, the service level (GS or BES), flow control space, the slot table, etc.

4.2 MANGO

In [Bje05] the MANGO (Message-passing Asynchronous Network-on-chip providing Guaranteed services over open core protocol (OCP) interfaces) architecture is presented, which is a clockless NoC implementation with guaranteed communication services. It uses an individual clock domains for each IP-core while the network links and routers are kept entirely clockless. This approach is also known as globally-asynchronous locally-synchronous (GALS) [MVK+99]. In Figure 4.2 the structure of a MANGO network is illustrated. It comprises network adapters attached to the IP-cores, and routers that are interconnected by links.

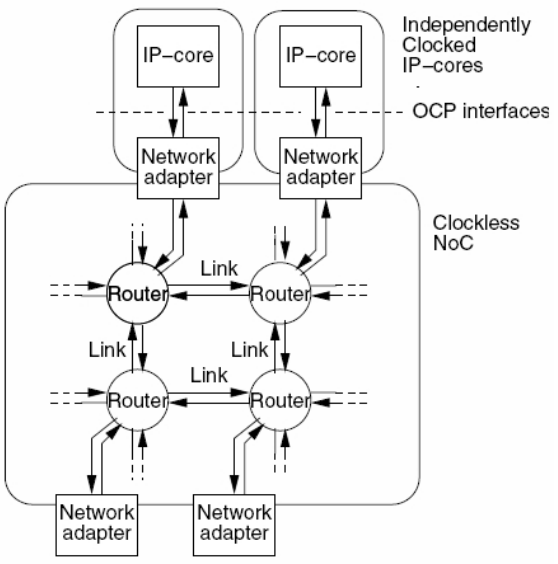

Figure 4.2: Structure of MANGO network. From [BS05a].

The MANGO architecture offers connection-oriented guaranteed services (GS) – with respect to hard latency and bandwidth bounds – and connection-less best-effort (BE) routing.

4.2.1 Network adapter

The network adapter (NA) [BMOS05] is the interface between the clocked IP-core and the asynchronous clockless NoC. At the side of the IP-core, a socket-based OCP interface – the core interface (CI) – is provided. On the other side, the network interface (NI) connects the NA to the network. In the NA, messages sent by the IP-core, which are realized as OCP transactions, are packetized, switched through the network as a stream of datagrams – called *flits* – and depacketized at the receiver. Thus, the NA implements *transaction handshaking* at the CI, *encapsulation* of transactions for the packet-switched network, and *synchronization* of the clockless NoC with the local clock region.

Memory-mapped read/write transactions are used for communication. Therefore two types of NAs are required: an *OCP initiator* attached to a master IP-core, and an *OCP target* at the slave IP-core. This structure is depicted in figure 4.3. It can be seen, that the NA comprises a *request* and a *response* path.

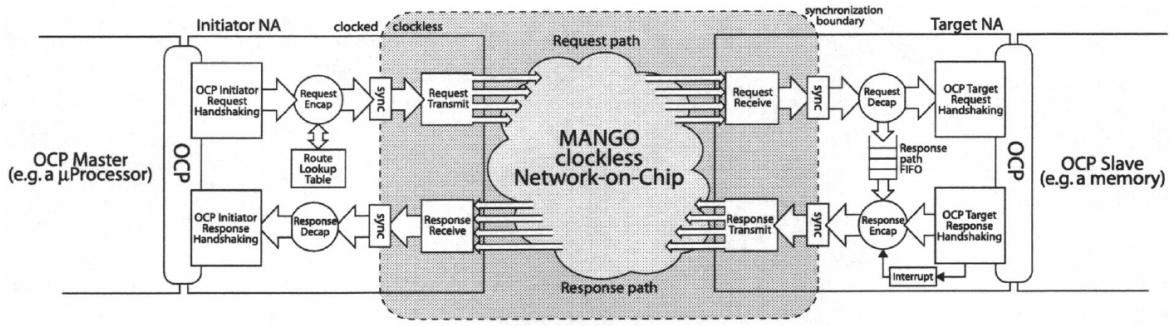

Figure 4.3: Master/slave communication structure. From [BMOS05].

OCP transactions are transformed to packets – and vice versa – by the *handshaking* and *encap/decap* modules. The *transmit* and *request* modules serialize and reassemble, respectively, packets in the clockless domain. *Sync* modules conduct the synchronization of the clocked domain with the clockless domain.

4.2.2 Router

MANGO routers [BS05a] are composed of two types of sub-routers, according to the service class [BS06]: connection-oriented *guaranteed service (GS) routers* and

connection-less *best-effort (BE) routers*. GS routers switch data streams on GS connections, while BE routers use routing information in the header of packets for dynamic routing.

Physical links are shared by multiple, logically independent virtual channels (VC). Each of the VCs is buffered individually, which implies that a router with P input ports and an equal number of output ports contains $P \cdot V$ buffers, where V is the number of VCs.

For GS and BE services distinct flow control mechanisms are implemented, which prevent that flits stall on the links, and the overflow of buffers. GS flow control uses a *share-based* concept. When the buffer at receiver side is free, a *sharebox* at the output port of the sending router admits a flit to the shared media. The sharebox locks, to prevent subsequent flits to be transmitted. At the other side, the flit is received by an *unsharebox*. After the flit leaves the unsharebox, an unlock control wire is toggled. This signals to the sharebox that the next flit can be received, and the sharebox is unlocked to admit the subsequent flit to the media. BE routers use *credit-based flow control* where the sender keeps track of free buffer space at the receiver.

While BE packets use source routing, where the path is contained in the packet header, GS routers implement virtual circuits through the network [BS06]. *Steering bits* determine the input-to-output VC mapping of a router. These are programmed to the GS routers via BE packets routed to the GS programming interface. The entire route of a packet from sender NA to the receiving NA is given by the sequence of VCs. The programming is centrally managed by a dedicated system programming unit.

Link arbitration is done by a dedicated *link arbiter* that controls access to the physical link and guarantees a fairly shared bandwidth between the VCs. Hard per connection latency and bandwidth guarantees are provided by the *Asynchronous Latency Guarantee (ALG)* algorithm [BS05b].

4.3 Nostrum

In [MNT02] the Nostrum NoC architecture is presented. Nostrum provides two distinct Quality-of-Service guarantees for NoC communication: *guaranteed bandwidth and latency (GB)* and *best-effort (BE)* packet delivery. It uses a *mesh* topology for *packet switched routing*. The architecture distinguishes between *resources* (e.g., processors, memory) and *switches*. The latter ones are connected to – at most four – nearest switch neighbors and to its corresponding resource.

A layered approach is used by the Nostrum backbone [MNT+04], that is based on the NoC. Resources are equipped with a *network interface (NI)*, that is layered

above the network. NIs implement a set of services that can be used by the *resource network interface (RNI)*, or directly by the IP-core. The RNI is a customized interface between the standard services of the NI and the internal communication infrastructure of the IP-core.

GB traffic uses virtual circuits (VC) which are implemented using *looped containers* and *temporally disjoint networks (TDN)* [MNTJ04]. Looped containers are data packets traveling on a pre-defined route through the network. They are used to guarantee access to the network, as an empty container can be loaded by a resource. TDNs result from the *deflective routing* [FR92] policy applied, which introduces an implicit time-division-multiplexing in the NoC.

Due to the concept of deflective routing, packets are not explicitly queued in the switches. Packets remain only for one clock cycle in a switch. This implies that packets must leave a switch in the same order as they entered. Furthermore, packets that enter a switch simultaneously also must leave simultaneously. As the switching decision is made for each packet individually in the switches, packets may be transported on different routes with different lengths. Thus, the reordering of packages is possible.

4.3.1 Temporally Disjoint Networks

As, due to the deflective routing, packets cannot be reordered within switches, an implicit time-division-multiplexing schema is created – referred to as temporally disjoint networks (TDN). This is the result of the network *topology* and the *number of buffer stages* in switches.

Topology: When packets are sent at the same clock cycle, a collision may only appear if both are on a multiple distance of the smallest round-trip delay. A coloring scheme is used by Millberg et al. [MNTJ04] to confirm this statement.

Switches in the mesh network are colored black and white in that way, that black switches are only connected to white ones, and vice versa – see figure 4.4a. Thus, a packet on the path to its destination need to traverse black and white switches in an alternating order, and packets residing in distinct colored switches will never meet.

The colored network can be restructured to obtain the bipartite graph illustrated in figure 4.4b. This graph in turn can be collapsed to a graph with one black and one white switch and two uni-directional links. When two packets are in differently colored switches it is logically the same as both packets were in different networks – i.e., TDN.

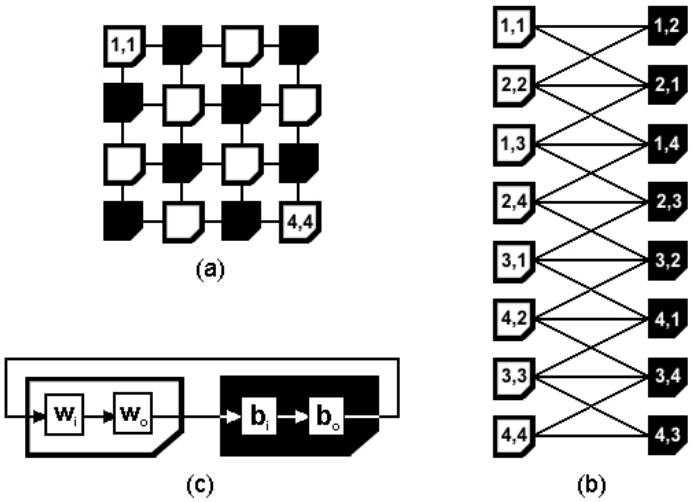

Figure 4.4: Temporally disjoint networks of Nostrum: (a) colored 4x4 mesh network; (b) network drawn as bipartite graph; (c) disjoint networks with buffer stages.

Buffer stages: When additional buffering in the switches is applied, a second set of TDNs is created. This can be seen in figure 4.4c. Hence, a packet routed on the network must visit buffers in the following cyclic order: $w_i \rightarrow w_o \rightarrow b_i \rightarrow b_o$.

The total number of TDNs can be calculated by TDNs introduced by the topology multiplied with TDNs resulting from additional buffer stages. In each of the TDNs a different communication type can be implemented, e.g., different priorities, traffic types or load conditions.

4.3.2 Looped Containers

In Nostrum bandwidth and latency guarantees are achieved with looped containers (LC) which implement virtual circuits (VC). Such a container travels in a loop between source and destination resource and can be loaded with information. This concept is based on two policies:

- Packets already in transit have precedence compared to packets waiting to be launched to the network.
- The difference between the number of packets entering a switch and those which leave after being switched, is zero after some point in time.

The consequence is, that a switch which always receives a packet at each of its input ports must route these packets to the output ports in the next cycle. Hence, it will never be able to serve packets of its corresponding resource and this resource cannot send messages.

When an empty LC arrives at the input of a switch, it can be used to transport a package of the resource connected to that switch. This is, because an output port is reserved to that LC. The LC is marked as loaded and sent to the receiver, where it is discharged and possibly reloaded. Otherwise, it is sent back and flagged as empty.

4.3.3 Theory of Operation

LCs and TDNs are combined to enable a set of VCs to share the same link. One TDN can be used by only one VC, while a VC can subscribe to multiple TDNs. A switch can handle as many simultaneous VCs as the number of TDNs.

A VC's route, and thus of its LCs, is defined during system design. To adapt for bandwidth requirements, the actual number of LCs on a VC can be changed at run-time.

When multiple resources read the content of a LC without freeing the container, also multi-cast communication can be realized.

4.4 HERMES

HERMES [MCM+04] is a NoC infrastructure intended for low area overhead *packet switching* networks. The architecture actually only supports *best-effort* services using source routing. Individual packets can experience arbitrarily long delays as each application is treated equally. It uses a *mesh* topology of interconnected routers on which the XY routing strategy is based. On its local port, a router connects one IP-core with the NoC. Adaptive time-division-multiplexing (TDM) is used to implement virtual channels (VCs) – also referred to as lanes [MTCM05].

Wormhole routing is applied for transmission of packets, where each packet is divided into *flits*, which is the smallest transportable unit. The header flit contains the address of the target router, that is based on the X and Y coordinates in the mesh network structure. In the second flit, the number of flits in the packet payload is specified. The actual packet content follows the path of the header flit.

A HERMES router with two VCs is depicted in figure 4.5. It comprises five bidirectional ports: four to connect neighboring routers and one local port for an IP-core. Except for the local port, all ports support individually buffered VCs

to increase the utilization of physical channels and throughput. These VCs are implemented using TDM on the physical channel. In case of n distinct VCs, a packet can use at least $1/n$ of the available bandwidth of the physical channel. If no other packets need to be transferred on this channel, the whole bandwidth can be used by one sole packet.

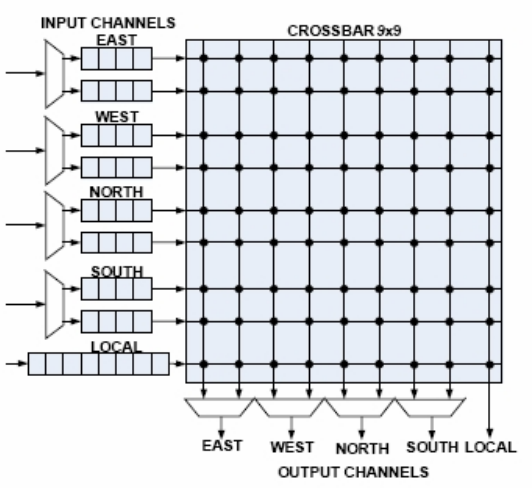

Figure 4.5: HERMES router with two VCs. From [MTCM05]

Credit based flow control is used between the routers to prevent the overflow of buffers. Each router keeps track of the available buffer space associated with a VC. When the input buffer is full, the neighboring router is denied to send further flits.

Packets received at the input ports of a router are checked, whether this is already the target router or not. In the latter case, an XY routing algorithm is applied, that first aligns the packet on the horizontal line to the coordinates of the target router – packet is sent on the east or west output of router. A horizontally aligned packet is sent on the vertical outputs of the router – north or south – to the destination. If there is no output VC free, the packet must wait until a connection on that output is closed.

The established connection in the router, from input VC to output VC, is stored in a routing table. A connection remains in the routing table until all of the flits – specified in the second header flit – have been transmitted. Afterwards, the connection is automatically closed and the VC can be used for another packet.

4.5 Comparison

Here, distinct characteristics of the architectures presented above will be compared to the TTSoC architecture. Beside some general characteristics, the main focus is on properties that relate to temporal and spatial partitioning as well as the reconfiguration of communication patterns.

Service classes: The four NoC architectures described in this chapter distinguish two types of services: communication with guarantee of bandwidth and latency, and best-effort communication. Æthereal, MANGO and Nostrum support both types of service, while the HERMES architecture in the actual version only provides best-effort services.

The TTSoC architecture does not differentiate service classes, but it supports the exchange of state and event messages. Guaranteed bandwidth and bounded latency are ensured by the concept of encapsulated communication channels. A priory knowledge of communication instants is used to reserve physical channels at determined instants for each connection. Thus no network contention can appear. BE services can be implemented using a channel for event communication. New messages from the host that arrive at the network interface of a micro component are placed in a queue. At predefined instants, the first message in the queue is transported to the receiver. When the queue is empty, no message is sent. In case of a full queue, the message must either be discarded or the sender must wait until a message from the queue is sent.

Clock domain: Æthereal, Nostrum and HERMES are based on a single clock domain for communication. Thus, synchronization of all communication partners is necessary. Unlike this, MANGO uses a clockless network which does not require synchronous participants of communication. IP-cores can operate completely asynchronous with respect to each other.

The TTSoC architecture supports multiple clock domains. Beside individually clocked IP-cores, the communication system uses its own system-wide clock domain for data transmission. Another independent clock domain is the system-wide global time base. It is used to coordinate and synchronize actions and communication activities in the whole system. Multiple clock domains have the advantage, that not the complete system must be adopted when the frequency requirements of one component are changed.

Arbitration method: Similarly to the TTSoC architecture, the Æthereal architecture uses TDMA for access control to the physical communication channel. The arbitration of the other architectures is based on the actual traffic on the network. The TDMA scheme of Æthereal is implemented by a slot table, where an entry corresponds to a predefined time slot for sending. This table equals one global period. In contrast, the TTSoC architecture supports multiple concurrent periods. Thus, the period of communication can be customized to the needs of periodic applications.

Predictability: Established guaranteed services in Æthereal, MANGO and Nostrum provide predictable communication with respect to bandwidth and latency. On the other hand, the predictable reconfiguration of communication to another application mode is not supported. Æthereal and MANGO transport reconfiguration information using BE traffic, which depends on the load on the communication channels. In Nostrum additional bandwidth can only be reserved if another looped container can be launched. This is only possible if there is free capacity on the network.

The TTSoC architecture provides both, predictable communication due to the a priory knowledge of communication instants and predictable reconfiguration. Also for reconfiguration encapsulated communication channels are used, for which bandwidth is reserved.

Message order: Temporal ordering of messages within a communication channel is assured by each of the architectures. As the switching decision for BE packets in Nostrum depends on the current network load, individual packets may take different routes, and hence, can be reordered.

Reordering of packets due to different network paths can also appear in the TTSoC architecture. However, the high abstraction level of messages from the underlying communication structure prevents the delivery of messages in a wrong order. Additionally, the TTSoC architecture ensures the consistent delivery of messages across the whole NoC. This means, that messages are delivered at each micro component at the same instant. Different messages from diverse senders are delivered in the same order in each micro component, which is a prerequisite for replica determinism [Pol94].

Reconfiguration: All architectures, except for the HERMES NoC which only supports BE traffic, can be reconfigured according to new communication demands. In MANGO, virtual channels are connected

to open or close a connection for guaranteed services. Creation or destruction of looped containers in Nostrum can be used for reconfiguration of bandwidth and latency depending on new demands. Æthereal allows the reconfiguration of individual channels, while other channels remain unchanged and operational. Thus, a smooth change from one application mode to another one is possible. Reconfigurations in the TTSoC architecture are executed at a global reconfiguration instant and at all affected micro components simultaneously.

Temporal alignment: An important feature that is unique to the TTSoC architecture comes with the introduction of the global time base. Applications in the TTSoC architecture can be *temporally aligned* [OSHK08]. This is important when a short latency between sender and receiver is required – like in many real-time systems. For example in a control loop where the sensor data is acquired by one micro component, processed by another micro component and afterwards a third micro component must operate an actuator. With the TTSoC architecture it is possible to temporally coordinate these three tasks to reduce the end-to-end latency of the control loop.

Chapter 5

Experiments & Fault Injection Framework

The fault injection experiments of the TTSoC architecture are based on the prototype implementation discussed in [Pau08]. This prototype consists on one hand of the network components implemented by VHDL code. On the other hand, a software driver is provided which can be used to access the hardware components from the micro processor's program code. A field programmable gate array (FPGA) onto which the hardware implementation of the TTSoC prototype was downloaded builds the heart of the experiments. The hardware design consists of several micro processors connected to the TTNoC.

At first, the hypotheses about the behavior of the system in general and in case of an error are stated. Upon these hypotheses different experiments are arranged to evaluate whether the hypotheses on the system behavior are correct. The second section explains the scope of each of the experiments carried out and the methods used to obtain results about the system behavior.

In the following sections the complete framework for the accomplishment of the experiments is presented. Section 5.3 presents an overview of the experimental structure together with an introduction of the functional purpose of each component in the structure. Then section 5.4 describes the environment employed during the evaluation. Afterwards, the detailed internal structure of the TTSoC, that is common for all experiments, will be explained. In section 5.6 parameters are defined, that are individually set for each type of experiment. Thereafter comes the description of the actual test procedure used to obtain the required communication data. At last, the program, that was implemented to do the scheduling of the communication channels, will be characterized in section 5.8.

5.1 Hypotheses

Four hypotheses are in the scope of this thesis. In this section these hypotheses are elaborated and will be evaluated by the subsequent experiments. Essentially, the independence of temporal and spatial properties of one micro component's communication from the behavior of other micro components and the reconfiguration functionality of the TTSoC architecture are of interest.

5.1.1 Temporal partitioning

> *The temporal properties of messages exchanged by components are not influenced by the behavior – i.e., the communication and the operations in the UNI – of other components.*

The temporal properties of messages comprise the transport time (latency), the latency jitter, message omission, message order and message duplicates. The *latency* gives the time between the generation of the message and its delivery at the receiver. In case of any variation of the latency on one communication channel, the duration between the lowest latency and the maximum latency is referred to as *jitter*.

The communication behavior of a component means the amount of messages generated and transmitted. If the component produces more messages than the communication channel is capable to transport, the messages are placed in an output buffer. Finally, when the buffer is full, some of the messages generated must be dropped but other communication channels must be independent from this behavior.

Operations in the UNI – particularly write operations – can alter the interactions of a component with the communication network. As the UNI contains both, control and status information, a modification may either change the message content, alter local communication parameters or even disrupt and turn off communication channels. However, even a malicious behavior of a component inside the UNI may not affect communication channels of other components.

5.1.2 Spatial partitioning

> *The behavior of a component – i.e., the communication and the operations in the component's UNI – cannot affect the integrity of the communication of other components.*

The integrity of communication is determined by the fractions of correct and corrupted messages. A message is corrupted (invalidated), when its content was

altered during its transport. This comprises the time, which the message is in the output buffer, the network and in the input buffer of the receiving micro component, until it is delivered to the host. A modified or even malicious behavior of a component (see 5.1.1) may not bring a change of this property in the communication channels of other components.

5.1.3 Stability of communication during reconfiguration

> *The reconfiguration of one communication channel has no impact on the temporal properties and the data integrity of messages exchanged on another channel, that was not reconfigured.*

When an encapsulated communication channel is reconfigured the schedules of both sides – the sender and all receivers – need to be updated. As it is very likely that there also exist different communication channels, the reconfiguration of one channel must not influence the temporal and spatial properties (see 5.1.1 and 5.1.2) of the other communication channels.

5.1.4 Bounded reconfiguration delay

> *The reconfiguration of communication channels is predictable and completed within a bounded interval of time.*

It is of importance that the process of reconfiguration has finished within determined bounds. The reconfiguration of one communication channel in one component often requires changes in the whole schedule. Thus, also communication channels may be concerned which are not reconfigured. Especially for these channels the duration of the reconfiguration process is crucial, as an unpredictable and long duration of reconfiguration may also halt the communication of uninvolved communication channels.

For example, when one micro component needs to be exchanged by a spare micro component due to a permanent fault, also all of the communication channels used by the failed component must be reconfigured. If this micro component is part of the control system of a car that drives at around 100 km/h, the output on the actuators will be frozen until the reconfiguration is completed. An upper bound must exist for the reconfiguration as otherwise – when the reconfiguration needs too much time – an accident could be the consequence.

5.2 Evaluation of hypotheses

To evaluate the correctness of the hypotheses above, different experiments were applied. The first concentrates on the traffic load and the inter-arrival time (IAT), that is the duration between the arrival of two successive messages at the encapsulated communication channel. The second one is based on a bit-flip model to simulate the effect of transient, intermittent and permanent physical faults as well as software design faults of one host. As last experiment a reconfiguration scheme serves to determine possible side-effects of reconfiguration on the communication of static encapsulated communication channels, when the communication channels of the hosts are reconfigured.

As there is a difference in the semantics of event and state communication and, additionally, event messages can either be periodic or sporadic, each experiment is repeated with *periodic event messages*, *sporadic event messages* and *periodic state messages*.

In the following subsections the purpose and functionality of these experiments are explained in more details. The exact parameters and communication details can be found later in chapter 5 along with the complete experimental framework.

5.2.1 Traffic load experiment

The traffic load experiment addresses the question whether the communication behavior of one micro component can influence the temporal properties (latency, jitter, message order etc.) and the data integrity of the messages exchanged by other micro components. According to the first two hypotheses neither a component with its involved encapsulated communication channels that is added to the system nor any malfunctioning component, that generates an arbitrary amount of messages at any frequency, should be able to impose changes to the temporal properties and data integrity of the communication of other components.

To find out if the hypotheses can be violated a *reference communication* with two distinct periods will be established. Each of two components (*reference component 1 (*RC1*)* and *reference component 2 (*RC2*)*) sends messages containing a sequence number, the time of message generation – the time when the message was placed in the output buffer – and a checksum via this dedicated reference communication channels. Two other components (*gateway component (*GW*)* and *reference component 3 (*RC3*)*) receive this information and add the time of reception. During this communication a third communication channel is used on which another component (the *fault injection component (*FIC*)*) sends messages with a varying IAT starting from slow communication ($IAT_{slow} = 100ms$) to fast ($IAT_{fast} = 61\mu s$). This latter communication channel will be referred to as *probe*

communication. At a certain rate of the probe communication's IAT, messages will be lost as they are produced faster than the channel can transmit its packages. Nevertheless, this fact is not in conflict with the temporal and spatial partitioning property as it does not concern the reference communication channels.

From the recorded information the latencies, jitter, lost and corrupted messages can be calculated. If a change in the temporal properties of the reference communication can be observed when varying the IAT of the probe communication, then the hypothesis of temporal partitioning is disproved. On the other hand, if the number of lost and corrupted messages increases with shorter IAT, so spatial partitioning is not valid.

5.2.2 Bit-flip experiment

The bit-flip experiment is also aimed at the hypotheses of temporal and spatial partitioning. It investigates whether it is possible for one host in a micro component to affect the integrity or temporal properties of messages exchanged by other components. A component that writes to its UNI either in a correct way or due to a malfunction may not disrupt the communication of other micro components. No data messages should be lost nor any modification of a data field may occur. A change of temporal properties of the reference communication when bit-flips appear at the fault injection component are also forbidden, as temporal partitioning is assumed.

In order to determine whether this hypothesis is correct or if one channel can disturb another one, two different reference communication channels are observed. The components RC1 and RC2 each send messages containing a sequence number, the instant of global time of the message generation and a checksum via its corresponding communication channel. These packages are received at the components GW and RC3, which add a time stamp of the moment of reception to the data package. Also the FIC component sends messages with the same content on its own communication channel to the same receivers. Meanwhile, some bit-flips at an arbitrary position are introduced to the UNI of the FIC component. This may lead to the loss or corruption of messages on the probe communication of the FIC component or even cause its total breakdown. But on the other hand, the reference communication should not notice any of these effects.

The collected data packages are analyzed to find lost messages or data corruption. In case of lost or corrupted data packages the hypothesis of spatial partitioning is disproved. Additionally, latencies and jitter can be calculated to underline the results of the other experiments w.r.t. temporal partitioning.

5.2.3 Reconfiguration experiment

With the reconfiguration experiment the side-effects of reconfiguration of communication channels should be evaluated. According to the hypothesis of independence of reconfiguration, the reconfiguration of one encapsulated communication channel does not affect the communication on other communication channels, when temporal and spatial partitioning is given.

In order to prove that the reconfiguration of communication channels does not influence the communication of independent communication channels, the period and schedule of the FIC component – especially the probe communication channel – is reconfigured repeatedly at equidistant points in time. Meanwhile, the data packages on the reference communication channels are observed at the components GW and RC3. Although the probe communication channel is received at all other hosts and, hence, the schedules of all hosts need to be reconfigured simultaneously, no impact should be visible at the reference communication channels.

The reconfiguration is done by the trusted network authority (TNA). It receives a set of schedules for each processor which are then used to replace the old schedules. The TNA must send the schedules to each component in the similar period, so all components start execution of the new message descriptor list (MEDL) at the same point of global time. The MEDL contains all send and receive instants of the distinct communication channels.

Similarly to the traffic load experiment, the IAT of the probe communication is changed from slow ($IAT_{slow} = 10ms$) to fast message generation ($IAT_{fast} = 61\mu s$) from testrun to testrun. This may lead to the loss of data packages of probe communication when the IAT is shorter than the period, but it must not influence the reference communication channels.

After the experiment, the data of the reference and probe communication channels is inspected to find lost or corrupted data packages as well as changes of latency and jitter, respectively. Any discontinuity in the temporal properties or data integrity indicates the violation of temporal and/or spatial partitioning induced by reconfiguration.

5.3 Structural overview of framework

The structure of the TTSoC during the experiments was chosen to contain six micro components interconnected by the NoC that consists of two fragment switches (FS) and one TISS for each of the host processors. A schematic overview of the structure is presented in figure 5.1.

The *gateway micro component* (GW component) is connected via ethernet to a host

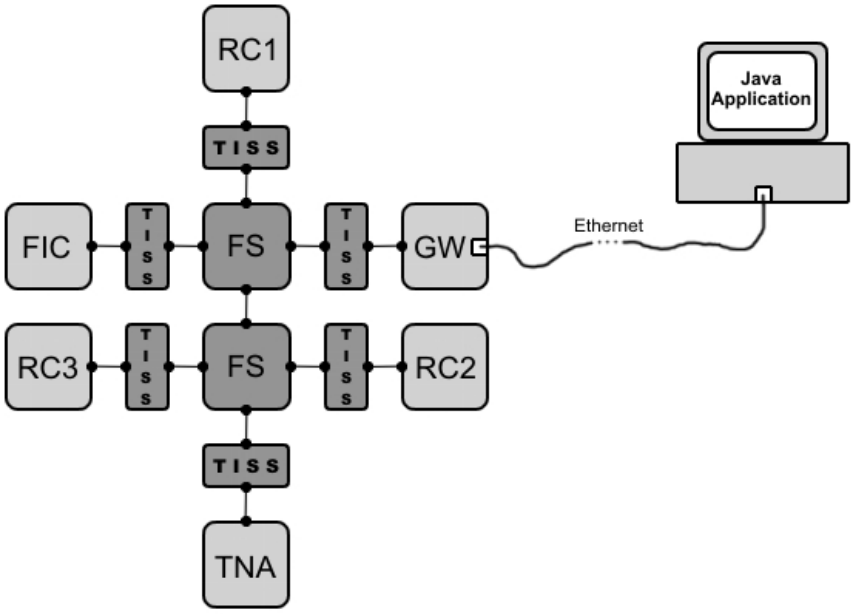

Figure 5.1: Structure of the experimental TTSoC setup: *each host processor (light gray) is connected to the communication subsystem (dark gray) by its local TISS*

PC which sends experimental parameters at the beginning of each testrun. Before the testrun actually starts the GW component communicates these parameters to the other micro components by the NoC. While the test is in progress the GW component stores communication packages both from reference communication and probe communication – that comes from the pretended faulty component. When a particular testrun is over the GW component pushes the experimental data, which was recorded during the testrun and collected by the GW via the NoC, through the ethernet interface to the host PC.

In the TTSoC architecture the *trusted network authority (*TNA*)* is part of the trusted and certificated communication subsystem. Here, it serves to enable fragment switches and to carry out the reconfiguration of the TISSes of the individual micro components. Especially, during the reconfiguration experiment this component performs the required changes of the schedules.

The *fault injection component (*FIC*)* builds one of the central components of the experiments. This component attempts to disrupt the communication of the reference components by intentionally introducing errors. As the TISS of the FIC is already part of the trusted subsystem the intentional misbehavior of the FIC may only address faults in the temporal and spatial domain, e.g. frequency of message generation and arbitrary bit flips in the uniform network interface (UNI).

In figure 5.1 the *reference components* (RC) are denoted by RC1, RC2 and RC3. These components establish a reference communication using different periods. While RC1 and RC2 continuously generate data packages that are sent via the NoC, RC3 receives this data and stores it in the component's local memory, similarly as the GW component does during the testrun. At the end of each testrun the GW component requests the collected data which is then transported over the NoC to the GW component. There it is forwarded via ethernet to the host PC.

On the *host PC* runs a Java application that drives the experiment. It is responsible for the reconfiguration of the experiment hardware, software download to the micro components, the parameterization of each testrun as well as for the logging of experiment data.

5.4 Fault injection environment

This section contains a description of the environmental elements of the fault injection experiments. This includes the used hardware as well as software tools utilized for development and experiment management.

5.4.1 Hardware

The prototype implementation of the TTSoC architecture was tested on an Altera® Stratix® III FPGA Development Kit. This board is a commercial off-the-shelf product which contains the required FPGA device as well as external memory and communication interfaces. Due to the high number of logic elements in the FPGA the entire hardware design fits in a single FPGA chip. Some important features of the Stratix® III Development Kit are detailed in table 5.1 (a full description can be found at Altera's product homepage[1]).

The development board was interconnected with a standard PC – that was used for experiment coordination and data logging – via the on-board USB-Blaster™ of the development kit. This connection served as configuration interface for all automatic testruns, i.e. hardware design download to the FPGA and programming of micro processors, and for debugging during the development of the experiments. A second connection was established using an Ethernet interface which was responsible for parameter setup at the beginning of each testrun and for data collection when a testrun had finished. The ethernet connection uses the on-board Marvell® 88E1111 Ethernet PHY base-T device that is directly connected to the FPGA.

[1] http://www.altera.com/products/devkits/altera/kit-siii-host.html

Feature	Description
FPGA	Stratix III EP3SL150F1152 142 500 equivalent logic elements 744 user I/O pins 384 18 x 18 multipliers
Memory devices	128 Mbyte DDR2 SDRAM DIMM 16 Mbyte DDR2 SDRAM devices 36 Mbit QDRII SRAM device 4 Mbyte PSRAM 64 Mbyte flash memory
Clocking	125 MHz oscillator 50 MHz oscillator
Configuration	On-board USB-Blaster™ JTAG download port
User I/O	System reset pushbutton Reset pushbutton DIP switch (x8) LEDs (x8) Quad 7-segment display 128 x 64 dot pixels graphics display LCD (16 character x 2 line)
Interfaces	USB 2.0 10/100/1000 Ethernet HSMC interfaces

Table 5.1: Features of the Stratix® III FPGA Development Kit

5.4.2 Development software and configuration

The FPGA hardware design was generated using Altera's Quartus® II v8.1 Subscription Edition design software. It is a tool that transforms source code written in an hardware description language (e.g., VHDL) into a chip layout either for downloading onto an FPGA or for manufacturing of an application specific integrated circuit (ASIC). Additionally, the SOPC Builder (System on a Programmable Chip Builder) tool, that comes with the Quartus® software, was applied for assembling TTSoC network components and micro components. The SOPC Builder is a graphical tool that provides a wide range of standard components as micro processors, memory units, interfaces, etc. which can be interconnected to compose larger systems or even systems-of-systems.

For the development of the micro component's software the Altera Nios® II Embedded Design Suite (EDS) was used. It is a C/C++ software development tool

especially designed for Nios® micro processor systems that were composed by the Quartus® tool chain. If desired the Nios® II EDS automatically adds the MicroC/OS-II real-time operating system or the NicheStack® TCP/IP Stack in case of an implemented ethernet connection, which are both free in case of non-commercial usage.

Both, hardware configuration and software download to the micro processors are done with an JTAG interface connected to the on-board USB-Blaster™. For hardware configuration setup the Quartus® II Programmer was used which was repeatedly called by the Java experiment management application. Similarly, the Nios® II download tool was called after the hardware was configured. This stores the binary program code into the corresponding instruction memory of each micro processor and resets the processor afterwards to start its execution.

5.4.3 Experiment coordination and data logging

As every experiment consists of thousands of individual testruns that produce a huge amount of logged data, an application is needed to automatically start each testrun and store the data for further examination. Thus, a simple Java application was developed that allows the variation of the experiment's parameters to find the optimum values for the actual testruns. This Java software runs on the Eclipse platform[2] which is an open source developing environment that already incorporates the necessary Java Runtime Environment.

For each testrun the software downloads the hardware design via JTAG UART into the FPGA by invoking the Quartus® Programmer with the binary file as parameter. Afterwards a batch file is called that copies the download script into the execution directory of the Nios® II EDS and launches the Nios® II Shell which automatically starts executing the previously copied script. Step-by-step the Nios® II downloader stores the binary program code of each processor in the corresponding instruction memory of each processor in the SoC. When the program was downloaded the processor is reset, so it starts execution at its reset vector which points to the newly installed instructions and the processor is doing its tasks.

After all processors are equipped with their programs it can be assumed that the GW component already created an ethernet server socket. Hence, the Java application tries to establish a connection to the GW component. If this was successful the parameters for the actual testrun are calculated and sent via ethernet to the GW component. Otherwise, in case the connection could not be established, the testrun is aborted.

[2]Downloaded from: http://wwww.eclipse.org

During the actual testrun on the development board the software waits until the GW component starts sending the recorded communication data. This data is stored in an *.csv-file* for each testrun individually. Subsequently, the next testrun starts by downloading the hardware design to the FPGA board. All data files are finally processed by some Matlab scripts to obtain the results.

5.5 Experimental TTSoC structure

The following sections present a more detailed description of the TTSoC structure, which was already introduced in section 5.3. At first, the concrete implementation of the components illustrated in figure 5.1 is explained. Afterwards, the NoC with its communication channels and the communication structure will be defined.

5.5.1 Micro components

As each of the micro components has its specific tasks in the experiments the components are equipped with different CPUs and peripherals. Additionally, the equipment of the Stratix® III Development Kit impose restrictions on the design of micro components. Especially, the number of FPGA internal memory blocks is limited what implies that host processors with a high amount of stored data – GW component, RC3 and the TNA in the reconfiguration experiment – need to be extended with external memory devices. That, in turn, entails that these processors may not run at a frequency as high as it is possible for processors with internal memory, because the access time of external memory devices is higher than for internal memory blocks. Table 5.2 summarizes the features of the micro components in the experimental TTSoC design.

	GW	TNA	FIC	RC1	RC2	RC3
Processor	Nios II/f	Nios II/s (Nios II/f)	Nios II/s	Nios II/s	Nios II/s	Nios II/s
CPU clock freq.	100 MHz	100 MHz	200 MHz	200 MHz	200 MHz	100 MHz
Instruction cache	32 kB	2 kB	512 Byte	512 Byte	512 Byte	8 kB
Memory type	external	external	internal	internal	internal	external
Memory size	16 MB	1 MB	104 kB	88 kB	88 kB	15 MB
Port memory size	65 kB	8 kB	16 kB	8 kB	8 kB	16 kB
Interrupt timer	1 ms	1 ms	15 μs	1 ms	1 ms	1ms

Table 5.2: Features of the micro components in the experimental TTSoC design

The different components are composed as follows:

GW component: This component uses a Nios II/f CPU which is a 32-bit RISC processor that supports additional data caches. Data caching was necessary since otherwise the data sent by ethernet was corrupted by conflicting access of CPU and ethernet controller to the data and instruction memory. Hence, a data cache with a size of 8 kB was implemented beside the 32 kB of instruction cache. The CPU runs at a frequency of 100 MHz and a timer interrupt period of 1 ms.

Due to the high amount of data accumulated during each testrun, this component is connected to an external 16 MByte DDR2 SDRAM device which serves as instruction as well as data memory. This brings the advantage of a big data storage with the expenses of slower data access.

Additionally, the GW component is equipped with the Opencores 10/100 Ethernet MAC 8.03 with Avalon interface [3]. It is an open source media access controller which drives the Marvell® 88E1111 Ethernet PHY that is mounted on the Development Kit board. The Ethernet PHY is responsible for the generation of the electrical signals on the physical ethernet medium.

TNA component: The TNA component uses the less advanced Nios II/s CPU with 2 kB instruction cache during the traffic load and bit-flip experiments. It is also a 32-bit RISC processor but does not support data caches. For the reconfiguration experiment the processor type was changed to the Nios II/f CPU, as due to the high amount of reconfiguration information the caching of data became necessary. Hence, a data cache with a size of 2 kB was added.

In both cases the processor runs at a frequency of 100 MHz and has a timer interrupt period of 1 ms. As the amount of internal memory blocks is rather limited and the MEDL information in the reconfiguration experiment requires several kB of memory the TNA is also connected to an external DDR2 SDRAM device which it shares with the RC3 component. Nevertheless, the memory it may use for instructions is restricted to 1 MB.

Due to the ability of the TNA to enable/disable the fragment switches, a two bit wide I/O port is needed. Hence, each of the bits enables/disables one of the fragment switches.

FIC component: A Nios II/s processor that runs at 200 MHz builds the heart of this component. It comprises 512 bytes of instruction

[3]Downloaded from: http://www.niosforum.com/pages/project_details.php?p_id=115&t_id=18

cache. Due to the message generation paradigm of the FIC, which is based on a timer interrupt, the timer needs the short period of 15 μs to be able to generate the desired number of messages per second.

The 104 kB of data and instruction memory are built of fast internal memory blocks.

RC1 and RC2: Both, RC1 and RC2 are identical components that only differ in the message generation period. They are composed of NiosII/s processors with 512 bytes of instruction cache that run at 200 MHz. Because of the small size of program code and a nearly vanishing amount of stored data the whole memory consists of 88 kB internal memory. The interrupt timer has a period of 1 ms as for this components new messages are automatically generated when the old one was successfully transmitted.

RC3 component: This component primarily has to accumulate data that was transmitted on the NoC. Hence, it is connected to an external DDR2 SDRAM device which it shares with the TNA component, however, 15 MB are reserved to the RC3 component. The processor used is also a Nios II/s with a frequency of 100 MHz and 8 kB instruction cache. The interrupt timer of the RC3 component is set to 1 ms.

Most of the interrupt timers are only used to determine the end of the experiment, hence, they need no fine granularity. Only the FIC component uses its timer for message generation that's why its timer period is that short. Additionally to the mentioned peripherals all components are equipped with a JTAG UART for program download and debugging.

5.5.2 Network-on-Chip (NoC)

To connect a component with the TTNoC a TTSoC frontend module is appended to each processor. This module is already part of the NoC and offers the possibility to the micro component's host processor to access its TISS. While the TTSoC frontend contains the control and status interface of the TISS, the actual state or event information is placed into and read from the port memory.

The port memory is implemented as dual-ported memory which can be accessed by the processor as well as by the NoC. Because of the different communication demands of the components the size of the port memory is varying from component to component – it can be found in table 5.2.

In figure 5.1 the network structure of the experimental setup is shown. There are two fragment switches linked together to form the backbone of the communication

network. On each of these fragment switches three micro components are connected, where GW, FIC and RC1 share the first fragment switch and TNA, RC2 and RC3 the second one.

5.5.3 Communication channels

Different encapsulated communication channels are used to perform the testruns. Each of them is dedicated to a destined task during the experiments. Table 5.3 gives an overview of the encapsulated communication channels that were used to perform the experiments. For each channel the sender and all receivers are given. The *periods* – i.e., the time between two successive messages on a channel – are indicated in seconds along with the size of the messages in 32-bit data words. The port type gives the semantics of the data transmitted by the corresponding channel. An event port only transfers data if a new message was explicitly pushed into the interface by the host. On the other side, a state port sends the data that was most recently created once in a period. At last the sender port id and the receiver's port id are presented which can be arbitrarily assigned except for port 127 that is used for reconfiguration purposes on receiver-side.

The encapsulated communication channels of table 5.3 are explained as follows:

ProbeCom: Communication channel used by the FIC component to transmit messages containing a sequence number, a time stamp of message generation and a checksum – resulting in three 32-bit words. The communication on this channel is accelerated during the experiment to test the temporal and spatial partitioning of the encapsulated communication channels. For the traffic load and bit-flip experiments a short period should result in the transmission of a high amount of messages while the test is running. During the reconfiguration experiment the period of the probe communication is changed by the TNA to find possible side-effects of reconfiguration.

Depending on the actual experiment parameter this channel is either interfaced as event port or state port.

RefCom: Both channels are needed as constant communication during the experiment. The same messages as for the probe communication are exchanged. A short and longer period are used to simulate fast and a slower communication while enough messages are transmitted to observe any discontinuity. If there is any change in the temporal properties (latency, jitter, etc.) of messages exchanged on this channels then the temporal partitioning of the TTSoC architecture would be disproved. In the same way, if there are any lost or corrupted messages

Channel	Sender	Receiver(s)	Period (in sec)	Message size (in words)	Port type	S-Port	R-Port
ProbeCom	FIC	GW	2^{-12}	3	event &	16	16
		RC1	(2^{-5}		state		16
		RC2	to				16
		RC3	2^{-14})				16
RefCom1	RC1	GW	2^{-12}	3	event &	17	17
		RC3			state		17
RefCom2	RC2	GW	2^{-9}	3	event &	18	18
		RC3			state		18
MgtCom	GW	FIC	2^{-2}	1	event	0	0
		RC1					0
		RC2					0
		RC3					0
SchedCom	GW	TNA	2^{-3}	128	event	100	100
DataCom1	RC1	GW	2^{-6}	361	event	1	1
DataCom2	RC2	GW	2^{-6}	361	event	2	2
DataCom3	RC3	GW	2^{-6}	361	event	3	3
DataCom4	FIC	GW	2^{-6}	361	event	4	4
CfgCom1	TNA	GW	2^{1} (2^{-3})	832 (100)	event	64	127
CfgCom2	TNA	FIC	2^{1} (2^{-3})	832 (100)	event	65	127
CfgCom3	TNA	RC1	2^{1} (2^{-3})	832 (100)	event	66	127
CfgCom4	TNA	RC2	2^{1} (2^{-3})	832 (100)	event	67	127
CfgCom5	TNA	RC3	2^{1} (2^{-3})	832 (100)	event	68	127

Table 5.3: Encapsulated communication channels of the experiments. The numbers in brackets denote that this value is changed for other experiments.

on this channels which can be traced back to the actual type of fault injection, then the spatial partitioning of the TTSoC architecture need to be questioned.

Similarly to the probe communication, this channel is implemented both as event and as state port, depending on the actual parameters.

MgtCom: The GW component uses this channel to transmit the parameters of the actual testrun to all participants. As all recipients get the message at the same instant of global time, the involved components are synchronized by the reception of a management message. Before the end of a testrun the GW component broadcasts a message on this channel which is only processed by the RC3 component. This message signifies that the GW component is ready to receive the accumulated

data packages from RC3 for relaying them to the host PC.

As this channel is not needed very often and with only a little amount of data a large period with one 32-bit data word is used. For not transmitting useless management messages during the testrun, this channel works with event semantics.

SchedCom: Only in the reconfiguration experiment this channel between the GW component and the TNA is needed. It is used to transmit the new reconfiguration data – ten different schedules for each host – to the TNA component. Due to the high amount of communicated data this channel uses a medium period with large packages.

After the schedules are sent to the TNA this channel is not needed any more. Furthermore, the transmitted data has event semantics, hence, this channel is implemented as event port.

DataCom1,2 and 4: These channels are applied to send a periodic sign of life to the GW component until the actual testrun is started. Hence, the GW component knows which of the processors are already booted and thus are ready to receive the testrun parameters. A middle period helps to shorten the start-up phase while the message size allows to use this channels also for data transfer and debugging. As after the start no more communication is needed on these channels, they are implemented for event communication.

DataCom3: While the system starts up, this channel is used by the RC3 component to generate a sign of life, like the other data communication channels do. Later, the recorded communication messages of probe and reference communication are transported through this channel to the GW component. Several kilo bytes of data need to be transferred by this channel. Thus, a middle period with high message size – that can carry 72 messages of probe and reference communication – was chosen for the channel. Also here an event port is applied as the communication is irregular during the experiment.

CfgCom: All five reconfiguration channels are used by the TNA to enable the different periods at the TISS of each micro component and to change the schedules of the encapsulated communication channels. Because this is done only once during the testruns of the traffic load and bit-flip experiments the largest period possible is sufficient. For the reconfiguration experiment this period is shortened to achieve a higher rate of reconfigurations. The message size was reduced for the reconfiguration experiment, as the port memory space was needed for queuing purposes.

This channel also serves for synchronization during the reconfiguration experiment. The actual test phase starts after the first reconfiguration is achieved.

At receiver-side the reconfiguration needs to be at port 127 as this is done automatically by the TISS. Event semantics best fit for the reconfiguration purposes of this communication channels.

5.5.4 Communication structure

The structure of communication of the fault injection experiments is presented in figure 5.2. The numbers in the figure represent the sequence of communication in the experiment.

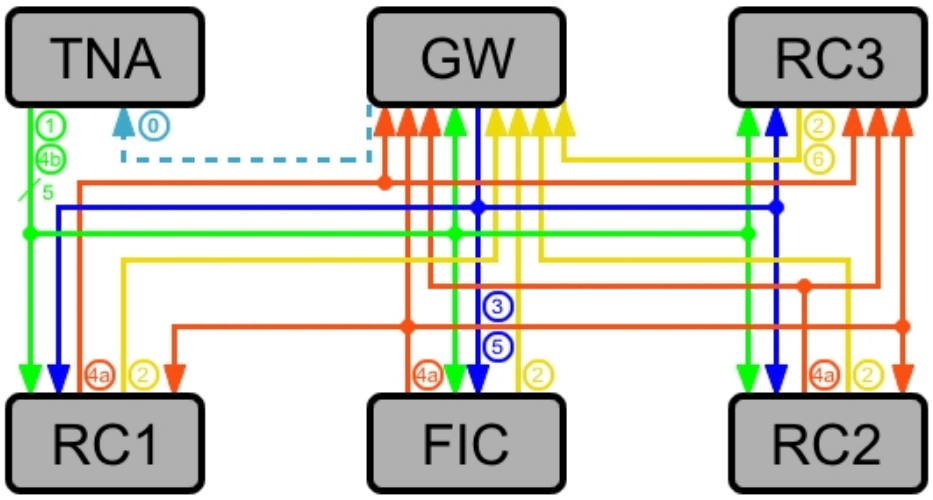

Figure 5.2: Communication structure of the experiments. The dashed line is only used in the reconfiguration experiment.

The workflow is as follows:

0. **Only in reconfiguration experiment:** The GW component sends the schedules for reconfiguration to the TNA component
1. Reconfiguration of TISSes by the TNA component – five different channels are used though only one is depicted
2. Components FIC, RC1, RC2 and RC3 signal that the processor is online

3. The GW component pushes the testrun parameters on the MgtCom channel

4. During the test phase:

   (a) Components FIC, RC1 and RC2 send data messages which are recorded at GW and RC3

   (b) **Only in reconfiguration experiment:** The TNA reconfigures the TISSes of the micro components

5. A request for experiment data is sent from the GW component to RC3

6. RC3 sends all collected RefCom and ProbeCom messages to the GW component

5.6 Experiments in detail

This section details characteristics of the experiments, like data structures, experiment parameters, message generation schematics, etc., that are individual for each experiment.

5.6.1 Traffic load experiment

The general setup, that is already described in the previous sections, builds the basis for this experiment. To carry out the traffic load experiment a software for each of the micro components was developed that uses the encapsulated communication channels, discussed in subsection 5.5.3 for management purposes, synchronization and the actual experimental communication.

The data structures of the actually transmitted messages are presented in the following paragraphs. Also the exact parameters used to accomplish the traffic load experiment are detailed here. One of the most important property for this experiment – the scheme for generation of new messages – builds the last paragraph.

Data structures

Different data structures were used on the distinct communication channels and the ethernet connection. In the following tables the data fields of the most important structures are detailed. Each table contains the name of the data field in the C program code, the field size in bytes and a short description.

Name	Size in bytes	Description
cfg_duration	2	Duration of the test phase
cfg_kValue_low	2	inter-arrival time (IAT) in μs – low value
cfg_kValue_high	1	IAT in μs – high byte
cfg_random	1	Random value for the testrun
cfg_type	1	Actual type of the testrun (event, state, periodic, sporadic)

Table 5.4: *experiment_config:* Configuration data structure for the traffic load experiment from the PC to the FPGA board.

Name	Size in bytes	Description
type_random	1	Type of testrun and random value
kvalue_high	1	High byte of the IAT in μs
kvalue_low_duration	2	Low value of the IAT in μs and duration of test phase

Table 5.5: *ttsoc_mgt_msg:* Management data structure for the traffic load experiment from the GW to the other components.

Name	Size in bytes	Description
msg_seqNr	4	Sequence number of the reference and probe communication channel
msg_timestamp	4	Global time base timestamp of the generation instant
msg_checksum	4	Sum of sequence number and timestamp

Table 5.6: *ttsoc_com_msg:* Communication data structure of reference and probe communication.

Name	Size in bytes	Description
msg_rcv_time	8	Global time base timestamp of the reception instant
msg_received	12	Message of type ttsoc_com_msg that is received from reference or probe communication channel

Table 5.7: *ttsoc_incoming_msg:* Data structure for incoming messages of reference and probe communication.

Name	Size in bytes	Description
sender_id	1	Identification of the sending component
channel_nr	1	Identification of the channel this data is coming from
msg_type	1	Signal if there are further messages following or if all data of the channel was transmitted
pkgs_transmitted	1	Number of messages transmitted in the actual data message
data_pkg	1440	Array of data messages of type ttsoc_incoming_msg

Table 5.8: *ttsoc_data_msg:* Data structure to transmit the collected messages from the RC3 component to the GW and to the PC.

Experiment parameters

Table 5.9 presents the parameters that were applied for the traffic load experiment. The inter-arrival time (IAT) – that denotes the duration between the arrival of two successive messages at an encapsulated communication channel – is given for the three different communication channels used during the actual test phase. Especially the parameters of the probe communication channel are defined more precisely because they are varied during the experiment.

The whole experiment is repeated three times for:

- event communication with periodic messages
- event communication with sporadic messages
- state communication with periodic messages

Each of these three sub-experiments consists of 2001 testruns where everyone of the testruns is composed of three phases. The reconfiguration of the test hardware, the test phase with the test communication on the board and the final collection of communication data. The actual duration of the communication during a test phase was defined to be one second.

In all testruns where event communication was used, the reference communication messages are periodically generated. Only the probe communication is switched from periodic to sporadic behavior. If the messages of the reference communication would be sent in a sporadic way – in form of message generation at random instants in time – one could not know whether the changes in the temporal properties of messages exchanged come from the moment of message generation or from influences of the probe communication channel.

Parameter	Event port & periodic messages	Event port & sporadic messages	State port & periodic messages
Number of testruns	2001	2001	2001
Duration of testrun	1000 ms	1000 ms	1000 ms
IAT of RC1 (reference comm.)	0.24 ms	0.24 ms	0.24 ms
IAT of RC2 (reference comm.)	1.95 ms	1.95 ms	1.95 ms
Max. IAT of FIC (probe comm.)	100 ms	100 ms	100 ms
Min. IAT of FIC (probe comm.)	61 μs	61 μs	61 μs
Msg. transmission period (probe comm.)	0.24 ms	0.24 ms	0.24 ms
Max. random value (RV_{max})	–	2 x IAT	–
Generation scheme (probe comm.)	IAT	IAT + IAT_{RV}	IAT

Table 5.9: Parameters of the traffic load experiment.

Message generation scheme

Messages for both reference communication channels are produced with a constant IAT in all testruns. Hence, the temporal properties of these channels should be constant as well. Otherwise, the temporal partitioning property of the TTSoC architecture would be disproved. RC1 uses *period 2* with 4096 messages per second while for RC2 *period 5* with 512 messages per second (msg/sec) is employed.

The probe communication channel uses *period 2* for message transmission during the whole experiment. On the other side, the generation of messages is varied from 10 msg/sec (or 100 ms IAT) to 16 384 msg/sec (or 61 μs IAT). In case the FIC component generates more than 4096 msg/sec it produces its messages faster than they can be transmitted by the communication subsystem. When the send buffer is full with messages to send, new messages are thrown away and this data is lost. The rate of change of the IAT (Δ_{IAT}) from testrun to testrun is constant on the number of msg/sec. It is calculated by the following equation:

$$\Delta_{IAT} = \frac{16\ 384\ msg/sec - 10\ msg/sec}{2001\ testruns\ - 1} \tag{5.1}$$

For periodic communication the calculated IAT is used to set the timer for the next message generation. Differently to this scheme, the simulation of sporadic

communication is done by requesting a new random value for each generated message and adding this random value to the IAT.

The actual IAT (IAT_{act}) between two messages can be calculated by the following equations:

$$IAT_{RV} = (RAND \ mod \ (IAT * RV)) \tag{5.2}$$

$$IAT_{act} = IAT + IAT_{RV} \tag{5.3}$$

In the equation $RAND$ means the random value generated by the C-command *rand()*. As this command returns at least a 15-bit value it must be calculated modulo the maximum random value (IAT * RV) to get the actual random value. Due to the fact that the random parameter $RV = 2$ and the *rand()* function is uniformly distributed, the mean value of the random part is around one IAT. Hence, the average IAT (IAT_{avg}) of the sporadic communication is about $2 * IAT$.

5.6.2 Bit-flip experiment

As in the traffic load experiment, the general setup discussed in the preceding sections builds the fundament of the bit-flip experiment. The particular structure of this experiment is very similar to the traffic load experiment. Even the data structures are quite similar and only modified very slightly. Hence, the paragraph about data structures just contains the changes from the data structures already listed for the traffic load experiment.

Data structures

The changes in the data structures origin from the value of the IAT that was left out. Instead a random initial value was introduced to initialize the random generator of the FIC processor. Without this new value the pseudo random generator would produce the same sequence of random values in each testrun. This is because of the automatic reset of the hardware after each testrun. As especially for the bit-flip experiment different sequences of random values are vital the initial value for the random generator comes from the PC.

Experiment parameters

The parameters of the bit-flip experiment are presented in table 5.12. Nearly all parameters are the same for the three sub-experiments. The IATs for reference and probe communication are constant except for the sporadic message generation,

Name	Size in bytes	Description
cfg_duration	2	Duration of the test phase
cfg_rand_init	2	Initialization value for the rand() function
cfg_random	1	Random value for the testrun
cfg_type	1	Actual type of the testrun (event, state, periodic, sporadic)

Table 5.10: *experiment_config:* Configuration data structure for the bit-flip experiment from the PC to the FPGA board.

Name	Size in bytes	Description
rand_init_duration	2	Initialization value for the rand() function and duration of the testrun
type_random	1	Type of testrun and random value

Table 5.11: *ttsoc_mgt_msg:* Management data structure for the bit-flip experiment from the GW to the other components.

where the IAT is influenced by a random value. Similarly to the traffic load experiment, the reference messages are always periodically generated, where *period 2* is used by RC1 and *period 5* by RC2. The message transmission of the probe communication is also done with *period 2*.

Parameter	Event port & periodic messages	Event port & sporadic messages	State port & periodic messages
Number of testruns	2000	2000	2000
Duration of testrun	1000 ms	1000 ms	1000 ms
IAT of RC1 (reference comm.)	0.24 ms	0.24 ms	0.24 ms
IAT of RC2 (reference comm.)	1.95 ms	1.95 ms	1.95 ms
IAT of FIC (probe comm.)	0.24 ms	0.24 ms	0.24 ms
Max. random value (RV_{max})	–	5	–
Generation scheme (probe comm.)	IAT	$IAT * RV_{act}$	IAT
Bit-flip interval	1 ms	1 ms	1 ms

Table 5.12: Parameters of the bit-flip experiment.

Sporadic messages are generated by simply multiplying the IAT with the actual random value (RV_{act}), which is generated by the processors pseudo random gen-

erator. This value is limited to RV_{max}, that is set to 5 and hence, an average IAT (IAT_{avg}) of $3 * IAT$ is reached.

The bit-flip interval in table 5.12 gives the time between two successive bit-flips. As it is set to 1 ms about 1000 bit-flips occur during each testrun, which last 1000 ms.

Bit-flip model

The bit-flip experiment simulates the incidental change of one or more values in the UNI of a host processor. Such a bit-flip may occur either by any malfunction of the host as well as by transient environmental upsets. As these events do not follow any rule, the bit-flips need to be at an arbitrary position but uniformly distributed.

According to [Pau08] the UNI of a host consists of four different memory regions:

- Port Memory (Data Memory)
- Port Configuration Memory
- Port Synchronization Memory
- Register File

Because not all of the available memory space is actually used by the TISS, the experiment concentrates on those regions in memory, where a bit-flip may have any influence on the system behavior. A schema of this memory structure is shown in figure 5.3. The port memory is physically separated from the rest of the UNI and consists of 4096 words of 32-bit length. Port configuration and port synchronization memory both use 128 words of memory while the register file has a width of 15 words.

As can be seen in table 5.12, a bit-flip occurs every milli second. To do so, firstly the memory region for the value change is chosen by the random generator. After one of the four regions is selected, the concrete memory word (32-bit) is calculated by requesting a random value and then doing a modulo operation with the actual memory region width. At last, an arbitrary position for the bit-flip inside the data word is found by using the random generator a third time. Finally, the selected word is read from the memory, the bit value at the required position is changed and the data word is written back to memory.

A bit-flip in the port memory may only lead to the corruption of data messages, because this memory region does not contain any control or status information. Contrary to this are bit-flips in other memory regions, which can result in data

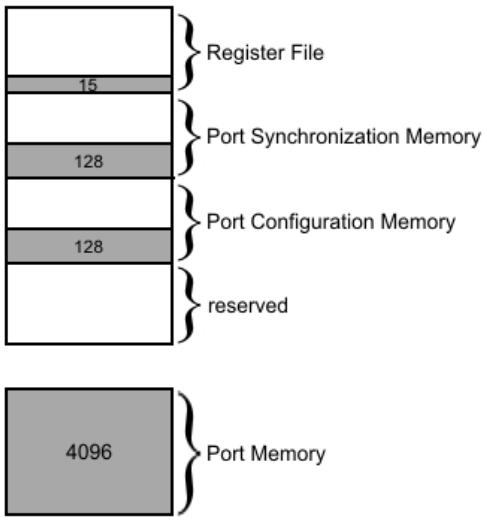

Figure 5.3: Memory structure of the UNI of the bit-flip experiment. Gray regions are used by the TISS. The port memory is physically separated from the other regions.

loss or the shut down of a channel or the host's complete communication subsystem. Due to the uniform distribution of the random generator output each of the memory regions will be selected around the same number of times. Hence, a smaller region is covered better with bit-flips than a bigger memory space. Table 5.13 presents the figures about the statistical coverage of each memory region.

Memory region	Memory size (bits)	Coverage (times)
Port Memory	131 072	3.81
Port Configuration Memory	4 096	122.07
Port Synchronization Memory	4 096	122.07
Register File	480	1 041.66
Total	139 744	14.31

Table 5.13: Statistical bit-flip coverage of different memory regions.

During each sub-experiment 2000 testruns are executed with each containing 1000 bit flips. This results in a total of about 2 Mio. bit-flips. Statistically a quarter of them happen at every distinct memory region. In total all bit positions of the UNI are flipped around 14.31 times, which means that each bit changes its value more than 14 times during the whole sub-experiment. Due to the fact, that corrupted data messages may not disturb the communication of other channels, the coverage

of the portmemory is kept relatively low compared to control and status interfaces.

5.6.3 Reconfiguration experiment

Also for the reconfiguration experiment the background setup has already been discussed. It uses some additional channels and some modifications of the hardware design. Apart from this, the reconfiguration experiment is very similar to the traffic load experiment, except that the TNA component periodically reconfigures the schedules of all components. The next paragraphs describe the differences to the data structures of the previous experiments, the actual parameters of the reconfiguration experiment and the process of reconfiguration.

Data structures

The data structures used for this experiment are the very similar to the structures already presented. Only for the communication from the PC to the TTSoC had been adapted to enable the transmission of the schedules. The information about the size and a checksum of the schedules is sent in the first data block from the PC to the FPGA board. Afterwards, the actual schedules are pushed as a byte stream which is finally examined at the FIC component to find the real beginning of each individual schedule. The checksum byte – which is a sum over all bytes of the schedules modulo 256 – is first calculated at the host PC. The same calculation is done at the FIC component. In case of a checksum mismatch at the receiver side the testrun can not be started and must be repeated.

Name	Size in bytes	Description
cfg_duration	2	Duration of the test phase
cfg_rand_init	2	Initialization value for the rand() function
cfg_random	1	Random value for the testrun
cfg_type	1	Actual type of the testrun (event, state, periodic, sporadic)
cfg_schedule_size	2	Size of schedules in byte
cfg_check_byte	1	Byte for end-to-end check of schedule
cfg_kValue_high	1	IAT in μs – high byte
cfg_kValue_low	2	IAT in μs – low value

Table 5.14: *experiment_config:* Configuration data structure for the reconfiguration experiment from the PC to the FPGA board.

Experiment parameters

As the reconfiguration experiment is based on the traffic load experiment, also most of the parameters are kept the same. In table 5.15 all parameters of the experiment are listed. The duration of the testruns was changed to 1500 ms to allow 10 reconfigurations during one testrun. Reference communication periods and message generation scheme remain unchanged and a description can be found in subsection 5.6.1.

Due to the reconfiguration process, the period of the probe communication can be varied from *period 9* with 32 msg/sec down to *period 0* with 16 834 msg/sec. The very slow period at the beginning of each testrun results in the loss of data packages of the probe communication as more messages are generated than can be transmitted. With an increased period the percentage of lost messages must decrease. For reconfiguration purposes *period 11* is applied. Hence, the duration between two reconfigurations is 125 ms or eight reconfigurations per second.

Schedule structure

Basically, a particular schedule that is sent to one micro component is a bit stream of alternating configuration flits and blocks of actual scheduling information. The configuration flit is a header with information about the memory region to which the following data is written, together with the size of this upcoming data. At last a – so called – terminal flit is introduced to signal the end of scheduling information to the receiver's TISS.

A TISS that receives such an information on *port 127* automatically interprets the data and stores it to the according memory region. Each of this memory regions has a specific purpose.

The memory regions to which is written during the reconfiguration process are:

- message descriptor list (MEDL): contains information about when, which port is sending or receiving data
- routing information memory (RI): the route from the sender to all receivers is stored in this memory
- burst configuration memory (BCFG): specifies the position in port memory where the data of a channel is loaded from or stored to
- Register file: only used to enable the required periods

Parameter	Event port & periodic messages	Event port & sporadic messages	State port & periodic messages
Number of testruns	2001	2001	2001
Duration of testrun	1500 ms	1500 ms	1500 ms
IAT of RC1 (reference comm.)	0.24 ms	0.24 ms	0.24 ms
IAT of RC2 (reference comm.)	1.95 ms	1.95 ms	1.95 ms
Max. IAT of FIC (probe comm.)	100 ms	100 ms	100 ms
Min. IAT of FIC (probe comm.)	61 μs	61 μs	61 μs
Max. transmission period (probe comm.)	31.25 ms	31.25 ms	31.25 ms
Min. transmission period (probe comm.)	61 μs	61 μs	61 μs
Max. random value (RV_{max})	–	2 x IAT	–
Generation scheme (probe comm.)	IAT	IAT + IAT_{RV}	IAT
Reconfiguration interval	125 ms	125 ms	125 ms
Number of reconfigurations	10	10	10

Table 5.15: Parameters of the reconfiguration experiment.

Reconfiguration process

All schedules for each host are calculated in advance by the host PC (see section 5.8). There are 10 different schedules for every host, which differ in the period of the probe communication. The other encapsulated communication channels remain unchanged in period and instance of port operation. At the beginning of each testrun these schedules are transmitted via the GW component to the TNA. After that, the TNA enables the necessary periods of the remaining micro components. Before the actual test phase can start, the TNA must wait some instants until the GW component transmitted all management data.

It is essential for the correct function of the TTSoC that all micro components switch to the new schedule at the same instant of global time. Otherwise, the sending operations of the communication subsystem may be unsynchronized and messages can be lost. To transmit the schedules for each host within the same in-

stant of global time, the communication subsystem of the TNA is disabled. Then, the schedules are written into the port memory and afterwards the communication subsystem is enabled again. The first one of the schedules of each host triggers the start of the actual test phase at each host. At the beginning the probe communication period is 31.25 ms.

After one reconfiguration is completed, the next schedule is sent to the reconfiguration queue of the TNA. This ensures that the schedules of all hosts are already in the port memory when the next reconfiguration is done by the TNA. The fastest period (61 μs) of the probe communication is when the last schedule is transmitted.

The newly reconfigured schedule is activated when the reconfiguration instant is triggered. This instant must be at the same time at all components and lets the scheduler restart at the initialization vector which points to the new schedule.

5.7 Test procedure

The Java application on the host PC starts with downloading the hardware design to the FPGA board and installing the programs of all micro components. In case of the reconfiguration experiment before the design and programs are downloaded, the schedules must be calculated for all hosts and reconfiguration steps. This is done only once at the beginning of the experiment because the schedules are similar for all testruns.

As the GW component is one of the first processors that start execution – while other components are still waiting for their program code – it can be assumed that it already created an ethernet server socket when the Java management application finishes installing the programs of all processors. The PC creates an ethernet client socket and establishes an ethernet connection to the GW component on the FPGA development board. Parameters for the actual testrun are calculated and pushed – together with the schedules, in case of the reconfiguration experiment – via ethernet to the GW component.

For the reconfiguration experiment the schedules are transmitted to the TNA component via the schedule communication channel. If there would be a failure in this communication the checksum at the TNA component would not match and the TNA stops its operation. Then a timeout is raised as the micro components can not be reconfigured and the testrun must be repeated.

Before the real test communication is done, the communication channels – with periods faster than $2s$ – need to be enabled by the TNA. This is done by sending a reconfiguration message through the NoC to each of the micro components. For this purpose an encapsulated communication channel is established between the TNA and each component named configuration channel (see CfgCom1-5 in

subsection 5.5.3).

Whenever a component started execution and has done its initialization routines it sends a sign of life to the GW component to signal that it is ready for the testrun parameters. The GW component waits until all participating components have sent such messages. If one component cannot be started correctly it does not send this sign of life and a timeout restarts the testrun. Afterwards, the GW component transmits the parameters of the actual testrun which it received previously from the PC. The data is split up into two messages which are received at all participants of the experiment simultaneously. The channel used for the distribution of the testrun's parameters will be referred to as management communication channel (MgtCom).

At the beginning the components are not synchronized to each other as each processor starts executing its program immediately after its individual reset. Because the messages in the TTNoC are received at the same moment of global time by all receivers, the second management message of the GW component serves as synchronization instant. Upon the reception of this message the *port operation complete* interrupt is raised what in turn triggers the start of the experiment. Only for the reconfiguration experiment the start of the experiment is triggered by the first of ten reconfigurations.

During the testrun, the components RC1 and RC2 repeatedly generate messages which are stored in the port memory by the TTSoC frontend. Such a message consists of a sequence number starting at zero, the timestamp of the global time base when this message was created and a checksum which is the sum of sequence number and timestamp. The message descriptor list (MEDL) in the TISS determines the instant in which the message is read from the port memory and actually sent over the TTNoC. Whenever a message was physically transmitted via the NoC the *port operation complete* interrupt on the sender-side is activated what on the other hand initiates the automatic generation of the next message. The first of this messages is generated at the beginning of the testrun to start this chain reaction. Both component's communication channels will be referred to as reference communication – RefCom1 and RefCom2 respectively.

At the same time the FIC component generates its messages triggered by its timer interrupt. This is the probe communication (ProbeCom) where the frequency of message generation is varied from testrun to testrun during the IAT and reconfiguration experiments. According to the current testrun parameters the FIC creates the messages in either equidistant time intervals (for periodic communication) or to simulate sporadic communication at a random instant within a time window. The structure of the messages is the same as for the reference communication.

Components GW and RC3 receive the data from reference communication and probe communication. When a new message comes in the TISS automatically

adds a time stamp of reception to the message. Then the *port operation complete* interrupt is raised what causes the host processor to copy the message from the port memory to its local storage where it is appended to the queue of incoming messages.

After the duration of the testrun is over, the GW component creates packages with the recorded messages and sends them via ethernet to the host PC. Subsequently, it sends a request message to the RC3 component via the management communication channel. Thus, the RC3 component knows that the GW is ready to receive its data and it also packs its accumulated messages and transmits them through the NoC to the GW component. The communication channel used for this transmission is the data collection channel (DataCom). The GW receives the packages and relays them via ethernet to the PC.

When the last package of messages is received by the host PC, the Java application stores the recorded data in a comma-separated text file for each testrun separately. Thereafter, the next testrun is initiated until all testruns of the experiment are completed.

The resulting files with the testrun data are evaluated using a Matlab script that opens each data file, reads the used testrun parameters and checks if there was any lost data package in the reference communication. For each testrun the minimum, maximum and average latency are calculated to observe potential influences of the probe communication on the reference communication.

5.8 Communication scheduling

To simplify the scheduling of encapsulated communication channels and the generation of the resulting message descriptor list (MEDL) a Java application was developed. After the structure of the NoC with its fragment switches and micro components was specified and the desired parameters of encapsulated communication channels were entered, this software calculates the scheduling of message transport and automatically creates a routing table. The software was used for two purposes. On one hand, the result are the three files that are necessary as memory initialization files during synthesizing the hardware design with Quartus: *medl.hex*, *bcfg.hex* and *ri.hex*. On the other side, it was slightly modified and added to the Java management application to receive the schedules for the reconfiguration experiment.

The scheduler simply generates an array of instants that spans the duration of one period of the slowest period defined in the communication channel list. Figure 5.4 depicts an array with scheduled communication channels. A scheduled communication channel periodically occupies the instants within this array that are

one period away from each other – e.g., a period of 8 *instants* is scheduled two times within a period of 16 *instants*. If all communication channels in the list can be scheduled without any conflicts, then the schedule is guaranteed to be free of interference.

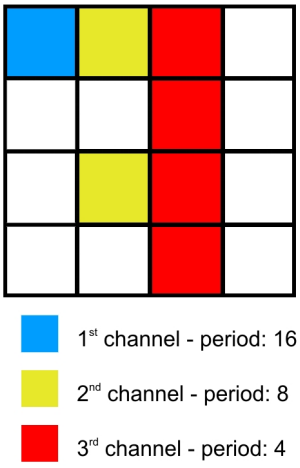

Figure 5.4: Array with scheduled communication channels. Each channel has a distinct period.

The algorithm starts to assign a 12-bit phase value to the first communication channel in the list. This channel has, preferably, the shortest period because its more difficult to schedule short periods than longer ones. The phase partitions the period into 2^{12} finer time slices and determines the instant within the period when the channel operation shall be triggered. The scheduler first assigns high instant values to prevent itself from a lock situation where no more channel can be scheduled.

If the algorithm would start with phase zero for short periods, conflicting situations may arise when different periods are used simultaneously for diverse communication channels. Such a situation is illustrated for the short period 0 and the large period 12 in figure 5.5.

In the figure it is assumed that the small period 0 – whose period bit is at 2^{-14} in the global time format – has a phase with all bits equal 0. This means, that whenever the bit at position 2^{-14} is toggled and all bits less than this position are equal 0 – due to the phase of period 0 –, then the sending of new data of this encapsulated communication channel (with period 0) is triggered. On the other hand, the least significant bit (LSB) of period 12 is higher than the most significant bit (MSB) of period 0 – in this case the LSB of period 12 is equal to the period bit of period 0. The sending of messages of period 12 is only triggered when all bits less than the LSB of the phase of period 12 are equal to 0 – otherwise none of the

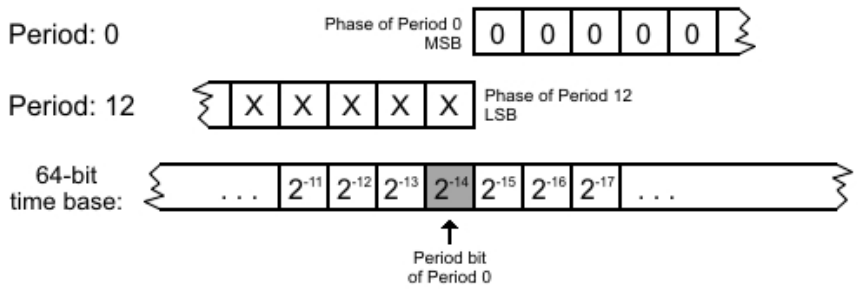

Figure 5.5: Conflicting phase values of different periods. In case of a zero phase of the short period 0 no communication channel with the large period 12 or higher can be scheduled

bits in the phase value of period 12 could toggle to the desired value. Consequently, whenever the phase of period 12 matches and thus the communication is triggered also the phase of period 0 matches and, hence, both try to send at the same time.

To not run into this conflict, the algorithm first fills the phase with 1 bits. Only if the encapsulated communication channel cannot be scheduled with this constraint, some phase bits are set to 0.

Chapter 6

Results

This chapter presents the results of the three different experimental scenarios. Each of them was further partitioned into three sub-scenarios for periodic and sporadic generation of event messages as well as for the periodic generation of state messages. The results of the experiments were stored in a file for each testrun individually. A Matlab script helps to calculate mean-, max- and min-values of latencies as well as to detect irregularities (message loss, data corruption, etc.) in the communication.

First of all, the different types of message faults are defined which are distinguished in the subsequent sections. Afterwards, in an individual section for each experimental scenario – i.e., the traffic load experiment, the bit flip experiment and the reconfiguration experiment – diagrams and tables are illustrating the actual outcomes of the experiments.

6.1 Classification of message faults

During the evaluation of the experiment data the messages received are classified according to their correctness. In figure 6.1 this classification of messages is illustrated. A messages is either *correct* or *incorrect*. The latter class is further partitioned into *invalid messages*, which are incorrect in the value domain (due to corruption), and *untimely messages*, that are incorrect in the time domain. A message can be untimely when it is *late*, *lost* (*omission*), received in the *wrong order* or *duplicated*.

Messages are categorized as follows:

Correct messages: These are data packages received with correct sequence number, timestamps and checksum. The sequence number is

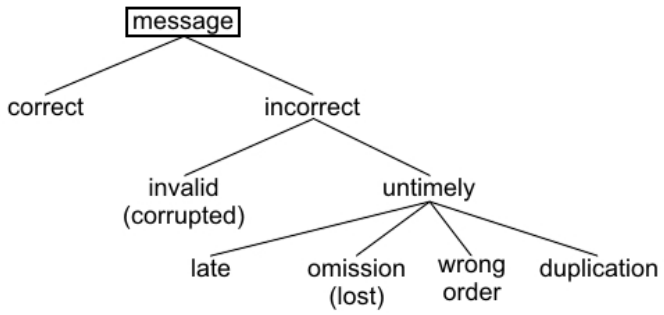

Figure 6.1: Classification of message faults.

an incremental value. The correctness of the message content – i.e., sequence number, send timestamp and checksum – is proved by adding the sequence number and the send timestamp. If this value equals the checksum, then the message content is correct in the value domain.

To evaluate the correctness of the receive timestamp an estimated time window is calculated. If the receive time of a message lies inside this window, it is classified to be correct, otherwise the message is untimely. This window of time ($\tau_{rcv_{1,2}}$) is calculated by the following equation:

$$\tau_{rcv_{1,2}} = \tau_{old} + (\Delta_{seq} * \delta_{period}) \pm \Theta_{wnd} \tag{6.1}$$

In the equation τ_{old} is the receive timestamp of the last correct message. Δ_{seq} denotes the distance between the sequence number of the last correct message and the actual one. The length of the send period is given by δ_{period}, and Θ_{wnd} is added or subtracted, respectively, to define a window of time to allow a small variance of the receive timestamp. Θ_{wnd} is defined to equal one tick of the global time base to cover imprecision of clock synchronization.

In case of sporadic event communication, due to the random time between two successive messages, the receive time is defined by the following equation:

$$\tau_{rcv_{1,2}} = \tau_{old} + ((\Delta_{seq} + x) * \delta_{period}) \pm \Theta_{wnd} \ , \ x \in \mathbb{N}_0, \ x < x_{max} \tag{6.2}$$

The variable x denotes, that the receive timestamp is valid at distinct points on the timeline, which are at a distance of one period to each other. It is in the range from 0 to x_{max}, where x_{max} give the maximum random value.

Lost messages: A lost message means, that a message was either not transmitted at all or a data package actually traversed the communication network but it was damaged in such a way that it could not be associated with any original message anymore, e.g. when sequence number and send timestamp are out of the allowed range. Due to this, the group of lost messages also comprises packages included in another group, especially, the group of corrupted messages.

Invalid messages: When the content of a message received was corrupted during transmission, it is said to be invalid. This can be proved, using the checksum. In case the sum of sequence number and send timestamp is not equal to the checksum, the message is invalid.

Message duplicates: This class contains messages that were received more than once. The very first of these messages will be classified to be correct while each following data package that can be associated with a correct message is added to this category. A message corresponds to another one if the entire message content is equal – i.e., sequence number, send timestamp and checksum. This implies, that the message was actually correct and that the message was transmitted more than once, rather than corrupted.

If no new message was generated, state messages are repeated automatically after one duration of the send period. Due to this fact, in case of state communication, this class contains only multiple sent messages when at least one other correct data package was transmitted between two multiple messages. For instance, when messages are ordered as follows (the notation is <*sequence number, timestamp, checksum*>):

$$\ldots, < 136, 1000, 1136 >, < 136, 1000, 1136 >, \ldots$$

$$\ldots, < 137, 1010, 1147 >, \ldots, < \mathbf{136,1000,1136} >, \ldots$$

Message order faults: A message order fault is, when a message that was generated before another message is actually transmitted later, e.g. the sequence

$$\ldots, < 136, 1000, 1136 >, \ldots, < 139, 1030, 1169 >, < 137, 1010, 1147 >, \ldots$$

indicates such a fault. The sequence number and the send timestamp must be in between of the two neighboring correct messages, and the checksum must be valid.

Late messages: A message is late, when it is received after the expected time, and hence, the receive timestamp is outside of the specified receive window $\tau_{rcv_{1,2}}$.

6.2 Latency and jitter

The *latency* is the duration between the generation of a message in one micro component and its actual delivery at another micro component. This includes the time, a message remains in the output buffer, and the time this message is in the NoC. Figure 6.2 presents a timeline to illustrate the latency of messages. After a message x was transmitted to its receiver, the new messages $x + 1$ is generated. The gray region around the message generation denotes a small imprecision of the generation instant, that comes from uncertainties of the operating system to switch to the interrupt service routine plus one clock cycle of the global time base for the instant, at which the timestamp is acquired. Also the send instant of message $x+1$ as well at the delivery (with the acquisition of the receive timestamp) contain imprecisions due to the imperfection of clock synchronization. The sum of all uncertainties is called *jitter*.

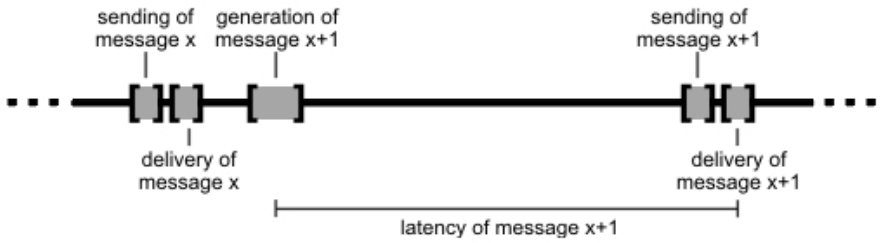

Figure 6.2: Timeline illustrating the latency of messages.

The latency is composed of a computational and a communication latency. The computational latency is the time, the message is generated and kept in the output buffer. Communication latency is the duration the message is transported on the network. As in the experiments a new message is generated immediately after the preceding message was sent, the message is in the output buffer for nearly one complete period.

Due to the concept of *shadow buffers* [Pau08] for state messages, the latency of state messages is the length of one period longer than the latency of event messages. While the communication system repeatedly sends the message from one half of the output buffer (the active buffer), the host is allowed to write a new message to the second half of the buffer (the shadow buffer). In case of a new valid message in the shadow buffer, the buffers are switched, and the shadow buffer becomes active.

As the buffers are switched at the end of the period, the latency of the message is extended by the length of one period. The jitter is not affected by the usage of shadow buffers.

6.3 Traffic load experiment

The traffic load experiment is based on the variation of the inter-arrival time (IAT) of the probe communication. Hence, the influence of the IAT on the reference communication channels is of highest interest. A table is given for each sub-scenario that states information about the recorded data packages within this test time. In particular, the number of total messages generated, the percentage of lost and corrupted messages as well as the percentage of other message faults can be found there. For the traffic load experiment the results are presented by figures showing the mean, minimum and maximum value of the latencies of the reference communication channels versus the IAT of the probe communication channel. The IAT of the probe communication was adapted during the experiment where the range of the variation corresponds to the width of the x-axis.

At the end of each sub-scenario a figure with the lost messages of the probe communication channel depicts the rise of data loss when the messages are produced at higher rate than the communication subsystem is able to transport them.

6.3.1 Periodic event communication

In table 6.1 statistics about the traffic load experiment with periodic event messages are presented. During the experiment about 8.20 Mio. messages of RefCom1 and 1.02 Mio. data packages of RefCom2 were collected. From the table it is apparent that no reference communication channel experienced either any data loss nor corruption nor any other failure of a message during the experiment, hence, no further diagram is shown with this information. The probe communication channel produced more than 20 Mio. messages in total but nearly two thirds of them have not been transmitted.

Figures 6.3 and 6.4 depict the latency of RefCom1 and RefCom2, respectively. This latency gives the duration from the moment of message generation until the instant of its reception at the GW or RC3 component. In each figure the mean-, min- and max-value of the latency is shown on the y-axis while the x-axis represents the IAT of the probe communication – which is decreased from testrun to testrun.

Both figures present a mean-value of the latency that remains constant while the FIC's IAT is decreased. In figure 6.3 the maximum difference from the found max-value to the corresponding min-value – which will be called the message transport

	RefCom1	RefCom2	ProbeCom
Messages total	~ 8.20 Mio.	~ 1.02 Mio.	~ 20.01 Mio.
Messages lost	0.0%	0.0%	63.9%
Messages corrupted	0.0%	0.0%	0.0%
Message duplicates	0.0%	0.0%	0.0%
Other message faults	0.0%	0.0%	0.0%

Table 6.1: Data integrity of the traffic load experiment with periodic event messages of the communication channels.

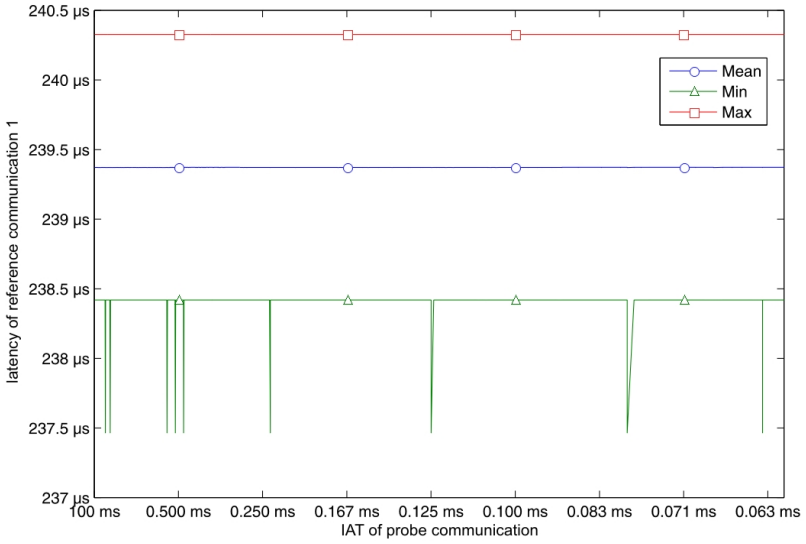

Figure 6.3: Diagram of the latency of RefCom1 in the traffic load experiment with periodic event communication.

jitter – is around $3\mu s$ which equals three cycles of the global time base. The figures show that also the jitter is constant in a range of two to three cycles of global time in all testruns.

The same is valid for the RefCom2 channel depicted in figure 6.4. Again, there is a jitter of at maximum three global time base cycles.

As the IAT of the probe communication is decreased during the experiment it reaches a value where more messages are generated than can be transported. This behavior can be seen in figure 6.5. Beginning at about 0.250 ms some messages cannot be transmitted as the send buffer is full. The percentage of lost messages increases until the end of the experiment where it reaches a maximum value of 82% of lost messages.

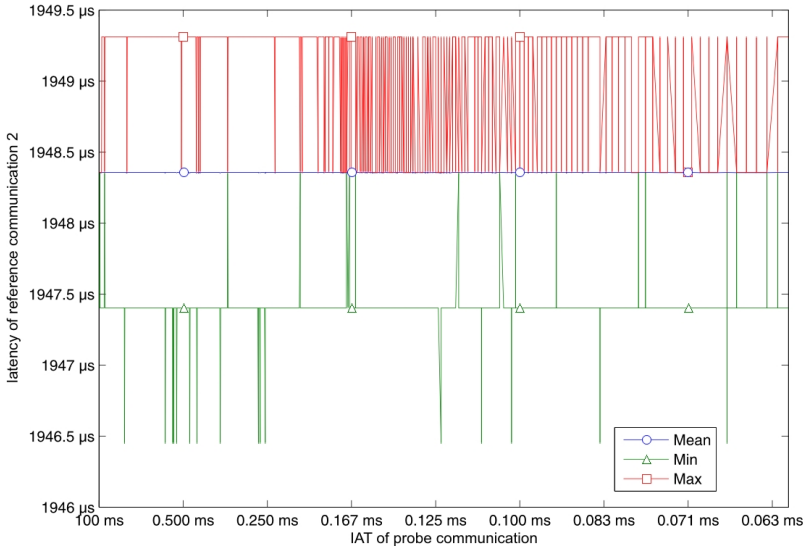

Figure 6.4: Diagram of the latency of RefCom2 in the traffic load experiment with periodic event communication.

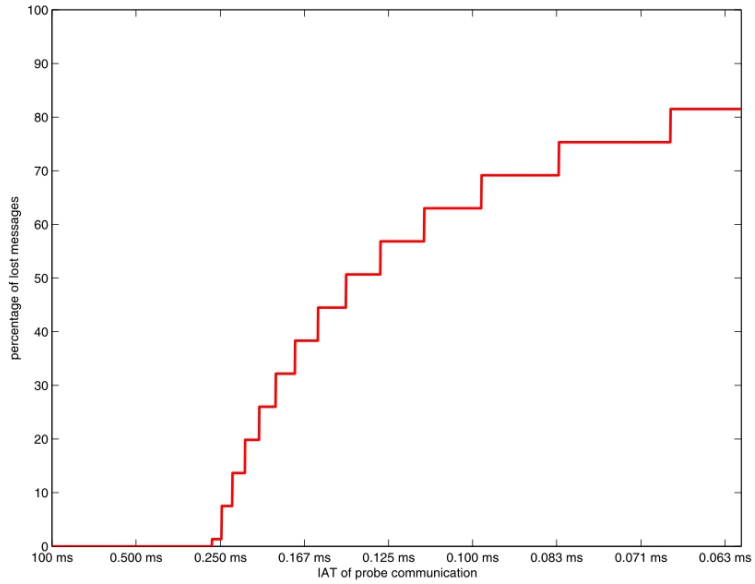

Figure 6.5: Diagram of lost messages of the probe communication in the traffic load experiment with periodic event communication.

6.3.2 Sporadic event communication

Similar results were observed for the event communication with sporadic messages. Table 6.2 shows that again no reference communication message was lost nor corrupted. For both reference communication channels the total number of messages transported is the same as it was for the periodic event communication. Due to the message generation scheme for sporadic messages, the number of probe communication messages generated is less than a half of the previous sub-scenario. More than a quarter of these messages have never been transported by the network. This is because more messages were generated than the communication subsystem was capable to transport with the selected configuration of communication periods.

	RefCom1	RefCom2	ProbeCom
Messages total	$\sim$ 8.20 Mio.	$\sim$ 1.02 Mio.	$\sim$ 8.61 Mio.
Messages lost	0.0%	0.0%	28.1%
Messages corrupted	0.0%	0.0%	0.0%
Message duplicates	0.0%	0.0%	0.0%
Other message faults	0.0%	0.0%	0.0%

Table 6.2: Data integrity of the traffic load experiment with sporadic event messages of the communication channels.

The latencies of the RefCom1 and RefCom2 channels are presented in the figures 6.6 and 6.7, respectively. Mean-, min- and max-values of the latency can be found on the y-axis while the x-axis gives the IAT of the probe communication.

As already observed for the periodic event communication, the mean-value is constant when the IAT of the probe communication is varied. Also the jitter from min- to max-value shows the same pattern like before and does not deviate from the known values.

In figure 6.8 the percentage of lost messages on the probe communication channel is shown. When the IAT of this channel is decreased, more messages are generated and – beginning at an IAT of about 0.125 ms – some of the data packages are never transmitted. Because less than a half of the messages are generated in this sub-scenario, also the IAT value, at which the first messages are lost, lies around a half of the value in the first sub-scenario.

6.3.3 Periodic state communication

The values and figures for the periodic state communication appear to be very similar to the first two sub-scenario. In table 6.3 numbers about total messages generated, lost and corrupted are presented. For the reference communication,

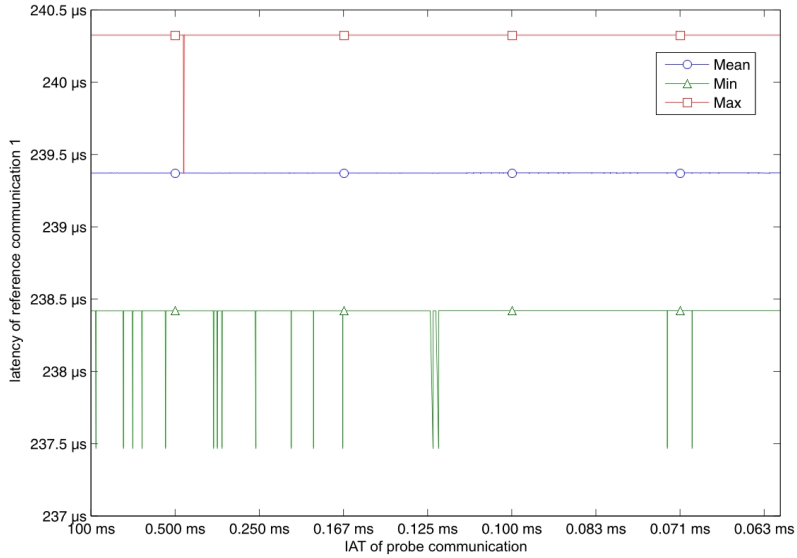

Figure 6.6: Diagram of the latency of RefCom1 in the traffic load experiment with sporadic event communication.

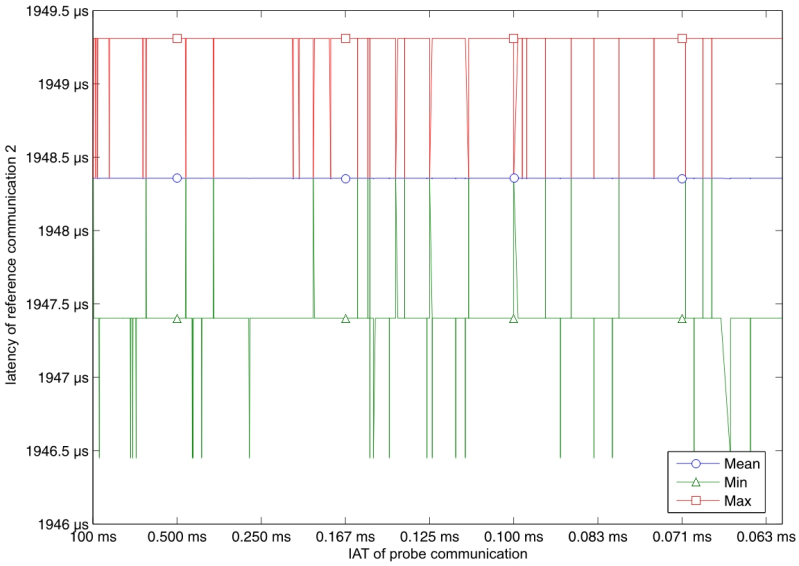

Figure 6.7: Diagram of the latency of RefCom2 in the traffic load experiment with sporadic event communication.

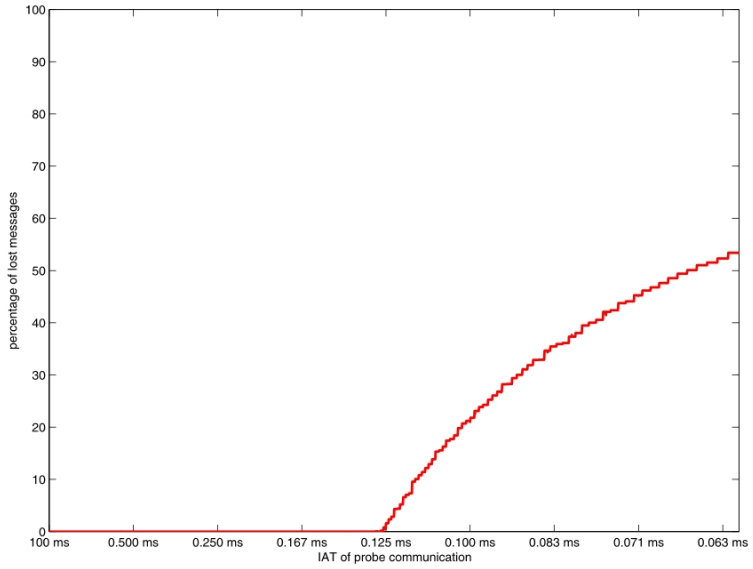

Figure 6.8: Diagram of lost messages of the probe communication in the traffic load experiment with sporadic event communication.

this data is equal to the previous experiments. The FIC component generated around 18.78 Mio. messages, where more than a half of them is lost. None of the communication channels experienced data corruption.

	RefCom1	RefCom2	ProbeCom
Messages total	$\sim$ 8.20 Mio.	$\sim$ 1.02 Mio.	$\sim$ 18.78 Mio.
Messages lost	0.0%	0.0%	56.4%
Messages corrupted	0.0%	0.0%	0.0%
Other message faults	0.0%	0.0%	0.0%

Table 6.3: Data integrity of the traffic load experiment with periodic state messages of the communication channels.

The figures 6.9 and 6.10 depict the latencies of the reference communication channels 1 and 2. Both diagrams are nearly the same as for the event communication experiments. The mean values are constant in the whole range of the IAT of the probe communication. Also the jitter is constant and does not contain irregularities.

The diagram in figure 6.11 shows the percentage of lost messages of the probe communication channel, when the IAT is decreased. Beginning at 0.250 *ms* some messages that were generated by the test application are not transmitted. The

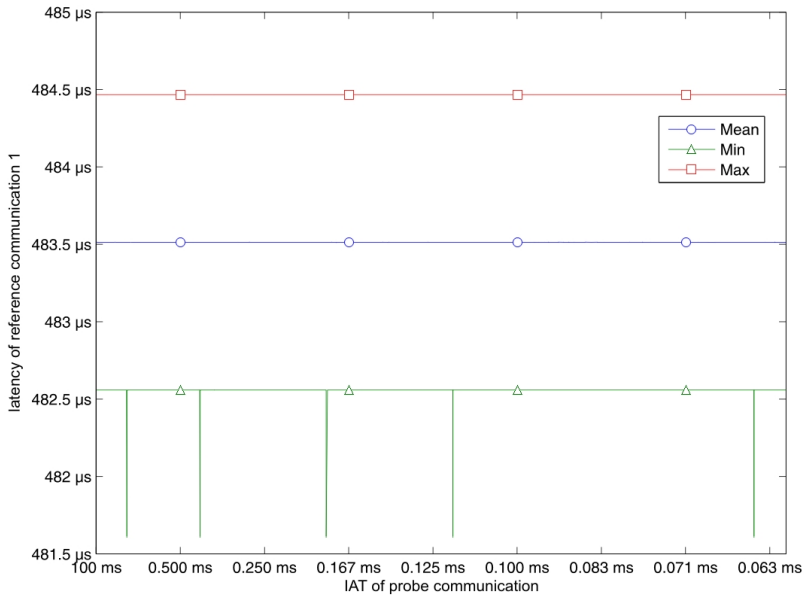

Figure 6.9: Diagram of the latency of RefCom1 in the traffic load experiment with periodic state communication.

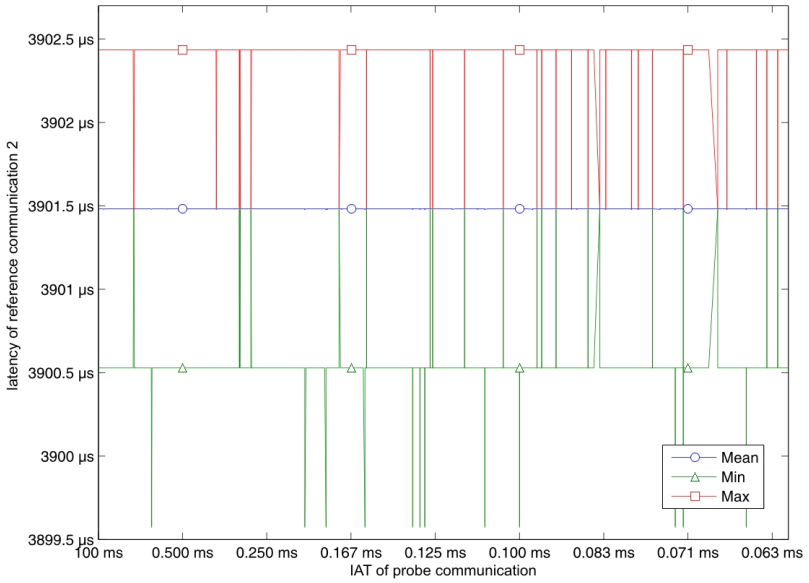

Figure 6.10: Diagram of the latency of RefCom2 in the traffic load experiment with periodic state communication.

highest value of data loss is when the IAT is set to the shortest value and hence, four times more messages are generated than can be transmitted. In case that less messages are generated – due to the state semantics of this communication channel – the last message transmitted is repeated.

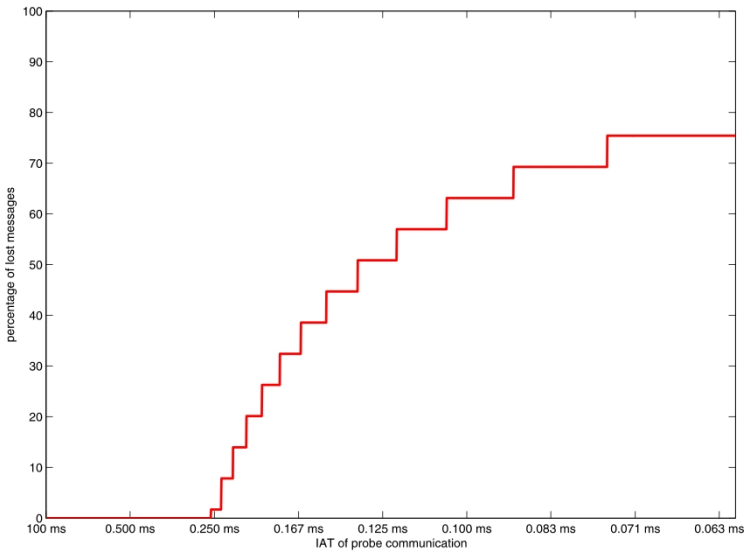

Figure 6.11: Diagram of lost messages of the probe communication in the traffic load experiment with periodic state communication.

6.4 Bit flip experiment

In the bit flip experiment the FIC component experiences the change of bit values in the network interface at random positions. During each testrun – with a duration of 1000 ms – one bit is flipped every 1 ms.

At first, tables will be given that provide information about the total number of messages sent, the percentage of lost and corrupted messages as well as the percentage of other message faults. Then diagrams are presented that show the mean, minimum and maximum values of the latency of the reference communication channels while bit flips are injected to the network interface of the FIC component. On the x-coordinate the number of these bit flips is plotted. As the bit flips appear in equidistant points in time, the abscissa can implicitly be interpreted as a timeline. Beginning from the left, the system starts in a fault free state. Incrementally more bit flips occur until the maximum of 1000 $flips$ is reached at the right side of the figure. This means that to calculate the according latency

value in the diagrams, the messages received during one testrun are indexed from zero to the maximum number of messages. All messages containing the same index build one set of data for which the latency values (mean, min and max) are calculated. In the diagrams, these values are ordered by the index, starting from index zero to the maximum index value.

Each sub-scenario closes with the presentation of the message faults in the output channel of the FIC component. The first figure indicates the decreasing percentage of correctly transmitted messages while the number of bit flips rises. Then the percentages of lost, corrupted and message duplicates is shown. The last two diagrams give the percentages of messages received in the wrong sequence and the messages received too late.

6.4.1 Periodic event communication

In the bit flip experiment with periodic event communication around 8.20 Mio. messages of RefCom1 and 1.02 Mio. messages of RefCom2 are transmitted. This can be seen in table 6.4. Obviously, there were no faults in any data package of the reference channels. Hence, no further diagrams of message faults in the reference communication are needed. In contrast, the probe communication channel of the FIC component had to deal with distinct types of faults. About 8.07 Mio. messages were generated in total. 71.0% of them were never transmitted or were corrupted in that way, that it was impossible to relate the received data package with the original message. In around 10.4% of all data transmissions the probe communication channel had to deal with faults of the remaining types (message duplicates, message order faults and late messages).

	RefCom1	RefCom2	ProbeCom
Messages total	$\sim$ 8.20 Mio.	$\sim$ 1.02 Mio.	$\sim$ 8.07 Mio.
Messages lost	0.0%	0.0%	71.0%
Messages corrupted	0.0%	0.0%	39.6%
Message duplicates	0.0%	0.0%	8.0%
Other message faults	0.0%	0.0%	2.4%

Table 6.4: Data integrity of the communication channels in the bit flip experiment with periodic event messages.

The latencies of the reference communication channels RefCom1 and RefCom2 versus the number of bit flips in the FIC's network interface are illustrated in the figures 6.12 and 6.13, respectively. The bold blue line depicts the mean value of the latency while the green one is the minimum value and the red one is the maximum value of the message transmission latency. In both figures all three latency values

remain in a constant range – between lowest minimum value to highest maximum value – which is a window of less than $3\mu s$. This corresponds to a jitter of three cycles of the global time base.

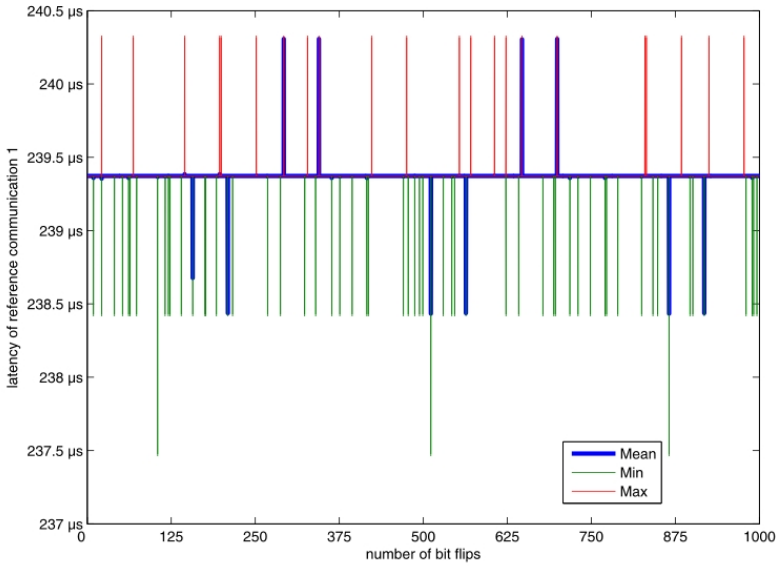

Figure 6.12: Diagram of the latency of RefCom1 in the bit flip experiment with periodic event communication.

The diagrams in figure 6.14 present the impact of bit flips in the FIC's network interface. With an increased number of bit flips (x-axis of the diagrams) the percentage of correctly transmitted messages decreases (a). Simultaneously, the probability of lost and corrupted data packages increases, (b) and (c). As the number of lost data packages is not only based on the actual reception of messages, both, the set of lost and the set of corrupted messages are not disjunct. The remarkable drop at the high end of the percentage of corrupted messages can be explained by the rapid increase of messages that where never transmitted, at the end of the testruns. Thereby, comparatively fewer messages can be classified as corrupted and the corresponding percentage drops down.

Diagrams (d) and (e) show the percentage of messages that were sent more than once and the percentage of message order faults. The last chart (f) indicates the fraction of late messages when the number of bit flips rises.

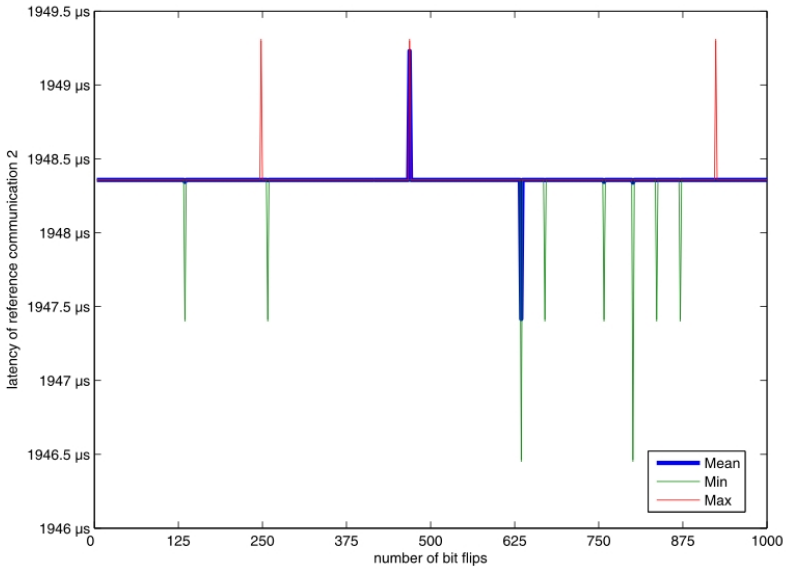

Figure 6.13: Diagram of the latency of RefCom2 in the bit flip experiment with periodic event communication.

6.4.2 Sporadic event communication

During the testruns with sporadic event communication around 8.20 Mio. data packages of RefCom1 and 1.02 Mio. messages of RefCom2 were exchanged. Also in table 6.5 this information can be found along with a summary about the found message faults. The analysis of the experiment data turned out, that none of the information exchanged by the reference communication channels was classified as faulty. The probe communication channel, on the other hand, lost 72.3% of its 4.71 Mio. messages generated. Around 65.0% of all received messages were corrupt and 17.1% were transmitted more than once. A sum of 0.3% of message order faults and late messages appeared.

	RefCom1	RefCom2	ProbeCom
Messages total	$\sim$ 8.20 Mio.	$\sim$ 1.02 Mio.	$\sim$ 4.71 Mio.
Messages lost	0.0%	0.0%	72.3%
Messages corrupted	0.0%	0.0%	65.0%
Message duplicates	0.0%	0.0%	17.1%
Other message faults	0.0%	0.0%	0.3%

Table 6.5: Data integrity of the communication channels in the bit flip experiment with sporadic event messages.

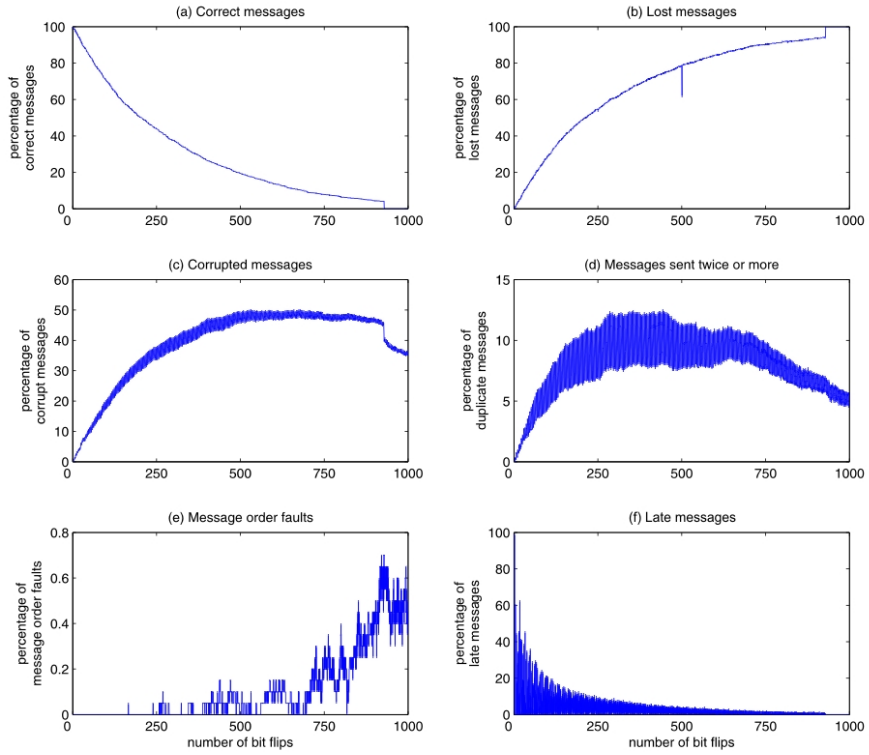

Figure 6.14: Diagrams of data integrity of ProbeCom in the bit flip experiment with periodic event communication. The x-axis represents the number of bit flips in the FICs network interface.

Figures 6.15 and 6.16 indicate the latencies of the messages exchanged by RefCom1 and RefCom2, respectively. On the x-axis the advancing number of bit flips in the FIC's network interface is displayed. The mean value of the latency is represented by the bold blue line. The minimum and maximum value of the latency are drawn by the green and the read line. Again, the range from the lowest minimum value to the highest maximum value – the jitter – is constant in the entire duration of each testrun of the experiment.

Diagrams with the different types of message faults in the probe communication channel are presented in figure 6.17. In the first chart (a) the decreasing percentage of correctly transmitted messages is depicted. The next graphs give the increasing number of lost messages (b) and the rising number of corrupted messages (c) when the number of bit flips is advanced. The fractions of message duplicates, message order faults and late messages are shown in figures (d), (e) and (f).

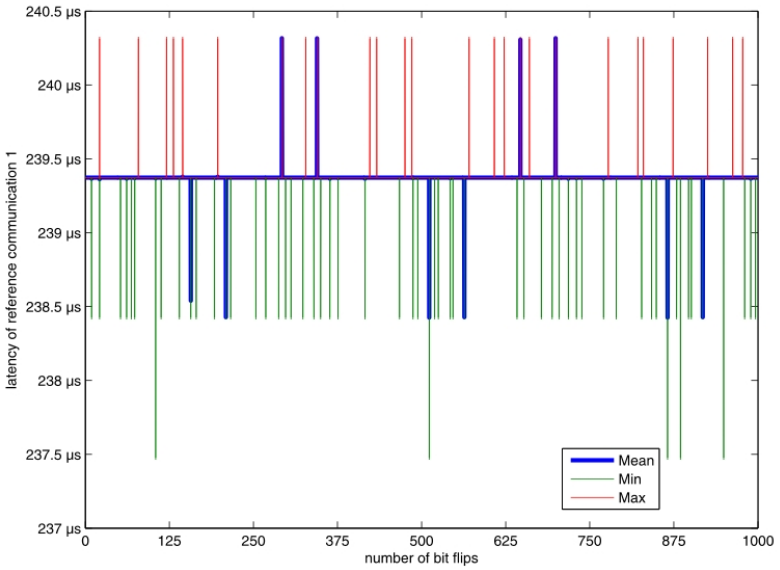

Figure 6.15: Diagram of the latency of RefCom1 in the bit flip experiment with sporadic event communication.

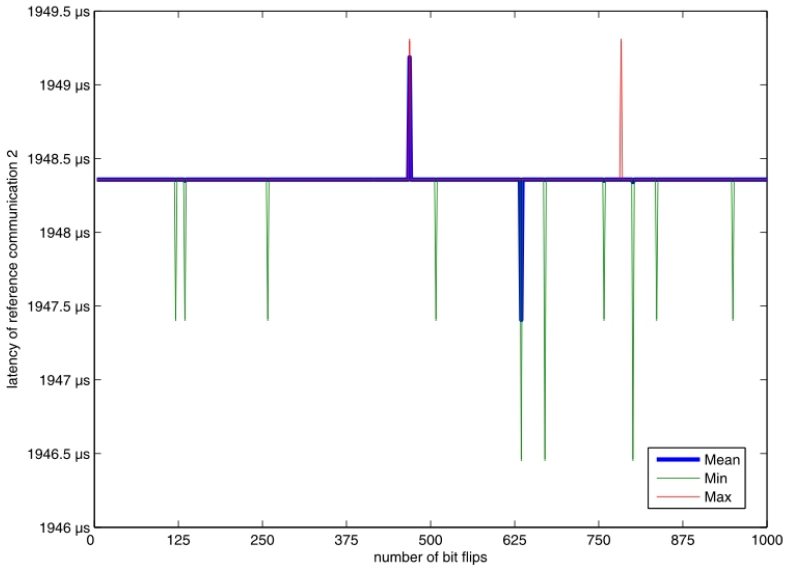

Figure 6.16: Diagram of the latency of RefCom2 in the bit flip experiment with sporadic event communication.

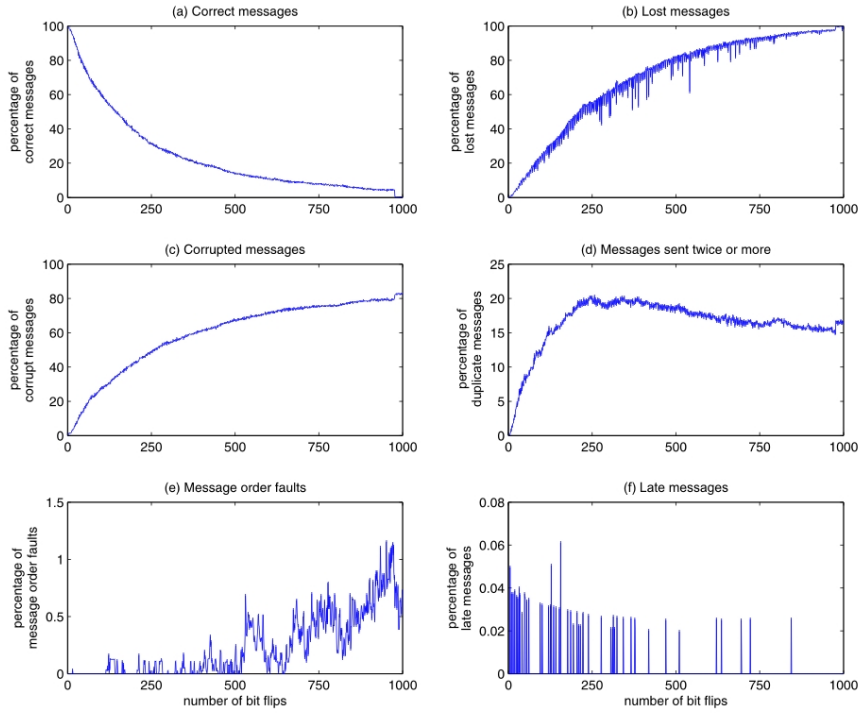

Figure 6.17: Diagrams of data integrity of ProbeCom in the bit flip experiment with sporadic event communication. The x-axis represents the number of bit flips in the FICs network interface.

6.4.3 Periodic state communication

Table 6.6 provides a summary about the messages transmitted during the experiment with periodic state communication. On the RefCom1 channel 8.20 Mio. messages traversed the communication network while on RefCom2 only 1.02 Mio. data packages were sent. As for the previous two variants of the experiment, also here no faulty message could be found on the reference communication channels. In the table no entry for message duplicates exists as due to the state semantics of the messages, the very last message is automatically repeated if no new message was put into the output buffer. Hence, messages are indeed transmitted more than once but in this variant of the experiment this is no fault.

Also for the probe communication channel 8.20 Mio. messages were generated. Approximately 49.0% of them where lost or could not be associated with the originally generated message. The corrupted part of the total messages is 23.8% and 4.0% of other faults were identified.

	RefCom1	RefCom2	ProbeCom
Messages total	~ 8.20 Mio.	~ 1.02 Mio.	~ 8.20 Mio.
Messages lost	0.0%	0.0%	49.0%
Messages corrupted	0.0%	0.0%	23.8%
Other message faults	0.0%	0.0%	4.0%

Table 6.6: Data integrity of the communication channels in the bit flip experiment with periodic state messages.

The charts 6.18 and 6.19 present the latencies of the messages exchanged on the reference communication channels 1 and 2. When the number of bit flips in the FIC's UNI is increased (x-axis), the minimum latency value (green line) and the maximum value (red line) stay an a constant range. The mean value of the latency is drawn as a bold blue line. Similarly to the previous sections, the jitter of the latencies is three clock cycles of the global time.

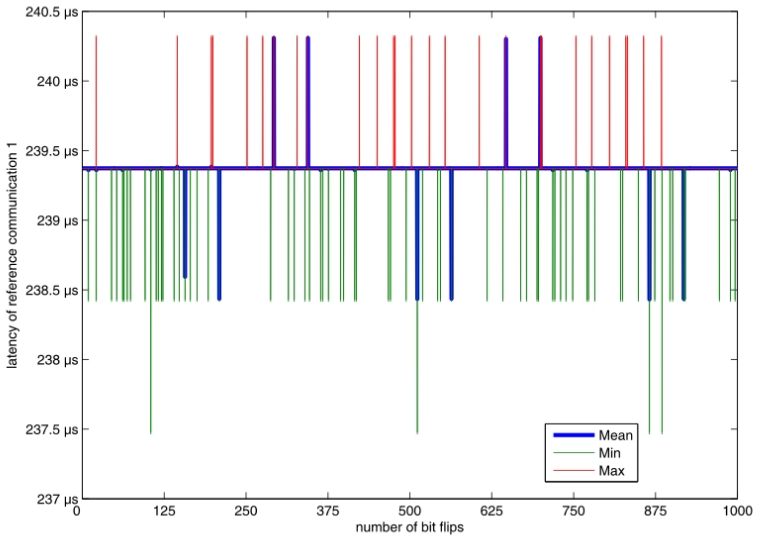

Figure 6.18: Diagram of the latency of RefCom1 in the bit flip experiment with periodic state communication.

The last figure of this section (figure 6.20) contains diagrams with the individual classes of message faults. First of all, the percentage of correctly transmitted messages is shown versus the number of bit flips in the system (a). Then the fractions of lost and corrupted messages compared to the total number of messages are presented in (b) and (c). Finally, (e) and (f) depict the percentages of message order faults and late messages. No line with the message duplicates is given as this is not relevant in case of messages with state semantics.

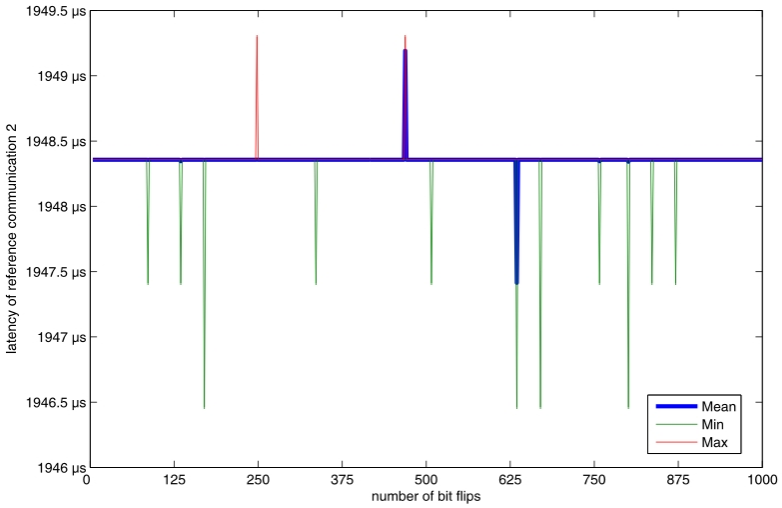

Figure 6.19: Diagram of the latency of RefCom2 in the bit flip experiment with periodic state communication.

6.5 Reconfiguration experiment

The reconfiguration experiment was accomplished with two distinct configurations concerning the probe communication channel. In the first configuration the probe communication channel is received at the components GW, RC1, RC2 and RC3. This implies, that the TISSes of these micro components have to be reconfigured when the period of the probe communication is changed. During the second configuration the probe communication channel is only received by the RC3 component. Thus, the other micro components are not affected by the reconfiguration of probe communication.

Ten different transmission periods are used for the probe communication channel in the reconfiguration experiment. For all other communication channels, the period remains constant during the entire experiment. To switch the TISSes of the distinct hosts to the new period, the TNA component initiates the reconfiguration of the schedules. As the reconfiguration channel with the actual parameters is capable of executing eight reconfigurations per second the duration of one testrun was set to 1500 ms to cover a wide range around the reconfiguration instants. Simultaneously to the reconfiguration of the periods, the IAT of the probe communication channel is increased from testrun to testrun, similarly to the traffic load experiment (see section 6.3).

Each subsection starts with a table containing an overview of the data integrity in the corresponding sub-scenario. Afterwards, diagrams that show the latency of

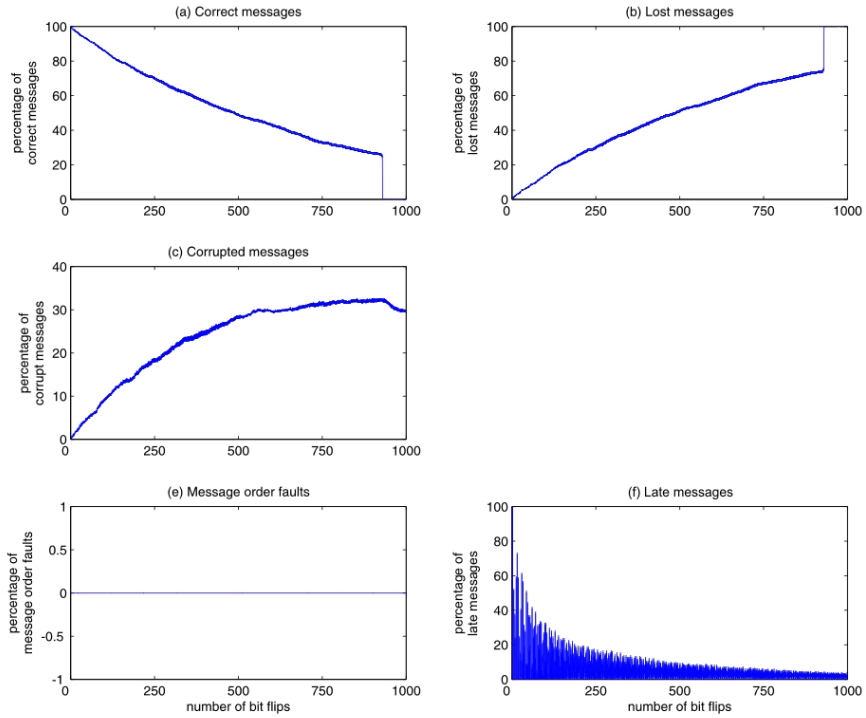

Figure 6.20: Diagrams of data integrity of ProbeCom in the bit flip experiment with periodic state communication. The x-axis represents the number of bit flips in the FICs network interface.

the reference communication channels versus the period length of probe communication using the first configuration, will be presented. To obtain the values for the charts, the corresponding data records of all testruns were related to each other, similarly to the bit flip experiment. This means, that each entry in the output of a testrun is indexed from zero to the maximum number of messages transmitted. Afterwards, the entries of all testruns that have the same index build one set of data. The latency values (mean, min, max) of this data set correspond to one data point in the diagram.

As the data of the first experimental configuration included some lost messages as well as late messages, figures illustrating the occurrence of such errors are presented. These figures show the errors from two distinct perspectives:

- *errors versus the IAT of the probe communication channel*: a data point equals the accumulated number of errors of one individual testrun.
- *errors versus the period length of probe communication*: a data point

is obtained by adding errors that appear at the same instant inside the testruns.

Afterwards, the latency of the reference communication channel using the second configuration is presented. It is calculated the same way as for the first experimental configuration. At the end of each sub-scenario, the number of lost messages in the probe communication for both configurations is illustrated in two charts for each configuration. The figures represent lost messages versus IAT of probe communication and versus the period length of probe communication, respectively.

6.5.1 Periodic event communication

Table 6.7 presents information about the total number of messages sent during both experimental configurations with periodic event communication and the data integrity of the communication channels. In the first configuration, around 11.26 Mio. messages of RefCom1 and 1.28 Mio. messages of RefCom2 were sent. Data evaluation turned out, that 0.06% of the RefCom1 channel messages have been lost. Additionally, 0.02% of the messages of RefCom1 and 0.16% of RefCom2 have been received too late. This means, that though a message was in the output buffer of the sending micro component, no message was actually transmitted at the scheduled instant.

In the second configuration, around 10.86 Mio. messages of RefCom1 and 1.23 Mio. messages of RefCom2 have been transmitted. None of the reference communication channels experienced any kind of message error, which is in contrast to the first configuration.

On the probe communication channel about 25.05 Mio. messages were generated in the first experimental configuration and 22.54 Mio. in the second configuration. In both cases more than 84% of all messages have not been received. No other types of message errors were found in the probe communication channel.

In figures 6.21 and 6.22 the latencies of messages of the reference communication channels RefCom1 and RefCom2 are presented for the first configuration. On the x-axis the period length of probe communication is given. Both diagrams contain a distinctive peak immediately after a reconfiguration instant. There the send time slot was missed and the message was transmitted one duration of the send period later.

		RefCom1	RefCom2	ProbeCom
First config.	Messages total	~ 11.26 Mio.	~ 1.28 Mio.	~ 25.05 Mio.
	Messages lost	0.06%	0.00%	85.60%
	Messages corrupted	0.00%	0.00%	0.00%
	Late messages	0.02%	0.16%	n/a
	Other message faults	0.00%	0.00%	0.00%
Second config.	Messages total	~ 10.86 Mio.	~ 1.23 Mio.	~ 22.54 Mio.
	Messages lost	0.00%	0.00%	84.65%
	Messages corrupted	0.00%	0.00%	0.00%
	Late messages	0.00%	0.00%	n/a
	Other message faults	0.00%	0.00%	0.00%

Table 6.7: Data integrity of the reconfiguration experiment with periodic event messages of the communication channels.

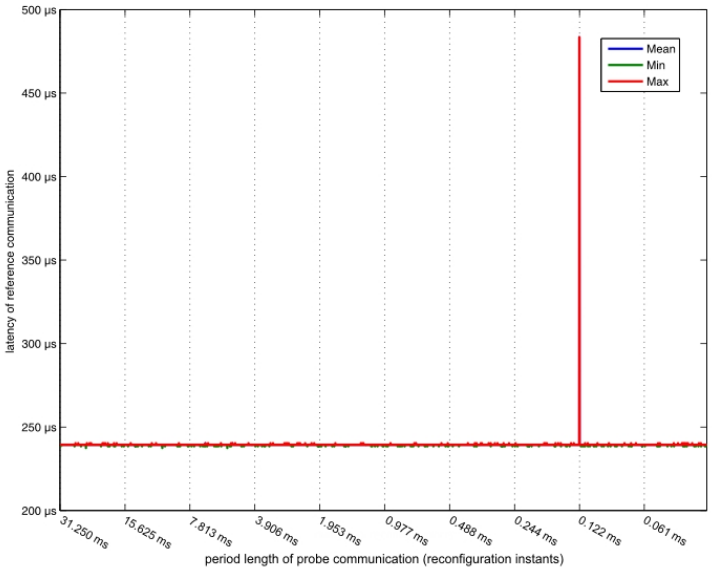

Figure 6.21: Diagram of the latency of RefCom1 in the first experimental configuration of the reconfiguration experiment with periodic event communication.

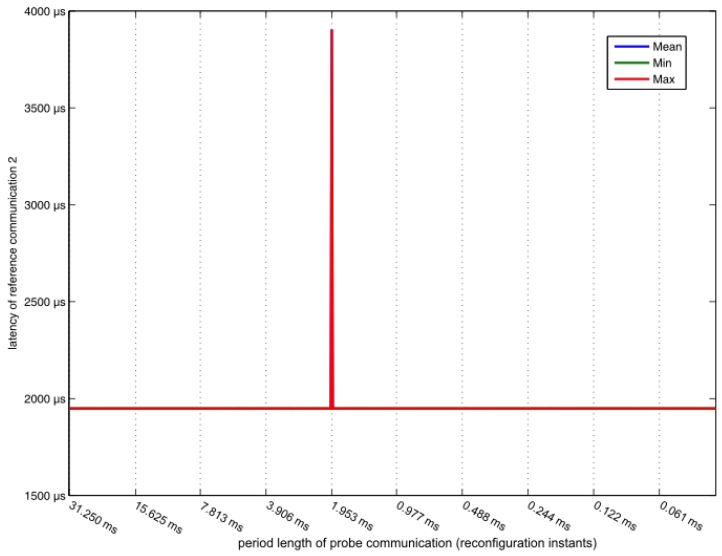

Figure 6.22: Diagram of the latency of RefCom2 in the first experimental configuration of the reconfiguration experiment with periodic event communication.

As during the first configuration messages of the reference communication channels were lost and late, charts are generated that indicate the number of errors and the moment when these occur (figure 6.23 and 6.24). For each of both channels the number of lost messages is depicted versus the IAT of the probe communication channel and versus the period length of probe communication. In RefCom1, the number of lost messages is constant until the IAT of the probe communication reaches a certain value (figure 6.23a). The second perspective – figure 6.23b – gives the same errors. It shows that these errors are located at two of the reconfiguration instants. RefCom2 does not contain lost messages, that's why the line constantly equals zero.

The diagrams containing the number of late messages (figures 6.23 and 6.24, c and d, respectively) present a comparable picture for both channels. For each IAT value of the probe communication channel, one send time slot is missed. Similarly to lost messages, late messages are located at one reconfiguration instant.

The latency of the reference communication channels in the second configuration can be found in figures 6.25 and 6.26. As one reconfiguration is accomplished after the other, no significant change of the latencies can be observed. There are no peaks as in the first experimental configuration of the sub-scenario. The range from the lowest minimum latency to the highest maximum latency is less than $3\mu s$ and equals three clock cycles of global time.

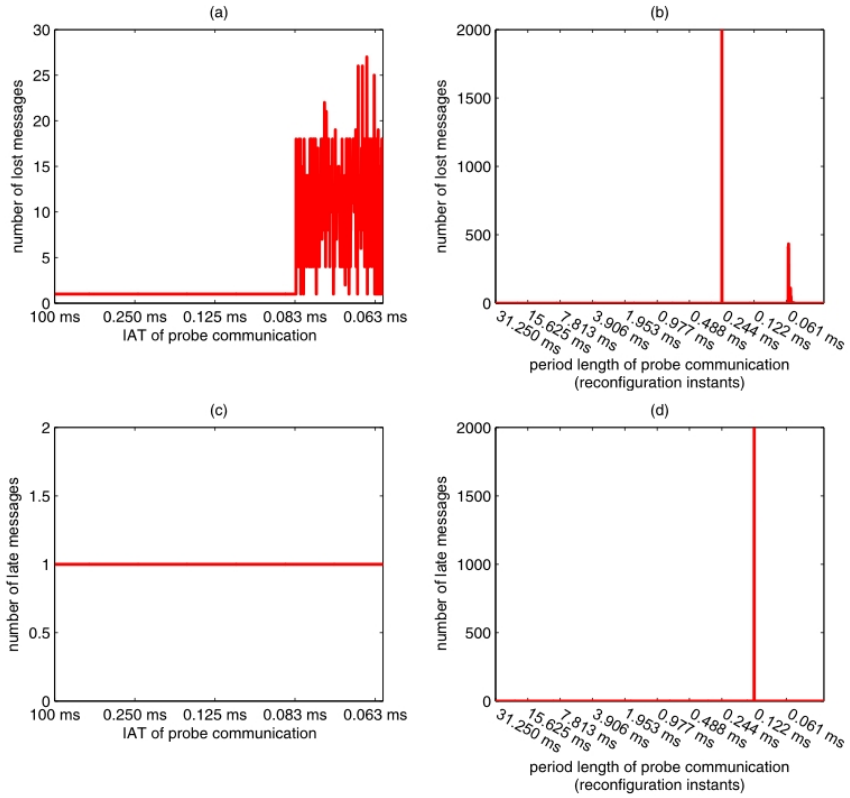

Figure 6.23: Errors in the communication of RefCom1 in the first experimental configuration of the reconfiguration experiment with periodic event communication. (a) and (b) show lost messages; (c) and (d) late messages.

At last, for each experimental configuration of the sub-scenario two diagrams with the percentage of lost messages are presented (figures 6.27 and 6.28). The first ones (a) show the rise of lost messages when the IAT of the probe communication channel is increased. In the second charts (b), no messages are lost until the third reconfiguration is accomplished and the percentage jumps to around 50% of lost messages. Then, the percentage of lost messages decreases stepwise, as more reconfigurations are accomplished, and hence, the transmission speed of the probe communication is raised.

6.5.2 Sporadic event communication

The total number of sent messages and information about data integrity of the communication channels in both experimental configuration of the sub-scenario with sporadic event communication is presented in table 6.8. Around 11.27 Mio. messages of RefCom1 and 1.28 Mio. messages of RefCom2 have been transmitted

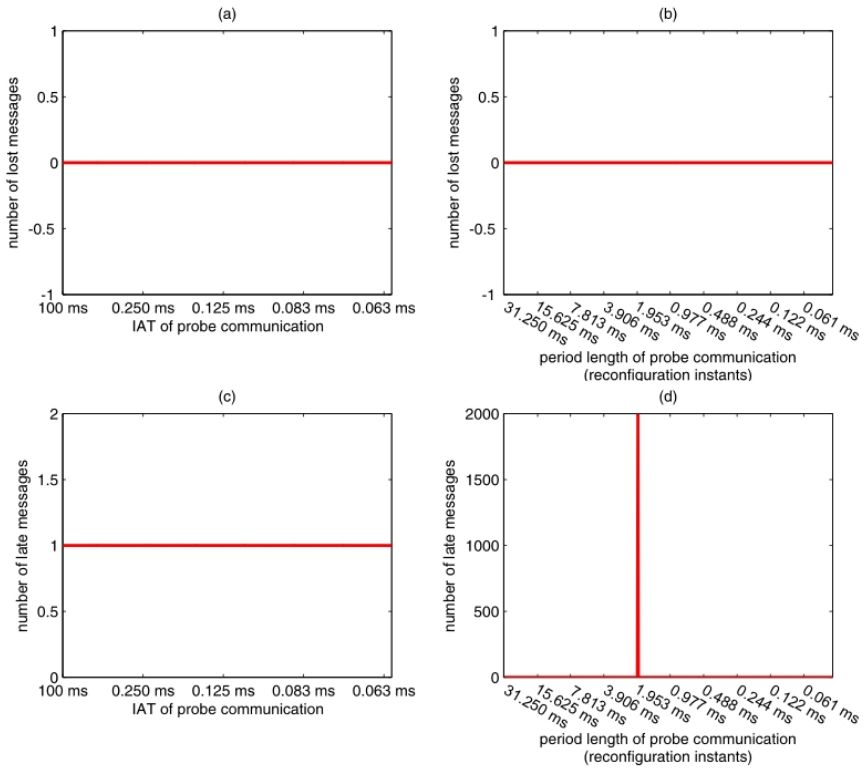

Figure 6.24: Errors in the communication of RefCom2 in the first experimental configuration of the reconfiguration experiment with periodic event communication. (a) and (b) show lost messages; (c) and (d) late messages.

using the first configuration. 0.02% of the messages of RefCom1 were lost and 0.02% of messages were received too late. On RefCom2 no messages were lost but 0.16% of messages were received too late.

With the second configuration 10.86 Mio. messages of RefCom1 and 1.23 Mio. messages of RefCom2 were produced. In this configuration none of the reference communication channels contained any kind of erroneous message.

The probe communication channel had to transmit 10.76 Mio. messages in the first configuration and 9.73 Mio. messages in the second configuration. About 37.38% of the message using the first configuration and 71.18% of messages in the second configuration have not been received. The probe communication channel did not contain other types of errors.

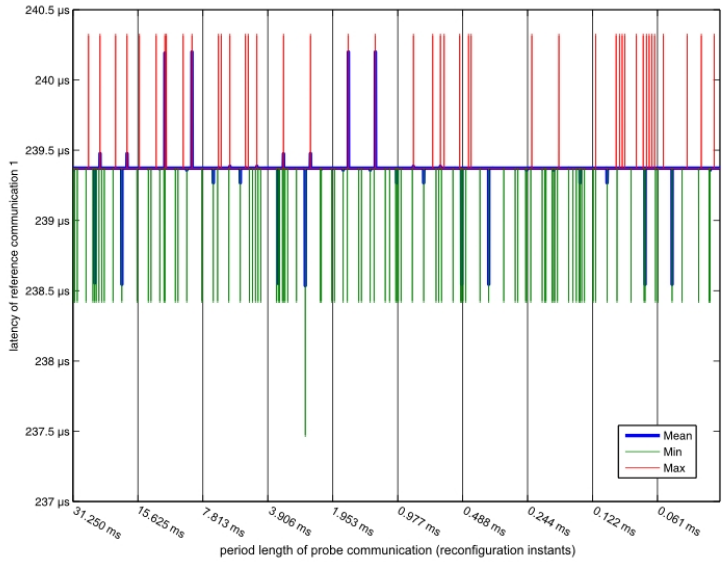

Figure 6.25: Diagram of the latency of RefCom1 in the second experimental configuration of the reconfiguration experiment with periodic event communication.

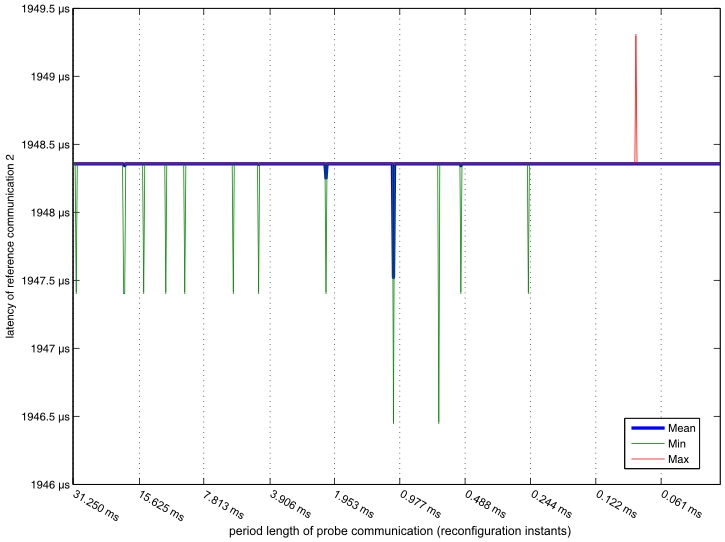

Figure 6.26: Diagram of the latency of RefCom2 in the second experimental configuration of the reconfiguration experiment with periodic event communication.

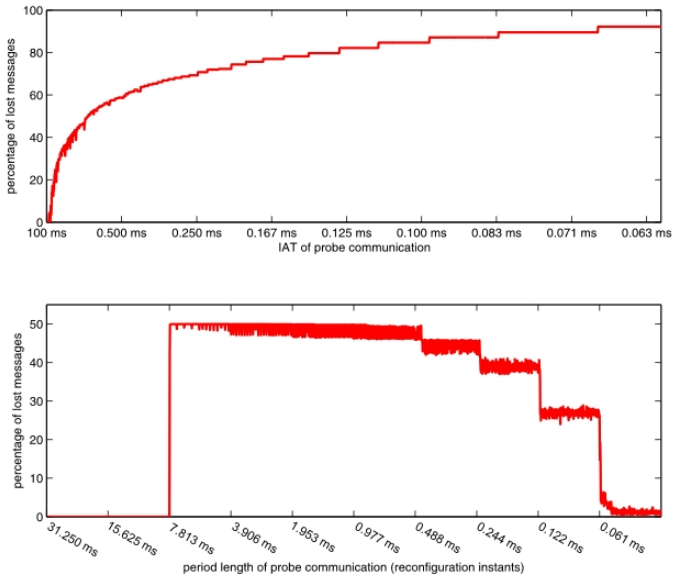

Figure 6.27: Lost messages in the probe communication channel during the first experimental configuration of the reconfiguration experiment with periodic event communication.

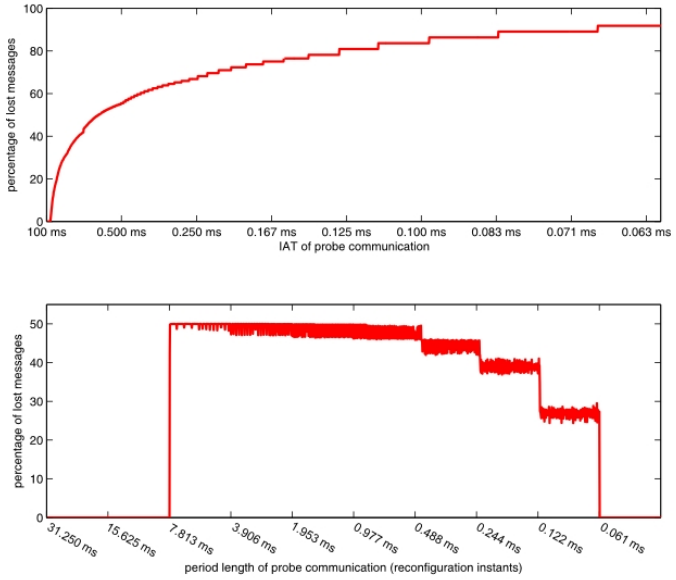

Figure 6.28: Lost messages in the probe communication channel during the second experimental configuration of the reconfiguration experiment with periodic event communication.

		RefCom1	RefCom2	ProbeCom
First config.	Messages total	~ 11.27 Mio.	~ 1.28 Mio.	~ 10.76 Mio.
	Messages lost	0.02%	0.00%	37.38%
	Messages corrupted	0.00%	0.00%	0.00%
	Late messages	0.02%	0.16%	n/a
	Other message faults	0.00%	0.00%	0.00%
Second config.	Messages total	~ 10.87 Mio.	~ 1.23 Mio.	~ 9.73 Mio.
	Messages lost	0.00%	0.00%	71.18%
	Messages corrupted	0.00%	0.00%	0.00%
	Late messages	0.00%	0.00%	n/a
	Other message faults	0.00%	0.00%	0.00%

Table 6.8: Data integrity of the reconfiguration experiment with sporadic event messages of the communication channels.

The latencies of messages of RefCom1 and RefCom2 versus the period length of probe communication in the first experimental configuration are depicted in figure 6.29 and 6.30, respectively. Similarly to the first sub-scenario, the diagrams contain a peak at the instant of a reconfiguration. The maximum value of the latency has a peak when the send time slot was missed, and thus, the message was transported one period length later.

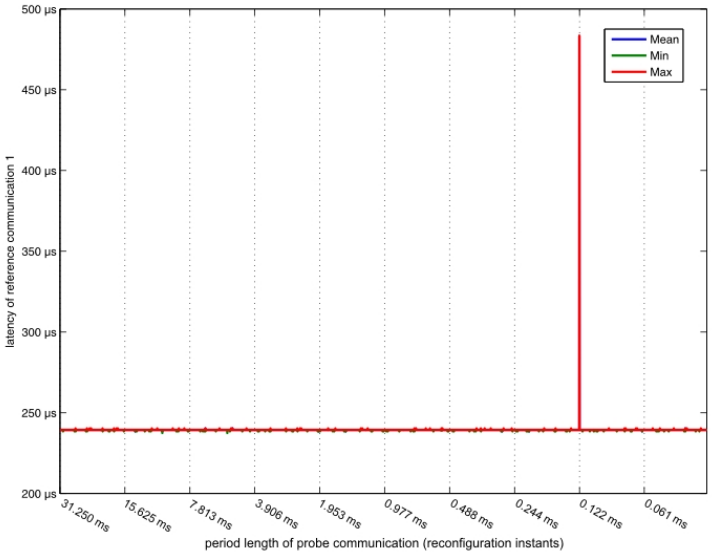

Figure 6.29: Diagram of the latency of RefCom1 in the first experimental configuration of the reconfiguration experiment with sporadic event communication.

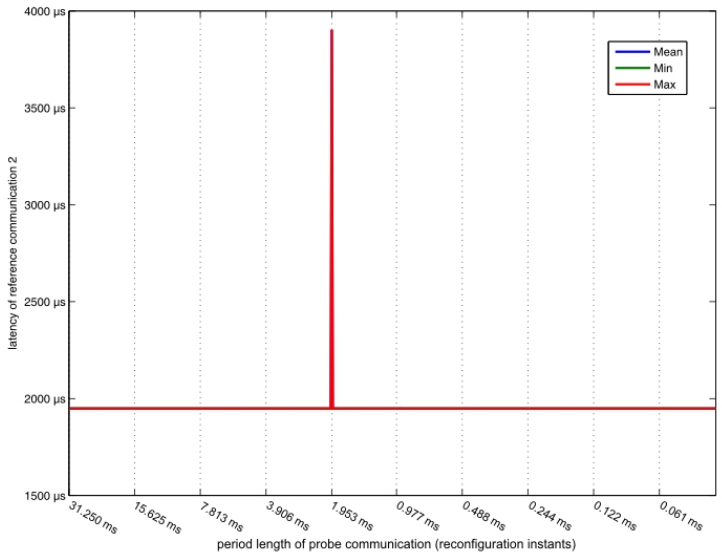

Figure 6.30: Diagram of the latency of RefCom2 in the first experimental configuration of the reconfiguration experiment with sporadic event communication.

The number of lost messages as well as the number of late messages in the first configuration are depicted in the figures 6.31 and 6.32. Each type of error is shown versus the IAT of the probe communication channel and versus the period length of probe communication. For RefCom1, the number of lost messages versus the IAT of the probe communication is constant one. In RefCom2 no lost message was found. Hence, the line is constantly zero.

The number of late messages versus the IAT of probe communication of both channels equals one. As can be seen in the diagrams, errors only occur at reconfiguration instants.

Figures 6.33 and 6.34 present the latency of the reference communication channels in the second experimental configuration. The minimum and maximum latency remain in a constant range, while one reconfiguration is accomplished after the other. The distance between the lowest minimum latency and the highest maximum latency equals three clock cycles of global time.

For each experimental configuration of this sub-scenario, the number of lost messages versus the IAT of probe communication (a) and versus the period length of probe communication (b) is given in figures 6.35 and 6.36. There is an increase of lost messages when more messages are produces on the probe communication channel. On the other hand, no messages are lost until the third reconfiguration, where a jump to 50% lost messages occurs. Afterwards, the number of lost messages is

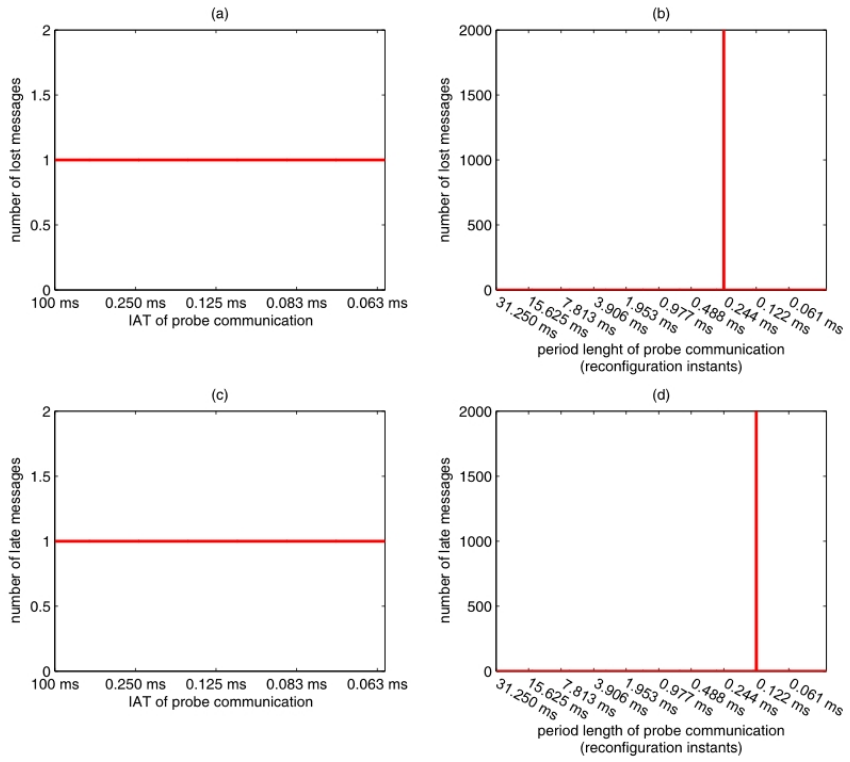

Figure 6.31: Errors in the communication of RefCom1 in the first experimental configuration of the reconfiguration experiment with sporadic event communication. (a) and (b) show lost messages; (c) and (d) late messages.

stepwise decreased when the period length is reduced, and thus, communication speed is accelerated.

6.5.3 Periodic state communication

In table 6.9 for both experimental configurations of the sub-scenario the total number of messages produced and information about the data integrity of communication channels with periodic state messages can be found. The reference communication channels RefCom1 and RefCom2 generated in the first configuration 16.34 Mio. and 2.05 Mio. messages, respectively. 21.88% of the messages of RefCom1 and 5.80% of RefCom2 have not been transmitted. 0.02% of messages of RefCom1 and 0.10% of RefCom2 were received too late.

The reference channels generated 9.71 Mio. messages for RefCom1 and 1.08 Mio. messages for RefCom2, using the second configuration. Both channels were free of any type of message errors.

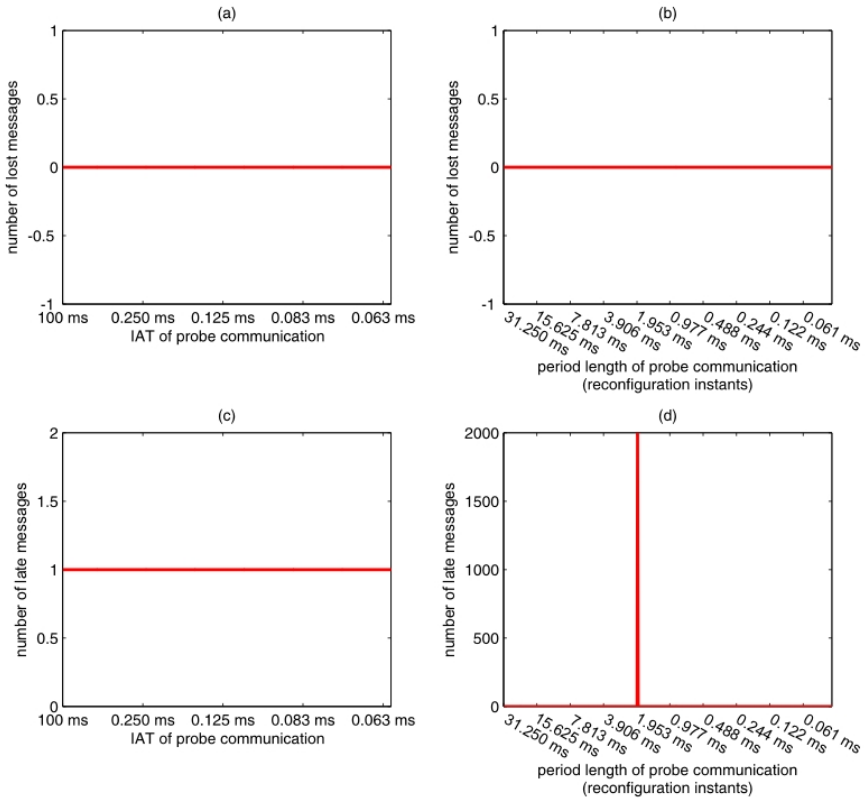

Figure 6.32: Errors in the communication of RefCom2 in the first experimental configuration of the reconfiguration experiment with sporadic event communication. (a) and (b) show lost messages; (c) and (d) late messages.

On the probe communication channel 24.25 Mio. messages in the first configuration and 21.14 Mio. messages using the second configuration were sent. Of those messages 37.38% were lost using the first configuration and 71.18% with the second configuration. No other kind of error was found in the analyzed data.

The diagrams 6.37 and 6.38 illustrate the latency of messages of the reference channels versus the period length of probe communication in the first experimental configuration of the sub-scenario. There are several peaks of the maximum latency in both diagrams. Each peak equals the double of the period length. When a send time slot is missed, such a peak occurs.

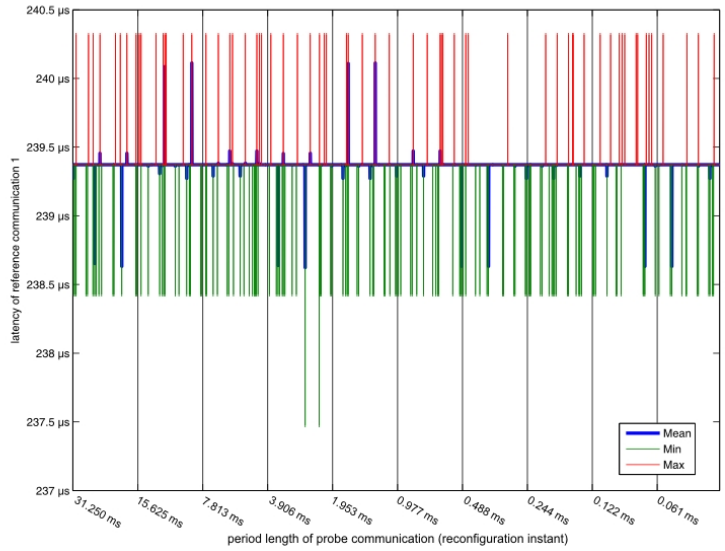

Figure 6.33: Diagram of the latency of RefCom1 in the second experimental configuration of the reconfiguration experiment with sporadic event communication.

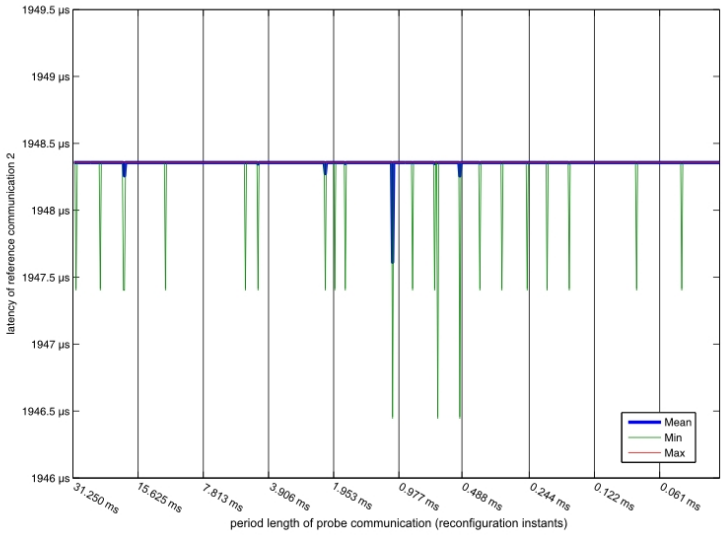

Figure 6.34: Diagram of the latency of RefCom2 in the second experimental configuration of the reconfiguration experiment with sporadic event communication.

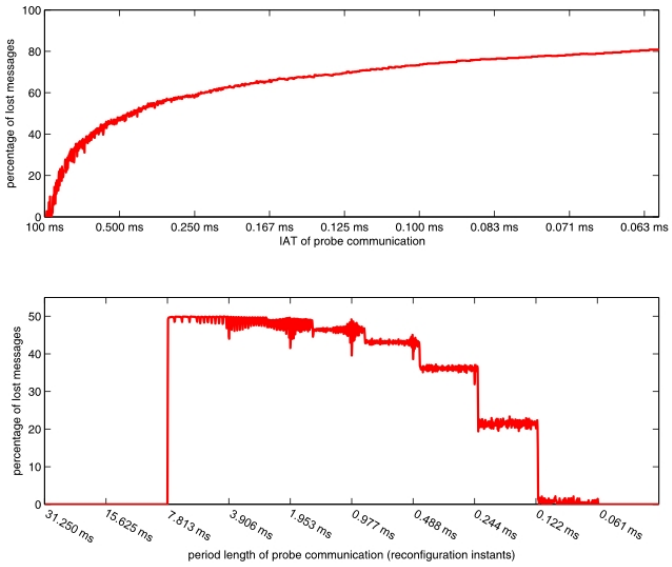

Figure 6.35: Lost messages in the probe communication channel during the first experimental configuration of the reconfiguration experiment with sporadic event communication.

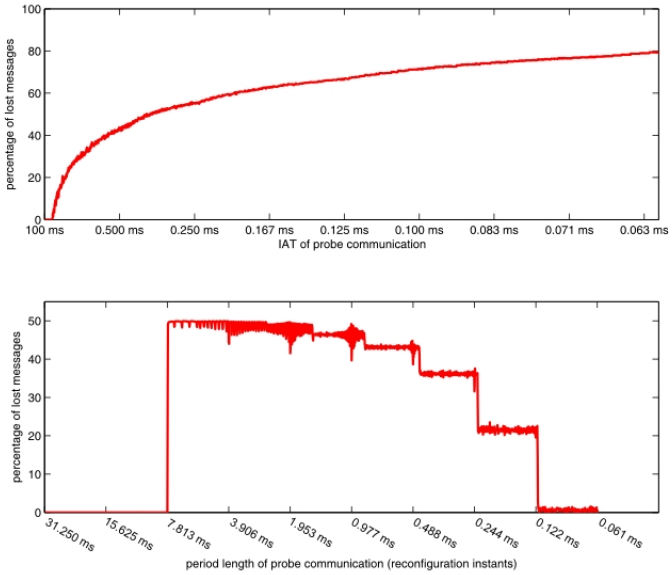

Figure 6.36: Lost messages in the probe communication channel during the second experimental configuration of the reconfiguration experiment with sporadic event communication.

		RefCom1	RefCom2	ProbeCom
First config.	Messages total	~ 16.34 Mio.	~ 2.05 Mio.	~ 24.25 Mio.
	Messages lost	21.88%	5.80%	77.66%
	Messages corrupted	0.00%	0.00%	0.00%
	Late messages	0.02%	0.10%	n/a
	Other message faults	0.00%	0.00%	0.00%
Second config.	Messages total	~ 9.71 Mio.	~ 1.08 Mio.	~ 21.14 Mio.
	Messages lost	0.00%	0.00%	76.55%
	Messages corrupted	0.00%	0.00%	0.00%
	Late messages	0.00%	0.00%	n/a
	Other message faults	0.00%	0.00%	0.00%

Table 6.9: Data integrity of the reconfiguration experiment with periodic state messages of the communication channels.

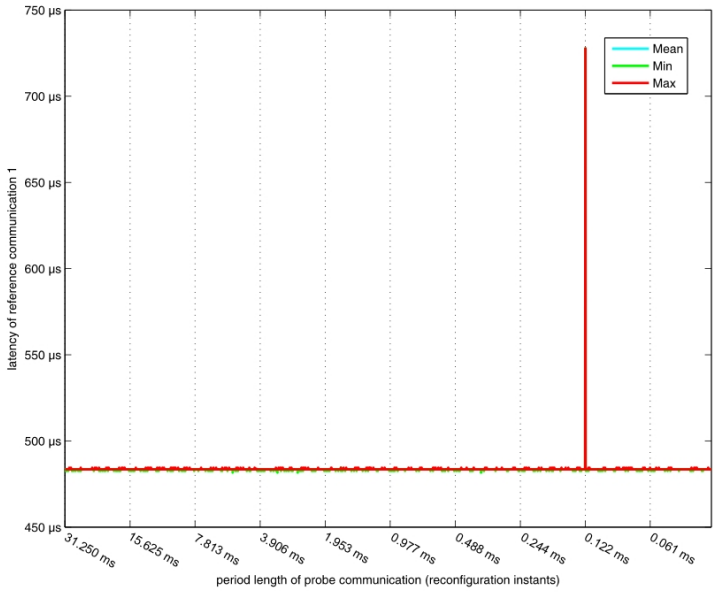

Figure 6.37: Diagram of the latency of RefCom1 in the first experimental configuration of the reconfiguration experiment with periodic state communication.

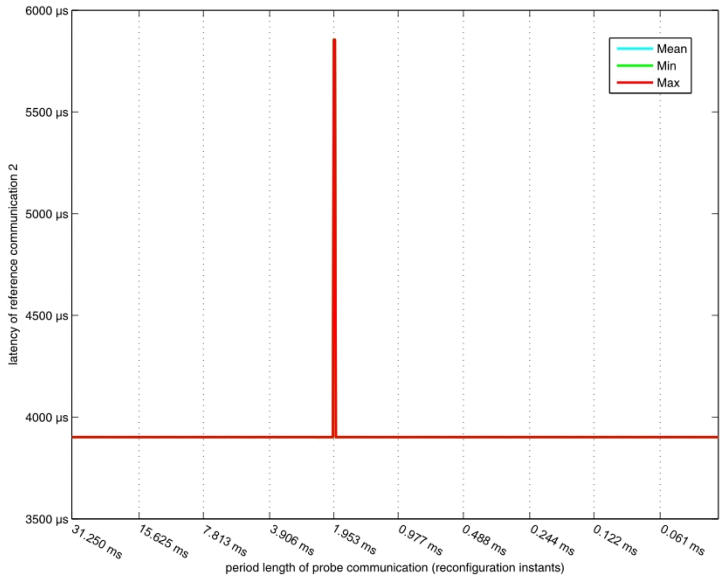

Figure 6.38: Diagram of the latency of RefCom2 in the first experimental configuration of the reconfiguration experiment with periodic state communication.

The charts of the figures 6.39 and 6.40 present for the reference communication channels in the first experimental configuration the number of lost and late messages. Both error types are depicted versus the IAT of the probe communication and versus the period length of probe communication.

The latency of the reference communication channels in the second experimental configuration is shown in figures 6.41 and 6.42. When more reconfigurations are accomplished, the minimum latency and maximum latency remain in a constant range. The distance from the lowest minimum latency to the highest maximum latency equals three clock cycles of global time.

At last, for both experimental configuration of the sub-scenario the number of lost messages of the probe communication channel versus the IAT of the probe communication (a) and versus the period length of probe communication (b) is presented in figures 6.43 and 6.44. With higher IAT of probe communication, also the number of lost messages increases. In contrast, the number of lost messages decreases, when the period length of the probe communication is decreased.

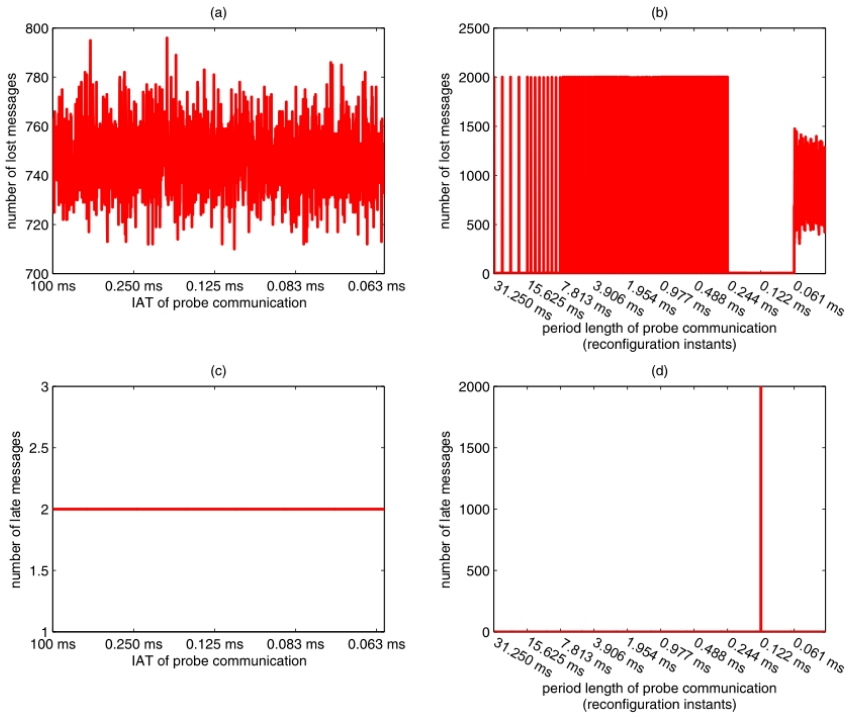

Figure 6.39: Errors in the communication of RefCom1 in the first experimental configuration of the reconfiguration experiment with periodic state communication. (a) and (b) show lost messages; (c) and (d) late messages.

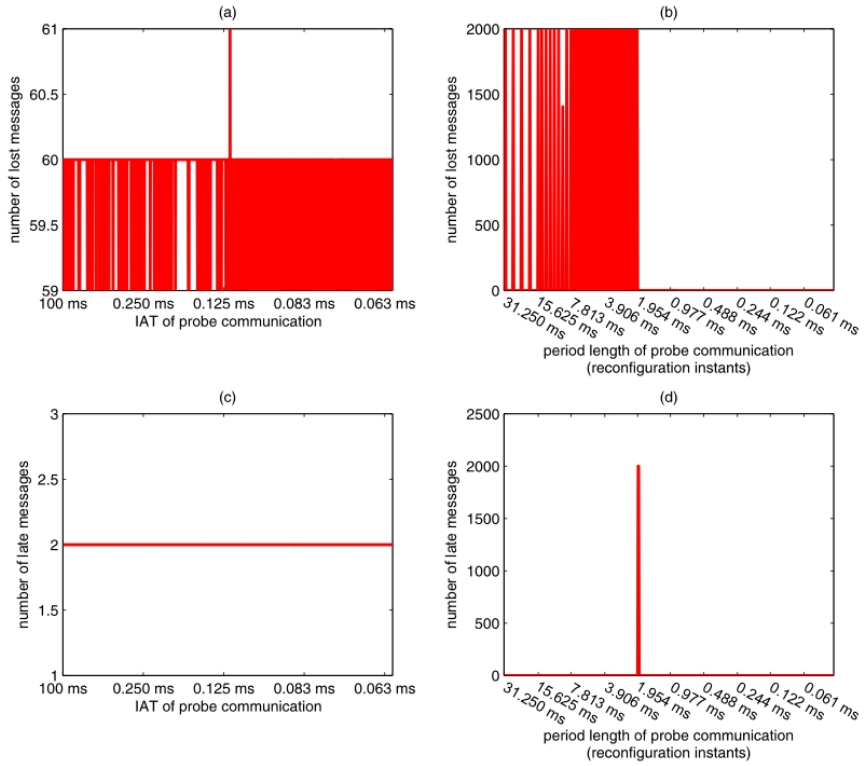

Figure 6.40: Errors in the communication of RefCom2 in the first experimental configuration of the reconfiguration experiment with periodic state communication. (a) and (b) show lost messages; (c) and (d) late messages.

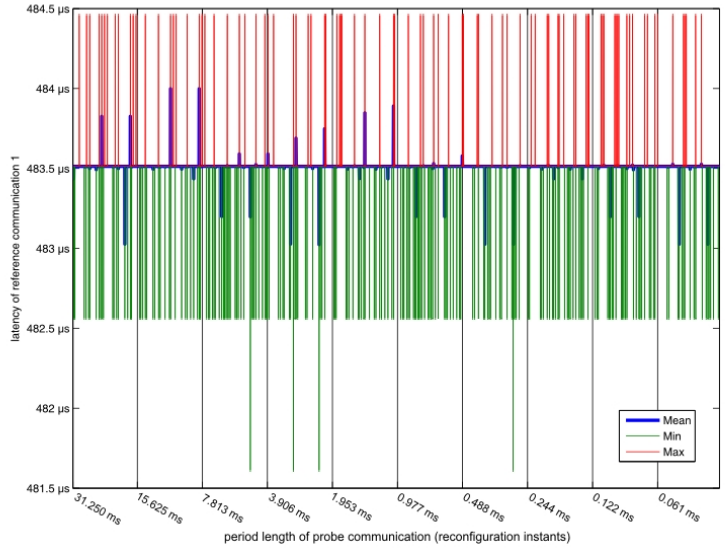

Figure 6.41: Diagram of the latency of RefCom1 in the second experimental configuration of the reconfiguration experiment with periodic state communication.

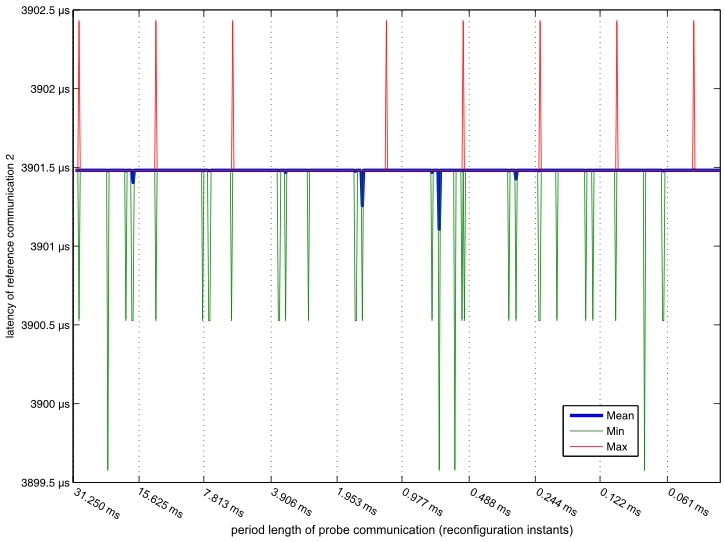

Figure 6.42: Diagram of the latency of RefCom2 in the second experimental configuration of the reconfiguration experiment with periodic state communication.

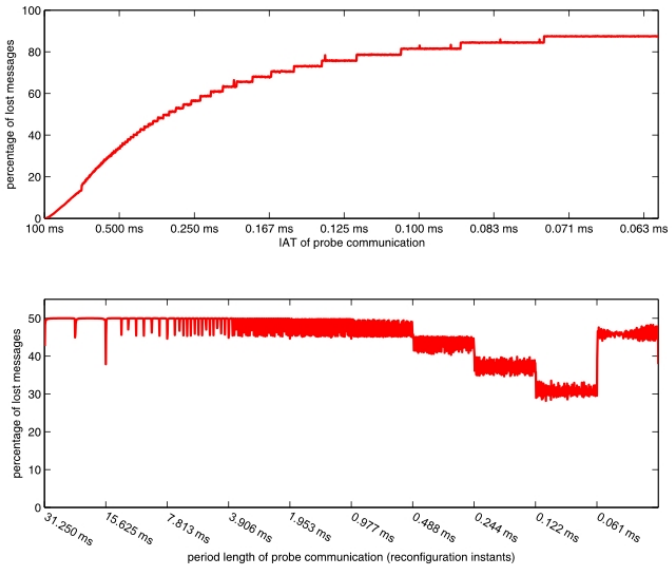

Figure 6.43: Lost messages in the probe communication channel during the first experimental configuration of the reconfiguration experiment with periodic state communication.

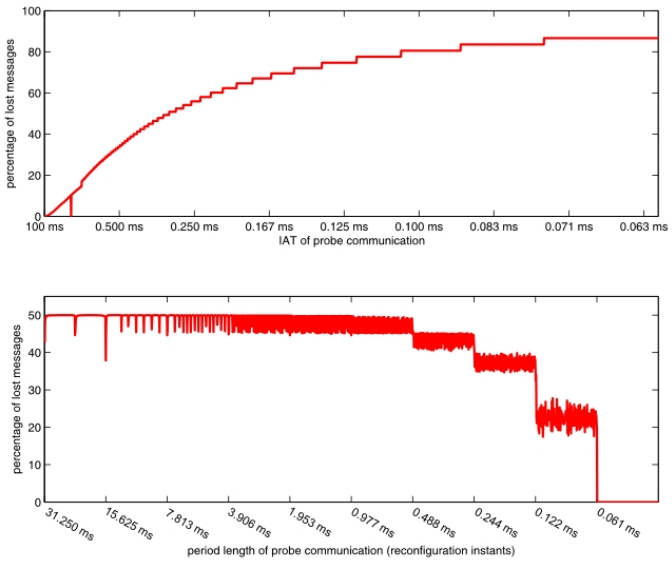

Figure 6.44: Lost messages in the probe communication channel during the second experimental configuration of the reconfiguration experiment with periodic state communication.

Chapter 7

Interpretation of Results

This chapter focuses on the interpretation of the results with respect to the hypotheses presented in section 5.1. At first, general aspects of the results are provided. Then, in section 7.2, based on the results of the experiments, a discussion about the validity of the hypotheses is presented.

7.1 General aspects

In this section, observations are presented, that cannot be associated with one individual hypothesis. Especially, the appearance of faults in the probe communication will be commented.

7.1.1 Latency and jitter

As already discussed in section 6.2, the latency of messages is defined by the duration between the generation of a message in one micro component and its delivery at another component. Due to the generation of messages for the reference communication channels immediately after the preceding message was sent, the new message waits in the output buffer nearly for the duration of the channel's period.

Except for the reconfiguration experiment using the first configuration, where the probe communication is received by RC1, RC2, RC3 and GW, the jitter of the reference communication channels is less than $3\mu s$. This equals three cycles of the global time base. The three cycles are composed of:

- one cycle for the operating system to switch to the interrupt service routine (ISR), that generates new messages

- one cycle to acquire the send timestamp at message generation
- one cycle to acquire the receive timestamp at message delivery

However, the jitter value is small compared to the average latency, which is around $239.4\mu s$ for RefCom1 and $1948.4\mu s$ for RefCom2.

7.1.2 Traffic load experiment

As expected of the faulty component, on the probe communication channel, a lot of messages have been lost during the experiments. The fraction of message omission increased with an decreasing IAT of the probe communication channel. As already mentioned in previous chapters, messages are placed in the output buffer until they are eventually sent. In case of more messages entering the buffer than are actually transmitted, the buffer will be full and all subsequent messages will be discarded until a message leaves the buffer.

Periodic and sporadic sub-scenarios differ in the fraction of lost messages, as due to the sporadic message generation scheme, statistically one half of the probe messages are generated than for periodic sub-scenarios. Thus, the output buffer first overflows with lower IAT value of the probe communication.

7.1.3 Bit flip experiment

During the bit flip experiment, the probe communication channel experienced distinct types of message faults. The appearance of these faults can be explained as follows:

Lost messages: In the bit flip experiment, messages for the probe communication channel are generated not faster than the period of the channel. Hence, the output buffer is never full. Message omission appears, when due to a bit flip in the sender's UNI, the output channel is completely deactivated though further messages are generated, or when the output buffer settings are manipulated in a way, that messages already in the buffer are not transmitted any more.

As the probability, that a bit flip manipulates one of these control registers, rises with the total number of bit flips in the system, also the percentage of message loss rises.

Corrupted messages: A bit flip in the output buffer directly alters the content of a message. Alternatively, the bit flip can corrupt the

control registers indicating the range of the output buffer. Then, parts of a memory region can be transmitted as message although these do not belong to the output buffer, and hence, the faked message is classified as invalid. The probability of such incidents rises with the number of bit flips that occurred in the system.

Message duplicates: These appear, when a bit flip manipulates the control registers of the communication channel that indicate which messages in the buffer already have been sent and which must be sent. Thus, the TISS can send a message that already has been transmitted.

Message order faults: The order of messages can be changed in a similar way as message duplicates appear. A bit flip affecting the control registers, can cause a message to be ignored by the TISS. Some of the subsequent messages are transmitted until the control registers are altered a second time. The previously skipped message is now transmitted and the order of message delivery is violated.

Due to the fact that state communication uses shadow buffers, where not more than two messages can remain in the buffer simultaneously, the reordering of messages cannot appear as the old message is replaced immediately by the next new message. This is also supported by the results depicted in figure 6.20e, where no message order faults were observed.

Late messages: The reason for a message to arrive too late is, when the channel is switched off for some time and then it is reactivated. Thus, the messages in the buffer are sent some time later.

7.1.4 Reconfiguration experiment

Three types of faults were observed during the reconfiguration experiments. The first type of faults is message loss in the probe communication, which is similar to the traffic load experiment. In the reference communication two distinct types of faults appear, when the first experimental configuration was applied. On one hand, the latency of messages is increased after the reconfiguration of the probe communication channel. On the other hand, messages of the reference channels are lost.

Message omission of probe communication

Message omission in the probe communication channel during the reconfiguration experiment originates from the overflow of the output buffer, as already discussed

for the traffic load experiment. The probability of a full buffer increases when the IAT of the probe communication decreases. Due to the usage of different communication periods for the probe communication, message omission is also observed for IATs around $100ms$, which is in contrast to the traffic load experiment. Especially at the start of each testrun, the output buffer is susceptible to an overflow, as only 32 messages per second are transmitted.

On the other hand, the percentage of lost messages decreases with faster communication periods. In the diagrams presenting lost messages in the probe communication with event messages (figures 6.27, 6.28, 6.35 and 6.36), it can be seen that within the first two periods (with $31.250ms$ and $15.625ms$) no message is lost. This is caused by the output buffers that hold more messages than with the first two periods are transmitted (4 messages with the first period and 8 with the second). As for state communication no such output buffers are used, this behaviour cannot be observed in figures 6.43 and 6.44.

Increased latency in reference communication after reconfiguration

In the diagrams of the reconfiguration experiment, peaks in the latency of the reference communication channels were observed. This is depicted in a different way in the diagrams containing the number of late messages (figures 6.23, 6.24, 6.31, 6.32, 6.39 and 6.40).

The increase of latency of reference communication messages is restricted to the first configuration of the reconfiguration experiment. In this configuration, the probe communication channel is received by RC1, RC2, RC3 and GW. Hence, all of these micro components need to be reconfigured at each reconfiguration instant. Whenever the latency has a peak in the diagrams, the amount of increase equals the length of the period of the corresponding channel ($244.1\mu s$ for RefCom1 and $1953.1\mu s$ for RefCom2). This means, that the sender delayed a message, that has already been in the output buffer, for the length of the period.

The peak in the latency value appears in those messages, that are sent immediately after the reconfiguration instant, where the probe communication adapts its period to be equal to the period of the reference communication. In other words, where reference communication and probe communication have the same period.

This behavior is explained by figure 7.1. The initialization vector points to the first time slot of the specified period, where messages are sent or received. This initialization vector is used only once after the reconfiguration of the TISS to instruct the TISS, when to start with the first communication action. All following activities are organized as a cyclic linked list, where one entry points to the next one. After the reconfiguration is completed, the initialization vector is loaded and communication activities may begin.

The figure depicts two cycles of an example period, and each cycle is partitioned into eight time slots. Within gray slots, communication activities are scheduled. Reconfiguration starts at the same instant as the first cycle begins, and it needs about the duration of three time slots to finish. Hence, when the initialization vector is loaded, the first three time slots already passed by. The initialization vector points to the second time slot, which is already over within the actual cycle. Thus, the communication system cannot execute communication activities with that period, until the second time slot of the next cycle. That's why in the first cycle no message can be transported, and thus, messages waiting in the output buffer are delayed for the length of the period.

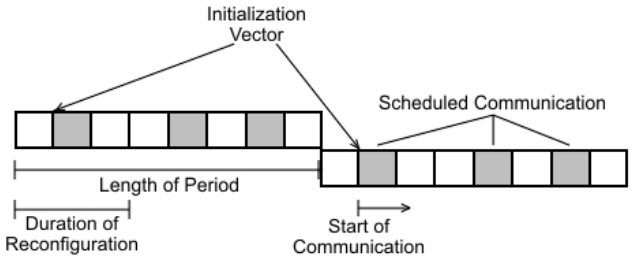

Figure 7.1: Omission of period cycle after reconfiguration.

Due to the message generation scheme, where messages are produced after the actual transmission of the preceding message, in the meanwhile no further message is generated. Consequently, also the generation of subsequent messages is delayed, and thus, the latency of only one message is affected by this behavior.

In the diagrams of state communication, each peak has a width of two messages, which means that two subsequent messages have been delayed. This originates from the shadow buffering of the state communication channels. In contrast to event ports, where only one message is kept in the output buffer, due to the message transmission as fast as the message generation, state ports hold two messages in the buffer – one that is actually transmitted, and the other one acts as shadow buffer. As both messages are already in the buffer, both of them are delayed when the above mentioned phenomenon arises.

Using the second experimental configuration, no messages are delayed. The probe communication channel is only received by the RC3 micro component, and hence, other TISSes are not concerned by the reconfiguration of the probe communication channel. Especially not those micro components, that send reference messages.

Message omission in the reference communication

Omission of messages in the reference communication channel only appears in the first experimental configuration, where the probe communication channel was received by all micro components. No lost messages in the reference communication were found during the execution of the second experimental configuration, where the probe communication channel was only received by the RC3 component.

In the first experimental configuration, message omission results from the capacity overload of the receiving micro components. The GW micro component received and stored the reference communication channel 1 (RefCom1) and the probe communication channel, while RC3 did the same with the second reference communication channel and the probe communication channel (ProbeCom). Both receiving micro components (GW and RC3) are thus coupled with the probe communication channel, as a control signal is raised each time a message arrives on the probe communication channel.

Looking at the results of the sub-scenarios with event communication, it is obvious, that on the reference communication channel 2 (RefCom2) no messages are lost. With full load, the RU3 micro component has to receive and store around 16 500 messages/second (about 500 from RefCom2 and 16 000 from ProbeCom). In figure 6.23a, which depicts the number of lost messages of RefCom1, the number of lost messages is constant until a certain value of the IAT of the probe communication. There, the total number of messages received exceeds around 17 000 messages/second, which the GW micro component is not capable to handle. In case of sporadic probe communication, less than a half of the messages on the ProbeCom channel are generated, and thus, there is no such phenomenon.

The high amount of lost messages in the reference communication during the first experimental configuration with state messages – presented in the figures 6.39b and 6.40b – also originates from capacity overload. In this case, shadow buffering is used, which implies that a messages is overwritten when the interrupt handler is not fast enough to read and store the message before. Whenever a message on the ProbeCom channel arrives, a message on the RefCom channels is lost. Thus, the number of lost messages increases with the number of messages on the ProbeCom channel. After a certain reconfiguration, i.e., when reference and probe channel transmit the same number of messages, interrupts are interleaved and no message is lost. After the subsequent reconfiguration, additional messages on the ProbeCom channel are lost, but no messages on the reference channel. Message loss in the last period of figure 6.39b results from the same phenomenon as described in the previous paragraph.

This problem does not arise with event messages, as the input buffer is able to hold up to 15 messages. Incoming messages are buffered – instead of overwritten

– and read all at once, when the interrupt handler services the request.

In the second experimental configuration, where RefCom1 and RefCom2 are received by the GW micro component and the ProbeCom channel by the RC3 component, no messages of the reference communication channels were lost. This results from the fact, that no micro component must handle more messages than it is able to. As the GW micro component is not connected to the control signals of the probe communication, the GW component is decoupled from the probe communication, and it does not react to messages sent on the probe communication channel.

7.2 Hypotheses

For the discussion of the first two hypotheses – i.e., temporal and spatial partitioning – primarily the results of the first two experiments – i.e., the traffic load experiment and the bit flip experiment – are used. In case of the hypotheses concerning the reconfiguration property of the TTSoC architecture, basically the third experiment is in the focus of the argumentation. But for the comparison of a system with and without reconfiguration, also the other two experiments will be referred to.

7.2.1 Temporal partitioning

The results of the traffic load experiment and the bit flip experiment show that the temporal properties of messages exchanged by one micro component are not affected by the behavior of other micro components. No matter how many messages were generated and sent on the probe communication channel, no faulty message was received on anyone of the reference communication channels. The latency of messages on the reference channels is nearly constant with an expected jitter (see section 7.1.1). There is no increased variability in the latency, even when the IAT of the probe communication is decreased.

Also during the bit flip experiment no faulty message on the reference communication channels was received. Both reference channel's latency was in the same range as for the traffic load experiment. The number of injected bit flips to the UNI of the fault injection component (FIC) does not have any influence on the temporal properties of the messages exchanged between the reference components.

Hence, the results of both types of experiment are an evidence that the hypothesis of temporal partitioning is valid.

7.2.2 Spatial partitioning

The data integrity of messages exchanged by one micro component is not influenced by the behavior of another micro component. In none of the three experimental scenarios a corrupted message of any reference communication channel was received. When one micro component – i.e., the FIC – decrements the IAT of its communication channel, and thus sends more messages, no messages that are exchanged between two other micro components are altered. This is verified by the traffic load experiment and the reconfiguration experiment, where the IAT as well as the channel period of the probe communication channel were decreased.

A faulty host of a micro component that arbitrarily or maliciously writes to its UNI cannot disturb or modify messages exchanged by other micro components. This has been proved by the bit flip experiment. Thus, with the three experimental scenarios also the hypothesis of spatial partitioning is validated.

7.2.3 Stability of communication during reconfiguration

Results of the reconfiguration experiment show that the reconfiguration of one communication channel does not affect temporal properties and data integrity of messages exchanged on another communication channel. Only when both channels use the same period and share a common micro component, messages can be delayed for the length of the period.

During the experiments using both experimental configurations, no message has been corrupted by the communication system, what proves the stability of spatial partitioning. On the other hand, temporal properties of communication channels are unaffected when period and micro components of both communication channels are different. This has been shown by the second configuration of the experiments. Even when the participating micro components are the same, but with a different period to be reconfigured, no violation of temporal partitioning appeared. That can be seen when looking at the results with the first experimental configuration, especially for the second reference communication channel.

7.2.4 Bounded reconfiguration delay

Reconfiguration activities during the experiments finished within a bounded interval. There are no more effects of reconfiguration, after the first cycle of a period. As explained in section 7.1.4, communication may stall for the length of one period, immediately after the reconfiguration. This occurs only, when the initialization vector of the period points to an instant smaller than the duration of the reconfiguration. Thus, the dispatcher must wait until the same instant within the next

period, to start communication. Due to the fact that the reconfiguration is triggered at all involved micro components at the same instant, all micro components consistently have to wait for the same time, until communication starts.

Therefore, the reconfiguration of communication channels is predictable and completed within a bounded interval of time.

Chapter 8

Conclusion

Within this thesis temporal and spatial partitioning in the TTSoC architecture have been evaluated. Different behaviors of micro components – i.e., load scenarios, faults of a host, and reconfiguration scenarios – have been simulated by means of fault injection. To observe the stability of communication under adverse conditions, reference communication channels between distinct micro components were established and monitored. A variation of data integrity or in the temporal behavior of these reference channels would indicate violations of temporal and spatial partitioning.

Experiments with a micro component trying to send more messages, than its communication channel is intended for, turned out, that this channel actually encapsulates temporal requirements and other channels do not notice any difference. Faulty behavior of a host has been simulated in another experiment. It showed, that output channels of the faulty component are corrupted in distinct ways, but all other communication channels remain completely unaffected. A third experimental scenario investigated the stability of communication during reconfiguration activities, as well as the bounded reconfiguration delay. Exchange of messages on channels, that were not reconfigured, was proved to be stable during the reconfiguration process. Additionally, it was found that the reconfiguration of communication channels is completed within a predictable bound.

These results present the TTSoC architecture as a predictable and robust embedded execution platform and communication infrastructure for the interconnection of multiple heterogenous IP-cores. The properties of temporal and spatial partitioning in the architecture enable the individual design, verification and integration of distinct application subsystems into one single SoC.

Acronyms and Abbreviations

ASIC ..... application specific integrated circuit Electrical circuit produced on semi-conductors that performs a specified functionality which can not be changed after its manufacturing.

BCFG ..... burst configuration memory Part of the TISS memory which holds the start and end address of each encapsulated communication channel in port memory.

FCR ..... fault containment region A self-contained component in which a failure is encapsulated and may not propagate to cause failures in other components.

FIC ..... fault injection component The micro component in the experimental setup that tries to disrupt the communication of other micro components.

FPGA ..... field programmable gate array Integrated circuit whose interconnections and logical function can be electronically configured so that a specified functionality is executed. A configured FPGA acts like an ASIC.

GW ..... gateway component Micro component of the experimental setup, which is responsible for data exchange with the host PC and for experiment management.

IAT ..... inter-arrival time Time between the arrival of two successive messages at an encapsulated communication channel.

MEDL ..... message descriptor list List that contains the instants in time when to fetch and deliver a message in the time-triggered network.

MTBF ..... mean-time-between-failures The average duration (in hours) between the occurrence of failures.

MTTF ..... mean-time-to-failure The average duration (in hours) until a failure occurs in a system.

MTTR ..... mean-time-to-repair The average duration (in hours) to repair the system – or one of its components – after the system failed.

NoC ..... network-on-chip Communication infrastructure on a single die to interconnect computational components.

RC1 ..... reference component 1 First micro component that periodically sends data packages on a reference communication channel.

RC2 ..... reference component 2 Second micro component that periodically sends data packages on a reference communication channel.

RC3 ..... reference component 3 Micro component that receives and collects data packages that were exchanged on the reference and probe communication channels.

RI ..... routing information memory Part of the TISS memory that holds information about the route of encapsulated communication channels in the NoC.

RMA ..... resource management authority Dedicated micro component of the TTSoC architecture that receives requests for resource relocation from other micro components in the system and calculates the new resource allocation.

RTL ..... Register transfer level Level of abstraction in the design of integrated circuits where the system is considered by the signal flow between registers.

SoC ..... system-on-chip A multicore computer system on a single chip.

SoS ..... slightly-off-specification Failure mode, where temporal or value specifications do not securely hold. Different communication partners may judge the results differently.

TDMA ..... time-division-multiple-access Communication paradigm where only one component/computer is allowed to send data at each point in time.

TISS ..... trusted interface subsystem Interface between micro components and the time-triggered NoC. It protects the communication subsystem from unauthorized access.

TMR ..... triple-modular redundancy Fault tolerance mechanism that uses three different computational units that all do similar calculations. A voting unit decides if he outcomes are correct or not and if there is a faulty computational unit.

TNA ..... trusted network authority Part of the certificated communication subsystem of the TTSoC architecture. It decides whether a proposed schedule is in conflict with the existing schedule and if not, it reconfigures the TISSes of the micro components.

TTNoC ..... time-triggered network-on-chip Communication infrastructure on a single die to interconnect computational components whose medium access is controlled by a predefined schedule.

TTSoC ..... time-triggered system-on-chip A multicore computer system on a single chip interconnected by a time-triggered communication network.

UNI ..... uniform network interface Interface between the TISS and its corresponding host.

List of Symbols

Symbol	Description	Page
Δ_{IAT}	rate of change of the inter-arrival time (IAT)	59
Δ_{seq}	difference between two sequence numbers	74
δ_{period}	duration of a period	74
Θ_{wnd}	window of time to allow some variance of the timestamp	74
λ	constant failure rate of a system (failures/hour)	5
μ	constant repair rate in a system (repaires/hour)	5
τ_{old}	receive timestamp of last correct message	74
$\tau_{rcv_{1,2}}$	estimated window for receive timestamp	74
A	availability of a system	5
IAT_{act}	actual IAT value between two messages	60
IAT_{avg}	average IAT reached with sporadic communication	60
IAT_{RV}	IAT value calculated by the random generator	66
$M(d)$	probability that the system can be restored within the duration d after a failure	5
$R(t)$	reliability of a system	5
RV_{act}	actual random value returned by the random generator	61
RV_{max}	maximum random value to limit the output of the random generator	66

Bibliography

[AAA+90] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.C. Laprie, E. Martins, and D. Powell. Fault Injection for Dependability Validation: A Methodology and Some Applications. In *IEEE Trans. on Software Engineering*, volume 16(2), pages 166–182, 1990. 10

[Ade03] A. Ademaj. *Assessment of Error Detection Mechanisms of the Time-Triggered Architecture Using Fault Injection.* Dissertation, Institute of Computer Engineering, Vienna University of Technology, Vienna, Austria, 2003. 4, 6, 8, 11, 13

[ALR00] A. Avizienis, J.C. Laprie, and B. Randell. Fundamental Concepts of Dependability. In *Proc. of the 3rd IEEE Information Survivability Workshop*, pages 7–12, 2000. 3

[Avi71] A. Avizienis. Fault-Tolerant Computing: An Overview. In *Computer*, volume 4, pages 5–8, 1971. 7

[Bje05] T. Bjerregaard. *The MANGO clockless network-on-chip: Concepts and implementation.* Dissertation, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2005. Available at: http://www2.imm.dtu.dk/pubdb/p.php?4025. 28

[BM02] L. Benini and G. De Micheli. Networks on Chips: A new SoC Paradigm. *Computer*, 35:70–78, 2002. 16

[BMOS05] T. Bjerregaard, S. Mahadevan, R.G. Olsen, and J. Sparsø. An OCP Compliant Network Adapter for GALS-based SoC Design Using the MANGO Network-on-Chip. In *Proc. of the Int. Symp. on System-on-Chip*, pages 171–174, 2005. 29

[BS05a] T. Bjerregaard and J. Sparsø. A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip. In *Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)*, volume 2, pages 1226–1231, 2005. 28, 29

[BS05b] T. Bjerregaard and J. Sparsø. A Scheduling Discipline for Latency and Bandwidth Guarantees in Asynchronous Network-on-Chip. In *Proc. of the 11th Int. Symp. on Asynchronous Circuits and Systems (ASYNC 2005)*, pages 34–43, March 2005. 30

[BS06] T. Bjerregaard and J. Sparsø. Implementation of guaranteed services in the MANGO clockless network-on-chip. In *IEEE Proc.-Computers and Digital Techniques*, volume 153(4), pages 217–229, July 2006. 29, 30

[CA85] F. Cristian and H. Aghili. Atomic Broadcast: From simple message diffusion to Byzantine agreement. *Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS 15)*, pages 200–206, 1985. 24

[Cri91] F. Cristian. Understanding Fault-tolerant Distributed Systems. *Communications of the ACM*, 34(2):56–78, 1991. 7

[FR92] U. Feige and P. Raghavan. Exact Analysis of Hot-Potatoe Routing. In *Proc. of the 33rd Annual Symp. on Foundations of Computer Science*, pages 553–562, Oct. 1992. 31

[FSK98] P. Folkesson, S. Svensson, and J. Karlsson. A Comparison of Simulation Based and Scan Chain Implemented Fault Injection. In *Twenty-Eighth Int. Symp. on Fault-Tolerant Computing (FTCS 28)*, pages 284–293, 1998. 11

[GDR05] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal Network on Chip: Concepts, Architectures, and Implementations. In *IEEE Design & Test of Computers*, volume 22(5), pages 414–421, Sept.-Oct. 2005. 25

[GIJ+03] M. Gaudel, V. Issarny, C. Jones, H. Kopetz, E. Marsden, M. Paulitsch N. Moffat, D. Powell, B. Randell, A. Romanowsky, R. Stroud, and F. Taiani. Final Version of DSoS Conceptual Model. Technical report cs-tr-782, University of Newcastle, April 2003. Available at http://www.cs.ncl.ac.uk/research/pubs/trs/papers/782.pdf. 16

[GS95] J. Güthoff and V. Sieh. Combining Software-Implemented and Simulation-Based Fault Injection into a Single Fault Injection Method. In *Twenty-Fifth Int. Symp. on Fault-Tolerant Computing (FTCS 25)*, pages 196–206, 1995. 10

[HCG07] A. Hansson, M. Coenen, and K. Goossens. Undisrupted Quality-of-Service during Reconfiguration of Multiple Applications in Networks on Chip. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6, April 2007. 27

[HG07] A. Hansson and K. Goossens. Trade-Offs in the Configuration of a Network on Chip for Multiple Use-Cases. In *First International Symposium on Networks-on-Chip*, pages 233–242, May 2007. 27

[HTI97] M.C. Hsueh, T.K. Tsai, and R.K. Iyer. Fault Injection Techniques and Tools. In *Computer*, volume 30(4), pages 75–82, 1997. 10, 12, 13

[KO02] H. Kopetz and R. Obermaisser. Temporal composability. *Computing & Control Engineering Journal*, 13(4):156–162, 2002. 1, 15, 20

[Koo02] P. Koopmann. What's Wrong With Fault Injection As A Benchmarking Tool? In *DSN Workshop on Dependability Benchmarking*, pages 31–36, 2002. 10

[Kop92] H. Kopetz. Sparse Time versus Dense Time in Distributed Real-Time Systems. In *Proc. of 12th International Conference on Distributed Computing Systems*, pages 460–467, 1992. 17

[Kop97] H. Kopetz. *Real-Time Systems. Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, Messachusets (USA), 1st edition edition, April 1997. ISBN 0-7923-9894-7. 4, 5, 17

[Kop03] H. Kopetz. Fault Containment and Error Detection in the Time-Triggered Architecture. In *Proc. of the Sixth Int. Symp. on Autonomous Dezentralized Systems (ISADS)*, pages 139–146, 2003. 8

[Kop06] H. Kopetz. Pulsed Data Streams. In *5th IFIP Working Conference on Distributed and Parallel Embedded Systems*, pages 105–114, Braga (Portugal), 2006. Springer Verlag. ISBN 0-387-39361-7. 21

[Kop08] H. Kopetz. The Complexity Challenge in Embedded System Design. In *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 3–12, May 2008. 1, 16

[KPJ+01] H. Kopetz, M. Paulitsch, C. Jones, M.-O. Killijian, E. Marsden, N. Moffat, D. Powell, B. Randell, A. Romanovsky, and R. Stroud. Revised Concepts of DSoS Conceptual Model. Project Deliverable for Dependable Systems of Systems (DSoS). Research Report 35/2001. *Vienna University of Technology, Real-Time System Group, Vienna, Austria*, 2001. 6

[Lap92] J.C. Laprie, editor. *Dependability: Basic Concepts and Terminology - in English, French, German, and Japanese*. Springer Verlag, Vienna (Austria), 1992. ISBN 0-387-82296-8. 3, 4, 5

[Lap95] J.C. Laprie. Dependability of computer systems: concepts, limits, improvements. *Sixth Int. Symp. on Software Reliability Engineering*, pages 2–11, 1995. 3

[LH94] J.H. Lala and R.E. Harper. Architectural Principles for Safety-Critical Real-Time Applications. In *Proceedings of the IEEE*, volume 82, pages 25–40, 1994. 6

[LN09] D. Lee and J. Na. A Novel Simulation Fault Injection Method for Dependability Analysis. In *IEEE Design & Test of Computers*, volume 26(6), pages 50–61, 2009. 11

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982. 7

[MCM+04] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost. HERMES: an infrastructure for low area overhead packet-switching networks on chip. In *Integration, the VLSI Journal*, volume 38(1), pages 69–93, Oct. 2004. 33

[MNT02] M. Millberg, E. Nilsson, and R. Thid. The Nostrum Protocol Stack and Suggested Services Provided by the Nostrum Backbone. Technical report TRITA-IMIT-LECSR02:01, LECS, IMIT, KTH, Stockholm, Sweden, Nov. 2002. Available at: http://www.imit.kth.se/info/FOFU/NOC/docs1/Reports-2002/millberg.pdf. 30

[MNT+04] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum Backbone - a Communication Protocol Stack for Networks on Chip. In *Proc. of the 17th Int. Conference on VLSI Design*, pages 693–696, 2004. 30

[MNTJ04] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip. In *Proc. of the Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 890–895, Feb. 2004. 31

[MTCM05] A. Mello, L. Tedesco, N. Calazans, and F. Moraes. Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC. In *Proc. of the 18th Annual Symp. on Integrated Circuits and System Design*, pages 178–183, Sept. 2005. 33, 34

[MVK+99] J. Muttersbach, T. Villinger, H. Kaeslin, N. Felber, and W. Fichtner. Globally-Asynchronous Locally-Synchronous Architectures to Simplify the Design of On-Chip Systems. In *Proc. of the Twelfth Annual IEEE International ASIC/SOC Conference*, pages 317–321, Sept. 1999. 28

[Obe07] R. Obermaisser. Temporal partitioning of communication resources in an integrated architecture. *IEEE Transactions on Dependable and Secure Computing*, October 2007. 1, 9, 22

[OKSH07] R. Obermaisser, H. Kopetz, C. El Salloum, and B. Huber. Error containment in the time-triggered system-on-a-chip architecture. *International Embedded Systems Symposium (IESS)*, June 2007. 16, 23, 24

[OP06] R. Obermaisser and P. Peti. A Fault Hypothesis for Integrated Architectures. In *Int. Workshop on Intelligent Solutions in Embedded Systems*, pages 1–18, 2006. 6, 7, 8, 9

[OSHK08] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz. The Time-Triggered System-on-a-Chip Architecture. In *IEEE International Symposium on Industrial Electronics (ISIE)*, pages 1941–1947, 2008. 1, 15, 17, 19, 20, 21, 22, 37

[Pau08] C. Paukovits. *The Time-Triggered System-on-Chip Architecture*. Dissertation, Institute of Computer Engineering, Vienna University of Technology, Vienna, Austria, 2008. 15, 17, 18, 39, 62, 76

[Pol94] S. Poledna. Replica Determinism in Distributed Real-Time Systems: A Brief Survey. *Real-Time Systems, Kluwer Academic Publishers*, 6:289–316, 1994. 16, 17, 20, 36

[Pow92] D. Powell. Failure Mode Assumptions and Assumption Coverage. In *Twenty-Second Int. Symp. on Fault-Tolerant Computing (FTCS 22)*, pages 386–395, 1992. 6

[RDG+04] A. Radulescu, J. Dielissen, K. Goossens, E. Rijpkema, and P. Wielage. An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration. In *Proc. of the Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 878–883, Feb. 2004. 26

[RDP+05] A. Radulescu, J. Dielissen, S.G. Pestana, O.P. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens. An Efficient On-Chip NI Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, volume 24(1), pages 4–17, Jan. 2005. 26, 27

[Rus99] J. Rushby. Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999. Available at http://www.md.kth.se/RTC/SC3S/papers/Rushby-Partitioning-NASA-99-cr209347.pdf. 9

[UK03] O.S. Unsal and I. Koren. System-level power-aware design techniques in real-time systems. *Proc. of the IEEE*, 91:1055–1069, 2003. 19

[Voa97] J. Voas. Software Fault Injection: Growing 'Safer' Systems. In *IEEE Aerospace Conference*, volume 2, pages 551–561, Feb. 1997. 13

[WG02] P. Wielage and K. Goossens. Networks on Silicon: Blessing or Nightmare? In *Euromicro Symposium on Digital System Design*, pages 196–200, 2002. 16

[WH09] J. Windsor and K. Hjortnaes. Time and Space Partitioning in Spacecraft Avionics. In *Third IEEE Int. Conf. on Space Mission Challanges for Information Technology*, pages 13–20, 2009. 9

Index

Curriculum Vitae

Oliver Höftberger

October 24^{th}, 1982	Born in Haag/Hausruck (Austria)
September 1989 – June 1993	Primary School in Weibern (Austria)
September 1993 – June 1997	Secondary School in Hofkirchen/Trattnach (Austria)
September 1997 – June 2002	Upper Secondary Technical and Vocational School for Computing & Organization in Leonding (Austria)
January 2003 – September 2003	Military Service in Hörsching (Austria)
October 2003 – February 2007	Bachelor Studies of Computer Science at the Vienna University of Technology
February 2007 – June 2007	Semester abroad at the University of Alicante (Spain)
since June 2007	Master Studies of Computer Science at the Vienna University of Technology