Masterarbeit

Reliability Assessment of DECOS System-on-a-Chip Components

ausgeführt am

Institut für Technische Informatik 182/1

der

Technischen Universität Wien

unter der Leitung von

o. Univ - Prof. Dr.phil. Dr.h.c. Hermann Kopetz

und

Univ. Ass. Dr. Roman Obermaisser
als verantwortlich mitwirkendem Assistenten

durch

Hubert Kraut
Matr. - Nr. 0025471
Buchengasse 17-19/10, A–1100 Wien

Wien, im April 2008 .....

Reliability Assessment of DECOS System-on-a-Chip Components

The Dependable Embedded Components and Systems (DECOS) System-on-a-Chip (SoC) component model lays the foundation for a next-generation embedded architecture that provides a predictable integrated execution environment for the component-based design of many different types of embedded applications (e.g., consumer, avionics, automotive, industrial). At the core of this architecture is a time-triggered Network-on-a-Chip (NoC) for the predictable interconnection of heterogeneous components which offers inherent fault isolation to support the seamless integration of independently developed micro components, possibly with different criticality levels. Furthermore, mechanisms for integrated resource management will support dynamically changing resource requirements (e.g., different operational modes of an application), fault-tolerance, and power-aware system behavior.

In the scope of this work, the reliability of a single DECOS SoC component, as well as clusters containing multiple DECOS SoC components are quantitatively assessed by means of dependability modeling. This work takes into account the increasing importance of transient faults by Single Event Upsets (SEUs) due to shrinking semiconductor geometries and lower power voltages and also focuses on the consequences of design faults in the context of mixed criticality application systems implemented by DECOS SoC components. Significant parameters for the reliability assessment are identified (e.g., probability for a transient failure of a micro component, error containment coverage of a TISS) and used to construct a generic dependability model, thus permitting a quantitative evaluation of design decisions and technological/application constraints.

Zuverlässigkeitsanalyse von DECOS System-on-a-Chip Komponenten

Das DECOS SoC Komponentenmodell beschreibt eine Architektur der nächsten Generation, die eine Plattform für die einfache Integration verschiedenster Typen eingebetteter Applikationen (z.B. Unterhaltungs-, Avionik-, Automobil- und Industrieelektronik) bietet.

Den Kern dieser Architektur bildet das zeitgesteuerte NoC, das eine deterministische Kommunikation zwischen den heterogenen Komponenten über einen gemeinsamen Bus untersttzt. Die dadurch inheränte Fehlerisolation ermöglicht eine nahtlose Integration unabhängig entwickelter Komponenten mit möglicherweise unterschiedlichen Zertifizierungsgraden.

Zusätzlich unterstützt das DECOS SoC Komponentenmodell die dynamische Rekonfiguration von Komponenten bei sich ändernden Applikationsanforderungen, z.B. im Hinblick auf Ressourcenverteilung, Fehlertoleranz und Energiemanagement.

In dieser Arbeit wird ein generisches Zuverlässigkeitsmodell für eine DECOS SoC Komponente präsentiert mit dem von einer einzelnen Komponente sowie von verteilten Applikationen aufbauend auf solchen Komponenten die Zuverlässigkeit quantitativ evaluiert wird. Die Parameter für das Modell werden von einer Prototypimplementierung übernommen. Durch die Variation der Modellparameter werden Designentscheidungen und Technologie/Applikations-Beschränkungen analysiert.

Besonderes Augenmerk wird auf transiente Fehler gelegt, die durch Partikeleinschläge (kosmische Strahlung und Rückstände vom Herstellungsprozess) verursacht werden. Diese haben einen gewichtigen Anteil an der Ausfallsrate elektronischer Systeme und gewinnen weiter an Bedeutung durch die fortschreitende Miniaturisierung der Halbleitertechnologien und der Verringerung der Spannungsversorgung. Ein weiterer Fokus liegt bei der Betrachtung von Designsfehlern im Kontext der integrierten Ausführung von Applikationen mit unterschiedlichen Zertifizierungsstufen und bei dem Nutzen von Designdiversität für redundanten Komponenten.

Acknowledgments

Sincerely I want to thank Roman Obermaisser for his excellent mentoring, Prof. Hermann Kopetz for the chance to write this thesis, Astrit Ademaj, Bernhard Huber and Christian El Salloum for the numerous discussions and Prof. William H. Sanders and his team at the University of Illinois at Urbana/Champaign for their hospitality and support of my work with M"obius.

Hearty thanks to my family for their backing of inestimable value, especially to my father who inspired my wish to study computer science during many stimulating discussions.

Last but not least I want to express my deep gratitude to my girlfriend Petra Heiss for her love and support in all circumstances.

I dedicate this work to my mother.

Danksagung

Ich bedanke mich herzlichst bei Roman Obermaisser für seine ausgezeichnete Betreuung, bei Prof. Hermann Kopetz dafür, dass er mir ermöglicht hat diese Arbeit zu verfassen, bei Astrit Ademaj, Bernhard Huber und Christian El Salloum für die zahlreichen, hilfreichen Diskussionen. Auerdem bei Prof. William H. Sanders und seinem Team an der Universität von Illinois in Urbana/Champaign für die Gastfreundschaft und Unterstützung bei der Arbeit mit Möbius.

Besonderer Dank gebührt meiner Familie für ihren unbezahlbaren Rückhalt. Speziell meinem Vater, der in mir, durch unsere anregenden Diskussionen, die Freude an der Informatik erst geweckt hat.

Zu guter Letzt will ich mich bei meiner Freundin Petra Heiss für ihre Unterstützung in allen Lebenslagen bedanken.

Ich widme diese Arbeit meiner Mutter.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Dependability is a keyword in system design that comprises a set of attributes. For example, in the domain of safety-critical applications, e.g., as aircraft and spacecraft control, the absolute key attribute is the safety, i.e., the absence of catastrophic consequences on the user(s) and the environment [6][page 2]. If the considered system must be fail-operational then safety relies on the reliability attribute, i.e., the ability of the application to deliver service continuously.

Now the essential question to the system designer is how to reach the required dependability attributes at a minimal expense of system complexity at a minimal time-to-market.

One weighty part of the answer can be Dependability Modeling. With Dependability Modeling the system designer can model the failure behavior of the system to gain quantitative statements about the dependability of the system before the first (usually expensive) prototype is produced. By this, the designer gets a powerful tool to accelerate the development cycle due to the reduced number of design/implementation/test-iterations. Early during the development process the developer can identify dependability bottlenecks, compare alternative architectures and has a decision support for the trade-off between costs and reliability.

This thesis presents a generic dependability model for DECOS [7] systems based on Time-Triggered System-on-a-Chip (TTSoC) [8] components and shows reliability and availability results for two system examples. The first example examines the reliability of a single SoC and three different Triple Modular Redundancy (TMR) approaches and the second example shows the adaption of the generic dependability model to a system from the automotive domain.

1.2 Objectives

The tasks of this thesis can be structured as following:

- **Modeling of a DECOS SoC component:** A DECOS SoC component encompasses multiple possibly heterogeneous Intellectual Property (IP) blocks called micro components, which are interconnected by a Time-Triggered Network-on-a-Chip (TTNoC). The dependability model takes into account that the SoC component contains architectural elements that are critical for the correct operation of the entire SoC (TTNoC, Trusted Network Authority (TNA), Trusted Interface Subsystems (TISSs)), as well as elements that are specific to the application services provided by a particular micro component (i.e., hosts of micro components).

- **Assessment of component reliability in the presence of transient faults:** The effects of transient faults on a DECOS SoC component are quantified. The analysis is based on information from major publications on transient faults (SEU/SET rates, effects of transient faults) and significant parameters of the SoC component. Concrete component failure and recovery rates for the models are derived from the SoC prototype implemented by the Real-Time Systems Research Group at the Vienna University of Technology and from typical values used in the automotive and aeronautic domain.

- **Reliability assessment in the presence of design faults:** The effects of design reliability on the application reliability are evaluated and for TMR the influence of software diversity is shown.

- **Modeling of a cluster containing multiple DECOS SoC components:** The DECOS TTSoC architecture defines gateways for accessing chip-external networks. Thereby, the construction of a distributed system with node computers, implemented according to the time-triggered TTSoC architecture, becomes possible. In analogy to the modeling of a single SoC component, a generic dependability model for a complete cluster is constructed.

- **Assessment of reliability improvements through TMR:** The reliability improvements of on- and off-chip TMR are compared.

- **Reliability assessment of a cluster example from the automotive domain:** The example cluster contains 4 SoCs and three

Distributed Application Subsystems (DASs) with different safety-criticality levels, a Cruise-Controller, a Brake and an Entertainment DAS.

As model formalisms are used Stochastic Activity Networks (SANs) [9] (probabilistic structural based models) for the description of the SoC component failure behaviors and Rep- and Join models for the composition of SANs. As modeling and evaluation tool Möbius [10] was chosen.

1.3 Document Structure

The rest of this paper is organized as follows. Section 1.4 (*Related Work*) sketches out the different ways of reliability assessment and provides an overview about typical model formalisms and popular dependability modeling tools. Chapter 2 (*Basic Terms and Concepts*) describes the basics of dependability and some mathematic fundamentals necessary to understand the dependability model. Next, chapter 3 (*TTSoC Architecture*) gives an overview of the DECOS SoC component model for which chapter 4 (*Fault Model*) elaborates on the fault hypothesis, the error containment capabilities of the TTSoC architecture and the estimation of failure and recovery rates for the model. Chapter 5 (*Möbius*) describes the dependability modeling tool Möbius and the model formalisms used for this paper. Upon this foundations chapter 6 (*SoC Model with Möbius*) describes the model framework for the DECOS SoC and presents two concrete DECOS cluster model examples, one which presents the DECOS SoC deployment in the automotive domain and one which allows a comparison of three different TMR approaches. Based on this model, chapter 7 (*Results*) presents results and interpretations of the reliability assessments and chapter 8 presents the conclusion.

1.4 Related Work

1.4.1 Soft Error Rate Estimation of FPGA-based Designs

A lot of previous studies examined the high sensitivity of Static Random Access Memory (SRAM)-based devices to high energetic alpha- and neutron particles (e.g., [11]) and that their overall failure rates are increasing with process scaling improvements [12]. There are several methods to measure the sensitivity of a particular device on such single-event effects.

The simplest one is the observation of a large amount of devices over a statistically sufficient long period in which the number of counted SEUs allows a reasonably

confident estimation of the Soft Error Rate (SER). Historically the first estimations about SERs were made by the analysis of stored log-files from large computer systems [11]. On the one hand, this method allows a direct measurement with no extrapolation and no assumptions about the energetic spectrum but on the other hand, this method is only possible after the product market introduction or at least if a sufficient amount of mature prototypes are available. This testing method is generally referred to as Real-Time System SER test [13].

One step further from Real-Time System SER testing is Accelerated-Soft Error Rate (ASER) testing. The JEDEC standard for measurement and reporting of particle induced soft errors defines ASER testing as follows:

> In ASER testing, devices are exposed to a specific radiation source whose intensity and energy spectrum is defined and typically much higher than the ambient levels of radiation the device would normally encounter [13][page 6].

Therefore, ASER testing needs much less testing time and only a few prototypes to gain ASER results from which the real SER can be extrapolated. Although much faster than Real-Time System SER testing this method is still very time consuming, as well depends on a relative mature prototype and additionally needs an expensive laboratory environment. Examples for ASER testing can be found in [14] and [15].

The following presented methods aim for a further decrease of costs and testing duration for SER estimations. They use SEU rate results gained from Real-Time System SER and ASER tests for the concrete SRAM-based device which will eventually host the design and combine them with the analyses about the error propagation for the considered design.

[16] presents an analysis framework for the estimation of Single-Event Functional Interrupts (SEFIs) of Field Programmable Gate Arrays (FPGAs), based on the composition of two complementary environments. The first environment uses radiation testing to identify the most sensitive regions of the device and to estimate the rate of SEUs within the configuration memory. The second environment injects single bit flips into the configuration file of the FPGA and simulates the changed functionality of the device. From the results of these two steps, the SEFI rate is estimated over the product of the SEU rate and the probability that the device fails after the occurrence of an SEU within the configuration memory.

The method in [17] also evaluates for a concrete FPGA-based design but uses an analytical approach to estimates the error propagation probabilities of the FPGA's nodes. The structural paths from each potential error site to all reachable outputs are examined and combined with the signal line probabilities. With the error propagation probabilities and the raw error rate of an SRAM cell the overall failure

rate for the FPGA-based design is computed.

1.4.2 Dependability Modeling Tools

A good classification of dependability modeling tools is given by the overview for system designer in [18] that elaborates on the methods and benefits of dependability modeling during the system design process. A distinction between three types of models can be made: parts-count models, combinatorial models and state-space models.

Parts-count models assume that each component failure leads to an overall system failure and so the overall failure rate is the sum of all component failure rates.

One step higher in complexity rank combinatorial models, including fault trees[19], success trees[19] and reliability block diagrams[20]. With these formalisms, it is already possible to describe simple redundant structures but many aspects of fault-tolerant systems are not considered including repair, reconfiguration and fault coverage.

The third class is the state-space model. Such models map each possible combination of failed and running components to a corresponding model state. By numerical analysis or simulation, the probability of being in a dedicated state can be calculated and so different reliability or performance metrics can be evaluated easily.

The last model type offers the best flexibility and accuracy for the model but also comes with a bunch of new problems. The first one is the state-space explosion, which means that a system model described by states and transitions easily reaches an infeasible amount of states. The second problem is the ratio between relatively fast and relatively slow transitions, called stiffness which leads to infeasible high calculation times when modeling ultra-reliable systems (see 5.5). There are some different ways to address these problems and actually, there are many tools on the market using the state-space variant with different types of model formalisms, Markovian-variants as structures for the analytical solving process and solution techniques. Besides Möbius very popular are Hybrid Automated Reliability Predictor (HARP), Semi-Markov Unreliability Range Estimator (SURE)[21] and SHARPE[22].

Chapter 2

Basic Terms and Concepts

2.1 Basic Terminology of Dependability

Avizienis, Laprie and Randell gave the following definition for dependability in their paper about the fundamental concepts of dependability:

> Dependability of a computing system is the ability to deliver service that can justifiably be trusted. A systematic exposition of the concepts of dependability consists of three parts: the threats to, the attributes of, and the means by which dependability is attained. [6][page 1]

The following subsections go deeper into these topics.

2.1.1 Dependability Attributes

In [6], six main attributes of dependability are considered: availability, reliability, safety, confidentiality, integrity and maintainability. In the scope of this thesis, only the reliability and the availability are interesting.

Availability The availability of a system states the probability that a system is ready to deliver correct service at a given time. If the fault model of the system considers means of reparation and recovery it is possible to give a time-invariant measure for the availability, the so-called steady-state availability. Knowing the Mean-Time-To-Failure (MTTF) and the Mean-Time-To-Repair (MTTR) the steady-state availability can be calculated by the following formula:

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

Reliability The reliability describes the systems's ability to provide continuous service. In the result section of this thesis the reliability of a DAS is given either by its MTTF or by the reliability function R(t) which states the probability that the system delivers its service continuously at least up to the time point t. In many papers the reliability is given as a constant probability $R_m = R(\text{mission time})$ regarding a specific mission time [18], e.g., the duration of the warranty of commercial systems or the maximum flight time of an aircraft.

Usually MTTFs are measured in failures in 10^9 hourss (FITs) whereby one FIT means that the device suffers one failure in 10^9 device-hours.

2.1.2 Dependability Threats: Faults, Errors and Failures

The correctness of a system can be defined over the correctness of the expected service at its service interface. These services can be specified in various domains like the time, value and power consumption domain. Based on the definition of a correct system, [6] differs between the following definitions:

A failure names the deviation of the delivered service from its specification. An example for a failure is a micro component which omits to deliver correct data. That part of the system state, which caused the failure, is called error. An error could be a flipped bit within the memory of the micro component that caused the subsequent micro component failure. The cause of an error is called fault. As continuation to the former examples, a fault could be a cosmic particle striking a memory cell. Since the failure of a component can be seen as a fault for the encompassing system, the fault-to-failure chain works recursively: fault→ error→ failure→ fault...

Fault Classifications [23] defines many classifications for faults and their sources. In the scope of this document the following are necessary to know:

The classification after the phenomenological causes provides the following fault types:

- Physical Faults: Such faults are generated during system operation and are due to adverse physical phenomena. The fault model within this paper considers particle strikes and electromagnetic interferences.
- Design Faults: Within this paper a design fault refers to a fault within program or description code of either a micro controller or an FPGA and corresponds to the definition of a software fault as e.g. made from Laprie in [24]. Accordingly, a design fault is the result of a designer's failure when implementing the system's specification. A fault is called

dormant as long as the system does not reach a state in which the fault was activated and caused an erroneous system state.

The consideration of the temporal persistence of faults leads to the following discriminations:

- Permanent Faults: The presence of a permanent fault does not depend on point wise conditions. An example for a permanent fault is the thermal destruction of a transistor within a device.
- Transient Faults: The presence of a transient fault depends on point wise conditions. For example, Electro Magnetic Interference (EMI) and cosmic particles can be counted to physical transient faults.

Failure Classifications According to [23] a system can fail in two different domains:

- Value Domain Failures: The service delivers values that do not comply with the specification.
- Temporal Domain Failures: The timing of the service delivery deviates from its specification.

Since the fault model in this thesis also considers SoC power mode failures, the list above can be enhanced by:

- Power Domain Failures: The power consumption of the service-delivering system does not comply with the specification.

Additionally to the failure classification above, it is useful to distinguish refined modes of failures. A general failure is called an arbitrary failure. If the system persistently stops to deliver its service this is called a crash failure.

2.1.3 FCRs and ECRs

A Fault Containment Region (FCR) as defined in [25] is used as the smallest unit of consideration for the fault model of this thesis. The fault model partitions the considered system into non-overlapping FCRs where each forms a subsystem that is considered to operate correctly regardless of the state of other FCRs. Therefore, it is reasonable to put components, which use common resources, into the same FCR to prevent the sharing of resources between FCRs.

Fault-tolerant systems are built with the intention to tolerate a specified amount of component failures. For this purpose, the system contains structures (e.g., error detection and masking) which confine the propagation of errors to the service interface of the system. The region in which the error propagation is confined is called Error Containment Region (ECR) [26].

2.2 Mathematical Backgrounds of Reliability

As described in [27] the cumulative distribution function for the reliability, R(t), states the probability that the system delivers its service continuously at least throughout a period of duration t. Opposite to that the failure probability Q(t) is defined as

$$Q(t) = 1 - R(t) \tag{2.2}$$

The corresponding failure probability density function f(t) is defined as

$$f(t) = \frac{dQ(t)}{dt} = -\frac{dR(t)}{dt} \tag{2.3}$$

The failure rate $\lambda(t)$ is specified as the relation between the number of failures and the number of correct components at time t:

$$\lambda(t) = \frac{f(t)}{R(t)} = -\frac{dR(t)}{dt}\frac{1}{R(t)} \tag{2.4}$$

Failure Rates characterized by the Bathtub Function

For a broad range of elements $\lambda(t)$ is often characterized as a "bathtub function" [27](see figure 2.1) which considers a failure rate to be divided into three phases: a "infant mortality" phase with a rapidly decreasing $\lambda(t)$, the "normal life" phase in which the failure rate is constant and a "wear-out" phase with a rapidly increasing $\lambda(t)$.

Constant Failure Rate: $\lambda(t) = \lambda$

A constant failure rate means that the failure behavior of a component knows no symptoms of old age. This assumption is often made to model failures caused by transient faults when such faults and the failure behavior of the component are assumed independent of the component's lifetime. The same assumption is usually used for failures caused by internal, physical faults, although the corresponding failure rate can rather be characterized by the "bath tub function". The usual

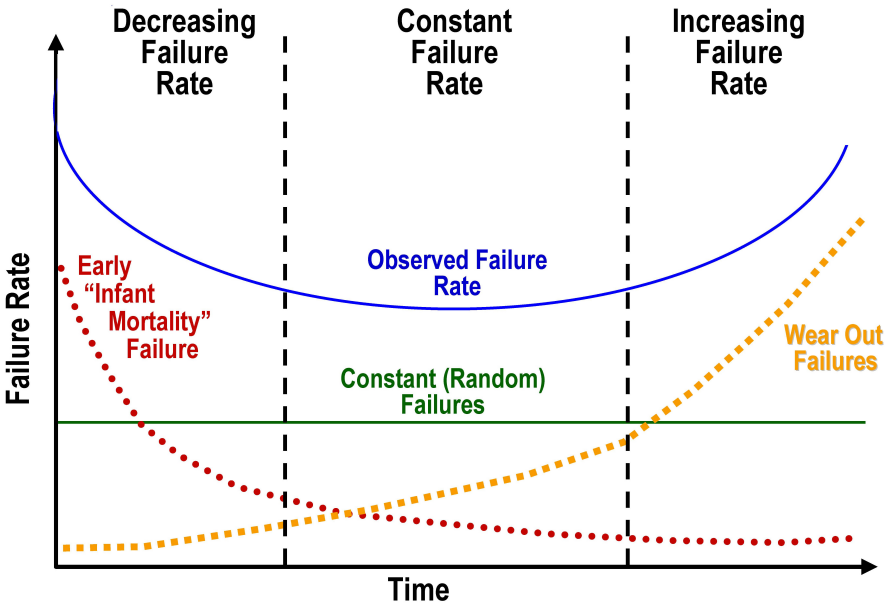


Figure 2.1: Bath Tub Function [1]

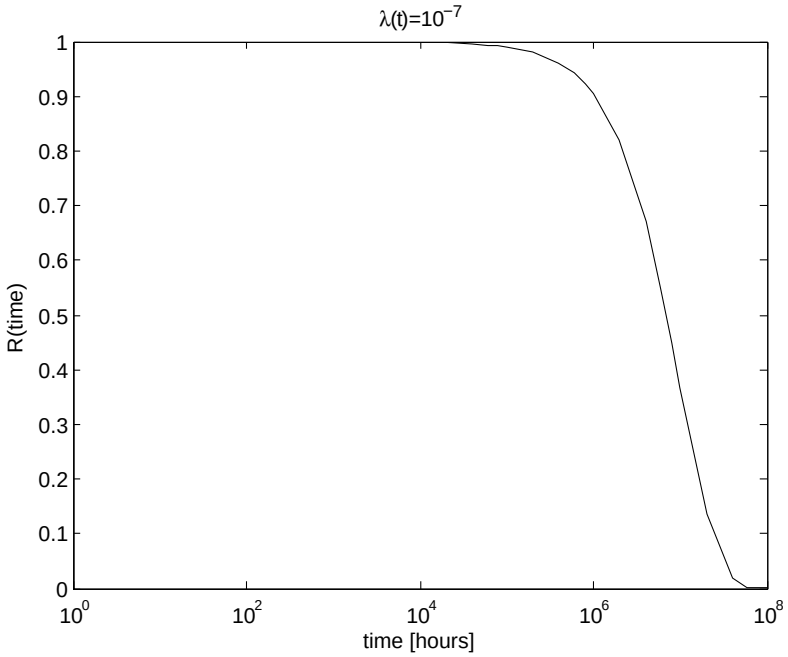


Figure 2.2: Reliability for constant Failure Rate

way is to argue that the "infant mortality" section can be removed by means of accelerated stress testing while the "wear out" section can be ignored due to proper maintenance actions.

A constant failure rate results the following reliability function:

$$\text{Reliability: } R(t) = e^{-t\lambda} \tag{2.5}$$

Therefore the failure probability function is:

$$\text{Failure Probability: } Q(t) = 1 - e^{-t\lambda} \tag{2.6}$$

Formula 2.6 equals the cumulative distribution function of an exponential distributed variable. Therefore, it follows that if a component has a constant failure rate then the stochastic variable that states the duration between component failures is exponential distributed. In the following, such a relation is symbolized by $X_1 \sim Ex_\tau$ which means that the stochastic variable X_1 follows an exponential distribution with the mean value τ.

The use of exponential distributed time functions brings some mathematical benefits for the model designer. One benefit is the simple relationship between MTTF and the failure rate λ:

$$MTTF = \frac{1}{\lambda} \tag{2.7}$$

Furthermore, be $X_1 \sim Ex_\tau$ and $X_2 \sim Ex_\mu$ then the time to the event that either one of the components fail is given by the following stochastic variable Y:

$$Y := min(X_1, X_2) \sim Ex_{\tau+\mu} \tag{2.8}$$

When one of the components failed, the probability that a particular component failed is given by the following probabilities [5]:

$$\boldsymbol{\mathcal{P}}\{\textit{Component 1 failed}\} = \frac{\tau}{\tau + \mu} \tag{2.9}$$

$$\boldsymbol{\mathcal{P}}\{\textit{Component 2 failed}\} = \frac{\mu}{\tau + \mu} \tag{2.10}$$

2.3 Dependability Modeling

Dependability Modeling is defined as the process of gaining quantitative statements about the reliability of a system through the analysis of a dependability model. Such a dependability model usually does not describe the whole functionality of the considered system but models the failure behavior based on the system's fault hypothesis. With Dependability Modeling, the system designer gets a powerful tool to accelerate the development cycle due to the reduced number

of design-implementation-test iterations. The results can be used as evidences in Safety Cases [28] which are requested by many safety standards for critical control applications as appear in the telecommunications, automotive, aircraft and spacecraft domain. But for safety-critical applications dependability modeling is only a good start but not the ultimate mean for the reliability evidence:

> The essence of model building lies in accuracy for the stated purpose, simplification and understandability. Given a set of models that describe a given phenomenon, the model that requires the smallest number of concepts and relationships to explain the issue involved is the preferred one. There is, however, the danger of oversimplification, or of omitting a relevant property [26] [page 72].

So it is essential do use additional evidences of dependability by means of formal verification of application algorithms, experimental verifications of fault tolerance mechanisms by fault injection and by accelerated stress testing of the complete system.

The steps of Dependability Modeling can be listed as following:

- Creating the Fault Hypothesis (Fault Model): The fault hypothesis specifies the number and types of faults the system is designed to tolerate and consists of the following points:
  - Identify FCRs and their Failure Modes
  - Identify Fault Tolerance Mechanisms
  - Estimate FCR Failure Rates and Component Down-Times
  - Appraise the Assumption Coverage: The assumption coverage is defined as the probability that the assumptions made in the model building process prove to be true in practice conditions. The assumption coverage limits the confidence into the conclusions derived from the model [26]. The fault hypothesis used for this paper can be found in chapter 4.
- Choosing Metrics for Analysis [18]: In this step it is decided which metrics reflects the dependability goals of the system at best. E.g., this paper states the results of the dependability evaluations of safety-critical applications by their reliability probability functions and by their MTTFs. These measurements concentrate on the continuousness of system's service because of the potential catastrophic effects of failures. Opposite to that, evaluation results for non safety-critical applications considers on the steady-state availability. More information about the attributes of dependability can be found in section 2.1.

- Creating the Dependability Model: Besides the fault model and the chosen metrics for the analysis there are some other factors that influence the final structure of the dependability model:

  - Constructibility: In the scope of this work constructibility means that validated and verified models of subsystems can be stick together to describe any complex system without the need to redesign or retest the particular submodels.

    So for dependability modeling constructibility brings on the one hand more model reusability and also a more comprehensible overall model, but on the other hand each submodel must implement some structures for state distributions (i.e. communications between the submodels, for an example see section 6.4.2). By this, the overall number of system states increase and leads to increased model calculation times.

    Möbius provides the composition of submodels by so-called Composed Models where the communication between the submodels run over shared state variables. Please refer to section 5.3 for a detailed explanation about the Replication-and-Join formalism in Möbius.

  - Simulation versus Analytic Solving: As explained in section 5.5 the decision between different solving methods leads to crucial constraints for the structure of the model.

  - Choosing Evaluation Tools and the Model Formalisms: For this thesis Möbius [10] was chosen as evaluation tool and the probabilistic structural based models are formulated as SANs. Refer to chapter 5 (*Möbius*) for more information about the used formalism and section 1.4 (*Related Work*) for a list of alternative modeling tools.

- Model Verification: None of the examined evaluation tools (Sharpe [22], SURE [21], Möbius) provides explicit model checking. Instead, in Möbius, the model designer can use the built-in simulator to debug the model. To monitor the state of the model during simulation the simulator can print the model state after each model transition. Additionally the SAN formalism in Möbius supports the placing of assertions and print-commands within output- and input-gateways. Refer to section 5.2 for a detailed explanation of the SAN formalism.

Altitude in meters	SER Multiplication Factor from sea level / New York
sea level	1
500	1.5
1000	2.3
9500	162

Table 2.1: SER Altitude Multiplication Factors described in JESD-89

2.4 Single Event Upset

A long known negative side-effect of the progressing miniaturization of semiconductor devices is their analogously increasing sensibility on strikes of high-energetic particles. The two main causes for such particles are cosmic ray neutrons and alpha particles released by contaminants within the chip packaging material.

2.4.1 Cosmic Ray Neutrons

Cosmic ray neutrons are created together with the less problematic pions and protons by reactions of solar particles in the upper atmosphere. Neutrons are particularly troublesome as they can penetrate most constructions (a neutron can easily pass through 1.5 meters of concrete [11]). The neutron pollution varies for latitudes and altitudes. For example in an airplane, the neutron flux can be 100 to 800 times worse than at sea level [11]. The JEDEC standard for the measurement of particle induced errors [13] provides a methods to estimates the neutron flux (given as number of neutrons per $cm^2 - MeV - s$) depending on altitude, location and sun activity. Table 2.1 shows some multiplication factors for different altitudes in relation to the neutron flux in New York at sea level.

2.4.2 Alpha Particles

Alpha particles are emitted from radioactive impurities in packaging, solder bumps, etc. left behind from the process of manufacture. The alpha problem was regarded seriously and, by agreement, chip vendors lowered it to tolerable levels by reducing the alpha particle flux emitted by packaging and processing materials to generally $< 0.01\alpha/cm^2 - hour$ [29].

2.4.3 Trends for Particle Induced Errors

One of the main critical factors that determines the sensitivity of a semiconductor device to high-energetic particles is the minimum critical charge that must be implanted by a particle strike to create a bit-flip within a memory-like element. Due to the industrial ambitions to make even faster and more power economic chips the technology trend inclines to lower transistor size, cell capacity and supply voltage. On the one hand, this trend leads to a decreasing of the minimum critical charge but on the other hand, the shrinking transistors decrease the probability that a particular transistor collects the minimum critical charge to upset the device. Overall, with process scaling the failure rate per bit is expected to decrease while the likelihood of particle-induced multi-bit errors will increase as well as the probability of soft errors within the combinational logic [30] [31].

2.4.4 Error Classification

According to where a particle strike changes the state within a semiconductor device, the following error classification can be made:

Non Destructive Events

- **SEU**: Single Event Upset are bit-flips in memory-type elements. If one particle strike causes multiple memory upsets this is called a Multi Bit Upsets (MBU). The rate for errors caused by SEUs is usually referred to as Soft Error Rate (SER).

- **SET**: Single Event Transients are "glitches" on signaling lines. Because of such glitches in the peripheral circuitry of memory elements also radiation-hardened memory can be affected from single events.

- **SEFI**: Single-Event Functional Interrupts are errors within configuration memory thus being permanent despite they were caused by transient faults. In comparison to soft errors, they are often referred to as firm errors since the data of the memory is not only corrupted but also the functionality of the device is affected. Especially vulnerable to SEFIs are SRAM based FPGAs. After a bit-flip within its configuration memory, the function of such an FPGA can be disturbed on arbitrary ways until a reconfiguration of the FPGA. Since not every upset in configuration memory leads to a failure, the SEFI rate is usually in the range of 10% to 20% of the SEU rate [16].

Destructive Events

Single-Event Gate Ruptures (SEGRs), Single-Event Latchups (SELs) and Single-Event Burnouts (SEBs) denote permanent errors caused by the thermal destruction of the transistor. This events are quite seldom and their occurrence can be minimized by adapted chip design [13].

Chapter 3

TTSoC Architecture

3.1 Motivation

A typical embedded application consists of many concurrently running jobs. One architectural approach could be the multitasking on a fast single-instruction set computer. But this solution carries many disadvantages with it: the operating system must establish an effective execution environment for the nearly independent application processes which means a worrying level of complexity and therefore a hardly affordable certification process for safety critical applications. Additionally the high overhead by the operating system needs a lot of extra computation time and die area. A faster computer typically needs a higher supply voltage, which leads through its quadratic effect on the power consumption to a non-linear increase of the power dissipation [32]. An architecture that provides a single hardware base for many different DASs is called an integrated architecture [33].

Especially popular in the automotive domain is the spatial isolation of application processes, i.e., each job gets its own electronic unit with a minimum of shared resources, e.g., power supply and network. The dependencies between jobs are minimized so that a job can adequately be described by its network interface to its other partners. An architecture where each DAS has its own dedicated hardware base is called a federated architecture [33]. This facilitates the integration of Intellectual Property (IP), the verification and the certification. On the other hand this approach leads to high costs because of the high number of units and networks and often leads to a waste of resources because of overdimensioned Electronic Control Units (ECUs) for small processes. Furthermore, more interconnections raises also the costs and the probability of interconnection faults. In the automotive domain a typical premium car hosts more than fifty ECUs and five different networks [34].

The TTSoC architecture [35] as depicted in figure 3.1 combines the benefits of both mentioned architectural approaches. The TTSoC architecture provides the

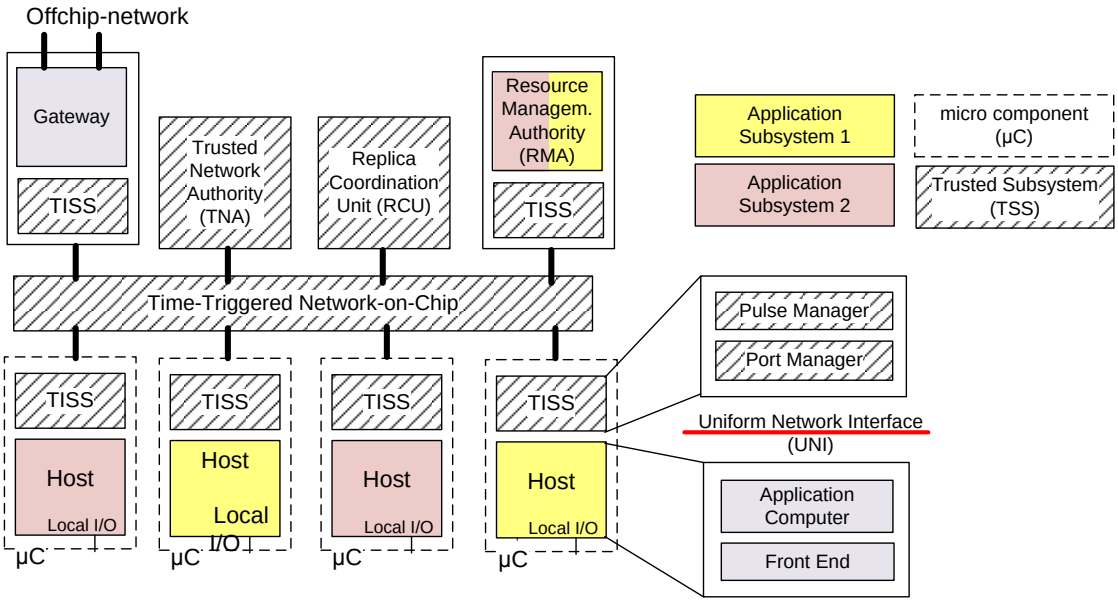


Figure 3.1: Structure of the Time-Triggered SoC Architecture [2][page 4]

partitioning into a set of nearly autonomous possibly heterogeneous Intellectual Property (IP)-blocks or micro components that only communicates with each other via a time-triggered Network-on-a-Chip (NoC). The NoC provides composability and error containment between DASs and so the inherent fault isolation facilitates the certification process immensely.

The following sections give as more detailed view of the architectural SoC components and see chapter 4 for the corresponding fault model.

3.2 Overview

Figure 3.1 depicts an SoC that hosts two application subsystems, perceptible in the figure by the different colors of the hosts. The shaded components deliver essential architectural services to the micro components and belong to the Trusted Subsystem (TSS), which is assumed to be free of design faults and therefore must be certified to the highest criticality level of any host within the SoC. The hosts implement application specific services and belong to a specific Distributed Application Subsystem (DAS). Each host uses the Trusted Interface Subsystem (TISS) to communicate via the NoC. Together they are forming a micro component.

3.3 NoC

The on-chip network interconnects the micro components within an SoC thus being the core of the SoC platform. The NoC provides the following essential services:

Clock Synchronization: To establish a time-triggered communication it is essential to establish a system-wide global time base [35] despite the existence of multiple clock domains. The time-triggered NoC uses a uniform time format which has been standardized by the OMG in the smart transducer interface [36].

Deterministic Message Transport: The NoC uses time-division-multiple-access (TDMA) for bus arbitration which divides the available network bandwidth into conflict free periodical communication slots, each assigned to a dedicated micro component. Therefore the NoC provides a deterministic communication with inherent fault isolation between the DASs.

3.4 Micro Component

A micro component is a self-contained computing element, e.g. it can be realized by a general purpose processor or by special purpose hardware. As depicted in figure 3.1 a micro component consists of an application specific host and a TISS to access architectural services. A host implements one job of a DAS and a DAS can be running on one micro component or be distributed over a group of possibly heterogeneous micro components.

The TISS on the one hand provides TTSoC architectural services like the global time base and on the other hand guards the temporal access to the time-triggered NoC, preventing a faulty host from interfering with the communication slots of other micro components. Thus, faulty hosts can only affect other hosts which belong to the same DAS but then only by providing faulty input values to them, via the sent messages. For this purpose the TISS must know exactly the sending slots of the micro component and so can act as temporal firewall. The TISS is essential to create fault isolation and therefore belongs to the TSS, i.e., it is considered to be free of design faults.

3.4.1 Structure of the TISS and the UNI

The TISS is structured into two layers, the port manager and the pulse manager (see figure 3.2). The pulse manager accesses the TTNoC. It sends and receives messages according to the definitions stored in the Message Descriptor List (MEDL) and consists of three interfaces:

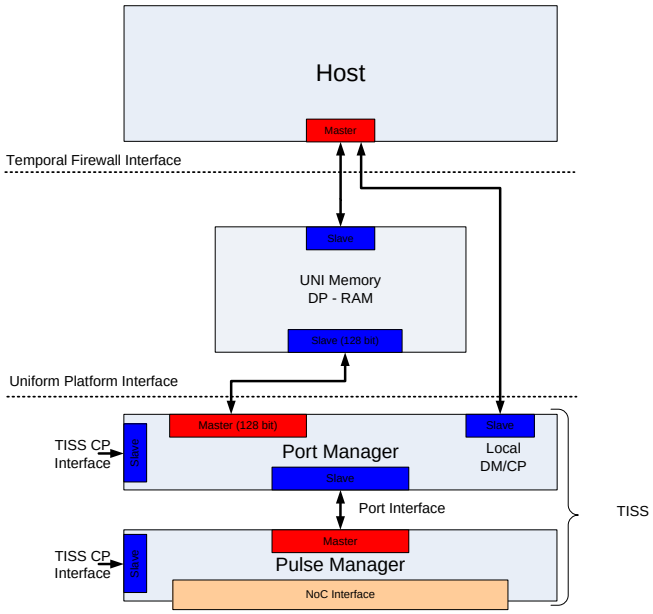


Figure 3.2: Structure of the TISS (based on a figure in [3])

- the TTNoC is accessed via the NoC interface.
- the TISS Configuration and Planning (CP) interface is accessed by the TNA in order to reconfigure the message schedule.
- the Port interface connects the pulse manager to the port manager.

Whenever a message is received, the pulse manager hands it over to the port manager. On the other hand, when a message is scheduled to be transmitted by the pulse manager, the pulse manager issues a request to the port manager.

Between host and port manager the Uniform Network Interface (UNI) is located. The port manager controls the access to the UNI memory. Based on the port configuration table the port manager maps each message that is received or sent by the pulse manager to the corresponding address in the UNI. Additionally, the port manager provides the programmable timer-interrupt service, the watchdog service and the power control service.

The port manager incorporates three interfaces:

- it exchange messages with the host over the UNI.
- it communicates with the pulse manager via the port interface.
- the configuration of the watchdog and the power mode of the host are set by TNA via the TISS CP interface.

- the port configuration table and the timer interrupt service are configured via the local Configuration and Planning (CP) and Diagnosis and Management (DM) interface.

The UNI is implemented as dual ported memory between host and port manager and therefore acts as temporal firewall for the host. Contrary to the TISS, the UNI is highly adaptable according to the specific requirements of a given micro component.

3.4.2 Structure of the Host

The host can be divided into the application computer and the front end. The application computer realizes the application functionality while the front end implements a domain-specific network interface to the application computer, e.g., event queues or the temporal firewall interface [37]. It can incorporate middleware bricks to add additional communication functionalities, e.g., for security (e.g., encryption, authentication) or fault-tolerance mechanisms (e.g., voting).

3.5 Trusted Network Authority

The TNA is part of the TSS and therefore has to be certified at least to the same level as the job of the most critical application subsystem hosted on the component. The TNA provides the following essential architectural services:

- **Establishment and Maintenance of the Global Time for all TISSs:** Because of the time-triggered NoC, a chip-wide global time base is a mandatory requirement for the SoC component. An additional option is the synchronization of the global time with an external time reference over a gateway.
- **Configuration of the Micro Components via their CP Interfaces:** During runtime the Resource Management Authority (RMA) is responsible to create new Time-Division Multiple Access (TDMA) schedules on demand, e.g. if a micro component issues a system mode switch. Because of the complexity of the RMA, it is not part of the TSS and so the job of checking the schedule feasibility and configuring of the micro components is done by the TNA. The TNA is the only component of the TTSoC architecture which can assign essential network configuration parameters to the micro components. So the TNA guarants that the micro components always access a conflict free channel.

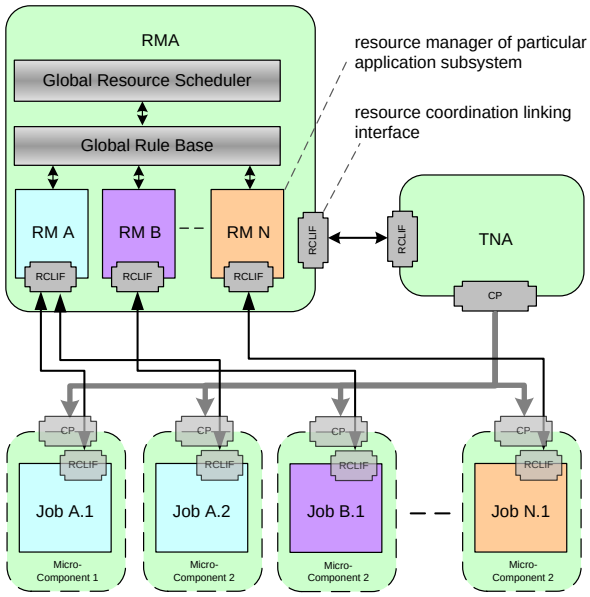


Figure 3.3: Structure of the RMA [4][page 36]

- **Protection of Statically Assigned Resources:** During design time a list of minimum resource requirements for privileged application subsystems like safety-critical ones can be established within the TNA. During runtime the TNA checks new schedules on this rules.

The TTSoC architecture is adaptable in that sense, that it can be dynamically reconfigured in order to meet changing requirements of the hosted application subsystems, e.g. a mode change of the application enforced by the user. The reconfiguration of the architecture is performed solely by the TNA, which forms together with the time triggered on-chip network and the TISS the trusted subsystem of the TTSoC architecture. Since failures within these core components are likely to cause failures of the entire SoC component, the TNA has to be certified at least to the same level as the job of the most critical application subsystem hosted on the component. In order to ease this certification, the complexity of the TNA is kept as low as possible.

3.6 RMA

The Resource Management Authority (RMA) is a dedicated micro component for the dynamically creation of new resource assignments to the micro components. Because of the high RMA complexity it is not part of the TSS and is not considered to be free of design faults. So the created configuration parameters must be passed

to the TNA which checks and forwards them to the micro components as depicted in figure 3.3. The following resources are essential for the RMA:

- **Communication Resources:** The RMA must build a conflict free schedule which at least fulfills the requirements (e.g., bandwidth, latency) for all safety-critical applications.
- **Computational Resources:** Computational resources, like clock frequency, can be dynamically reassigned if the host supports dynamic reconfiguration.
- **Power:** A power-aware SoC where the micro components supports dynamic reconfiguration can perform power management by regulation of the clock frequencies of hosts.

The structure of the RMA can be partitioned into the global resource scheduler, the global rule base and one Resource Manager (RM) per application subsystem (figure 3.3). The purpose of an application-specific RM is the interaction with a host to handle resource requests and to resolve resource conflicts between hosts within the application subsystem. In this way, the RM creates a configuration vector for the hosts of its application subsystem. The global rule base checks the acceptance of the resource allocations created by the RMs and resolves component-wide conflicts. In case of conflicts, the global rule base possesses a list of allowed degraded modes for each application subsystem. On the decisions made by the RMs and the global rule base, the global resource scheduler consolidates all host configuration vectors and forwards the resource allocation to the TNA.

3.7 RCU

The Replica Coordination Unit (RCU) is part of the TSS and has the purpose of detecting transient host failures and managing the recovery and reintegration of faulty hosts. It works together with the voter middleware in the host-front end of replicated hosts by comparing the redundant computational results.

3.7.1 Support of On-Chip TMR

The messages of all replicas are routed to the RCU which compares the values. In case of deviations between replicas, it sends a restart-request to the TNA which has direct access to the TISSs which in turn can reset their micro components. In case of transient faults, the faulty micro component can be immediately restarted and reintegrated into the TMR assemble by synchronizing its internal state with the

other replicas. Each replica periodically transmits its internal state and overwrites its internal state with the voted result synchronous (within the same global tick) with the other replicas.

In case of permanent faults it is likely that the micro component will fail again after a short time. Therefore, in order to detect permanent or intermittent faults, the RCU can make use of failure counters, which increments after failure occurrence and automatically decrements after some failure-free time. If the counter exceeds a certain threshold, the corresponding micro component is switched off and maintenance actions are triggered.

3.7.2 Support of Off-Chip TMR

If the replicas are located on different SoCs we are speaking of off-chip TMR. For off-chip TMR exists no central element which can decide to reset a faulty micro component. Instead, the RCU of each SoC must receive all messages to decide if the local micro component is faulty and must be reset. From a global point of view a SoC is a self-checking component in which only the local RCU can trigger recovery actions. The drawback of this solution is the dependence on the local TSS. Since there architecture does not provide a central element which can reset a complete SoC, no recovery actions in case of TSS faults can be taken. The reason for this architectural decision is, that a central element would facilitate a single point of failure and could not be used for ultra-high dependable systems.

3.8 Gateways

Since component failure rates are usually in the range of 10^3 to 10^4 FIT [38] it is imperative for ultra-dependable systems to use redundancy off-chip level to reach a system failure rate of less than one FIT. For this, the TTSoC architecture supports gateways to access public networks (e.g., the Internet), to construct distributed systems connected by off-chip networks and for the synchronization to external time reference.

When the gateway connects to off-chip network which also works with the time-triggered paradigm (e.g., TTP [39] or TTE [40]), then it is possible to align the periods and phases of the on-chip network with the transmission start instances of the off-chip network to reach bounded message delays with minimum jitter. Also the alignment of the pulsed data stream with the messages of the off-chip networks enables the receiving of the messages by all replicated SoCs at the same global time instance. This is a essential requirement for archiving replica determinism [41] as needed for TMR with exact voting.

Chapter 4

Fault Model

4.1 Introduction

As foundation for the Möbius SoC models, this chapter presents the fault model for a single SoC as well as clusters containing multiple DECOS SoC components. Because of the limited independence of IP cores within an SoC, the conventional approach for a fault model would be the consideration of the complete chip as a single FCR for physical- and design faults, as outlined in [26]. Justified by the distributed, fault-tolerant architecture of the TTSoC architecture, this fault model exhibits a more detailed structuring of FCRs. The resulting assumption coverage is bounded by the probability of common mode faults, e.g., power supply variations.

Because of the reusability of design, design faults are not subject to the same containment as physical faults and so the fault model for a SoC component is partitioned into a design fault model and a physical fault model. The design fault model comprises hardware and software design faults and the physical fault model describes faults caused by physical exposures.

Since the TTSoC architecture does not impose a specific technology for inter-SoC communication, the network fault model is given for a fault-tolerant, time-triggered off-chip network and is described apart of the SoC fault model.

4.2 Design Fault Model for a single SoC

4.2.1 Fault Containment Regions

The design fault model contains the following FCRs:

Replicated Host: This FCR includes the application computer and the front end. In case of replication, all replicas (possibly on different SoCs) belong to the same FCR.

TSS: This FCR comprises the safety-critical infrastructure of an SoC and consists of the RCU, the NoC, the TNA and the TISSs. The TSS is assumed to be free of design faults. The simplicity of the TSS's design in the TTSoC architecture facilitates a thorough validation (e. g., by means of formal verification) which substantiates this assumption.

RMA: The RMA is not part of the TSS. The reason for this is, that due to the high complexity of the RMA, which has to dynamically compute time-triggered communication schedules from the reconfiguration requests, validation to the highest criticality levels (e. g., class A according to DO-178B or SIL4 according to IEC61508) would generally be infeasible. So the RMA (or parts) can be developed by different vendors and exhibit a lower rigidity in the development process. The potential residue of design faults in the RMA is taken into account by means of a resource allocation protection by the TNA. The TNA protects safety-critical application subsystems from being disturbed by reconfiguration actions by a faulty RMA. Thus, only non safety-critical application subsystems can be affected by an RMA failure.

4.2.2 Failure Modes and Error Containment

Table 4.1 identifies all failure modes per FCR for the design fault model and the corresponding error containment mechanisms applied within the TTSoC architecture. The first column states the failing FCR, while the second column differentiates a FCR failure into failure modes. The third column states the component of the TTSoC architecture which is responsible for failure detection or masking and the forth column states the error containment mechanism and its success-probability (also referred to as *coverage*).

Since the TSS is assumed to be free of design faults, the TTSoC architecture possesses no error containment mechanisms for TSS failures.

[1]Value errors produced by faulty hosts can never exceed the boundary of the respective application sub system. Within this boundary a value error can propagate to other hosts.

[2]The coverage is bounded by the probability (=design fault correlation) that all host replicas concurrently suffer from a common design fault. Since more than one micro component would fail, TMR would not work.

[3]It is not a requirement on the TTSoC architecture to have a detection coverage of 100% for errors in resource allocations, which only affect non-safety critical sub systems.

failing FCR	Failure Modes	detecting FCR	Containment Mechanisms
Replicated Hosts	crash failure	Port Manager	**detection:** Watchdog Timer **coverage:** 100%
	temporal domain	UNI	**masking:** temporal firewall by UNI [1]; **coverage:** 100%
	value domain	receiving Host-Front Ends	**masking:** TMR **coverage:** $\leq$ 100% [2]
	power mode failure	Port Manager	**detection:** TISS power detection; **coverage:** 100%
RMA	crash failure	Port Manager	**detection:** Watchdog Timer **coverage:** 100%
	temporal domain	UNI	**masking:** temporal firewall by UNI; **coverage:** 100%
	value domain	TNA	**masking:** Resource Assertions **coverage(1):** 100% for safety-critical sub system requirements **coverage(2):** <100% [3] for non-safety-critical sub systems
TSS	arbitrary	no error containment!	

Table 4.1: ECRs for the Design Fault Model

4.2.3 Failure Rates

The design failure rates can be estimated by means of the desired Safety Integrity Levels (SILs) per FCR (according to the IEC61508 standard, see table 4.2). When using design diversity each application computer can be considered as own FCR. The according assumption coverage is bounded by the design fault correlation which is determined by the usage of common resources like development tools, compiler, specification representation etc. ([42]). Table 4.3 summarizes the design failure rates.

4.2.4 Design Fault Tolerance

The system architecture assures correct service for safety critical applications in case of any arbitrary design faults in the RMA and in hosts of other application subsystems. The architecture assumes zero design faults in safety-critical hosts

SIL	FIT	Severity of Failure
4	[1, 10)	Catastrophic
3	[10, 100)	Hazardous/Severe
2	[100, 1000)	Major
1	[1000, 10000)	Minor

Table 4.2: SILs according to IEC61508

FCR	SIL	Failure Rate
Safety Critical Host	4	10 FIT
Non Critical Host	1	1000 FIT
RMA	2	100 FIT
Trusted Subsystem	4	1 FIT

Table 4.3: Failure Rates for the Design Fault Model in FITs (one FIT is one failure in 10^9 device-hours)

and in the TSS.

In case of a RMA failure, the system architecture runs in a degraded service mode, which means that no more schedule changes and switches between operational modes of the system are possible. Because the TNA guarantees only correct resource allocations to safety critical applications, the correct service of non-safety critical applications is not any longer assured in case of RMA failures.

4.3 Physical Fault Model for a single SoC

4.3.1 Fault Containment Regions

The physical fault model contains the following FCRs:

Host + UNI + Port manager: For physical faults, a host in conjunction with the port manager constitutes an FCR. Neither the port manager nor the host can interfere with the correct operation of the pulse manager or the NoC.

Pulse Managers + TNA + RCU + NoC: The pulse manager is the sole part of a micro component which has direct access to the NoC. So a failure of one of the pulse managers can cause an overall SoC failure. Also the TNA can cause a

common mode failure and therefore it is reasonable to combine all pulse managers and the TNA together into one FCR.

RMA: Since the resources of safety-critical application subsystems are protected by the TNA, a fault in the RMA can affect exclusively non safety-critical application subsystems.

4.3.2 Failure Modes

Table 4.4 identifies all failure modes per FCR for the physical fault model and the corresponding error containment mechanisms applied within the TTSoC architecture. The first column states the failing FCR, while the second column differentiates a FCR failure into failure modes. The third column states the component of the TTSoC architecture which is responsible for failure detection or masking and the forth column states the error containment mechanism and its probability of success (coverage).

4.3.3 Failure Rates

The failure rates for the physical fault model refer to neutron and alpha-particle induced SEUs in SRAM based FPGAs at sea level in New York. Failure rates for permanent faults are estimated by multiplying the failure rates for transient faults with a constant permanent-to-transient fault ratio factor. A common assumption is a proportion of about 1:10000 from permanent to transient faults [26]. Further details to the estimation process of transient failure rates can be found in section 4.7.

4.3.4 Assumption Coverage

The assumption coverage is bounded by the probability of common mode failures: Faults, affecting the whole die like a damaged power supply, physical damage by force, etc.

[5]Value errors produced by faulty hosts can never exceed the boundary of the respective application sub system. Within this boundary a value error can propagate to other hosts.

[6]It is not a requirement on the TTSoC architecture to have a detection coverage of 100% for errors in resource allocations, which only affect non-safety critical sub systems.

failing FCR	**Failure Modes**	**detecting FCR**	**Containment Mechanisms**
Host	crash failure	Port Manager	**detection:** Watchdog Timer **coverage:** 100%
	temporal domain	UNI	**masking:** temporal firewall by UNI; **coverage:** 100%
	value domain	receiving Host-Front Ends	**masking:** TMR **coverage:** 100%
	power mode failure	Port Manager	**detection:** TISS power detection; **coverage:** 100%
UNI	value domain	receiving Host-Front Ends	**masking:** TMR **coverage:** 100%
Port Manager	temporal domain	Pulse Manager	**masking:** temporal firewall by Pulse Manager [5] **coverage:** 100%
	value domain	receiving Host-Front Ends	**masking:** TMR **coverage:** 100%
RMA	crash failure	Port Manager	**detection:** Watchdog Timer **coverage:** 100%
	temporal domain	UNI	**masking:** temporal firewall by UNI; **coverage:** 100%
	value domain	TNA	**masking:** Resource Assertions **coverage(1):** 100% for safety-critical sub system requirements **coverage(2):** <100% [6] for non-safety-critical sub systems
TSS	arbitrary	no error containment!	

Table 4.4: ECRs for the Physical Fault Model

4.4 Repair and Recovery Durations

In the fault model we distinguish between repair and recovery in that way that repair removes permanent faults and is executed by an external force while recovery happens autonomously after transient faults. We got concrete values for the durations from typical values of the aviation and the automotive domain: For the MTTR we took the typical airplane mission time of 10 hours and for the recovery time we took the maximum allowed activator freezing time of 50 ms for automotive systems.

Component	Transient Failure Rate	Permanent Failure Rate
Host	10^4 FIT	1 FIT
TISS-Port manager + UNI	276 FIT	0.03 FIT
TISS-Pulse manager	319 FIT	0.03 FIT
RMA	404 FIT	0.04 FIT
TNA	144 FIT	0.02 FIT

Table 4.5: Failure rates for the Physical Fault Model in FITs (one FIT is one failure in 10^9 device-hours)

4.5 Error Detection Mechanisms

4.5.1 TISS - Watchdog Timer

In order to detect crash failures of the host (respectively RMA) the TISS provides a watchdog service. The period by which the live sign has to be set by the host can be parametrized by the TNA. This detection mechanism does not detect the type of the underlying fault (permanent or transient) and always triggers a reset and reintegration of the micro component. In case of permanent faults, the permanently switching off and the replacement action must be initiated by the RCU.

4.5.2 TISS - Power Monitoring

The TISS has physical control over the power lines or the clock lines of the host in order to be able to reset the micro component in case it does not operate in conformance to its specification. The host can be powered down via the TISS Configuration and Planning (CP) interface which can be exclusively accessed by the TNA.

4.5.3 TMA - Schedule Error Detection

The TNA acts as a guard for the reconfiguration activities performed by the RMA and provides the following features:

- detection of potential collisions on the time-triggered NoC.
- detection of violations of resource reservations of privileged application subsystems.

- resetting of a faulty RMA.

- if an erroneous resource schedule is detected, the current configuration remains unchanged and the new schedule is rejected.

Detection Limitations: The TNA does not detect missing slot allocations for non critical sub systems.

4.5.4 RCU - Results Comparison

The RCU receives the computational results of all replicated micro components and triggers their recovery or replacement in case of detected failures (see chapter 3.7). For on-chip TMR, the RCU is capable to detect failures in the value domain and to trigger reset or replacement, as long as the TSS is running. For off-chip TMR, an RCU can also remove local micro component failures but since the architecture does not offer a central element which can reset a complete SoC (to save a distributed systems from single point of failures), an SoC can not be recovered after a TSS failure.

4.6 Fault Model for an Off-Chip Network

4.6.1 Introduction

This chapter explains the fault model for safety critical inter-SoC time triggered networks like TTP [39] and TTE [40] as a foundation for the SoC cluster models, later presented in this paper.

Figure 4.1 depicts schematically the interconnection of four SoCs over a time-triggered, star-coupled network.

For fault-tolerance purposes two switches are used, each enclosed from a guardian which protects the whole network from temporal domain SoC failures. Each Guardian has exact knowledge about the sending times of each SoC and acts as temporal firewall in the same way as the TISS for the NoC does. The connection between an SoC and the cluster network is made by the gateway micro component.

Figure 4.2 depicts the structure of a gateway. The gateway consists of a gateway host as intelligent mediator between intra and inter-SoC network and two redundant communication controllers.

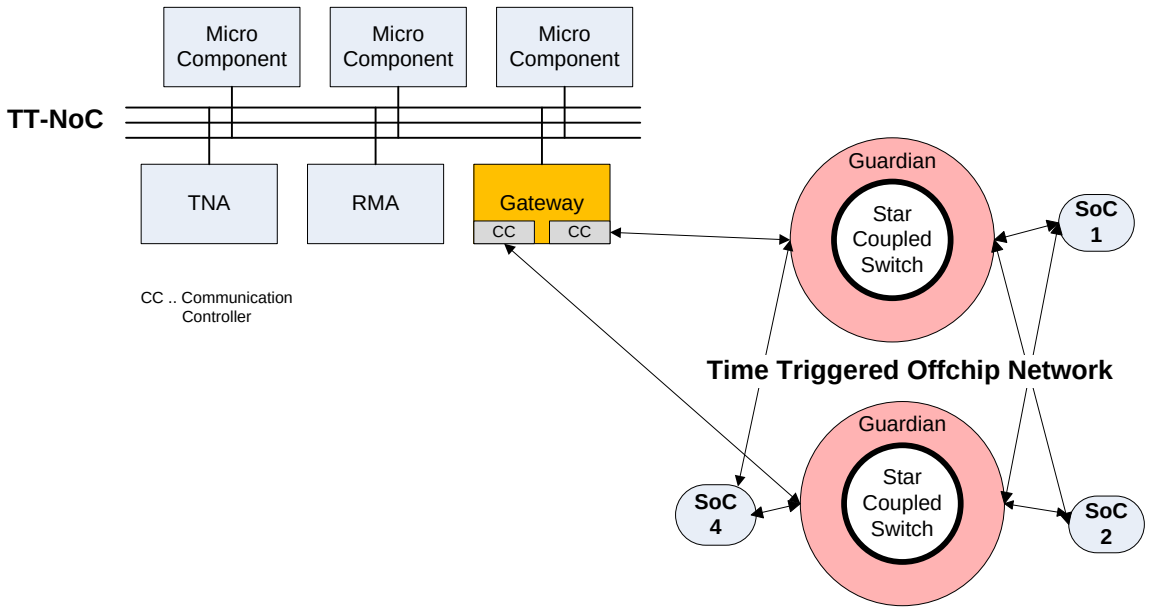


Figure 4.1: SoC Interconnection by a Fault-Tolerant Time-Triggered Network

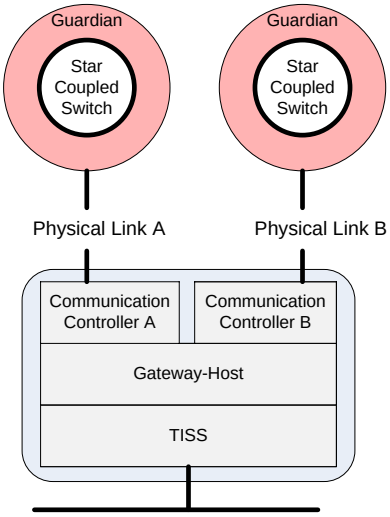


Figure 4.2: Gateway - Micro Component

4.6.2 Fault Containment Regions

The following table summarizes the FCRs of a fault-tolerant SoC Cluster connected by a time-triggered off-chip network:

Switch + Guardian: Physically the Guardian is a part of the switch and in this way, together, they form a FCR.

Physical Link: A physical link is a bidirectional physical connection between an SoC and a Switch. Because the two physical links to the redundant off-chip network are spatial close together, the assumption coverage is bounded by the probability of correlated channel failures (e.g. caused by EMI).

Communication Controller: A communication controller is part of the Gateway micro component. One controller per physical link is assumed.

Gateway Host: A gateway application computer has the purpose of mediation between the NoC and the off-chip network.

4.6.3 Error Containment

Table 4.6 identifies all failure modes per FCR for the physical fault model and the corresponding error containment mechanisms applied within the TTSoC architecture. The first column states the failing FCR, while the second column differentiates a FCR failure into failure modes. The third column states the component of the TTSoC architecture which is responsible for failure detection or masking and the forth column states the error containment mechanism and its success-probability (coverage).

4.6.4 Number of Tolerable Faults for the Off-Chip Network

In the following, the set of related communication controller, physical link and switch is called a channel. An SoC has a valid off-chip network interconnection if its TSS, the gateway host and at least one of its two channels are valid.

4.6.5 Failure-, Recovery- and Restart Rates

The failure rates for communication controller and gateway application computer are estimated over their resource usages as mentioned for the SoC fault model. Also the repair and recovery durations are the same as in the chip fault model.

The failure rate for switches are taken from typical assumptions about electrical devices while for simplification restart-, recovery rates and permanent-to-transient fault ratio are taken from the SoC fault model.

failing FCR	Failure Modes	detecting FCR	Containment Mechanisms
Gateway Host	crash failure	Port Manager	**detection:** Watchdog Timer **coverage:** 100%
	temporal domain	UNI	**masking:** temporal firewall by UNI; **coverage:** 100%
	value domain	receiving Host-Front Ends	**masking:** TMR **coverage:** 100%
	power mode failure	Port Manager	**detection:** TISS power detection; **coverage:** 100%
Comm. Controller	temporal domain	Switch	**masking:** temporal firewall by Switch-Guardian; **coverage:** 100%
	value domain	receiving Comm. Controller	**detection:** CRC check **coverage:** $\leq$100%
Physical Link	temporal domain	Switch	**masking:** temporal firewall by Switch-Guardian; **coverage:** 100%
	value domain	receiving Comm. Controller	**detection:** CRC check **coverage:** $\leq$100%

Table 4.6: ECRs for the Off-Chip Network Fault Model

FCR	Transient Failure Rate	Permanent Failure Rate
Switch + Guardian	10^4 FIT	1 FIT
Physical Link[7]	10^5 FIT	10^4 FIT
Communication Controller	700 FIT	0.07 FIT
Gateway Host	100 FIT	0.01 FIT

Table 4.7: Failure Rates for the Off-Chip Network Fault Model in FITs (one FIT is one failure in 10^9 device-hours)

For physical links the failure rate and failure duration are taken from typical EMI occurrences, which are assumed to occur with a rate of 10^5 FIT while existing for about two seconds. The rate for permanent link faults, i.e., broken links, is taken from the automotive domain and is valued at 10^4 FIT.

Table 4.7 summarizes the failure rates for the off-chip network fault model.

	FIT per MBit		
	SRAM configuration memory	**Flip Flops**	**SRAM application memory**
Neutron Induced	188 FIT	613 FIT	770 FIT
Alpha Induced	229 FIT	748 FIT	939 FIT

Table 4.8: Failure Rates for different Memory Classes of an Altera Cyclone II EP2C20 [43]

Component	Configuration Bits	Application Bits	Flip Flops
Host	1M	10M	7.2K
TISS-Port manager + UNI	400K	6.4K	600
TISS-Pulse manager	600K	16K	1.5K
RMA	600K	100K	1.5K
TNA	300K	15K	0.5K

Table 4.9: Resource Requirements of SoC Components

4.7 Failure Rate Estimation for SRAM-based FPGA

This paper uses a quite simple approach to estimate transient failure rates of SoC subcomponents implemented within an SRAM-based FPGA. With the FPGA radiation results of an Altera Cyclone II EP2C20 [43] (summarized in table 4.8) which examines the upset rates for configuration and application bits, Flip Flops and combinatorial logic (whereby the last point is negligible for the Cyclone II) and the resource usage per subcomponent (TISS, TNA, RMA, etc.) a rough estimation of subcomponent failure rates could be done. Since not each SEU causes a failure the subcomponent failure rates can rather be seen as upper bounds. Consequently the reliability assessments deliver conservative results. The necessary resource usage was estimated by means of a prototype implementation ([44] [45] [46]) (see table 4.9).

Chapter 5

Möbius

5.1 Overview

Möbius (for a complete description see [10]) is a software tool for modeling performance and dependability of complex systems. It uses an integrated multi-formalism, multi-solution approach, i.e., the overall model can be divided into smaller pieces and treated with different model formalisms and solution techniques. Möbius was developed by the Performability Engineering Research Group at the University of Illinois at Urbana-Champaign. Head and founder of this group is Prof. William H. Sanders.

Figure 5.1 shows how Möbius divides the workflow of system attribute assessment into successive work packages, each supported by its own model formalisms. The classical modeling work is done in atomic and composed models. Atomic models are the basic modeling elements which consist of states, state transitions and parameters. Composed models assemble a set of atomic- or composed models to an overall composed model of your system. On the basis of composed and atomic models the reward model describes the calculation of so-called performance variables, e.g., MTTF or Availability. Within theses models, parameters can be represented by global variables so the last step of defining a concrete model is the assignment of parameter values by means of an own study editor. A complete set of value assignments is called an experiment, while a set of experiments is called a study. Möbius provides two different solution approaches for the evaluation of the performance variables: analytical solving and simulation.

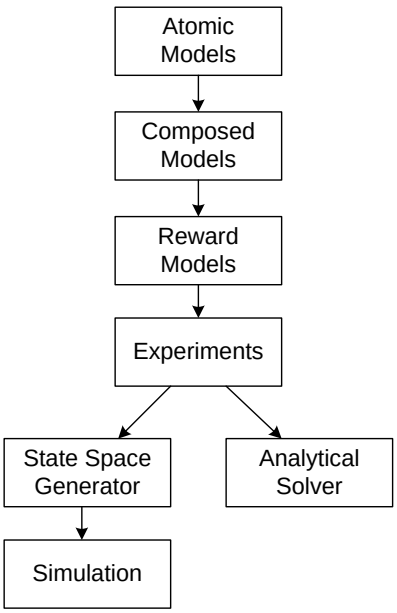


Figure 5.1: Möbius Workflow

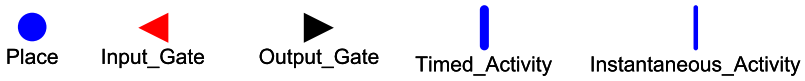


Figure 5.2: SAN Primitives

5.2 Atomic Models as SANs

For this thesis all atomic models for SoC reliability evaluations were created as SAN as presented in [9]. SANs are stochastic extensions to Petri-nets which can use arbitrary data structures instead of simple Petri-net places and allow the execution of functions on the model state before and after state transitions.

SANs consists of five basic elements: places, timed activities, instantaneous activities, input gates and output gates. Figure 5.2 shows their graphical representation in Möbius and the following sub-chapters describe them in detail:

5.2.1 Places

Places in SANs have the same meaning as in Petri nets and can hold a natural number greater or equal than zero. Usually this number is referred to as number of marks stored in the place. Additionally places are able to hold more complex

data structures like floating point numbers, timestamps and records.

Another feature of SAN places is that they are accessible from all over the whole SAN and from the rewards models by means of C expressions. Please refer to [47] for details about the programming language C. For example the place `PC_OK` can be accessed over `PC_OK->Mark()`.

5.2.2 Input Gateways

Graphically, the input gateway only must be connected to activities, but for the purpose of documentation input gateways can also be connected to places. Input gateways possess the following two parameters which both must be given in C code:

- **Input Predicate:** The input predicate defines an enable-condition for all connected activities. The predicate is a C expression defined on the state variables of the SAN. If the predicate is true, the lines to all connected activities are enabled.
- **Input Function:** If a connected activity completes, then the C expression of the input function is executed.

5.2.3 Output Gateways

Output gateways defines a function which is executed when a connected activity completes. They must be connected to activities, but may be connected to places for documentation purposes, as explained for input gateways the sub-section before.

5.2.4 Timed Activities

An activity represents a transition from one system state to another state within an either deterministic or stochastic distributed amount of time. The activity can be in three different states: Initially the activity is **disabled** and becomes enabled if all places and input-gates, connected to the activity by input-lines, contain at least one mark or are enabled, respectively. While being in the **enabled** state the activity is running down an internal counter which was initialized by the activity time. After the expiration of this term the activity is said to be **completed**. In this case each place connected towards the activity looses one mark. All places which are connected to the activity output gains a mark and all connected output-gates are activated. An additional feature of activities are

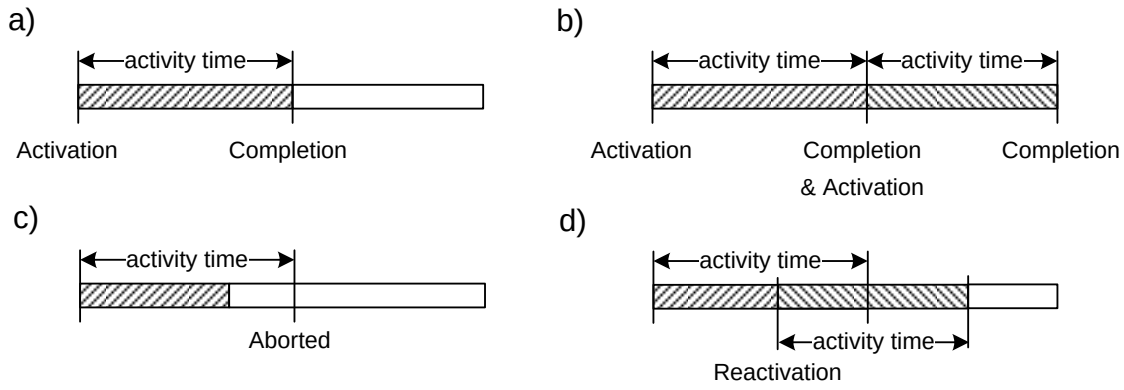


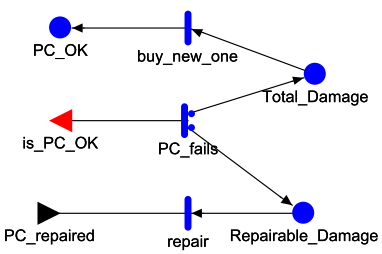
Figure 5.3: Execution of a Timed Activity [5]

case probabilities. Graphically, a case is indicated by a small circle at the activity output. From these cases originate output lines towards output-gateways and places. When the activity completes, determined by user-defined probabilities, a case is chosen and only output-gates and places connected to the chosen case are further considered.

An activity possesses the following properties:

- **Time Distribution Function**: This property gives the firing time distribution for the activity and can either be deterministic or stochastic, e.g., exponential distributed.
- **Rate, Mean, Variance**: According to the chosen distribution function the activity possesses different lists of distribution parameters. E.g., if the activity time is normal distributed, then the parameters *Mean* and *Variance* must be set.
- **Case 1, Case 2, ...**: To determine the case chances, each case must be weighted by a real number. So when the activity fires, Möbius calculates the probability for each case and then randomly chooses one of them.
- **Reactivation Function**: The reactivation function gives the necessary conditions of the system state to trigger a reactivation of the activity. This parameter is optional, figure 5.3-(d) shows a possible timeline.

Figure 5.3 shows four example time lines for the execution of a timed action. Shaded areas represent the enabled state and the even areas the disabled state of the activity. After the activity time in (a) the activity completes and the following actions changes the system state in that way that the activity becomes disabled. In (b) after the first activity completion the activity is still enabled so that a



a)

Place Names	Initial Markings
PC_OK	1
Repairable_Damage	0
Total_Damage	0

b)

Output Gate	PC_repaired
Function	PC_OK->Mark() = 1;

c)

Input Gate:	is_PC_OK
Predicate	PC_OK->Mark() > 0
Function	;

d)

Timed Activity:	PC_fails
Distribution Parameters	Rate: 10
Case Distributions	case 1: 50
	case 2: 50

e)

Timed Activity:	buy_new_one
Distribution Parameters	Rate: 50

f)

Timed Activity:	repair
Distribution Parameters	Rate: 20

Figure 5.4: SAN Example

new activation cycle begins. The activity in (c) looses its enabled state before completion and so the action is aborted. In (d) before the activation can complete the activity is reactivated and so a new activation cycle begins. This timeline is only possible, if the reactivation function of the activity is set.

Since the effects of actions can be marking-dependent, the following order of events after activity completion must be considered (as defined in [9]):

1. If the activity has cases, a case is (probabilistically) chosen.
2. The functions of all the connected input gates are executed (in an unspecified order).
3. Tokens are removed from places connected by input arcs.
4. The functions of all the output gates connected to the chosen case are executed (in unspecified order)
5. Tokens are added to places connected by output arcs connected to the chosen case.

5.2.5 Instantaneous Activities

Instantaneous activities have the same execution rules like the timed activities but with the difference that completion happens immediately after activation.

5.2.6 SAN Example

Figure 5.4 depicts the graphical representation and its corresponding configuration tables of a simple SAN example. It models a personal computer (PC) which either can suffer a total or a repairable damage and the PC can be replaced or repaired, respectively. In any cases, the model always reaches the state where the PC is running again, after some time.

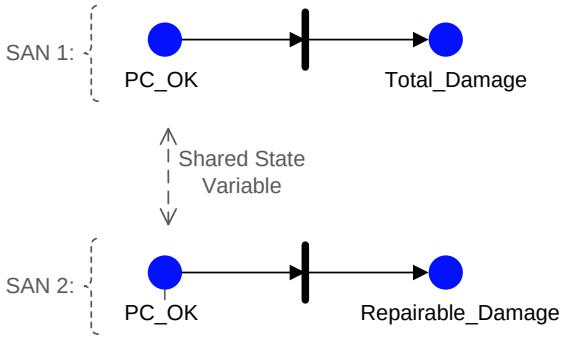


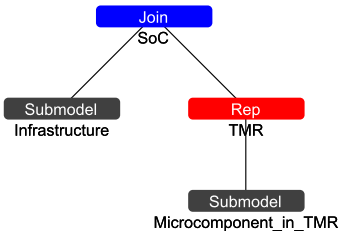
Figure 5.5: Shared State Variables in Composed Models

So the model consists of three places with the initial markings shown in table (a) of figure 5.4.`PC_OK` is one, if the PC is running, zero otherwise. `Total_Damage` is one if the PC is assumed to be non repairable, zero otherwise. And `Repairable_Damage` is one if the PC must become repaired.

The only input-gateway is `is_PC_OK`, described by table (c). It defines a predicate that checks the state of `PC_OK`. So if this place is greater than zero, then the input-gateway is enabled and by the connection to the activity `PC_fails` it defines the enable-condition for the activity. Since the initial marking of `PC_OK` is one, the activity is enabled from the start. After the activation time of `PC_fails`, the activity chooses between two cases. Table (d) shows their case distributions. By a fifty-fifty chance, either `Total_Damage` or `Repairable_Damage` is filled with a mark by the activity. If `Total_Damage` is filled with a mark, every input-line of `buy_new_one` is enabled and so the activity is enabled, which in turn fills `PC_OK` with a mark after the activation time. In the other case, `Repairable_Damage` is filled with a mark which enables repair and which in turn leads to an execution of the output-gateway function of `PC_repaired`. As one can see in table (b), the function sets `PC_OK` back to one. So in either case the whole model is back in its initial state where the whole cycle can start again.

5.3 Composed Models

Complex models can possess a great amount of redundant structures, e.g., when describing TMR. Therefore it is reasonable to define an own class of meta-models to describe submodel replications and their information exchange. For this purpose Möbius provides composed models with the so-called Rep & Join formalism ([5]). The building blocks of composed models are atomic or other composed models. These submodels can be connected by replication-operators (Rep-nodes) or join-operators (Join-nodes). In this manner connected submodels share information by



a)

Rep Node	Reps	Shared State Variables
TMR	3	DAS_OK
		Microcomponents_Failed
		TSS_OK

b)

Join Node	State Variable Name	Submodel Variables
SoC	TSS_OK	Infrastructure->TSS_OK
		TMR->TSS_OK

Figure 5.6: Example of a Composed Model

means of shared state variables as depicted in figure 5.5. This figure shows two SANs which both share the place `PC_OK`. Both SANs have access to the same variable. Since analytical solvable models must not have more than one deterministic activity at any time, the access collision issue is confined to simulation which offers no rules for mutual exclusions but uses the principle of contingency.

5.3.1 Example of a Composed Model

Figure 5.6 shows an example of a composed model. It describes a system with three replicated micro components (the corresponding sub model is named *Safety_Critical_Microcomponent_in_TMR*) on a chip, whereby the service functionality of the chip itself is modeled by the submodel called *Infrastructure*.

In the figure, table (a) describes the parameters of the Rep-node: The Rep-node `TMR` defines the triple replication (stated in the Reps column of the table) of the sub model *Safety_Critical_Microcomponent_in_TMR*. The column *Shared State Variables* of table (a) lists common used places of the replicated submodels. So ,e.g., by the sharing of the place `Microcomponents_Failed` each submodel can increment this variable in case of a failure and so a common view on the system state for all submodels is realized.

The Join-node has the purpose to connect the submodel *Infrastructure* with the replicated micro components (Rep-node `TMR`). Table (b) shows the corresponding parameters. The first column holds the name of the Join-node, the second column defines a new name for the state variable and the third column lists all variables from the connected submodels which shall share a common storage. So in the example if the *Infrastructure* model clears the variable `TSS_OK` then, because of the Join-node and the Rep-node, each submodel in the composed model has the same view on the cleared `TSS_OK`.

5.4 Reward Model

The reward model is built on top of atomic and composed models and specifies measures by defining so-called performance variables, e.g., measures to evaluate the availability or the MTTF. The core of a performance variable is determined by one or both (as sum) of the following rewards:

- **Rate Reward:** A rate reward specifies measures on the state of the model at an instant of time.
- **Impulse Rewards:** An impulse reward is a function on the state of the system and the state of a single action. The impulse reward function is evaluated each time the assigned action completes.

Either of the two reward types consists of a predicate on the model state and a reward function. So if the predicate is true at a point in time, then reward function is calculated and added to the performance variable. The reward can be specified to be calculated at the following times:

- **Instant Of Time:** Gives the performance variable for specific points in time.
- **Interval Of Time:** Gives the integral of the performance variable over an interval.
- **Time Averaged Interval Of Time:** Gives the integral divided by the interval duration.
- **Steady State:** Gives the mean value of the performance variable at an initial transient time and at some far away lying points of time.

In Möbius the reward must be given in C notation, whereby the predicate is formulated as IF-condition and the reward function is coded as return statement.

5.4.1 Rewards on Replicated Models

When a reward is defined on a replicated submodel and the reward function returns the value of a shared state variable, then it is necessary to divide the reward function by the number of replicas. The reason is that Möbius evaluates the reward over all replicas. The next code listing shows an example reward for a TMR system. The reward should return 1 if less than one replica failed and because Möbius would evaluate the reward three times it is necessary to divide the reward function by three:

```
if (TMR->replica_failed->Mark() < 1)
{
        return 1.0 / 3.0;
}
```

5.5 Solving Models with Möbius

5.5.1 Analytical Solving

Analytical solvers calculate their solutions by analyzing the underlying state space of the discrete-state stochastic Poisson process, inherent in a model. So the analytical approach of Möbius is divided into two steps. First the model state space must be generated by the so-called State Space Generator and then on this interim result one of Möbius's analytical solvers can work out the results for the performance variables. The big benefit of analytical solvers is the capability of providing statistically accurate solutions up to machine precision. One drawback are the strong restrictions on the model:

- All timed activities are deterministic or exponentially distributed.
- At most one deterministic activity is enabled at any time and its firing delay must not be state-dependent.
- The model must have a finite, small state space.
- When instantaneous activities are used:
  - The model must start in a stable state.
  - The model must be self-stabilizing and well-specified.

A second drawback are the high calculation times for systems with a mix of very slow and very fast transition rates (i.e. stiff model [48]) where the rate of the Poisson process is determined by the fastest rate, but the time points of interest are often in the scale of the slow rates causing a high number of solver iterations. Naturally models of ultra-reliable systems possesses two different groups of transitions: relatively fast transitions that characterize fault/error handling mechanisms and relatively slow transitions that describes the fault-occurrence mechanisms. A stiffness ratio of 10^10 together with a state space of 10^5 states still appears infeasible for numerical solving techniques [49].

5.5.2 Simulation

Instead of analyzing Markov chains the discrete event simulator gets its results by experimentation and statistical analysis of a number of simulation batches. Therefore the big benefit of the simulator is that it can solve every model without restrictions, even stiff models and non-Markovian models, and delivers statistically accurate solutions determined by a user-specifiable confidence interval. The drawback is that more result accuracy is paid in calculation time.

For the evaluations presented in this paper it was decided to use the discrete event simulator from Möbius because of the mix of very fast recovery rates and very slow failure rates in the SoC model. The analytical solver shows unbearable long calculation times. Instead the simulator is used with a confidence level of 95% and a confidence interval of 10%.

Chapter 6

SoC Model with Möbius

6.1 Notations

- $\mathcal{P}\{...\}$: Probability of ...
- $\mathcal{A}\{...\}$: Availability of ...
- Failure rates usually refer to transient failure rates, while permanent failure rates are estimated by multiplying the transient failure rate by the permanent-to-transient fault ratio. (Parameter name: *perm_trans_ratio*)
- The formula $(1 + perm\_trans\_ratio) * transient\text{-}failure\text{-}rate$ calculates the totalized failure rate for transient and permanent failures.
- `Place->Mark()`: `Mark()` allows the access to the value of the state variable *Place*.
- Notation for names of shared state variables: Names of shared places in SoC models are in uppercase.

6.2 Global Variables And Their Default Values

Möbius supports global variables within the models to allow the simple creation of studies by means of variable value assignments by so-called *Experiments*. Table 6.1 shows a description of the global variables of the SoC model. The table is partitioned into sections:

a) Failure Rates for Transient Physical Faults

FCR	Transient Failure Rate	Model Parameter
Host	10^4 FIT	host_fr
TISS-Port Manager + UNI	276 FIT	tiss_value_fr
TISS-Pulse Manager	319 FIT	tiss_temp_fr
RMA	404 FIT	rma_fr
TNA	144 FIT	tna_fr
Switch	10^3 FIT	switch_fr
Gateway Host	100 FIT	gateway_host_fr
Communication Controller	700 FIT	comm_controller_fr
Physical Link	10^4 FIT	tr_link_fr

b) Failure Rates for Permanent Physical Faults

FCR	Permanent Failure Rate	Model Parameter
Host	1 FIT	perm_trans_ratio * host_fr
TISS-Port Manager + UNI	0.03 FIT	perm_trans_ratio * tiss_value_fr
TISS-Pulse Manager	0.03 FIT	perm_trans_ratio * tiss_temp_fr
RMA	0.04 FIT	perm_trans_ratio * rma_fr
TNA	0.02 FIT	perm_trans_ratio * tna_fr
Switch	0.1 FIT	perm_trans_ratio * switch_fr
Gateway Host	0.01 FIT	perm_trans_ratio * gateway_host_fr
Communication Controller	0.07 FIT	perm_trans_ratio * comm_controller_fr
Physical Link	10^3 FIT	pm_link_fr

c) Error Rates for Design Faults

FCR	Design Error Rate	Model Parameter
Safety-Critical Application Computer	10 FIT	critical_host_design_fr
Non Safety-Critical Application Computer	10^3 FIT	non_critical_host_design_fr
RMA	100 FIT	rma_design_fr
Trusted Subsystem	1 FIT	tss_design_fr

d) Probabilities and Ratios

Parameter Description	Value	Model Parameter
Design Fault Correlation	0.1 %	design_fault_correlation
TSS Recovery Probability	80 %	tss_recovery_p
Permanent-to-Transient Fault Ratio	10^{-4}	perm_trans_ratio
Probability of correlated failures on both physical communication links	10%	correlated_channel_failure_p

e) Recovery Rates and the corresponding Mean Durations

Parameter Description	Rate	Duration	Model Parameter
Recovery Rate for Physical Link Failures	1800	2 s	physical_link_recovery_r
Recovery Rate for SoC Sub-Components	72000	50 ms	recovery_r
Repair Rate for SoC Components	0.1	10 hours	repair_r

Table 6.1: Global Variables and their Default Values

Sections a to c describe design and physical failure rates of SoC subcomponents with the first column stating the afflicted SoC subcomponent, the second column stating the failure rate and the third column stating the model parameter. If the introduction of a new parameter is not mandatory the model parameter is given as function (in C notation) of two other model parameters.

Section d depicts case probabilities and ratios.

Section e describes recovery rates and their corresponding mean durations.

6.3 Overview

To describe the whole TTSoC two different model formalisms are combined: The Rep & Join formalism for an abstract (composed) view of the SoC and the SAN formalism to describe the building blocks of the composed model in detail.

How the architecture is partitioned into atomic models, is shown by figure 6.1: RCU, TNA and RMA are combined together by the *Infrastructure* SAN. The *Gateway* SAN comprises the failure behavior of the gateway and the channels to the off-chip network, i.e., physical links and switches. Micro components are described, according to the safety-criticality of the corresponding application, either by a SAN for safety or non safety-critical micro components. Since the NoC is realized by the TNA and the micro component TISSes, the modeling of the NoC failure behavior is distributed over the micro component models, the *Gateway* model (which also possesses a TISS) and the *Infrastructure* model.

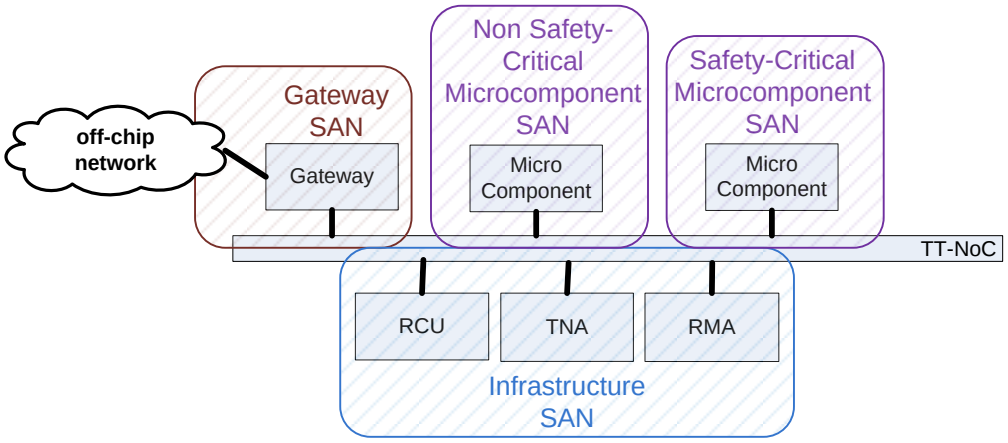


Figure 6.1: TTSoC Architecture Partitioning into Atomic Models

The overall SoC model is composed of SANs by means of model replication and combination. Figure 6.2 gives a schematic representation of the submodel relations.

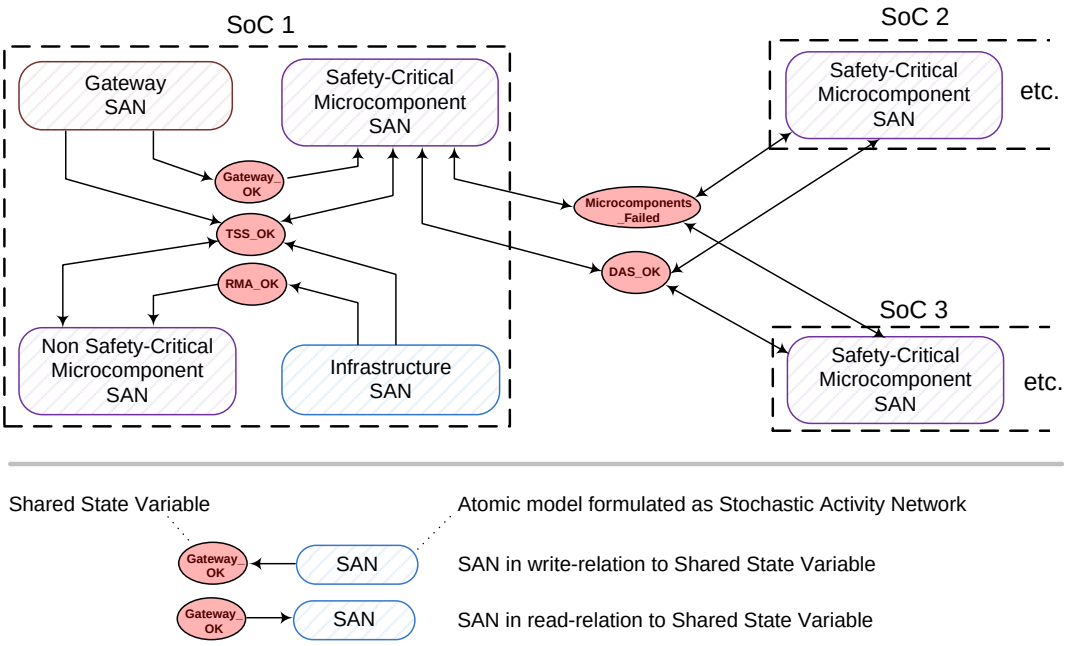


Figure 6.2: Composed Off-Chip TMR Model and its Shared State Variables

The left side of the picture holds a dashed box, representing SoC 1, which comprises all necessary atomic models to describe the failure behavior of an SoC. The data exchange between the submodels take place over shared state variables depicted as red circles, e.g., `TSS_OK` and `RMA_OK`. An arrow which points towards a shared state variable means a write-to-variable-relation, while an arrow towards a submodel means a read-by-SAN-relation. The combination of these relations is also possible.

Figure 6.2 shows an example for a composed model which consists of three SoCs where only SoC 1 is depicted completely. The states of the architectural services of each SoC are described by three shared state variables: `Gateway_OK`, `TSS_OK` and `RMA_OK`. Each of the SoCs has a set of these variables. Dependent on the requirements of a micro component, the associated micro component model shares some of these variables. The `Gateway_OK` is only shared between the *Gateway* model and the micro component models which depends on the state of the off-chip network. `RMA_OK` is only shared between the *Infrastructure* model and micro component models which describe the behavior of non safety-critical models. `TSS_OK` describes the state of the trusted sub-system and must be shared by all atomic models which belong to the same SoC.

Opposite to this SoC bounded variables, the figure also shows the use of application specific variables: `DAS_OK` and `MICROCOMPONENTS_FAILED`. Since each micro component belongs to a distributed application sub-system, each micro component

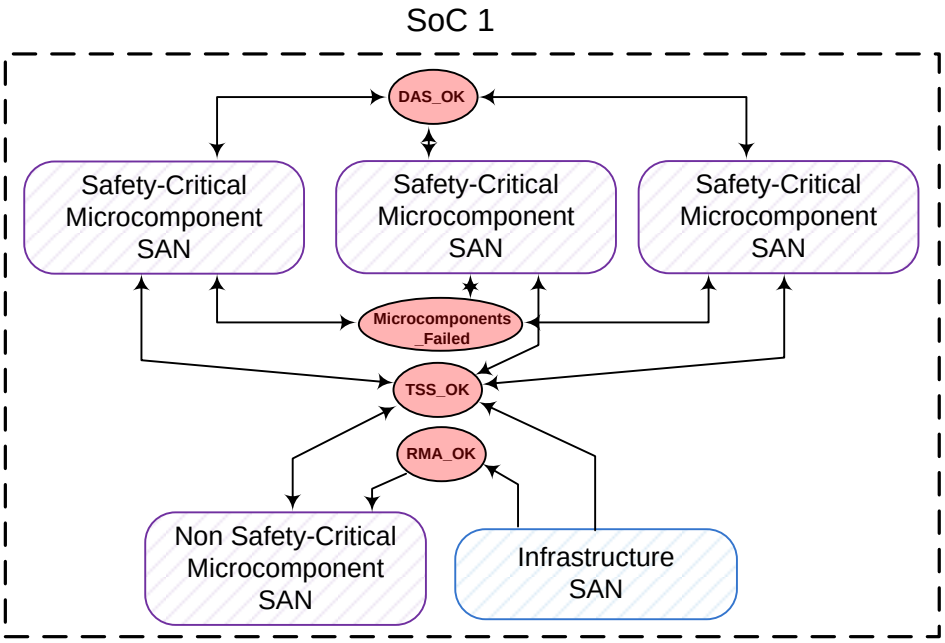


Figure 6.3: Composed On-Chip TMR Model and its Shared State Variables

model must have access to the according DAS state variable. In the example the variable is shared by micro component models belonging to different SoCs. If a micro component is part of a TMR (as in the example the three safety-critical micro component models), it must share the variable `MICROCOMPONENTS_FAILED` which states the number of failed replicas.

Figure 6.3 shows an example for a composed model similar to that depicted in figure 6.2 but with only one SoC. This SoC hosts three Safety-Critical Micro Component models, forming on-chip TMR. The big difference to the off-chip TMR example is that all three Safety-Critical Micro Component models now share the same architectural services state variable, i.e, `TSS_OK`.

Figure 6.4 shows the schematic of a composed model which combinates off-and on-chip TMR. To distribute the number of failed TMRs among all SoC models, the composed model possesses an additional shared variable, called `TMRS_FAILED`. If `TMRS_FAILED` is greater than one, the application is considered to be failed and the shared variable `DAS_OK` is cleared.

6.4 Common Constructs of SoC SANs

This section describes recurring structures within the atomic models of the SoC.

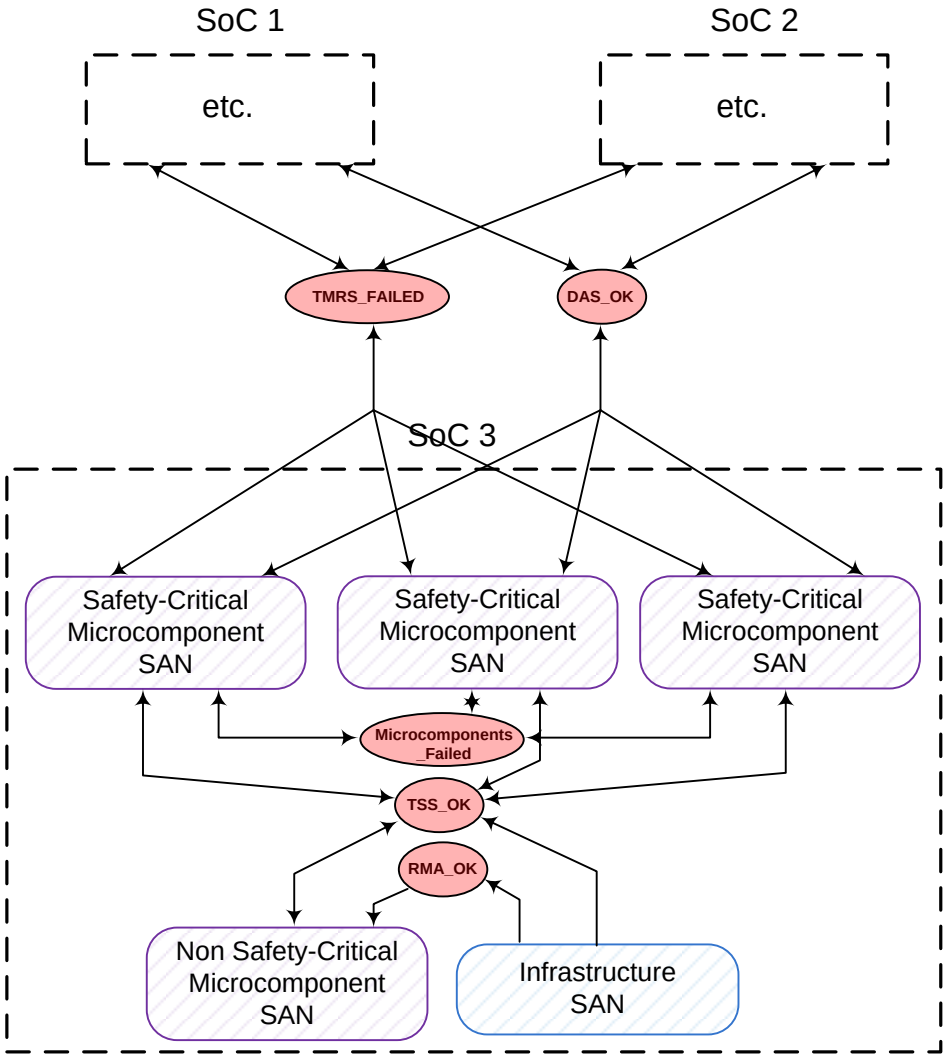


Figure 6.4: Composed Off-and-On-Chip TMR Model and its Shared State Variables

6.4.1 Combined Failure Rates and Case Probabilities

In some cases it is useful to combine the failure rates for transient and permanent failures of a component to reduce the number of activities. E.g. if $trans\_fr$ is the transient failure rate and $perm\_fr$ is the permanent failure rate of a component which has exponential distributed times-to-failure, the overall failure rate is given by $trans\_fr + perm\_fr$.

If the model should react differently for these two different failure classes, then this can be modeled by creating two cases for the overall activity: One for permanent

and one for transient failures. The case probabilities are calculated by the two following formulas:

$$\mathcal{P}\{transient\ failure\} = \frac{trans\_fr}{trans\_fr + perm\_fr}$$

$$\mathcal{P}\{permanent\ failure\} = \frac{perm\_fr}{trans\_fr + perm\_fr}$$

If $perm\_fr$ can be estimated from $trans\_fr$ over the permanent-to-transient-fault-ratio by $perm\_fr = trans\_fr * perm\_trans\_ratio$, then the overall failure rate equals $(1 + perm\_trans\_ratio) * transient\text{-}failure\text{-}rate$ and the case probabilities for a common activity are calculated by:

$$\mathcal{P}\{transient\ failure\} = \frac{1}{1 + perm\_trans\_ratio}$$

$$\mathcal{P}\{permanent\ failure\} = \frac{perm\_trans\_ratio}{1 + perm\_trans\_ratio}$$

6.4.2 Modeling of Temporal TISS Failures

The following structure is used in all atomic models. Figure 6.5 depicts the corresponding detail of a micro component model.

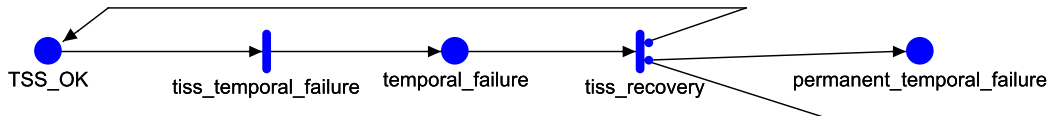


Figure 6.5: Detail of a SAN: TISS Failures and Recovery

Each micro component has an access point to the NoC by the pulse manager of its TISS. Since physical faults in the pulse manager can lead to temporal domain failures which disturb the on-chip communication, each additional micro component increases the failure rate of the whole NoC.

One model approach could consider a central activity within the *Infrastructure* model with an activation rate that equals the overall NoC failure rate. The drawback of this solution is, that the number of modeled pulse managers must be given explicitly as a model parameter which would be used for all instances of the *Infrastructure* model. As a consequence, this imposes a restriction on the reusability of the *Infrastructure* model within composed models of distributed applications: All SoCs would be restricted to contain the same number of micro components.

The solution used for this thesis allows a better composability of SoC subcomponents: Each micro component model describes failures and recoveries of its own pulse manager (by the `tiss_temporal_failure` and the `tiss_temporal_recovery` activities).

The TISS recovery-duration is described by the `tiss_recovery` activity which differs between two cases: The first case considers transient TISS failures, where the TISS down-time is given by the duration of an autonomous recovery. The second case considers permanent TISS failures which only can be removed by an external maintenance action. The maintenance time is described by the `system_repair` activity. The case probabilities of `tiss_recovery` are calculated as described in the following paragraph:

Case Probabilities of `tiss_recovery`: Since a distributed system build upon the TTSoC architecture offers no central element which allows the external reset of an SoC, the recovery depends on the type of the causing fault, i.e., if the resulting failure is limited in time. E.g., if a SEU alters some essential configuration data of the TSS, then the SoC fails permanently while an SEU striking some volatile application memory, would cause only a limited number of faulty messages. For this reason the case probabilities of the `tiss_recovery` activity takes into account that the probability of a recovery after a transient fault ($\boldsymbol{\mathcal{P}}\{\textit{TSS recovery after fault}\}$) can be smaller than one:

$$\boldsymbol{\mathcal{P}}\{\textit{transient failure}\} = \frac{1}{1 + \textit{perm\_trans\_ratio}} \textit{ (as described in sub-section 6.4.1)}$$

$$\boldsymbol{\mathcal{P}}\{\textit{recovery}\} = \boldsymbol{\mathcal{P}}\{\textit{no TSS recovery after fault}\} = 1 - \boldsymbol{\mathcal{P}}\{\textit{repair}\}$$

$$\boldsymbol{\mathcal{P}}\{\textit{repair}\} = \boldsymbol{\mathcal{P}}\{\textit{transient failure}\} * \boldsymbol{\mathcal{P}}\{\textit{TSS recovers after transient fault}\}$$

6.4.3 Model Reactions on TSS and Gateway Failures

The following structures are used in all micro component models and in the *Infrastructure* model. Figure 6.6 depicts the corresponding detail of a micro component model.

The functionality of each micro component depends on the TSS and in case of distributed applications also on the gateway. Therefore the states of TSS and the gateway are distributed by the shared state variables `TSS_OK` and `GATEWAY_OK`. Their values are used within the input-gateway `infrastructure_not_ok` which enables the activity `infrastructure_failed`. So in case of a TSS or gateway failure the function of the output-gateway `job_is_failing` is executed, which clears the value of `microcomponent_ok`, increments the value of `MICROCOMPONENTS_FAILED` and computes the new state of `DAS_OK`. After this, the place `infrastructure_failed`

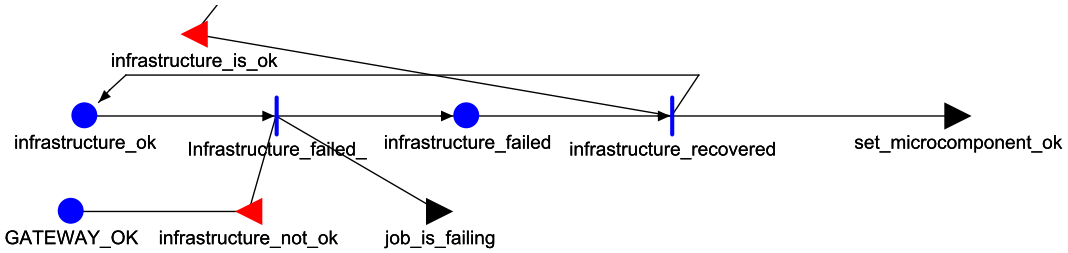


Figure 6.6: Detail of a SAN: Reactions on Infrastructure Failures and Recovery

contains a mark. The enable of the activity `infrastructure_recovered` now only depends on the input-gateway `infrastructure_is_ok` which is activated if the gateway and TSS are considered to be valid. After activation the output-gateway `set_microcomponent_ok` resets the micro component model.

6.5 Atomic Models

6.5.1 Infrastructure - SAN

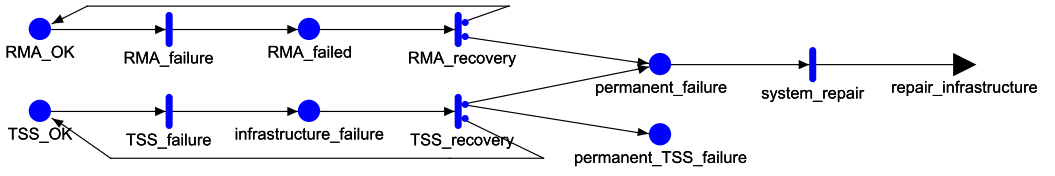


Figure 6.7: Infrastructure SAN

A collective model describes the failures and recoveries of TNA, RMA and RCU. The model summarizes the failure rates of TNA and RCU within the `TSS_failure` activity, while RMA failures are modeled by the `RMA_failure` activity, see figure 6.7. Tables 6.2 to 6.4 give the complete parameter set for the *Infrastructure* SAN.

The `RMA_failure` activity summarizes the failure rates for failures caused by transient and permanent physical faults and by design faults. The `RMA_recovery` activity possesses two cases: The first one considers transient failures and refills the place `RMA_OK` which represents the RMA recovery. The second case leads to the `permanent_failure` place which indicates that only an external maintenance action can recover the RMA. The case probabilities are given by the following

formulas:

$$\begin{aligned}\boldsymbol{\mathcal{P}}\{case\ 1\} &= \boldsymbol{\mathcal{P}}\{RMA\ recovery\ after\ fault\} = \boldsymbol{\mathcal{P}}\{transient\ failure\} \\ &= \frac{trans\_fr}{trans\_fr + perm\_fr}\end{aligned}$$

$$\begin{aligned}\boldsymbol{\mathcal{P}}\{case\ 2\} &= \boldsymbol{\mathcal{P}}\{no\ RMA\ recovery\ after\ fault\} = \boldsymbol{\mathcal{P}}\{permanent\ failure\} \\ &= \frac{perm\_fr}{trans\_fr + perm\_fr}\end{aligned}$$

TSS failures are similar modeled as RMA failures but with a additional factor to consider not recoverable failures caused by transient faults. So the case probabilities of the recovery activity are:

$$\begin{aligned}\boldsymbol{\mathcal{P}}\{case\ 1\} &= \boldsymbol{\mathcal{P}}\{recovery\} \\ &= \boldsymbol{\mathcal{P}}\{transient\ fault\} * \boldsymbol{\mathcal{P}}\{\ TSS\ recovery\ after\ fault\}\end{aligned}$$

$$\begin{aligned}\boldsymbol{\mathcal{P}}\{case\ 2\} &= \boldsymbol{\mathcal{P}}\{repair\} \\ &= \boldsymbol{\mathcal{P}}\{transient\ fault\} * (1 - \boldsymbol{\mathcal{P}}\{\ TSS\ recovery\ after\ fault\}) \\ &\quad + \boldsymbol{\mathcal{P}}\{permanent\ fault\}\end{aligned}$$

A detailed explanation for the formulas can be found in sub-section 6.4.2, paragraph *Case Probabilities of* `tiss_recovery`:.

The repair duration after a permanent failure of TNA, RMA or RCU is expressed by the `system_repair` activity, which resets the model by triggering `repair_infrastructure` (output gateway) after completion.

Place Names	Initial Markings
RMA_OK	1
RMA_failed	0
TSS_OK	1
infrastructure_failure	0
permanent_TSS_failure	0
permanent_failure	0

Table 6.2: Infrastructure SAN - Places

Output Gate	Function

repair_infrastructure	RMA_failed->Mark() = 0;
	RMA_OK->Mark() = 1;
	permanent_failure->Mark() = 0;
	
	if (permanent_TSS_failure->Mark())
	{
	permanent_TSS_failure->Mark() = 0;
	TSS_OK->Mark() = 1;
	}

Table 6.3: Infrastructure SAN - Output Gateways

Activity	Exponential Rate	Cases
RMA_failure	rma_fr * (1 + perm_trans_ratio) + rma_design_fr	
RMA_recovery	recovery_r	**case 1** 1 - perm_trans_ratio **case 2** perm_trans_ratio
TSS_failure	tss_design_fr // TSS design failure + (1 + perm_trans_ratio) * (tna_fr // TNA physical failure + tiss_temp_fr // TNA-DLL physical failure + tiss_temp_fr // RMA-DLL physical failure)	
TSS_recovery	recovery_r	**case 1** 1 - (1 - perm_trans_ratio) * tss_recovery_p **case 2** (1 - perm_trans_ratio) * tss_recovery_p
system_repair	repair_r	

Table 6.4: Infrastructure SAN - Activities

6.5.2 Gateway - SAN

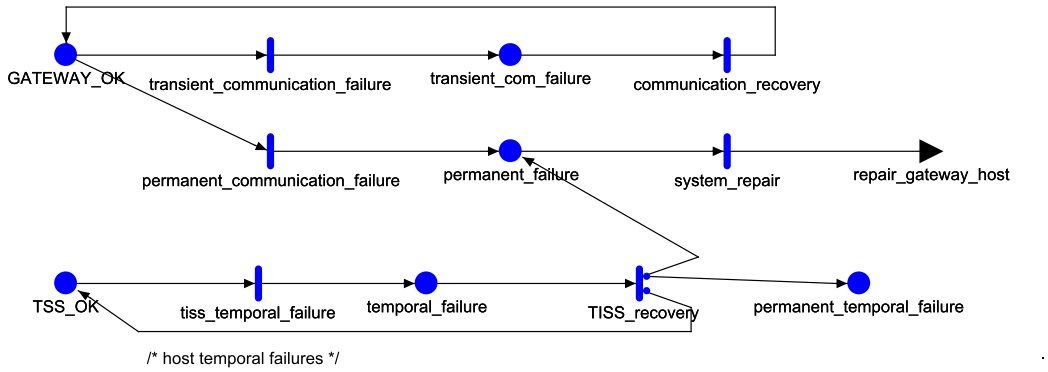


Figure 6.8: Gateway SAN

Place Names	Initial Markings
GATEWAY_OK	1
permanent_failure	0
TSS_OK	1
permanent_temporal failure	0
temporal_failure	0
transient_com_failure	0

Table 6.5: Gateway SAN - Places

The *Gateway* model (figure 6.8) describes the SoC gateway together with its interconnection to the off-chip network, i.e., the failure behavior of the physical links to the star-coupled switches, the communication controllers, the gateway host and the TISS (described in section 4.6). The model parameters are depicted in the tables 6.5 to 6.7.

A plain gateway model which maps one failure-cause to one SAN activity would create much computing time during simulation. The reasons for this are on the one hand the high failure rates of the two redundant channels and on the other hand the high number of activities in total within a composed distributed model. The high number of activities arises as a result of the redundant structures within a plain *Gateway* model and of the necessity to incorporate one *Gateway* model per SoC within a composed model of a distributed system.

To reduce the number of activities with high firing rates and consequently the simulation execution time, it is possible to combine failure and repair rates of the

Activity	Exponential Rate	Case Probabilities
system_repair	`repair_r`	-

TISS_recovery	`recovery_r`	**case 1**
		`1.0 - ( 1.0 - perm_trans_ratio) * tss_recovery_p`
		case 2
		`(1.0 - perm_trans_ratio) * tss_recovery_p`

tiss_temporal_failure	`tiss_temp_fr *(1 + perm_trans_ratio)`	
communication_recovery	`min(recovery_r, physical_link_recovery_r)`	
permanent_communication_failure	see sub-section *Permanent Gateway Failures*	
transient_communication_failure	see sub-section *Transient Gateway Failures*	

Table 6.6: Gateway SAN - Activities

Output Gate	Function

repair_gateway_host	`GATEWAY_OK->Mark() = 1;`
	`transient_com_failure->Mark() = 0;`
	`temporal_failure->Mark() = 0;`
	`permanent_failure->Mark() = 0;`
	
	`if (permanent_temporal_failure->Mark())`
	`{`
	`permanent_temporal_failure->Mark() = 0;`
	`TSS_OK->Mark() = 1;`
	`}`

Table 6.7: Gateway SAN - Output Gateways

redundant *Gateway* subcomponents by exploiting the mathematical benefits of exponential distributed activity times.

The `transient_communication_failure` activity models transient failures in the value domain of the whole gateway, while `permanent_communication_failure` does a similar job for permanent failures: The activities models the failures of the gateway host, the TISS and of both redundant channels (each comprising a communication controller, a physical link and a switch).

The `communication_recovery` activity models the recovery duration after transient link-, switch- and communication controller failures. To reduce the activity complexity, the rate is chosen to be the minimum of the link-recovery-rate (for recoveries after interferences on the physical link) and the switch/communication controller recovery rate. Therefore in some cases, the model uses a longer mean-time-to-recovery after gateway failures. But this circumstance carries no weight

for the system evaluation, since the two different recovery rates are assumed to be of a similar order of magnitude. The `system_repair` activity describes the repair duration after permanent failures.

Since TISS failures in the temporal domain affect the whole on-chip communication, their occurrence and their removal are covered by the activities: `tiss_temporal_failure` and `TISS_recovery`. A detailed explanation can be found in sub-section 6.4.2.

The following sub-sections describe the *Gateway* model activities and the calculation of their activation-rates. The theoretical backgrounds for the following equations are described in 2.2 (*Mathematical Concepts*).

Transient Gateway Failures

Transient gateway failures are modeled by the `transient_comm_failure` activity. Its overall failure rate is calculated by formula 6.13 which is build upon the formulas 6.1 to 6.12.

Combined Availability for a Communication Controller and a Switch: The combined transient failure rate for a communication controller and a switch is calculated by formula 6.1.

$$tr\_controller\_fr = comm\_controller\_fr + switch\_fr \tag{6.1}$$

The permanent failure rate can be estimated by multiplying the transient failure rate with the factor *perm_trans_ratio*. In sum the total (i.e., permanent + transient) failure rate for a communication controller and a switch is given by formula 6.2. Since all activity times are assumed to be exponential distributed, the failure rate equals the reciprocal MTTF.

$$controller\_fr = \frac{1}{MTTF_{\text{controller+switch}}} = tr\_controller\_fr * (1 + perm\_trans\_ratio) \tag{6.2}$$

Depending on whether the failure was caused by a transient or a permanent fault, the communication controller and the switch have different down-times:

$$\text{time-to-restart} = \begin{cases} \text{recovery-duration} & : \quad \text{case probability} = \frac{1}{1+perm\_trans\_ratio} \\ \text{repair-duration} & : \quad \text{case probability} = \frac{perm\_trans\_ratio}{1+perm\_trans\_ratio} \end{cases} \tag{6.3}$$

The corresponding MTTR is defined as the expectancy value for the time-to-restart in equation 6.3 and is calculated by formula 6.4:

$$MTTR_{controller+switch} = \frac{recovery\text{-}duration + perm\_trans\_ratio * repair\text{-}duration}{1 + perm\_trans\_ratio} \tag{6.4}$$

Applying preliminary findings on the common formula for the availability (formula 6.5) results in the combined availability for a communication controller and a switch, see formula 6.6.

$$Availability = \frac{MTTF}{MTTF + MTTR} \tag{6.5}$$

$$\begin{aligned}\boldsymbol{\mathcal{A}}\{controller+switch\} = 1 \;/\; (1 \\ &+ tr\_controller\_fr * recovery\text{-}duration \\ &+ tr\_controller\_fr * repair\text{-}duration * perm\_trans\_ratio \\ &)\end{aligned} \tag{6.6}$$

Physical Link Availability: The calculation of the availability for a physical link succeeds just like the availability calculation for the communication controller and the switch, but with the physical link interference rate as transient failure rate, pm_link_fr for permanent link failures and the interference duration instead of the recovery duration. For later calculations the link availability does not take correlated physical link failures (where both links are affected coincidently from an interference) into account since these are considered directly in the final transient gateway failure rate in formula 6.13. To eliminate correlated physical link failures from the availability calculation , tr_link_fr is multiplied with the probability that a transient fault does not hit both links coincidently, $\boldsymbol{\mathcal{P}}\{uncorrelated\ link\ failure\}$.

$$tr\_link\_fr_u = tr\_link\_fr * \boldsymbol{\mathcal{P}}\{uncorrelated\ link\ failure\} \tag{6.7}$$

$$MTTF_{physical\ link} = \frac{1}{pm\_link\_fr + tr\_link\_fr_u} \tag{6.8}$$

$$\text{time-to-restart} = \begin{cases} \text{interference-duration} & : \text{ case probability} = \frac{tr\_link\_fr_u}{pm\_link\_fr + tr\_link\_fr_u} \\ \text{repair-duration} & : \text{ case probability} = \frac{pm\_link\_fr}{pm\_link\_fr + tr\_link\_fr_u} \end{cases} \tag{6.9}$$

The corresponding MTTR is defined as the expectancy value for the time-to-restart in equation 6.9 and is calculated by formula 6.10:

$$\begin{aligned} MTTR_{physical\ link} = & \frac{tr\_link\_fr_u * interference\text{-}duration}{pm\_link\_fr + tr\_link\_fr_u} \\ & + \frac{pm\_link\_fr * repair\text{-}duration}{pm\_link\_fr + tr\_link\_fr_u} \end{aligned} \tag{6.10}$$

Applying preliminary findings on the common formula for the availability (formula 6.5) results in the availability for a physical link, see formula 6.11.

$$\begin{aligned} \boldsymbol{\mathcal{A}}\{physical\ link\} = & 1\ /\ (\ 1 + pm\_link\_fr * repair\text{-}duration \\ & + tr\_link\_fr_u * interference\text{-}duration) \end{aligned} \tag{6.11}$$

Combined Failure Rate for Redundant Channels: Having the availabilities of a physical link and of the combination of communication controller and switch, it is possible to calculate the overall failure rate for two redundant channels. From the equation $\boldsymbol{\mathcal{P}}\{channel\ failed\} = 1 - \boldsymbol{\mathcal{A}}\{channel\}$ and $\boldsymbol{\mathcal{A}}\{channel\} = \boldsymbol{\mathcal{A}}\{physical\ link\} * \boldsymbol{\mathcal{A}}\{controller{+}switch\}$ follows:

$$tr\_channels\_fr = \ (1 - \boldsymbol{\mathcal{A}}\{channel\}) * (tr\_contr\_fr + tr\_link\_fr_u) \tag{6.12}$$

Overall Failure Rate for Transient Gateway Failures: Finally, the gateway failure rate for transient failures is given by the sum of formula 6.12, the failure rates for gateway host, TISS failures (for TISS failures in the value domain; temporal failures are considered by an other activity) and correlated physical link failures ($tr\_link\_fr * \boldsymbol{\mathcal{P}}\{correlated\ link\ failure\}$):

$$\begin{aligned} transient\_gateway\_fr = & \ host\_design\_error\_rate \\ & + gateway\_host\_fr \\ & + TISS\_value\_fr \\ & + tr\_link\_fr * \boldsymbol{\mathcal{P}}\{correlated\ link\ failure\} \\ & + tr\_channels\_fr \end{aligned} \tag{6.13}$$

Permanent Gateway Failures

Permanent gateway failures are modeled by the `permanent_comm_failure` activity. The overall failure rate is calculated by formula 6.16, which is build upon the formulas 6.14 to 6.15.:

Combined Permanent Failure Rate for Redundant Channels: The combined, permanent failure rate for a communication controller and a switch is calculated by multiplying the failure rate for transient failures with the factor $perm\_trans\_ratio$:

$$pm\_controller\_fr = tr\_controller\_fr * perm\_trans\_ratio \tag{6.14}$$

With the availabilities of the physical link and of the combination, communication controller and switch (see formulas 6.11 and 6.6), it is possible to calculate the overall failure rate for permanent failures of two redundant channels.

$$pm\_channels\_fr = (1 - \boldsymbol{\mathcal{A}}\{channel\}_u) * (pm\_contr\_fr + pm\_link\_fr) \tag{6.15}$$

Overall Failure Rate for Transient Gateway Failures: Finally, the gateway failure rate for permanent failures is given by the sum of formula 6.15, the failure rates for gateway host and TISS failures (for TISS failures in the value domain; temporal failures are considered by an other activity):

$$\begin{aligned} permanent\_gateway\_fr = {} & pm\_channels\_fr \\ & + perm\_trans\_ratio * (gateway\_host\_fr + TISS\_value\_fr) \end{aligned} \tag{6.16}$$

6.5.3 Safety-Critical Micro Component in TMR - SAN

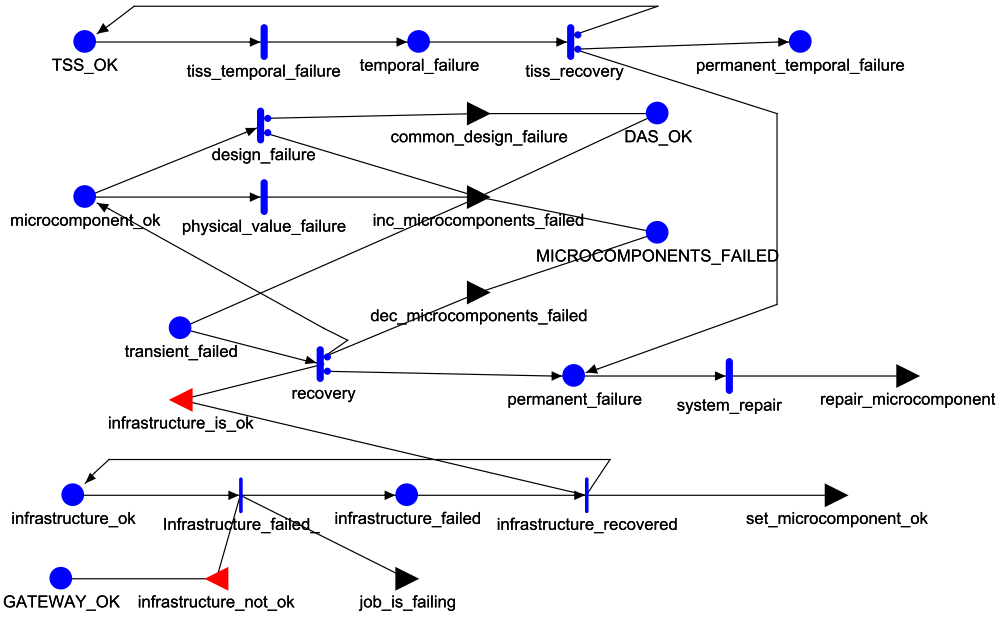


Figure 6.9: SAN for Micro Components in TMR

The SAN model depicted in figure 6.9 is considered to work within a composed TMR model. For this purpose the model shares a state variable to count the number of failed replicas and a state variable to set or reset the state of the corresponding DAS. To allow reactions on state changes of gateway and TSS, the model shares also the state variables `GATEWAY_OK` and `TSS_OK`. The SAN model parameters are depicted in the tables 6.5 to 6.7.

The model covers the following events:

- Physical Faults in the TISS-Pulse Manager:
  - Physical faults in the pulse manager can lead to temporal domain failures which can disturb the whole NoC and therefore must be signaled to all other submodels, associated with the same SoC. This is done by clearing the value of the shared place `TSS_OK`. A detailed explanation can be found in subsection 6.4.2 (*Modeling of Temporal TISS Failures*).
  - Also the model must react to temporal TISS–failures signaled from other models (e.g. *Infrastructure*, *Gateway* and other micro component models). Since, this model structure appears in all other models, too, this details got its own

subsection, see 6.4.3 (*Model Reactions on TSS and Gateway Failures*).

- Gateway Failures: The model gets the state of the off-chip interconnection by the state variable `GATEWAY_OK` shared with the *Gateway* model. If the gateway fails, the micro component would have no connection to the off-chip network. Within the model, this event is treated as if the micro component itself fails. For more details see sub-section 6.4.3. If the micro component model is considered to be independent of the off-chip network, then this case can be modeled within the encompassing composed model by not sharing the place `GATEWAY_OK`.

- Physical Faults in Host and TISS-Port Manager: Failures caused by such faults can only yield to failures in the value domain which only affects the involved micro component and the application. Since the modeled micro component is considered to be part of a TMR, the RCU detects value domain failures and resets the micro component if necessary. So in the model transient faults always lead to recoverable failures (see activity `recovery`) and permanent failures need maintenance actions (modeled by the activity `system_repair`) .

- Application Computer Design Faults: Failures caused by such faults are modeled by the `design_failure` activity which possesses two cases: The first case considers a common application computer fault in all micro components belonging to the same TMR. This case leads directly to a DAS failure. The second case considers only micro component local design faults which lead to a local micro component failure. The calculation of the activity rate and the case probabilities are done in paragraph *Failure Rate and Case Probabilities for Design Faults:*.

If one of the described activities fires, the place `MICROCOMPONENTS_FAILED` is incremented by one, except for common application computer design faults, which set `MICROCOMPONENTS_FAILED` to 3. Whenever a change of this place occurs, the model solver checks if $MICROCOMPONENTS\_FAILED \leq 1$, which is the requirement for TMR to work properly. The first time the number of failed micro components is to high, the place `DAS_OK` is permanently set to 0, which means that the whole DAS is considered to be failed because one of the ECUs of the distributed application system failed. In this way, by setting `DAS_OK` permanently to zero after the first DAS failure, the Möbius simulator can evaluate the MTTF.

Failure Rate and Case Probabilities for Design Faults: The `design_failure` activity combines the failure rates of failures caused be local and

common (i.e., among all micro components of the same TMR) design faults. The probability that a design fault within one micro component is not as well within other micro components of the TMR is given by the grade of design diversity $\boldsymbol{\mathcal{P}}\{\textit{common design fault}\}$ (model parameter: `design_fault_correlation`).

Therefore, the failure rate for a micro component in TMR which fails because of a local design fault is given by:

$$\textit{design-failure-rate}_{uncorr} = \textit{critical\_host\_design\_fr} * (1 - \boldsymbol{\mathcal{P}}\{\textit{common design fault}\})$$

and the failure rate for common design failures is given by:

$$\textit{design-failure-rate}_{corr} = \textit{critical\_host\_design\_fr} * \boldsymbol{\mathcal{P}}\{\textit{common design fault}\}$$

To spare an atomic model which would have only the purpose to model common micro component failures of a TMR, the value of the latter formula can be distributed amongst all TMR micro component models. This is done by adding the following formula to each local design failure rate of a micro component within TMR:

$$\frac{\textit{design-failure-rate}_{corr}}{\textit{Number-of-running-micro-components}}$$

So in sum, the rate of the `design_failure` activity is calculated by:

$$\textit{design-failure-rate} = \frac{\textit{design-failure-rate}_{corr}}{\textit{Number-of-running-micro-components}} + \textit{design-failure-rate}_{uncorr}$$

As explained in 6.4.1 (*Combined Failure Rates and Case Probabilities*), the case probabilities for the `design_failure` activity which combines the failure rates of common and local design faults can be calculated as following:

$$\boldsymbol{\mathcal{P}}\{\textit{case 1}\} = \frac{\textit{design-failure-rate}_{corr}}{\textit{design-failure-rate} * \textit{Number-of-running-micro-components}}$$

$$\boldsymbol{\mathcal{P}}\{\textit{case 2}\} = \frac{\textit{design-failure-rate}_{uncorr}}{\textit{design-failure-rate}}$$

Place Names	Initial Markings
DAS_OK	1
GATEWAY_OK	1
MICROCOMPONENTS_FAILED	0
TSS_OK	1
infrastructure_failed	0
infrastructure_ok	1
microcomponent_ok	1
permanent_failure	0
permanent_temporal_failure	0
temporal_failure	0
transient_failed	0

Table 6.8: SAN for Micro components in TMR - Places

Activity	Exponential Rate	Case Probabilities
design_failure	`critical_host_design_fr / (3 - MICROCOMPONENTS_FAILED->Mark())`	**case 1** `design_fault_correlation` **case 2** `1.0 - design fault correlation`
physical_value_failure	`(tiss_value_fr + host_fr) * (1 + perm_trans_ratio)`	
recovery	`recovery_r`	**case 1** `1.0 / (1.0 - perm_trans_ratio)` **case 2** `perm_trans_ratio / (1.0 - perm_trans_ratio)`
system_repair	`repair_r`	
tiss_recovery	`recovery_r`	**case 1** `tss_recovery_p / (1 + perm_trans_ratio)` **case 2** `(perm_trans_ratio + 1 - tss_recovery_p) / (1 + perm_trans_ratio)`
tiss_temporal_failure	`tiss_temp_fr * (1 + perm_trans_ratio)`	

Table 6.9: SAN for Micro components in TMR - Activities

Input Gate	Predicate
infrastructure_is_ok	`TSS_OK->Mark() && GATEWAY_OK->Mark()`
infrastructure_not_ok	`( !TSS_OK->Mark() || !GATEWAY_OK->Mark() ) && DAS_OK->Mark()`

Table 6.10: SAN for Micro components in TMR - Input Gateways

Output Gate	Function
common_design_failure	`MICROCOMPONENTS_FAILED->Mark() = 3;` `DAS_OK->Mark ()  = 0;`
dec_microcomponents_failed	`MICROCOMPONENTS_FAILED->Mark () --;`

inc_microcomponents_failed	`MICROCOMPONENTS_FAILED->Mark()++;`
	
	`if (MICROCOMPONENTS_FAILED->Mark() > 1)`
	`{`
	`  DAS_OK->Mark()  = 0;`
	`}`
	`else  transient_failed->Mark() = 1;`

job_is_failing	`if (microcomponent_ok->Mark())`
	`{`
	`  microcomponent_ok->Mark() = 0;`
	`  MICROCOMPONENTS_FAILED->Mark()++;`
	
	`  if (MICROCOMPONENTS_FAILED->Mark() > 1)`
	`  {`
	`  DAS_OK->Mark() = 0;`
	`  }`
	`}`

repair_microcomponent	`permanent_failure->Mark() = 0;`
	
	`if (!microcomponent_ok->Mark() && DAS_OK->Mark())`
	`{`
	`  transient_failed->Mark()  = 0;`
	`  microcomponent_ok->Mark() = 1;`
	`  MICROCOMPONENTS_FAILED->Mark()--;`
	`}`
	`if (permanent_temporal_failure->Mark())`
	`{`
	`  permanent_temporal_failure->Mark() = 0;`
	`  TSS_OK->Mark() = 1;`
	`}`

set_microcomponent_ok	`if (  !microcomponent_ok->Mark()`
	`  && !transient_failed->Mark()`
	`  )`
	`{`
	`  microcomponent_ok->Mark() = 1;`
	`  MICROCOMPONENTS_FAILED->Mark()--;`
	`}`

Table 6.11: SAN for Micro components in TMR - Output Gateways

6.5.4 Safety-Critical Micro Component within a 3-of-4 ensemble - SAN

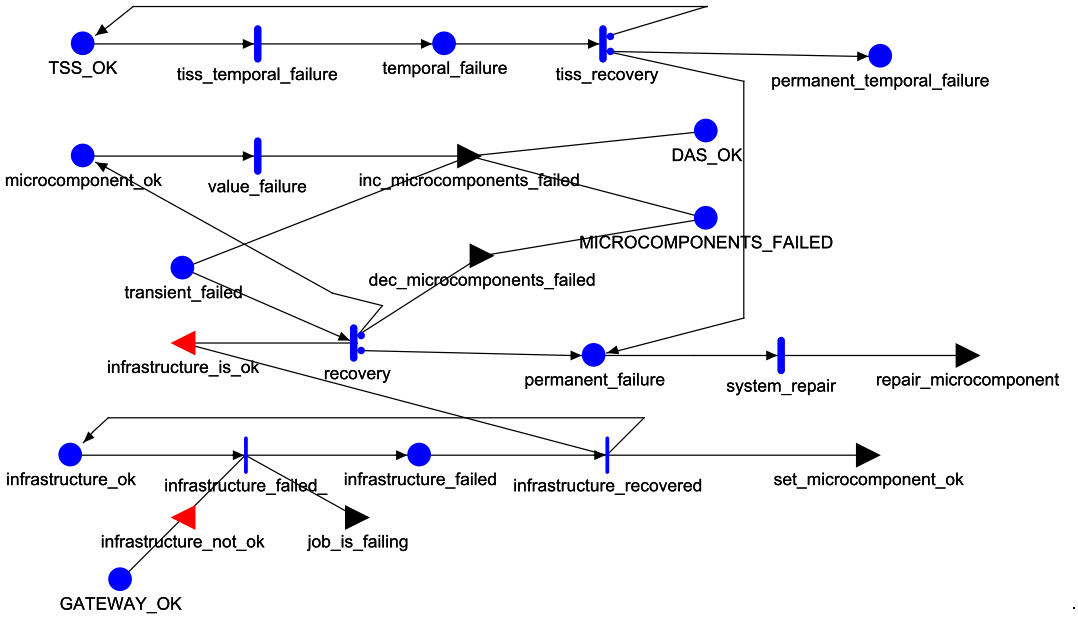


Figure 6.10: SAN for Micro Components in 3-of-4 Ensembles

This section describes the model of a micro component which works within an ensemble of four, not replicated, micro components, where the system is considered to be valid if at least three of four micro components are running. Each of the micro component application computers is assumed to have its own independent design, i.e., no common design failures must be considered within the model. In section 6.6.1 this micro component model is used to describe the brake system of an automotive system.

Figure 6.10 depicts the SAN of the micro component. The SAN is nearly the same as for the micro component model for TMR (sub-section 6.5.3), but differs in the following point:

- No Common Design Faults: The design failure rate for a single safety-critical application computer is added to the overall failure rate for failures in the value domain, which are modeled by the `value_failure` activity. Table 6.12 shows the parameter for the `value_failure` activity.

Activity	Exponential Rate
value_failure	`(tiss_value_fr + host_fr) * (1 + perm_trans_ratio) + critical_host_design_fr`

Table 6.12: Parameters of Activity `value_failure`

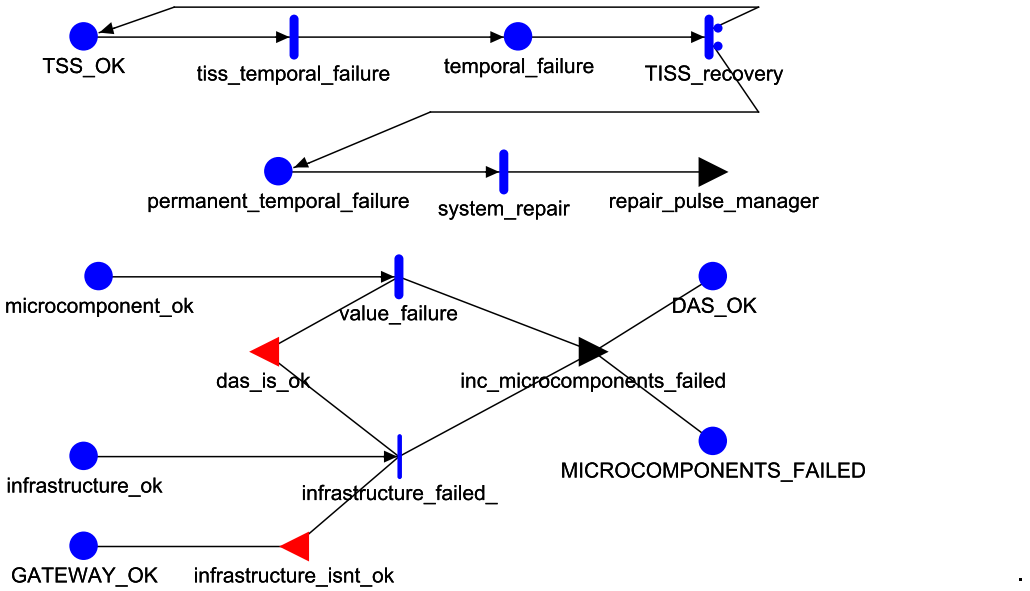


Figure 6.11: SAN for (simple) Safety Critical Micro Components

6.5.5 Safety-Critical Micro Component - SAN

The following SAN describes a micro component with a safety-critical application computer which forms no kind of an error containment region together with other micro components, i.e., the corresponding application relies completely on the functionality of the micro component. This model makes no sense for the modeling of safety-critical applications but establishes a base for comparisons between different TMR approaches, as presented in chapter 6.6.2. The graphical representation of the SAN is depicted in figure 6.11 and the model parameters are depicted in the tables 6.13 to 6.15.

The model considers two failure classes:

- Temporal Domain Failures: Like all other micro component SANs this model also contributes to the NoC failure rate, modeled by the activities `tiss_temporal_failure` and `tiss_recovery`. A detailed description

can be found in sub-section 6.4.2.

- Value Domain Failures: The `value_failure` activity comprises failures of host and port manager caused by transient and permanent faults. Additionally, if the place `GATEWAY_OK` is shared with the *Gateway*, then the model also reacts on gateway failures.

After each failure, the state variable for the distributed application, `DAS_OK`, is (permanently) set to zero, which indicates a DAS failure.

Place Names	Initial Markings
DAS_OK	1
GATEWAY_OK	1
TSS_OK	1
microcomponent_ok	1
permanent_temporal_failure	0
temporal_failure	0

Table 6.13: SAN for (simple) Safety Critical Micro Components - Places

Output Gate	Function
repair_pulse_manager	`TSS_OK->Mark() = 1;`

Table 6.14: SAN for (simple) Safety Critical Micro Components - Output Gateways

Activity	Exponential Rate	Case Probabilities
value_failure	`(tiss_value_fr + host_fr) * (1 + perm_trans_ratio) + critical_host_design_fr`	
system_repair	`repair_r`	
tiss_recovery	`recovery_r`	**case 1** `tss_recovery_p / (1 + perm_trans_ratio)` **case 2** `(perm_trans_ratio + 1 - tss_recovery_p) / (1 + perm_trans_ratio)`
tiss_temporal_failure	`tiss_temp_fr * (1 + perm_trans_ratio)`	

Table 6.15: SAN for (simple) Safety Critical Micro Components - Activities

6.5.6 Non Safety-Critical Micro Component SAN

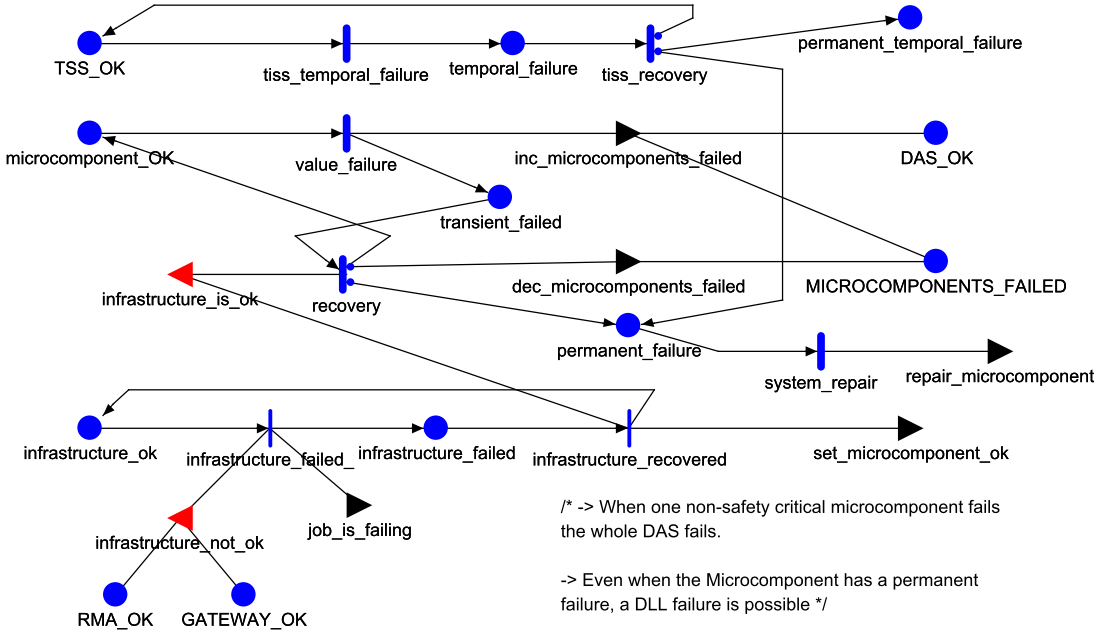


Figure 6.12: SAN for Non Safety-Critical Micro Components

This sub-section describes the SAN model for micro components with non safety-critical application computers, as depicted in figure 6.12. It is considered to be no part of an error containment unit and so the corresponding non safety-critical application is considered to fail if one of its non safety-critical micro components fail.

The SAN is similar to the micro component model for TMR (sub-section 6.5.3), but differs in the following points:

- No Common Design Faults: The design failure rate for a single non safety-critical application computer is combined with the failure rate for value domain failures caused by physical faults. Value domain failures are modeled by the `value_failure` activity. Table 6.12 shows the parameter for the `value_failure` activity.
- Reactions on RMA Failures: The model can share the place `RMA_OK` with the *Infrastructure* model and considers a micro component failure if the RMA is down.
- The model never reaches a stable state: In contrast to the micro component model for TMR, after the first DAS failure the model continues to model value domain failures and recoveries. In this way the

model enables the evaluation of the availability of the corresponding non safety-critical DAS.

The SAN model parameters are depicted in the tables 6.16 to 6.19.

Place Names	Initial Markings
RMA_OK	1
DAS_OK	1
GATEWAY_OK	1
MICROCOMPONENTS_FAILED	0
TSS_OK	1
infrastructure_failed	0
infrastructure_ok	1
microcomponent_ok	1
permanent_failure	0
permanent_temporal_failure	0
temporal_failure	0
transient_failed	0

Table 6.16: SAN for Non Safety-Critical Micro Components - Places

Activity	Exponential Rate	Case Probabilities
value_failure	(tiss_value_fr + host_fr) * (1 + perm_trans_ratio) + non_critical_host_design_fr	
recovery	recovery_r	**case 1** 1.0 / (1.0 - perm_trans_ratio) **case 2** perm_trans_ratio / (1.0 - perm_trans_ratio)
system_repair	repair_r	
tiss_recovery	recovery_r	**case 1** tss_recovery_p / (1 + perm_trans_ratio) **case 2** (perm_trans_ratio + 1 - tss_recovery_p) / (1 + perm_trans_ratio)
tiss_temporal_failure	tiss_temp_fr * (1 + perm_trans_ratio)	

Table 6.17: SAN for Non Safety-Critical Micro Components - Activities

Input Gate	Predicate

infrastructure_is_ok	`TSS_OK->Mark()`
	`&& GATEWAY_OK->Mark()`
	`&& RMA_OK->Mark()`

infrastructure_not_ok	`!TSS_OK->Mark()`
	`|| !GATEWAY_OK->Mark()`
	`|| ! RMA_OK->Mark()`

Table 6.18: SAN for Non Safety-Critical Micro Components - Input Gateways

Output Gate	Function
dec_microcomponents_failed	`MICROCOMPONENTS_FAILED->Mark()--;`
inc_microcomponents_failed	`MICROCOMPONENTS_FAILED->Mark()++;` `DAS_OK->Mark() = 0;`

job_is_failing	`if (microcomponent_OK->Mark())`
	`{`
	`  microcomponent_OK->Mark() = 0;`
	`  DAS_OK->Mark() = 0;`
	`  MICROCOMPONENTS_FAILED->Mark()++;`
	`}`

repair_microcomponent	`permanent_failure->Mark() = 0;`
	`transient_failed->Mark() = 0;`
	
	`if (infrastructure_ok->Mark())`
	`{`
	`  microcomponent_ok->Mark() = 1;`
	`  MICROCOMPONENTS_FAILED->Mark()--;`
	`}`
	`if (permanent_temporal_failure->Mark())`
	`{`
	`  permanent_temporal_failure->Mark() = 0;`
	`  TSS_OK->Mark() = 1;`
	`}`

set_microcomponent_ok	`if ( !transient_failed->Mark()`
	`&& !permanent_failure->Mark()`
	`)`
	`{`
	`  microcomponent_ok->Mark() = 1;`
	`  MICROCOMPONENTS_FAILED->Mark()--;`
	`}`

Table 6.19: SAN for Non Safety-Critical Micro Components - Output Gateways

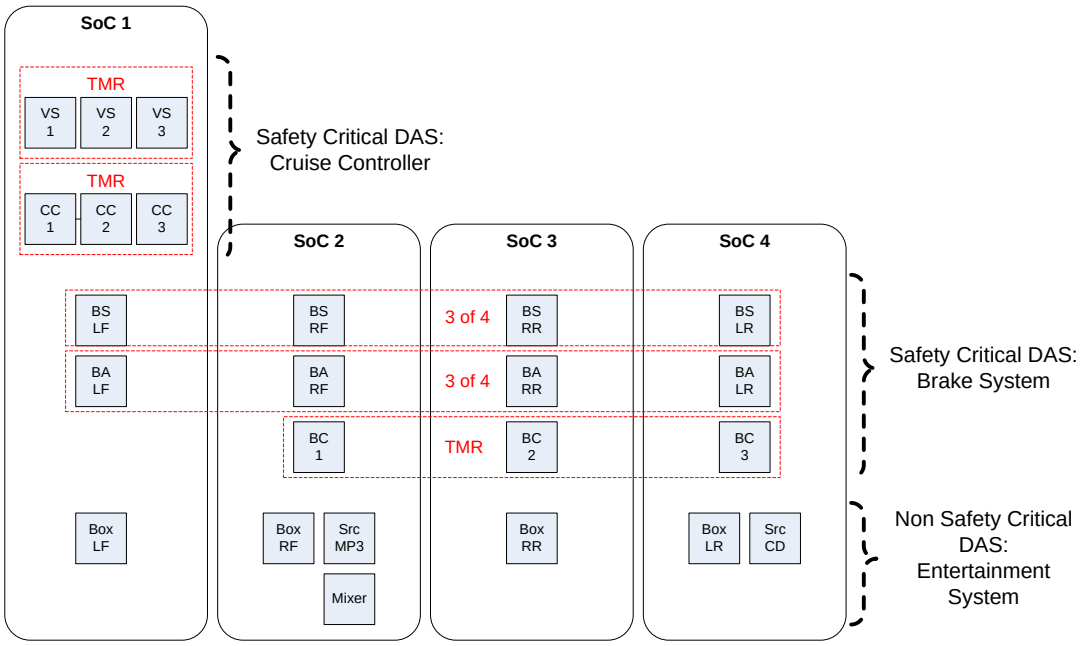


Figure 6.13: Micro Components of the Automotive Example

6.6 Composed Models

6.6.1 Automotive Example

Introduction to the Automotive Example

Figure 6.13 outlines an example for a safety-criticality mixed system with three logical independent DASs in an automotive environment. At each wheel an SoC is located, e.g., for brake controlling, audio signals decoding or throttle activation calculations. The SoC components are assumed to be physical independent and the three distributed DASs are logical independent.

The Cruise Controller DAS consists of two on-chip-TMR protected safety-critical tasks: The velocity-sensor (VS) and the cruise-controller task (CC). The DAS is considered to be valid as long as at least two replicas per task are working.

The safety-critical Brake System DAS consists of a brake-sensor (BS) and a brake-actuator (BA) task per wheel and an off-chip-TMR protected brake-control (BC) task. The DAS tolerates the loss of one sensor, one actuator and one brake-control job.

The Entertainment System DAS is non safety-critical. In this way the jobs have a minor SIL level and no redundancy. The DAS consists of two multimedia source

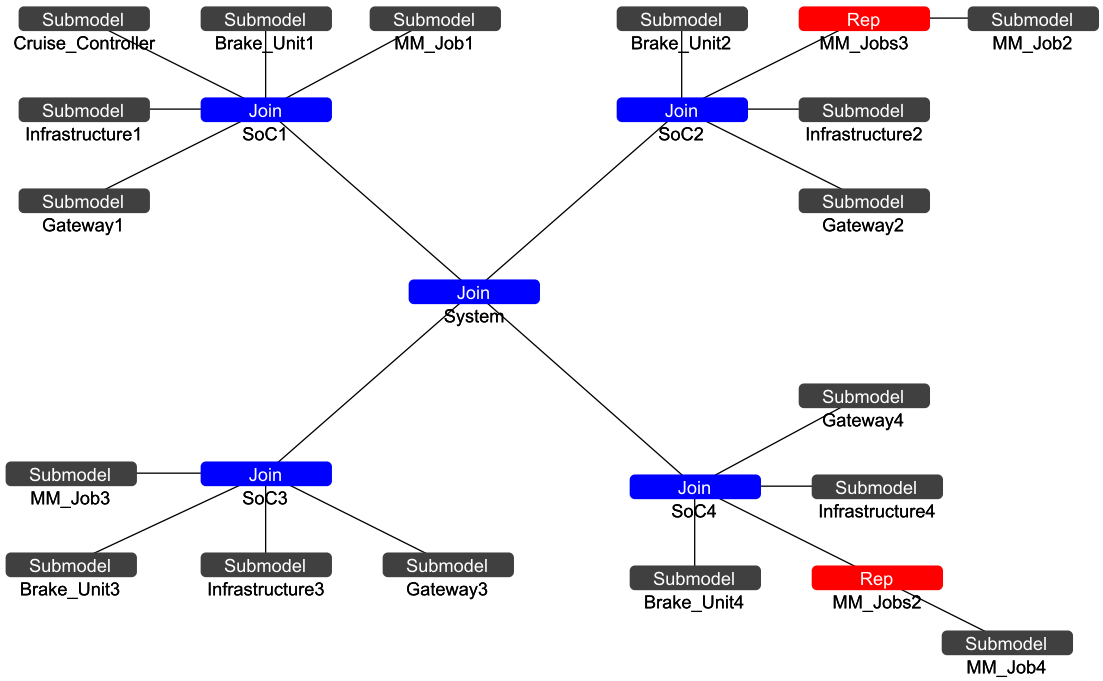


Figure 6.14: Composed Model of the Automotive Example

tasks (e.g., a CD jukebox and a MP3 player), a sound-mixer task and one music-box-task per SoC, with the purposes of decoding signals from different multimedia sources.

Composed Model of the Automotive Example

Figure 6.14 depicts the composed cluster model where each branch (originating from the Join-node `System`) represents an SoC. For each SoC a Join-node combines micro component models, a *Infrastructure* model and a *Gateway* model. See table 6.20 for a detailed explanations for the submodels. The first column states the caption of a submodel within the composed model while the second column specifies the name of the original SAN or composed model. The third and the fourth column describe the purpose of the submodel.

Table 6.21 specifies the Rep-node parameters for the composed model.

The tables 6.22 to 6.25 describe the shared state variables of each of the four SoC branches in the composed model.

Submodel	Instance of …	Description	Notes
Cruise_Controller	Cruise_Controller	Composes the micro components of the Cruise Controller DAS	The cruise controller is implemented by two on-chip-TMRs on SoC 1
SoC_Infrastructure	Infrastructure	Models the SoC Infrastructure: TNA, RMA and RCU	The overall system model incorporates one Infrastructure and one Gateway model per modeled SoC
Gateway{1,2,3,4}	Gateway	Describes the interconnection with the off-chip network.	
Brake_Unit1	Brake_Transducer	Describes the sensor- and the actuator tasks of the Brake-DAS at wheel 1	The brake-controller is implemented by means of off-chip-TMR. The sensors and actuators are part of 3-of-4 ensembles. I.e., the brake is available if 3 of 4 nodes are running.)
Brake_Unit{2,3,4}	Brake_Unit	Describes the sensor-, actuator and controller tasks of the Brake-DAS at the wheels 2 to 4	
MM_Job{1,2,3,4}	Non_Safety_Critical _Microcomponent	Each sub-model stands for an Entertainment task	-

Table 6.20: Composed Model of the Automotive Example - Submodels Description

Rep-Node	Reps #	Shared State Variables
MM_Jobs2	2	DAS_OK
		GATEWAY_OK
		MICROCOMPONENTS_FAILED
		RMA_OK
		TSS_OK
MM_Jobs3	3	DAS_OK
		GATEWAY_OK
		MICROCOMPONENTS_FAILED
		RMA_OK
		TSS_OK

Table 6.21: Composed Model of the Automotive Example - Rep-Node Parameters

Shared State Variable	Sub-model Variables
BA_MICROCOMPONENTS_FAILED	Brake_Unit1->BA_MICROCOMPONENTS_FAILED
BS_MICROCOMPONENTS_FAILED	Brake_Unit1->BS_MICROCOMPONENTS_FAILED
DAS1_OK	Cruise_Controller->DAS_OK
DAS2_OK	Brake_Unit1->Brake_DAS_OK
DAS3_OK	MM_Job1->DAS_OK
GATEWAY_OK	Brake_Unit1->GATEWAY_OK
	MM_Job1->GATEWAY_OK
	Gateway1->GATEWAY_OK
MM_MICROCOMPONENTS_FAILED	MM_Job1->MICROCOMPONENTS_FAILED
TSS_OK	Cruise_Controller->TSS_OK
	Brake_Unit1->TSS_OK
	MM_Job1->TSS_OK
	Infrastructure1->TSS_OK
	Gateway1->TSS_OK
RMA_OK	MM_Job1->RMA_OK
	Infrastructure1->RMA_OK

Table 6.22: Composed Model of the Automotive Example - Join Node Parameters of SoC 1

Shared State Variable	Sub-model Variables
BA_MICROCOMPONENTS_FAILED	Brake_Unit2->BA_MICROCOMPONENTS_FAILED
BC_MICROCOMPONENTS_FAILED	Brake_Unit2->BC_MICROCOMPONENTS_FAILED
BS_MICROCOMPONENTS_FAILED	Brake_Unit2->BS_MICROCOMPONENTS_FAILED
DAS2_OK	Brake_Unit2->Brake_DAS_OK
DAS3_OK	MM_Jobs3->DAS_OK
GATEWAY_OK	MM_Jobs3->GATEWAY_OK
	Brake_Unit2->GATEWAY_OK
	Gateway2->GATEWAY_OK
MM_MICROCOMPONENTS_FAILED	MM_Job1->MICROCOMPONENTS_FAILED
TSS_OK	MM_Jobs3->TSS_OK
	Brake_Unit2->TSS_OK
	Infrastructure2->TSS_OK
	Gateway2->TSS_OK
RMA_OK	MM_Jobs3->RMA_OK
	Infrastructure2->RMA_OK

Table 6.23: Composed Model of the Automotive Example - Join Node Parameters of SoC 2

Shared State Variable	Sub-model Variables
BA_MICROCOMPONENTS_FAILED	Brake_Unit3->BA_MICROCOMPONENTS_FAILED
BC_MICROCOMPONENTS_FAILED	Brake_Unit3->BC_MICROCOMPONENTS_FAILED
BS_MICROCOMPONENTS_FAILED	Brake_Unit3->BS_MICROCOMPONENTS_FAILED
DAS2_OK	Brake_Unit3->Brake_DAS_OK
DAS3_OK	MM_Job3->DAS_OK
GATEWAY_OK	Brake_Unit3->GATEWAY_OK
	MM_Job3->GATEWAY_OK
	Gateway3->GATEWAY_OK
MM_MICROCOMPONENTS_FAILED	MM_Job3->MICROCOMPONENTS_FAILED
TSS_OK	Brake_Unit3->TSS_OK
	MM_Job3->TSS_OK
	Infrastructure3->TSS_OK
	Gateway3->TSS_OK
RMA_OK	MM_Job3->RMA_OK
	Infrastructure3->RMA_OK

Table 6.24: Composed Model of the Automotive Example - Join Node Parameters of SoC 3

Shared State Variable	Sub-model Variables
BA_MICROCOMPONENTS_FAILED	Brake_Unit4->BA_MICROCOMPONENTS_FAILED
BC_MICROCOMPONENTS_FAILED	Brake_Unit4->BC_MICROCOMPONENTS_FAILED
BS_MICROCOMPONENTS_FAILED	Brake_Unit4->BS_MICROCOMPONENTS_FAILED
DAS2_OK	Brake_Unit4->Brake_DAS_OK
DAS3_OK	MM_Jobs2->DAS_OK
GATEWAY_OK	Brake_Unit4->GATEWAY_OK
	MM_Jobs2->GATEWAY_OK
	Gateway4->GATEWAY_OK
MM_MICROCOMPONENTS_FAILED	MM_Jobs2->MICROCOMPONENTS_FAILED
TSS_OK	Brake_Unit4->TSS_OK
	MM_Jobs2->TSS_OK
	Infrastructure4->TSS_OK
	Gateway4->TSS_OK
RMA_OK	MM_Jobs2->RMA_OK
	Infrastructure4->RMA_OK

Table 6.25: Composed Model of the Automotive Example - Join Node Parameters of SoC 4

Composed Model of a Cruise Controller

The Cruise Controller consists of two TMR systems: One that fetches sensor values and one that calculates servo values for the throttle activator. The corresponding composed model is depicted in figure 6.15. Each task is described by an instance of the safety-critical micro component model for TMRs. The Rep-node `TMR` forms a TMR by tripling the micro component models which in turn is duplicated by the Rep-node `Sensors_and_Controller`.

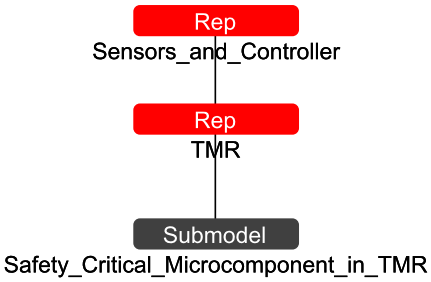


Figure 6.15: Composed Model of a Cruise Controller

Rep-Node	Reps #	Shared State Variables
Sensors_and_Controller	2	DAS_OK
		TSS_OK
TMR	3	DAS_OK
		TSS_OK
		TOTAL_MICROCOMPONENTS_FAILED

Table 6.26: Composed Model of a Cruise Controller - Rep-node Parameters

Composed Models of a Brake Unit and a Brake Transducer

For the Brake-DAS two composed models are used for the automotive example: The *Brake_Transducer* model shown in figure 6.16 describes the combination of a brake sensor and an actuator, each modeled by a safety-critical micro component model for 3-of-4 ensembles. The *Brake_Unit* model in figure 6.17 enhances the *Brake_Transducer* with a brake controller, which is specified by the safety-critical micro component model for TMR.

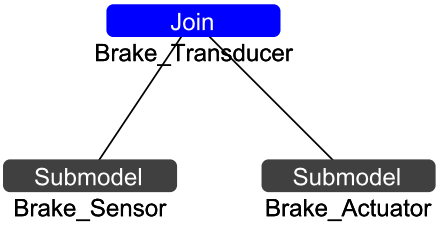


Figure 6.16: Composed Model of a Brake Transducer

Shared State Variable	Sub-model Variables
BA_MICROCOMPONENTS_FAILED	Brake_Actuator->MICROCOMPONENTS_FAILED
BS_MICROCOMPONENTS_FAILED	Brake_Sensor->MICROCOMPONENTS_FAILED
Brake_DAS_OK	Brake_Sensor->DAS_OK
	Brake_Actuator->DAS_OK
GATEWAY_OK	Brake_Sensor->GATEWAY_OK
	Brake_Actuator->GATEWAY_OK
TSS_OK	Brake_Sensor->TSS_OK
	Brake_Actuator->TSS_OK

Table 6.27: Composed Model of a Brake Transducer - Join Node Parameters

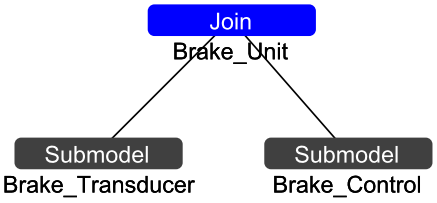


Figure 6.17: Composed Model of a Brake Unit

Shared State Variable	Sub-model Variables
BA_MICROCOMPONENTS_FAILED	Brake_Transducer->BA_MICROCOMPONENTS_FAILED
BC_MICROCOMPONENTS_FAILED	Brake_Control->MICROCOMPONENTS_FAILED
BS_MICROCOMPONENTS_FAILED	Brake_Transducer->BS_MICROCOMPONENTS_FAILED
Brake_DAS_OK	Brake_Transducer->Brake_DAS_OK
	Brake_Control->DAS_OK
GATEWAY_OK	Brake_Transducer->GATEWAY_OK
	Brake_Control->GATEWAY_OK
TSS_OK	Brake_Transducer->TSS_OK
	Brake_Control->TSS_OK

Table 6.28: Composed Model of a Brake Unit - Join Node Parameters

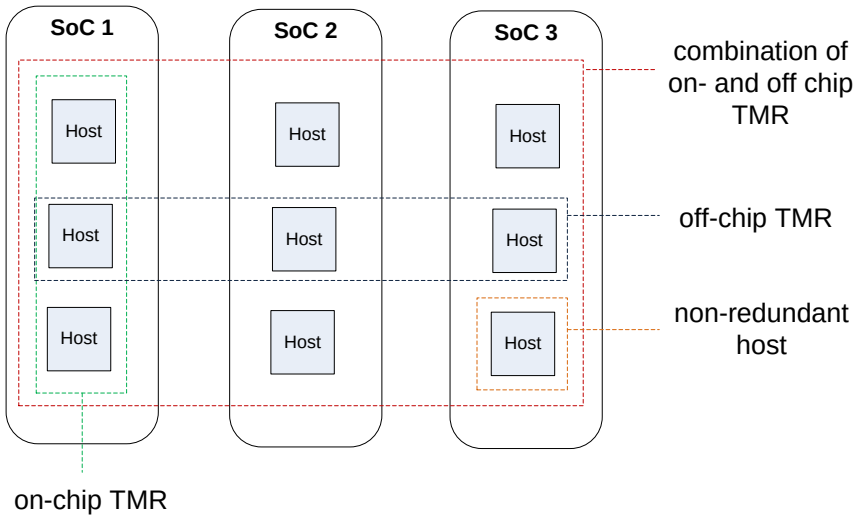


Figure 6.18: TMR Approaches

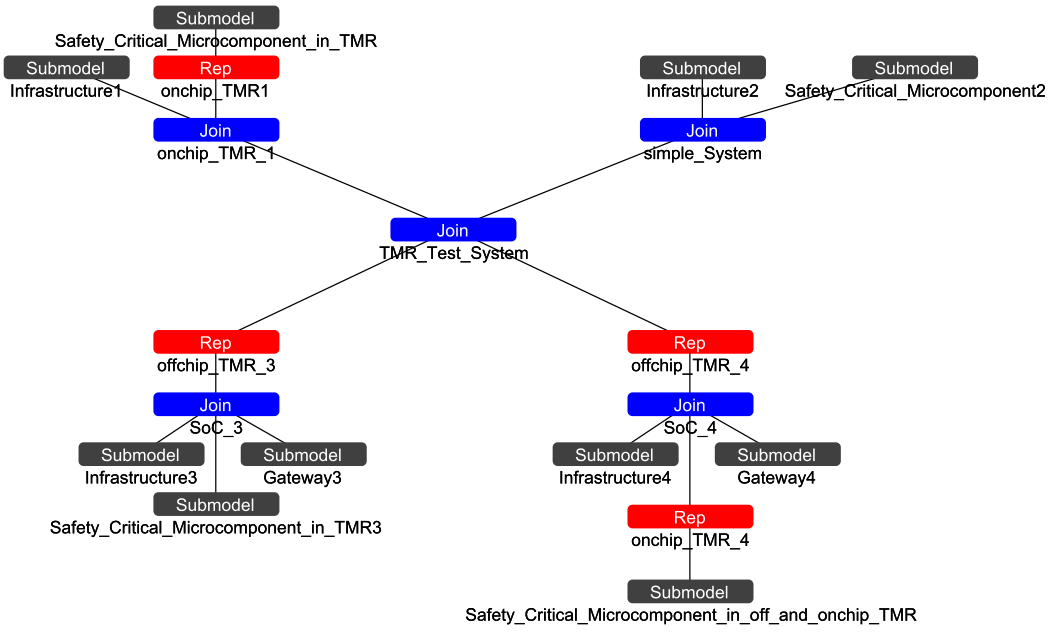


Figure 6.19: Composed Model for a TMR Approach Comparison

6.6.2 TMR Comparison

The composed model in figure 6.19 combines four different implementation approaches of an application which possesses only one task. Figure 6.18 depicts a schematic representation of the four approaches: A simple implementation with one job running on one SoC and three different approaches of TMR as described in sub-section 6.3. The tables 6.29 and 6.30 specify the complete parameter set for the composed model.

Rep-Node	Reps #	Shared State Variables
onchip_TMR1	3	DAS_OK
		GATEWAY_OK
		MICROCOMPONENTS_FAILED
		TSS_OK
offchip_TMR_3	3	DAS_OK
		MICROCOMPONENTS_FAILED
onchip_TMR_4	3	DAS_OK
		GATEWAY_OK
		MICROCOMPONENTS_FAILED
		TSS_OK
		TMRS_FAILED
		TMR_OK
		TOTAL_MICROCOMPONENTS_FAILED
offchip_TMR_4	3	DAS_OK
		TMRS_FAILED
		TOTAL_MICROCOMPONENTS_FAILED

Table 6.29: Composed Model for a TMR Approach Comparison - Rep Node Parameters

a) On-Chip Approach - Join-Element: `onchip_TMR_1`

Shared State Variable	Sub-model Variables
TSS_OK	onchip_TMR1->TSS_OK
	Infrastructure1->TSS_OK

b) Off-Chip Approach - Join-Element: `SoC_3`

Shared State Variable	Sub-model Variables
DAS_OK	Safety_Critical_Microcomponent_in_TMR3->DAS_OK
GATEWAY_OK	Safety_Critical_Microcomponent_in_TMR3->GATEWAY_OK
	Gateway3->GATEWAY_OK
MICROCOMPONENTS_FAILED	Safety_Critical_Microcomponent_in_TMR3->MICROCOMPONENTS_FAILED
TSS_OK	Infrastructure3->TSS_OK
	Safety_Critical_Microcomponent_in_TMR3->TSS_OK
	Gateway3->TSS_OK

c) Combined Off- and On-Chip Approach - Join-Element: `SoC_4`

Shared State Variable	Sub-model Variables
DAS_OK	onchip_TMR_4->DAS_OK
GATEWAY_OK	onchip_TMR_4->GATEWAY_OK
	Gateway4->GATEWAY_OK
TMRS_FAILED	onchip_TMR_4->TMRS_FAILED
TOTAL_MICROCOMPONENTS_FAILED	onchip_TMR_4->TOTAL_MICROCOMPONENTS_FAILED
MICROCOMPONENTS_FAILED	Safety_Critical_Microcomponent_in_TMR3->MICROCOMPONENTS_FAILED
TSS_OK	onchip_TMR_4->TSS_OK
	Infrastructure4->TSS_OK
	Gateway4->TSS_OK

d) Non-Fault_Tolerant_Approach - Join-Element: `non_fault_tolerant_approach`

Shared State Variable	Sub-model Variables
TSS_OK	Infrastructure2->TSS_OK
	Safety_Critical_Microcomponent2->TSS_OK

Table 6.30: Composed Model for a TMR Approach Comparison - Join Node Parameters

Confidence	**Level**	95%
	Interval	10%
Simulation Stop Time	10^{10} hours	

Table 6.31: Möbius Simulator Parameter

6.7 Reward Models

This section describes the performance variables used for the evaluations of the example SoC systems. As explained in section 5.5 (*Solving Models with Möbius*) the Möbius simulator is used as solver. The common simulation parameters for all performance variables are given by table reftab:simparameter.

Backgrounds to reward models in Möbius can be found in section 5.4.

6.7.1 Automotive Example

Tables 6.32 and 6.33 specifies performance variables to calculate the MTTFs and the reliabilities for the automotive example.

The `DAS_OK` variables in table 6.32, (c) is divided by 6 since the variable belongs to the *Safety_Critical_Microcomponent* submodel (as depicted in sub-section 6.6.1) which is replicated ternary for the controller task and additionally ternary for the sensor task. In sum the composed model contains 6 replications. Since the Möbius simulator summarizes the output of the reward functions for each of the replications, the performance variable result would be six times too high. The reward functions of the other performance variables can be explained similarly.

6.7.2 TMR Comparison

Figures 6.34 to 6.35 specifies performance variables to calculate the MTTFs and the reliabilities for the different TMR approaches.

a) Entertainment DAS – Mean Time To Failure

Reward Function	`return MM_Job1->DAS_OK->Mark();`
Type	Interval of Time

b) Brake DAS – Mean Time To Failure

Reward Function	`return Brake_Control->DAS_OK->Mark() / 3;`
Type	Interval of Time

c) Cruise Controller DAS – Mean Time To Failure

Reward Function	`return Safety_Critical_Microcomponent_in_TMR->DAS_OK->Mark() / 6;`
Type	Interval of Time

d) Entertainment DAS – Availability

Reward Function	`if (!MM_Job1->MICROCOMPONENTS_FAILED->Mark())`
	`{`
	`  return 1.0;`
	`}`
	`else return 0;`

Type	Time Averaged Interval of Time

Table 6.32: Performance Variables for the Automotive Example - Part 1

a) Brake DAS – Reliability

Reward Function	`return Brake_Control->DAS_OK->Mark() / 3;`
Type	Instant of Time
Time Points [hours]	1E1,1E2,1E3,1E4,1E5,1E6,2E6,4E6,6E6,8E6,1E7,2E7,4E7,6E7,8E7,1E8,2E8,4E8,6E8,8E8,1E9

b) Cruise Controller DAS – Reliability

Reward Function	`return Safety_Critical_Microcomponent_in_TMR->DAS_OK->Mark() / 6;`
Type	Instant of Time
Time Points [hours]	1E1,1E2,1E3,1E4,2E4,4E4,6E4,8E4,1E5,2E5,4E5,6E5,8E5,1E6,2E6,4E6,6E6,8E6,1E7

Table 6.33: Performance Variables for the Automotive Example - Part 2

a) Single Job – Mean Time To Failure

Reward Function	`return Safety_Critical_Microcomponent2->DAS_OK->Mark();`
Type	Interval of Time

b) On-Chip TMR – Mean Time To Failure

Reward Function	`return Safety_Critical_Microcomponent_in_TMR->DAS_OK->Mark() / 3;`
Type	Interval of Time

c) On- and Off-Chip TMR combination – Mean Time To Failure

Reward Function	`Safety_Critical_Microcomponent_in_off_and_onchip_TMR->DAS_OK->Mark() / 9;`
Type	Interval of Time

d) Off-Chip TMR – Mean Time To Failure

Reward Function	`return Safety_Critical_Microcomponent_in_TMR3->DAS_OK->Mark() / 3;`
Type	Interval of Time

Table 6.34: Performance Variables for the TMR comparison - Part 1

a) On-Chip TMR – Reliability

Reward Function	`return Safety_Critical_Microcomponent_in_TMR->DAS_OK->Mark() / 3;`
Type	Instant of Time
Time Points [hours]	1E1,1E2,1E3,1E4,2E4,4E4,6E4,8E4,1E5,2E5,4E5,6E5,8E5,1E6,2E6,4E6,6E6,8E6,1E7,2E7, 4E7, 6E7,8E7,1E8

b) Off-Chip TMR – Reliability

Reward Function	`return Safety_Critical_Microcomponent_in_TMR3->DAS_OK->Mark() / 3;`
Type	Instant of Time
Time Points [hours]	1E1,1E2,1E3,1E4,2E4,4E4,6E4,8E4,1E5,2E5,4E5,6E5,8E5,1E6,2E6,4E6,6E6,8E6,1E7,2E7, 4E7, 6E7,8E7,1E8,2E8,3E8,4E8,5E8,6E8,7E8,8E8,9E8,1E9

c) On- and Off-Chip TMR combination – Reliability

Reward Function	`Safety_Critical_Microcomponent_in_off_and_onchip_TMR->DAS_OK->Mark() / 9;`
Type	Instant of Time
Time Points [hours]	1E1,1E2,1E3,1E4,2E4,4E4,6E4,8E4,1E5,2E5,4E5,6E5,8E5,1E6,2E6,4E6,6E6,8E6,1E7,2E7, 4E7, 6E7,8E7,1E8,2E8,3E8,4E8,5E8,6E8,7E8,8E8,9E8,1E9

d) Single Job – Reliability

Reward Function	`return Safety_Critical_Microcomponent2->DAS_OK->Mark();`
Type	Instant of Time
Time Points [hours]	1E1,1E2,1E3,1E4,2E4,4E4,6E4,8E4,1E5,2E5,4E5,6E5,8E5,1E6,2E6,4E6,6E6,8E6,1E7,2E7, 4E7, 6E7,8E7,1E8

Table 6.35: Performance Variables for the TMR comparison - Part 2

Chapter 7

Results

7.1 TMR Approaches

This section compares the simulation results of four different system configurations: non-redundant, on-chip TMR, off-chip TMR, and combined on-chip and off-chip TMR. Table 6.1 (see page 50) states the applied default model parameters for the evaluations.

7.1.1 Results for Default Parameters

Table 7.1 depicts the simulation results as MTTFs for the different TMR approaches and the non-redundant system. Figure 7.1 shows the corresponding reliability in the form of a probability function. The x-axis states the systems lifetime while the y-axis states the probability that the system experienced no failure yet.

For default parameters the evaluations show clearly the improvement of system reliability by each form of TMR deployment. Furthermore, from the comparison with the exponential distributed reliability function example, showed in figure 2.2 on page 11 can be seen, that the reliability functions of the different systems are also following an exponential distribution.

System	MTTF in Hours
Non redundant host	9,0328E+04
On-chip TMR	5,7203E+05
Off-chip TMR	2,9805E+09
Combined Off- and On-chip TMR	3,8974E+09

Table 7.1: MTTFs of TMR Approaches for Default Parameters

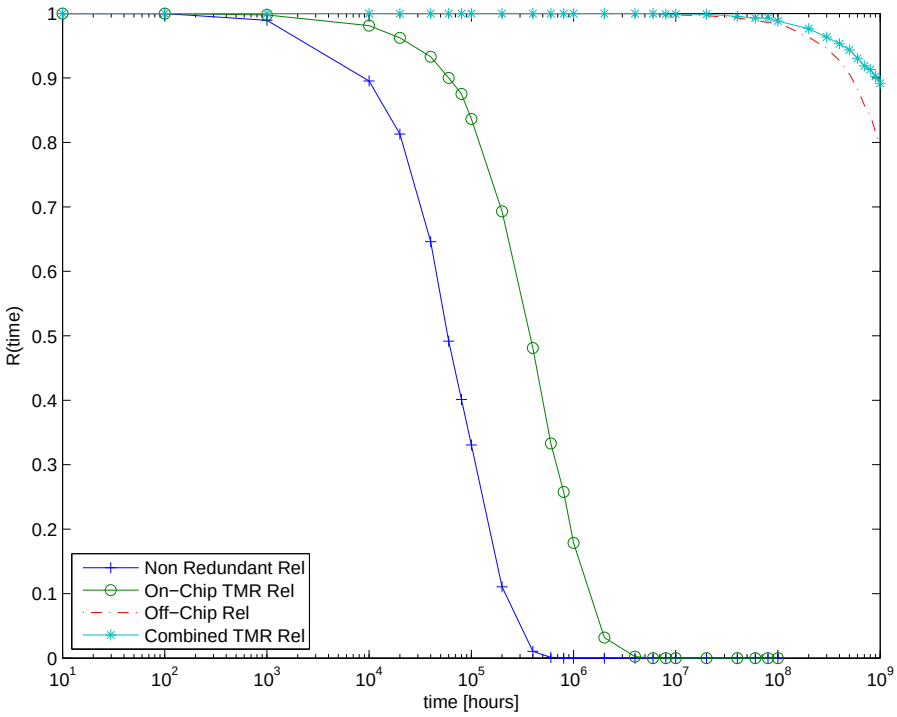


Figure 7.1: Reliabilities of TMR Approaches for Default Parameters

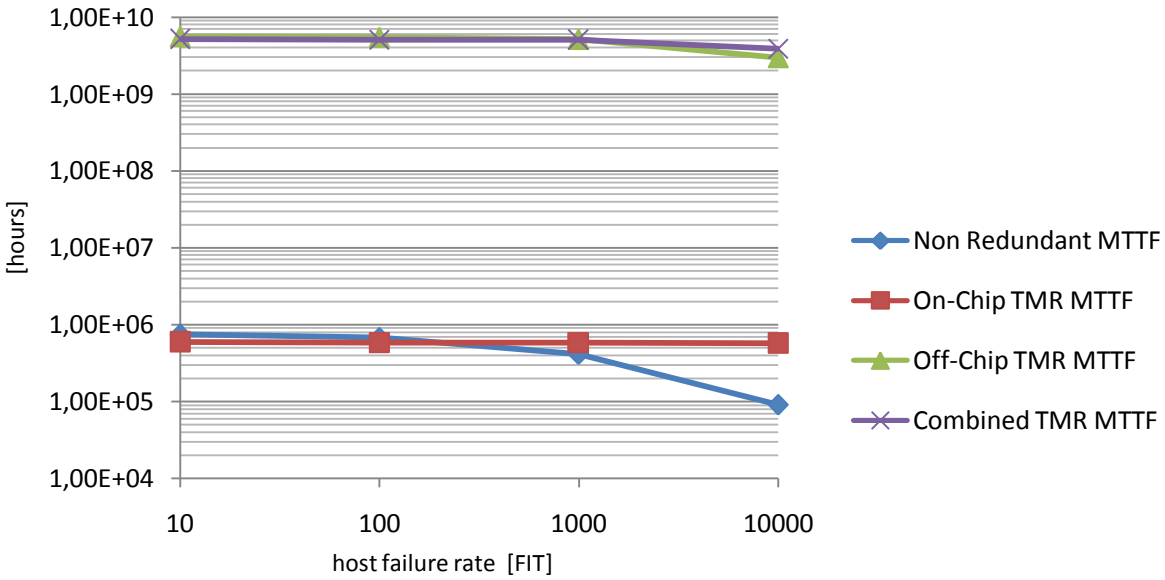


Figure 7.2: MTTFs over Physical Host Failure Rates of TMR Approaches

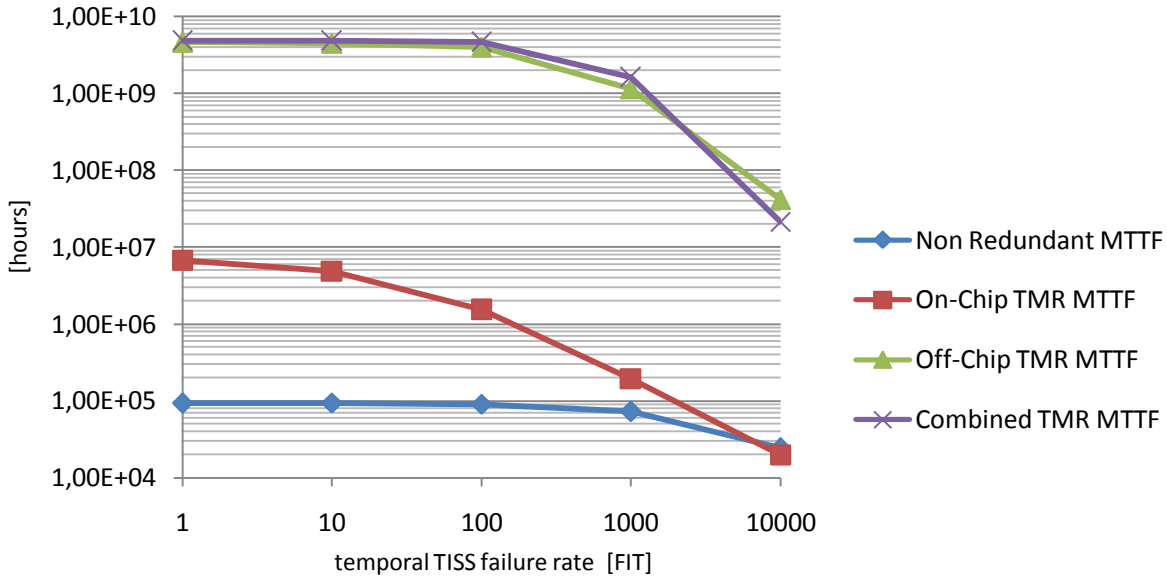


Figure 7.3: MTTFs over the Failure Rate of Physical Temporal TISS Failures

7.1.2 Physical Host Failures

Figure 7.2 shows the effect of the physical host failure rate on the reliabilities of the three TMR approaches and the non-redundant system. Along the horizontal axis, physical host failure rates are distinguished while the vertical axis shows the evaluated MTTFs.

As can be seen in figure 7.2 on-chip TMR outperforms a non-redundant solution for host failure rates of 300 FIT or worse. In case of host failure rates better than 300 FIT, the failure rate of the TSS is dominant and undermines potential reliability gains through active replication of hosts on the chip. In addition, in case of low host failure rates, compared to the failure rate of the TSS, on-chip TMR does not contribute towards a better reliability, because each additional TISS worsens the reliability of the TSS.

7.1.3 TISS-Pulse Manager Failures

Figure 7.3 shows the effect of the TISS-Pulse Manager failure rate on the reliabilities of the three TMR approaches and the non-redundant system. Along the horizontal axis physical failure rates for the TISS-Pulse Manager (which also can be interpreted as the failure rates for temporal domain failures of the TISS) are distinguished . The vertical axis shows the evaluated MTTFs.

Assuming that the failure rate for micro component failures in the value domain is about 10^4 FIT, the figure shows that the MTTFs of the non-redundant system, the

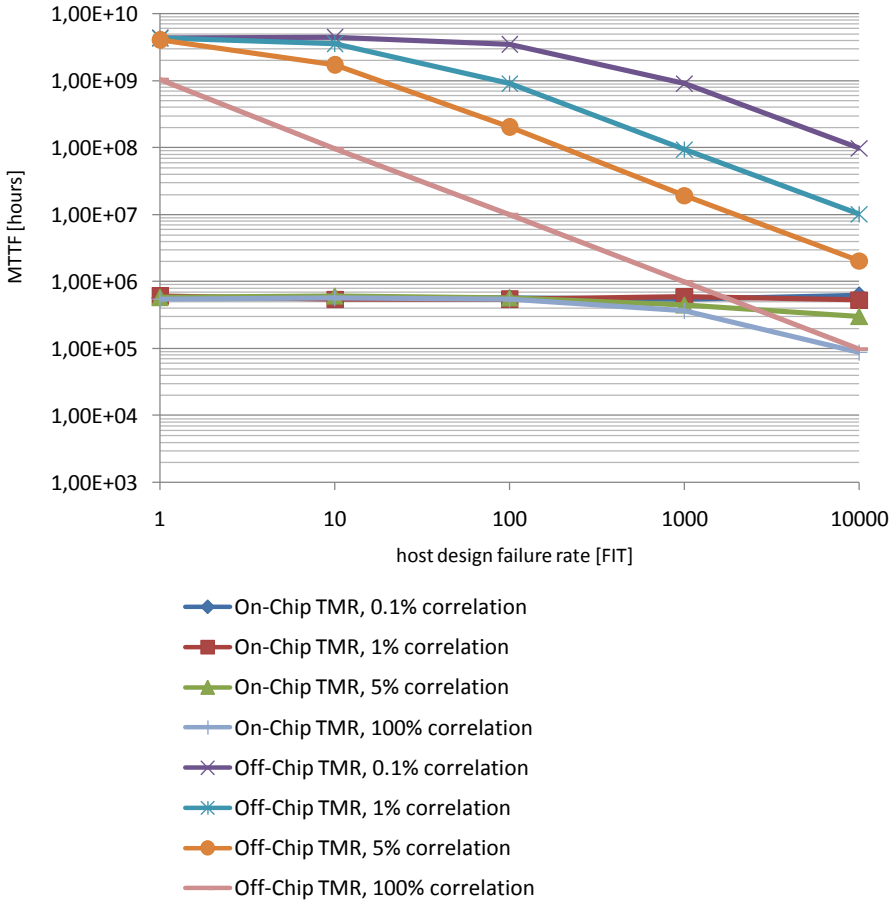


Figure 7.4: MTTFs over Host Design Failure Rates for different Diversity Coverages of TMR Approaches

off-chip TMR approach and the combined approach begin to decline from about a ratio of 1:10 (in the figure at 1000 FIT) of temporal to value domain failures. This reliability decline even worsens from a ratio of 1 (in the figure at 10000 FIT). Since the on-chip TMR approach has a tripled chance to fail due to the tripled failure rate of the NoC it shows the highest sensitivity to worsen TISS-Pulse Manager failure rates and at about 10000 FIT it even brings a worse reliability than the non-redundant system. Analogous is true for the comparison between the off-chip and the combined TMR approach. After some point, the off-chip TMR approach outperforms the combined TMR approach.

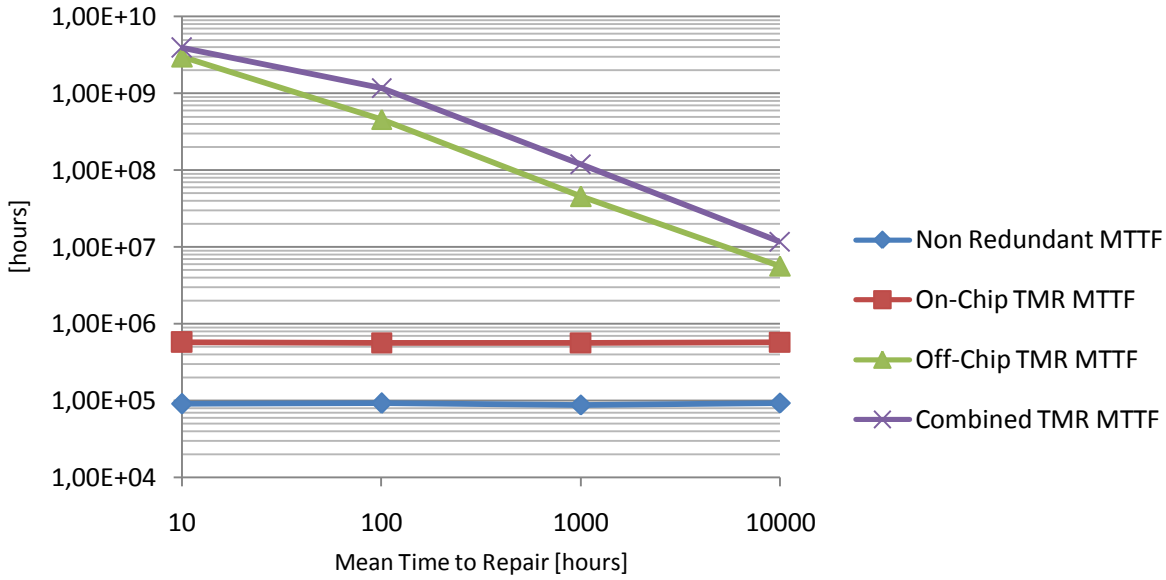


Figure 7.5: MTTFs over Mean-Time-to-Repair

7.1.4 Application Computer Design Failures & Diversity Coverage

Figure 7.4 depicts the MTTFs of different TMR approaches in comparison to a system with only one SoC. Along the horizontal axis different rates of failures caused by design faults are distinguished. The figure also shows the effect of different correlations between FCRs (between 0.1% and 100%). The rate of failures caused by physical faults is not varied.

The figure shows that for on-chip TMR, the reliability improvements of design diversity is less significant because of the dominant failure rates due to physical faults. For off-chip TMR, on the other hand, diversity has a significant positive impact for increasing failure rates due to design faults.

7.1.5 Repair Rate

Figure 7.5 shows the influence of MTTR on the reliability of the system. The MTTR has obviously no effect on the reliability of the two single-chip realizations while it significantly influences the off-chip and the combined TMR approach.

System	MTTF in Hours
Cruise Controller DAS	2,5626E+05
Brake DAS	3,4378E+08
Entertainment DAS	7,2721E+03

Table 7.2: MTTFs for Default Parameters

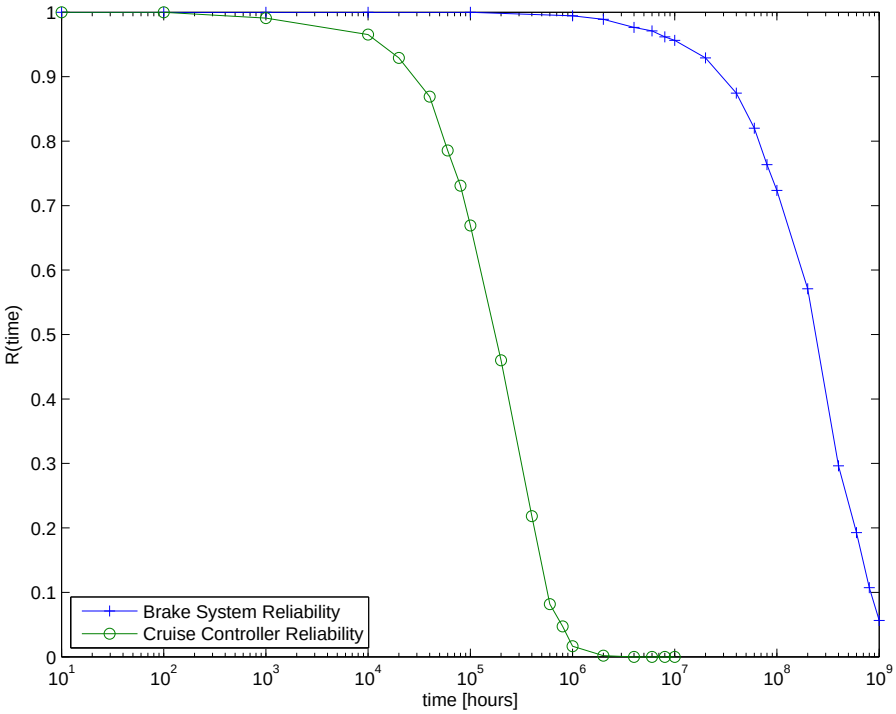


Figure 7.6: Reliability Functions for Default Parameters

7.2 Automotive Example

This section presents the simulation results for the three DASs of the automotive system example. Table 6.1 (see page 50) states the applied default model parameters for the evaluations.

7.2.1 Results for Default Parameters

Table 7.2 depicts the MTTFs evaluated by simulation for default parameters. Figure 7.6 shows the corresponding reliabilities as probability function. The x-axis states the systems lifetime while the y-axis states the probability that the system experienced no failure yet.

Although, compared with the other DASs, the Entertainment DAS showed a bad

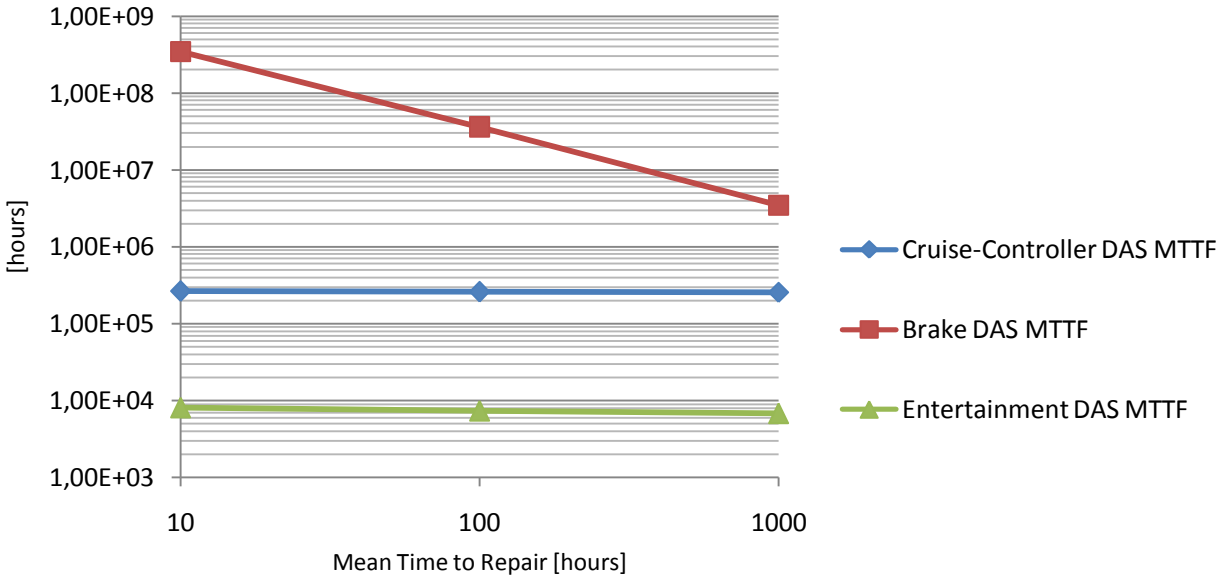


Figure 7.7: MTTFs over Mean-Time-to-Repair

Reliability, the simulation resulted a Availability of 99.9976%.

7.2.2 Repair Rate

Figure 7.7 shows the influence of MTTR on the reliability of the different DASs. The y-axis states the MTTFs in hours while the x-axis lists different MTTRs also measured in hours.

The Cruise Controller DAS is quite unaffected by the repair rate. The reason for this is, that the DAS is completely realized on only one SoC which means that the overall failure rate, caused by TSS failures, is dominant compared to permanent failures of replicas of the on-chip TMR.

Opposite to that the Brake DAS reliability shows a strong dependence on maintenance times. The reason for this is that the functionality of the two integrated off-chip TMRs are highly dependent on the state of the physical connections, which possess a quite high permanent failure rate.

7.2.3 RMA Failure Rates

The model was simulated with RMA failure rates from 10 FIT to 10000 FIT but because of the dominant effects of other failure rates and the fact that the Entertainment DAS deploys no kind of redundancy, the Entertainment Availability is quite uninfluenced by the RMA failure rate less or equal 10000 FIT and lies at

at about 99.9976%.

Chapter 8

Conclusion

This thesis showed that the TTSoC architecture enables the integrated execution of many different applications at a minimal loss of fault isolation (due to failures of the Trusted Subsystem) and without losing predictability. In consequence it was shown that ultra reliable systems can be built upon DECOS SoC components.

At architecture side, mainly responsible for this properties is the Time-Triggered Network-on-a-Chip (TTNoC), which can only be accessed over the TISSs. Since the inter- and the intra SoC communication takes place only at defined time points, the TISS forms a so-called temporal firewall that hinders an arbitrary failed host from disturbing the communication between other hosts. Therefore, the TTSoC architecture allows the integration of applications with different certification levels.

Being the sole part that directly accesses the NoC, the TISS also is the sole weak point. This means that even one temporal domain failure of the TISS can lead to an overall SoC failure and that each TISS contributes to the overall SoC failure rate. Especially for exponential distributed failure times, the overall NoC failure rate can be estimated by the sum of all TISS failure rates. From the reliability point of view the consequence is that only a bounded number of cores can be feasibly integrated on a single chip.

The dependability model of a complex system based on SoC components can be easily composed from the generic SoC model presented in this thesis. Two concrete SoC cluster examples were presented.

The first example compared the reliability of different TMR approaches.

With the chosen default parameters only the off-chip TMR and the combined approach (off-chip and on-chip TMR combined) reached a reliability which was adequate for ultra reliable applications. The on-chip TMR approach had only a 6 times better reliability than a non-redundant approach. When the host failure rate was experimentally reduced to the scale of the TISS failure rate the positive effect

diminished. The overall failure rate of the NoC grew dominant and undermined the potential reliability gains of the host replication. Nevertheless, the failure rate of the host is assumed to be much higher than the failure rate of the TISS because of its higher resource consumption. This assumption has been confirmed by the first prototype implementation.

Furthermore, the usefulness of different grades of design diversity for the TMR approaches in case of different design safety integrity levels was evaluated. The results have shown that for on-chip TMR, the reliability improvements of design diversity are less significant because of the dominant failure rates due to physical faults. For off-chip TMR, on the other hand, diversity has a significant positive impact for increasing failure rates due to design faults.

The second example for a DECOS SoC based cluster came from the automotive domain. The considered system mixes safety-critical and non-safety-critical applications and consists of a cruise controller DAS, a brake DAS and an entertainment DAS. This example showed how to compose a complex real-world example from the generic SoC model created within Möbius.

Appendix A

Acronyms

ASER Accelerated-Soft Error Rate

CP Configuration and Planning

DAS Distributed Application Subsystem

DECOS Dependable Embedded Components and Systems

DM Diagnosis and Management

ECR Error Containment Region

ECU Electronic Control Unit

EMI Electro Magnetic Interference

FCR Fault Containment Region

FIT failures in 10^9 hours

FPGA Field Programmable Gate Array

FTU Fault-Tolerant Unit

HARP Hybrid Automated Reliability Predictor

IP Intellectual Property

MBU Multi Bit Upsets

MEDL Message Descriptor List

MPSoC Multi-Processor System-on-a-Chip

MTTF Mean-Time-To-Failure

MTTR Mean-Time-To-Repair

NoC Network-on-a-Chip

PC personal computer

QoS Quality of Service

RCU Replica Coordination Unit

RM Resource Manager

RMA Resource Management Authority

SAN Stochastic Activity Network

SEB Single-Event Burnout

SEFI Single-Event Functional Interrupt

SEGR Single-Event Gate Rupture

SEL Single-Event Latchup

SER Soft Error Rate

SET Single Event Transients

SEU Single Event Upset

SIL Safety Integrity Level

SoC System-on-a-Chip

SRAM Static Random Access Memory

SURE Semi-Markov Unreliability Range Estimator

TDMA Time-Division Multiple Access

TISS Trusted Interface Subsystem

TMR Triple Modular Redundancy

TSS Trusted Subsystem

TNA Trusted Network Authority

TTE Time-Triggered Ethernet

TTNoC Time-Triggered Network-on-a-Chip

TTSoC Time-Triggered System-on-a-Chip

UNI Uniform Network Interface

References

[1] Wikipedia. Bathtub curve, 1. Feb 2008. At `http://en.wikipedia.org/wiki/Bathtub_curve`. 11

[2] R. Obermaisser, H. Kraut, and C. Salloum. A transient-resilient system-on-a-chip architecture with support for on-chip and off-chip tmr. Research report, Vienna University of Technology, Institute of Computer Engineering, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2007. 20

[3] H. Kopetz, B. Huber, R. Obermaisser, C. Salloum, G. Engleder, R. Seiger, and B. Weirich. Design of system-on-a-chip component. Project diliverable, Vienna University of Technology, Institute of Computer Engineering, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, Not finished. 22

[4] Deliverable d1.1, fit-it project 813299/7852. Research report, Vienna University of Technology, Institute of Computer Engineering, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, March 2007. 24

[5] Performability Engineering Research Group, Coordinated Science Laboratory, University of Illinois, USA. *Mobius Manual*. http://www.perform.csl.uiuc.edu/mobius/manual/MobiusManual.pdf. 12, 42, 44

[6] B. Randell A. Avizienis, J.-C. Laprie. Fundamental concepts of dependability. In *LAAS-CNRS*, 2001. 1, 7, 8

[7] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri. From a federated to an integrated architecture for dependable embedded systems. Research Report 22/2004, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2004. 1

[8] H. Kopetz, R. Obermaisser, C. El Salloum, and B. Huber. Automotive software development for a multi-core system-on-a-chip. *4th International ICSE workshop on Software Engineering for Automotive Systems (SEAS'07)*, May. 2007. 1

[9] J.F. Meyer A. Movaghar W.H. Sanders. Stochastic activity networks: Structure behavior and application. In *Proc. of the International Workshop on Timed Petri Nets*, pages 106–115, july 1985. 3, 40, 43

[10] "G. Clark T. Courtney D. Daly D. Deavours S. Derisavi J.M. Doyle W.H. Sanders P. Webster". "the mobius modeling tool". In *"Proc. of 9th International Workshop on Petri Nets and Performance Models"*, pages 241–250, sept 2001. 3, 14, 39

[11] E. Normand. Single event upsets at ground level. *IEEE Transactions on Nuclear Science*, 43(6):2742–2750, 1996. 3, 4, 15

[12] Cristian Constantinescu. Impact of deep submicron technology on dependability of vlsi circuits. *dsn*, 00:205, 2002. 3

[13] JEDEC. *JEDEC standard JESD89, Measurement and Reporting of Alpha Particles and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices*, August 2001. 4, 15, 17

[14] G. Asadi, S.G. Miremadi, H.R. Zarandi, and A. Ejlali. Fault injection into sram-based fpgas for the analysis of seu effects. *Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference on*, pages 428–430, 15-17 Dec. 2003. 4

[15] J. Arlat, Y. Crouzet, and J. Laprie. Fault injection for dependability validation of fault-tolerant computing systems. *Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'., Twenty-Fifth International Symposium on*, pages 400–, 27-30 Jun 1995. 4

[16] M. Ceschia, M. Violante, M.S. Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, and A. Candelori. Identification and classification of single-event upsets in the configuration memory of sram-based fpgas. *Nuclear Science, IEEE Transactions on*, 50(6):2088–2094, Dec. 2003. 4, 16

[17] M.B. Tahoori G. Asadi. Soft Error Rate Estimation and Mitigation for SRAM-Based FPGAs. *International Symposium on Field Programmable Gate Arrays*, 2005. 4

[18] M. Veeraraghavan A.-L. Reibman. Reliability modeling: An overview for system designers. *Computer*, 24(4):49–57, 1991. 5, 8, 13

[19] A. Wild. The synergistic twins: fault trees and success trees. *Reliability and Maintainability Symposium, 2005. Proceedings. Annual*, pages 445–450, Jan. 24-27, 2005. 5

[20] M. Sahinoglu, C.V. Ramamoorthy, A.E. Smith, and B. Dengiz. A reliability block diagramming tool to describe networks. *Reliability and Maintainability, 2004 Annual Symposium - RAMS*, pages 141–145, 2004. 5

[21] R. W. Butler. The sure reliability analysis program. pages 198–204, August 1986. 5, 14

[22] K. Trivedi. Sharpe 2002: Symbolic hierarchical automated reliability and performance evaluator. *dsn*, 00:544, 2002. 5, 14

[23] J.-C. Laprie. *Dependability: Basic Concepts and Terminology.* Springer Verlag, Vienna, Austria, 1992. 8, 9

[24] M. Lyu, editor. *Handbook of software reliability and system reliability.* McGraw-Hill, Inc., Hightstown, NJ, USA, 1996. 8

[25] Lala J.H. and Harper R.E. Architectural principles for safety-critical real-time applications. In *Proceedings of the IEEE*, volume 82, pages 25–40, jan 1994. 9

[26] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, Boston, Dordrecht, London, 1997. 10, 13, 27, 31

[27] Dhiraj K. Pradhan, editor. *Fault-tolerant computer system design.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. 10

[28] Peter Bishop and Robin Bloomfield. A methodology for safety case development. In Felix Redmill and Tom Anderson, editors, *Industrial Perspectives of Safety-critical Systems: Proceedings of the Sixth Safety-critical Systems Symposium, Birmingham 1998*, pages 194–203. Springer, 1998. 13

[29] A. Ditali A. Hasnain. Building-in reliability: Soft errors - a case study. In *Proceedings. 30 Intl Reliability Physics Symposium*, page 276, April 1992. 15

[30] C. Constantinescu. Trends and challenges in vlsi circuit reliability. *IEEE Micro*, 23(4):14–19, 2003. 16

[31] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinatorial logic. In *Proceedings. Intl Conference on Dependable Systems and Networks*, pages 389–398, 2000. 16

[32] T. Furuyama. Deep sub-100 nm design challenges. In *Proceedings of the 9th EUROMICRO Conference on Digital System Design*, pages 9–16, 2006. 19

[33] R. Obermaisser, H. Kopetz, C. Salloum, and B. Huber. Error containment in the time-triggered system-on-a-chip architecture. *International Embedded Systems Symposium (IESS'07)*, Jun. 2007. 19

[34] J. Leohold. Automotive systems architecture. In *Architectural Paradigms for Dependable Embedded Systems*, 2005. 19

[35] R. Obermaisser, H. Kopetz, C. Salloum, and B. Huber. Error containment in the time-triggered system-on-a-chip architecture. In *Proc. of the Int. Embedded Systems Symposium IESS2007*, May 2007. 19, 21

[36] Object Management Group. *Smart Transducers Interface Specification*, 2002. 21

[37] H. Kopetz and R. Nossal. Temporal firewalls in large distributed real-time systems. *Proc. of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '97)*, 1997. 23

[38] B. Pauli, A. Meyna, and P. Heitmann. Reliability of electronic components and control units in motor vehicle applications. In *VDI Berichte 1415, Electronic Systems for Vehicles*, pages 1009–1024. Verein Deutscher Ingenieure, 1998. 26

[39] H. Kopetz and G. Grünsteidl. TTP – a protocol for fault-tolerant real-time systems. *Computer*, 27(1):14–23, January 1994. Vienna University of Technology, Real-Time Systems Group. 26, 34

[40] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer. The Time-Triggered Ethernet (TTE) design. *Proc. of 8th IEEE Int. Symposium on Object-oriented Real-time distributed Computing (ISORC)*, May 2005. 26, 34

[41] S. Poledna. Replica determinism in distributed real-time systems: A brief survey. *Real-Time Systems*, 6:289–316, 1994. 26

[42] W. Schutz A. Avizienis, M.R. Lyu. In search of effective diversity: A six-language study of fault-tolerant flight control software. *Eighteenth International Symposium on Fault-Tolerant Computing (FTCS-18)*, 1988. 29

[43] iRoC Technologies. Radiation results of the ser test of actel fpga december 2005. Technical report, December 2005. 38

[44] R. Seiger. An extensible interface subsystem for a novel time-triggered system-on-a-chip architecture. Master's thesis, Vienna University Of Technology, 2007. 38

[45] B. Weirich. Resource management in an on-chip network. Master's thesis, Vienna University Of Technology, 2007. 38

[46] G. Engleder. Time-triggered network-on-a-chip. Master's thesis, Vienna University Of Technology, 2007. 38

[47] Int. Standardization Organisation, ISO/IEC 9899:1999. *ISO C99*, 1999. 41

[48] A.-L. Reibman, R. Smith, and K. Trivedi. Markov and markov reward model transient analysis: An overview of numerical approaches. *European Journal of Operational Research*, 40(2):257–267, May 1989. available at http://ideas.repec.org/a/eee/ejores/v40y1989i2p257-267.html. 47

[49] R. Geist and K. Trivedi. Reliability estimation of fault-tolerant systems: Tools and techniques. *Computer*, 23(7):52–61, 1990. 47