FEDERAL UNIVERSITY OF RIO GRANDE DO SUL
ENGINEERING SCHOOL
DEPARTMENT OF ELECTRICAL ENGINEERING

GUILHERME BAUMGARTEN

GRADUATE PROJECT

DEFINITION AND CLASSIFICATION OF FAULTS OF AN AUTOMOTIVE ELECTRICAL/ELECTRONIC SYSTEM BASED ON THE STANDARD ISO 26262

Porto Alegre
2012

FEDERAL UNIVERSITY OF RIO GRANDE DO SUL
ENGINEERING SCHOOL
DEPARTMENT OF ELECTRICAL ENGINEERING

DEFINITION AND CLASSIFICATION OF FAULTS OF AN AUTOMOTIVE ELECTRICAL/ELECTRONIC SYSTEM BASED ON THE STANDARD ISO 26262

Graduate Project submitted to the Department of Electrical Engineering of the Federal University of Rio Grande do Sul, as part of the requirements to graduate in Electrical Engineering.

SUPERVISOR: PROF. DR. MARCELO GÖTZ

Porto Alegre
2012

FEDERAL UNIVERSITY OF RIO GRANDE DO SUL
ENGINEERING SCHOOL
DEPARTMENT OF ELECTRICAL ENGINEERING

GUILHERME BAUMGARTEN

DEFINITION AND CLASSIFICATION OF FAULTS OF AN AUTOMOTIVE ELECTRICAL/ELECTRONIC SYSTEM BASED ON THE STANDARD ISO 26262

This project was judged appropriate to do justice to the credits of the Discipline "Graduate Project" of the Department of Electrical Engineering and approved in its final form by the Supervisor and the Examining Committee.

Supervisor: _____

Professor Marcelo Götz, UFRGS

Ph.D. by University of Paderborn – Paderborn, Germany

Examining Committee:

Professor Altamiro Amadeu Susin, UFRGS

Ph.D., Grenoble Institute of Technology – Grenoble, France

Professor Carlos Eduardo Pereira, UFRGS

Ph.D., University of Stuttgart – Stuttgart, Germany

Professor Marcelo Götz, UFRGS

Ph.D., University of Paderborn – Paderborn, Germany

Porto Alegre, June 2012

DEDICATION

The present work is dedicated to the people who are or were by my side, whether in a permanent or a temporary relationship, during this journey called life. Many are the objectives which we would like to achieve and usually few are the people who help us to accomplish them. This paradigm brings us to the conclusion that more important than a big interpersonal relationship network is to have a network of true collaborators, either in professional and private life. This work is dedicated to my network of true collaborators.

DEDICATÓRIA

Este trabalho é dedicado às pessoas que estão ou estiveram ao meu lado, seja em uma relação permanente ou temporária, nessa caminhada que se chama vida. Muitos são os objetivos que gostaríamos de alcançar e, muitas vezes, poucas são as pessoas que verdadeiramente nos ajudam a alcança-lôs. Este paradigma nos leva à conclusão de que, talvez, mais importante do que uma grande rede de relações interpessoais, o mais importante seja termos uma rede de verdadeiros colaboradores, seja na vida profissional, seja na vida pessoal. Este trabalho é dedicado a minha rede de verdadeiros colaboradores.

ACKNOWLEDGEMENTS

Firstly, I would like to express my special thanks to my parents, Renato José Baumgarten and Vera Regina Schneider Baumgarten, for being my best and most important fellows during my academic education as well as for always being my north.

Afterwards, I would like to thank Alexandre Baumgarten and Cristiano Baumgarten for the unconditional and reciprocal support in the most difficult moments of our lives. Thank you for being the best brothers ever.

I would like to thank Professor Marcelo Götz for his advice on the subjects related to the present work as well as for his readiness to help me in the most different matters, even when things were going in a different way in relation to the expected.

An important thanks may be given to Professor Achim Rettberg (*Carl von Ossietzky University Oldenburg* e *OFFIS*) and to Markus Oertel (*OFFIS*) for the intellectual advisement and the possibility of performing the present work in partnership with OFFIS (das Oldenburger Forschungs- und Entwicklungsinstitut für Informatik).

I would like to thank Angela Recchia for being always by my side and providing me with the necessary support, whether helping me to figure out a solution for my problems or making me briefly forget them.

Finally, I would like to thank the professors of the UFRGS Electrical Engineering Department for the education that was provided to me and for the sharing of their knowledge.

AGRADECIMENTOS

Primeiramente, gostaria de oferecer um agradecimento todo especial aos meus pais, Renato José Baumgarten e Vera Regina Schneider Baumgarten, pelo fato de terem sido os meus melhores e mais importantes companheiros nessa longa caminhada da minha formação e educação, bem como pelo fato de serem meu norte.

Logo em seguida, não posso deixar de agradecer a Alexandre Baumgarten e Cristiano Baumgarten, pelo apoio incondicional e recíproco nos momentos mais críticos de nossas vidas, sendo sempre os melhores irmãos que poderiam ser.

Gostaria de agradecer ao Prof. Marcelo Götz pela sua orientação neste trabalho, bem como pela sua prontidão a auxíliar-me nos mais diversos assuntos, mesmo quando as coisas pareciam que não sairíam como o esperado.

Agradecimentos importantes devem ser feitos ao Prof. Achim Rettberg (*Carl von Ossietzky University Oldenburg* e *OFFIS*) e a Markus Oertel (*OFFIS*) pela possibilidade de realizar este projeto em parceria com o OFFIS (das Oldenburger Forschungs- und Entwicklungsinstitut für Informatik) e pela orientação intelectual.

Gostaria de agradecer a Angela Recchia por estar sempre ao meu lado e me dar o apoio necessário, seja ajudando-me a encontrar a solução para os meus problemas, seja fazendo-me esquece-lôs momentaneamente.

Para finalizar, gostaria de agradecer aos professores do curso de Engenharia Elétrica da UFRGS pela formação que me foi proporcionada e pelo compartilhamento de seus conhecimentos.

ABSTRACT

The present work aims to construct a table that collects a large set of kinds of failures that can occur in an automotive embedded system and the respective coverage mechanisms of detection and correction of such failures. This table might be used to implement high level description models of next generation automotive vehicles which are intended to be safer and more comfortable. The whole developed work is based in the context of the ISO 26262, the functional safety standard for road vehicles. To support and prove the need of an improvement in the functional safety and security of automotive vehicles, a simulation of a failure in a car brake system is realized using the language VHDL-AMS and the results of this simulation are presented as well. Its objective is to show to the reader that reliable control hardware may avoid severe accidents and under any circumstances the very control system can lead to a generalized system failure which would finish putting people in danger.

Keywords: Automotive Embedded Systems; Fault Model; Reliable Hardware; Coverage Mechanisms of Detection and Correction of Faults; Functional Automotive Safety; ISO 26262; VHDL-AMS.

RESUMO

O presente trabalho tem por objetivo a construção de uma tabela que agrupa um grande número de tipos de falhas que podem ocorrer em um sistema embarcado automotivo e os respectivos mecanismos de cobertura responsáveis pela detecção e correção destas. Esta tabela poderá ser usada na implementação de modelos descritivos de alto nível dos veículos automotivos de próxima geração os quais devem ser mais seguros e confortáveis. Todo o trabalho desenvolvido é baseado no contexto da norma internacional ISO 26262, a norma vigente no que concerne os veículos rodoviários e sua segurança funcional. Para apoiar e provar a necessidade de melhorias referentes a segurança funcional dos veículos automotivos, uma simulação da ocorrência de falha no sistema de freios de um carro é realizada usando a linguagem VHDL-AMS e os resultados desta simulação são posteriormente apresentados. Seu objetivo é mostrar ao leitor que um hardware de controle confiável evita graves acidentes e que, sob hipótese alguma, o próprio sistema de controle pode conduzir a uma falha generalizada do sistema a qual poderia terminar colocando pessoas em risco de acidentes automobilísticos.

Palavras-chave: Sistemas Embarcados Automotivos; Modelo de Falhas; Hardware Confiável; Mecanismos de Cobertura de Detecção e Correção de Falhas; Segurança Automotiva Funcional; ISO 26262; VHDL-AMS.

ABSTRAKT

Die vorliegende Arbeit zielt darauf ab, eine Tabelle mit einer großen Menge von Fehlertypen, die in einem Automotive Embedded System auftreten können, und ihren Mechanismen der Erkennung und Korrektur zu konstruieren. Diese Tabelle kann verwendet werden, um High-Level Modelle der nächsten Generation von Automobilen, mit deren gestiegenen Anforderungen an Sicherheit und Komfort, zu implementieren. Die ganze entwickelte Arbeit ist im Kontext der ISO 26262, der funktionalen Sicherheitsstandard für Straßenfahrzeuge, angesiedelt. Um die Notwendigkeit einer Verbesserung der funktionalen Sicherheit von Kraftfahrzeugen zu unterstützen und beweisen, erfolgt eine Simulation eines Fehlers in einer Auto Bremsanlage mit VHDL-AMS und die Ergebnisse dieser Simulation werden erläutert. Ziel ist es, dem Leser zu zeigen, dass zuverlässige Steuerungshardware die Wahrscheinlichkeit von schweren Unfällen stark reduzieren kann.

Schlüsselwörter: Automotive Embedded Systems; Fehlermodell; Zuverlässige Hardware; Mechanismen der Erkennung und Korrektur von Fehler; Funktionale Automobil Sicherheit; ISO 26262; VHDL-AMS.

RÉSUMÉ

Ce travail vise à construire une table qui réunit un grand nombre de types de défaillances qui peuvent survenir dans un système automobile embarqué et leurs mécanismes de couverture responsables de les détecter et de les corriger. Ce tableau peut être utilisé pour mettre en œuvre des modèles descriptifs de haut niveau pour la prochaine génération de véhicules automobiles qui devraient être plus sûrs et plus confortables. Tout le travail est basé sur le cadre de la norme internationale ISO 26262, la réglementation en vigueur en termes de véhicules routiers et de leur sécurité fonctionnelle. Afin de soutenir et de prouver la nécessité d'améliorer en ce qui concerne la sécurité fonctionnelle des véhicules automobiles, une simulation de l'occurrence d'un défaut du système de freinage d'une voiture est réalisée en utilisant le langage VHDL-AMS et les résultats de cette simulation sont ensuite présentés. Son but est de montrer au lecteur qu'un matériel de contrôle fiable peut éviter des accidents sérieux et que, en aucun cas, le système de contrôle lui-même ne peut conduire à un échec général du système qui pourrait finir par mettre les personnes à risque.

Mots-clés: Systèmes embarqués pour l'automobile; Modèle de défaillances; Matériel fiable; Mécanismes de couverture de détection et de correction des défauts; Sécurité fonctionnelle automobile; ISO 26262; VHDL-AMS.

SUMMARY

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Here, the first column acronyms stand for the writing in the second column:

ALU	Arithmetic and Logic Unit
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
CPU	Central Processing Unit
EAST-ADL	Architecture Description Language (ADL) for automotive embedded systems
ECU	Electronic Control Unit
e.g.	*exempli gratia*[1] (for example)
H&R	Hazard analysis and Risk assessment
i.e.	*id est*[2] (in other words)
IEEE	Institute of Electrical and Electronics Engineers
IP-XACT	XML Schema for meta-data documenting Intellectual Property (IP)
ISO	International Organization for Standardization
ISO 26262	Standard named: Road vehicles – Functional safety
n/a	Not Applicable or Not Available, depending on the occasion
QM	Quality Management
RTE	Run Time Environment
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VHDL-AMS	Analog and mixed-signal extensions for VHDL

[1] A Latin expression.
[2] A Latin expression.

1. INTRODUCTION

1.1. Context and motivation

Nowadays, the automotive industry is reconsidering its products, starting by their basic concepts and going until their composition and final employment. Due to the high requirement by their clients and the future and inevitable lack of raw material used to produce the combustible fuel that are used to power their cars [1], the automobile manufacturers are changing the functional basis of their vehicles, starting in the processes of design and implementation of their cars passing through the final presentation to their clients, these ones being everyday more and more concerned about their safety, security and comfort.

On the one hand, there is a tendency to replace the current combustion system with an electric propulsion based system [2], enabling the next generation of automotive vehicles to be autonomous regarding the use of petroleum. In the other hand, due to the ascending demand for comfort, security and safety during the driving, more and more electronic solutions are being incorporated into the automotive embedded systems, these last ones already filled with plenty of ECUs (Electronic Control Unit). Both last points converge exactly in the idea that the automotive industry is using (and will keep doing it) electronic systems for the most diverse purposes, in an increasing way.

Exemplifying, between the 1970s and the 1990s, the automotive embedded systems modules had only one ECU (a microcontroller or a microprocessor) which was responsible to execute from one to some dozens of specific functions. The modern cars (after the 2000s) have into themselves embedded systems that have up to a hundred ECUs being able to exchange up to 2500 different messages between themselves [3] [4].

The extreme growing in the use of embedded systems on cars brought forth a concern: “if faults occur in the electronic systems, can these faults create risks for people during the driving?”

Taking into account this new concern (safety of drivers during the use of automotive vehicles), a new standard has been created: the ISO 26262 - *Road vehicles* - *Functional safety*. Being based on this new standard, the automotive industry started altering the conceptions and methodologies of the design of its vehicles [5], whether by choice to suit the standard or by the legal obligation and necessity to remain competitive against other similar companies.

However, there is a considerable inertia in the work of suitability, since the automotive industry moves billions of dollars a year and supports, in certain cases, the biggest part of the economy of some countries. Nevertheless, the companies that usually dictate the tendency of the automotive sector, aware of the inevitability of the change, are organizing and investing in new technologies and research.

In this context, it has started developing some research projects with the aim of creating new methodologies and tools to design the next generation of automotive vehicles [6]. To do so, it is essential the confluence of several disciplines as well as the cooperation between different professionals of each subject that is concerned in the project, in order to render these multidisciplinary projects effective and profitable.

1.2. Objectives

A good example of the exposed before is precisely the subject of the present work: an electrical engineering student with general abilities in microcontrollers, microprocessors and embedded systems and an insider with respect to research methodologies who is using his knowledge to perform a fault analysis and to construct a general fault classification table describing these potential faults, at the hardware level of an automotive embedded system, according to ISO 26262 international standard.

This way, the primary goal of the present work is settled:

- Define and classify faults of an automotive electrical/electronic system based on the standard ISO 26262 having as output a general fault table.

Also, as a secondary goal, it is aimed to:

- Demonstrate the possible severity of a car accident for people realizing a simulation of a real accident situation.

1.3. Engineering approach

To accomplish the goals listed prior to the present section, it is used an engineering approach to the task of defining and classifying possible faults that might occur in an embedded system.

Firstly, a research work based on many papers (including IEEE and PubMed[3] papers) and the very standard ISO 26262 is prepared in order to clarify the subject and to set some directions. The results of this research work are presented under the chapter "Theoretical Basis".

Then, to motivate the work and effort of doing such tasks, a simulation in VHDL-AMS [7] of a car accident is done and the results are presented for consideration. The aim here is to prove the severity of a car accident originated due to a fault in the automotive embedded system.

Afterwards, a study about the standard ISO 26262 is presented in order to simplify its operation mode and objectives and to point out to the reader the main ideas.

Next, a table in which the definition and the classification of the possible faults of an automotive embedded system is done and each of their factors is analyzed.

Finally, conclusions are woven and next steps for the present work are proposed.

---

[3] www.pubmed.gov
National Center for Biotechnology Information, U.S. National Library of Medicine
8600 Rockville Pike, Bethesda MD, 20894 USA

2. THEORETICAL BASIS (research-based work)

In this chapter the theoretical basis found in the research sources are presented to the reader. The AUTOSAR consortium is introduced and the industry motivation is exposed. Then the taxonomy concerning the security of given systems is studied. The last chapter of this section gives to the reader an overview of the VHDL-AMS language in order to allow readers, who do not have a specific knowledge about this language, to understand the first chapter of the development section.

2.1. The AUTOSAR consortium

The present work intends, as mentioned before, to develop a descriptive fault table for the hardware level of an automotive embedded systems.

In matter of faults and failures, there is a concern about the safety of people and the property involved. The scientific and industrial communities are increasingly using standards which aim to be a reference and a standardization pattern in several branches of the industry and the university. The main international institution responsible to create and share the standards is the ISO (*International Organization for Standardization*) [8].

When talking about safety for people in automotive vehicles, it is to be expected that there is a standard to regulate the matter. The ISO 26262[4] (*Road vehicles - Functional safety*) was precisely created to regulate the functional safety of automotive vehicles during the driving seeking the safety of people that use them. However, this standard was released not long ago (2011), demonstrating that:

1) Up to now, the industry had its own project and production methodologies and the concern with safety was something of intrinsic, focusing mainly on the car mechanical structure;

[4] This standard is an adaptation of IEC 61508 (Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems) for the automotive industry.

2) Today, the number of electronic components present in a car is very significant and it is inevitable that this number will keep rising during the next years;

3) The complexity of the electronic components and devices as well as the complexity of their interactions (information exchange, data transfer, interfaces, etc.) is increasing day by day.

By analyzing these findings, it can be supposed that the future of automotive vehicles is strongly depending on the embedded systems in it and that the standardization of all the technological aspects of a project, in matter of safety during the utilization of any equipment or device or even with respect to its composition, may always be taken into consideration in order to the proper development of industry and its practices.

Hereupon, the first step to execute this work was the reading and the research work on the standard mentioned before (ISO 26262).

To synthesize a fault analyses table of an automotive system, it is necessary to know how the industry works and its methodologies. Each one of the biggest automotive manufacturers has its own way of working and making cars and also its own project and design methodologies, fact that makes more difficult the generalization and interaction between different projects of diverse cars. Added to this last fact and with respect to the extensive use of electronics in cars, the following point is identified:

- The difficulty of reusing structures of old projects (mainly electric/electronic structures) is becoming a major impediment to innovation and to perform the abandonment of ancient structures: a changing in an automotive embedded system, partially or fully, might result in the need of development of a whole new system. The high level descriptions of these systems are highly hardware-dependents, making impossible any reuse or interchangeability of such high level models.

Consequently and combined with the topic of automotive safety, there is a big problem of reutilization of intellectual material already produced. With this in mind, a European consortium was created to propose solutions to these limitations of methodology, keeping always in mind the safety aspect of driving. This consortium is named AUTOSAR (*AUTomotive Open System ARchitecture*) [9]. This initiative aims at unification of automotive software architecture, by creating an open software architecture designed for the ECUs of the automotive embedded systems. The organization of this consortium can be seen in Figure 1.

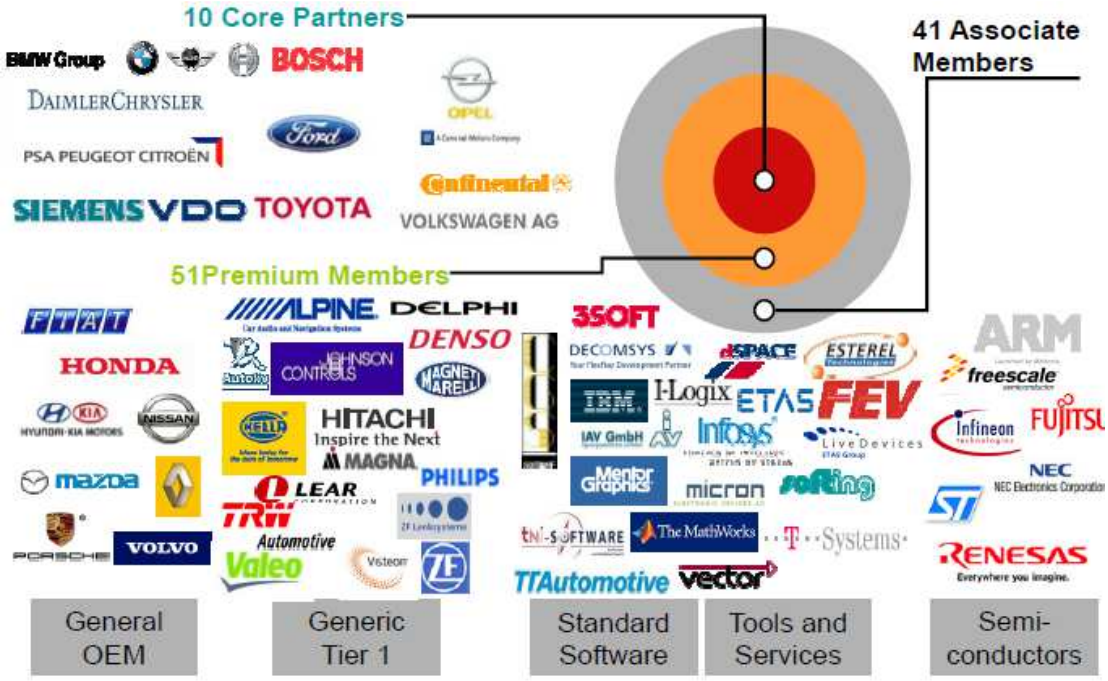


Figure 1 Composition of AUTOSAR [9]

Its main goal is to manage the rising complexity of electric/electronic architectures applied to automotive vehicles developing a project methodology that may promote the emergence of new technologies more efficiently, without affecting the quality of the final product. Alternatively, AUTOSAR wishes to develop a set of specifications that describe software architecture components and to define their interfaces. Its purpose is to improve the management of the complexity of embedded architectures by the improvement of interoperability and reusability of software components [10]. The earnings obtained are:

1) Certified standard software production platforms for suppliers (improvements in quality, easy integration between applications, etc.);

2) Software (hardware-independent) from different vendors reusable between different devices (reducing costs of development, validation, etc.);

3) Facilitation of cooperation between partners (improved formalized exchanges, etc.);

4) Eventual reduction in the number of controllers (cohabitation in the same controller of software from different vendors).

The idea is to separate high level layers (software) from low level layers (hardware), adding among them the AUTOSAR RTE (*AUTOSAR Run Time Environment*) and dealing with the interfaces of each layer, as can be seen from Figure 2.

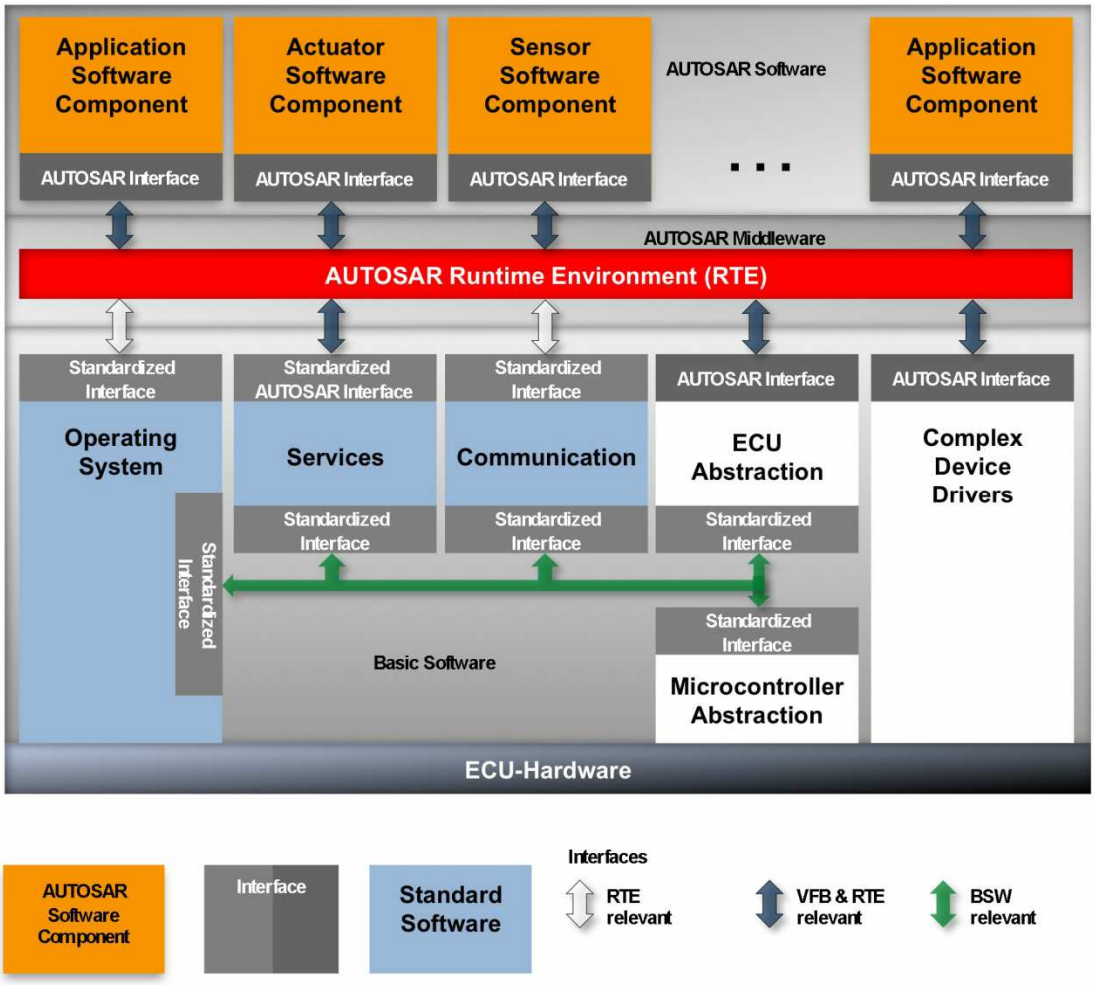


Figure 2 Conception structure of AUTOSAR software architecture [9]

2.2. Basic concepts and taxonomy of dependable and secure computing [11]

In this section, the basic concepts of the dependable and secure computing are discussed and its taxonomy (standardized set of terminologies used within the field of safety, in this case) is developed in order to render the discussion about the standard clear and comprehensible. Though, these concepts are defined in a large sense, enough to define from individual logical systems to complex virtual networks handled by humans and used by lots of users.

The employment of a correct taxonomy is important to realize a good study and classification of faults in a given system.

2.2.1. Function, behavior, structure, and service of a system

A system can be viewed as an entity that interacts with other entities (other systems, including hardware, software, humans, the physical world with its natural phenomena, etc.). These other systems are the environment of the given system.

Talking about computing and communication systems, these are characterized by fundamental properties:

- functionality;
- performance;
- dependability and security;
- cost;
- usability;
- manageability;
- adaptability.

The function of a system is what it is intended to do and is described by the functional specification in terms of functionality and performance. The behavior of a system is what it

does to implement its function and is described by a sequence of states. The total state of a system is the set of the following states:

- computation;
- communication;
- stored information;
- interconnection;
- physical condition.

The structure of a system is what allows it to generate its behavior. From a structural perspective, a system is composed of a set of components put together to interact, where all the components are another systems. The recursion stops when a component is considered to be atomic: any further internal structure cannot be discerned, or is not of interest and can be ignored. Consequently, the total state of such a system is the set of the states of its atomic components.

The service delivered by a system as a provider is its behavior as it is perceived by its user(s). A user is another system that receives service from the provider. The part of the provider's system boundary where service delivery takes place is the provider's service interface. The part of the provider's total state that is perceivable at the service interface is its external state and the remaining part is its internal state.

A system might sequentially or simultaneously be a provider and a user with respect to another system. It can deliver service to and receive service from other system(s).

2.2.2. The threats to dependability and security: failures, errors, faults

Correct service is delivered when the service implements the system function. A service failure (or simply failure) is an event that occurs when the delivered service deviates from correct service. Since a service is a sequence of the system's external states, a service

failure means that at least one external state of the system deviates from the correct service state. This deviation is called an error. The adjudged or hypothesized cause of an error is called a fault. The prior presence of a vulnerability means that an internal fault that enables an external fault to harm the system is necessary for an external fault to cause an error and possibly subsequent failure(s).

The failure of one or more of the services implementing the functions may leave the system in a degraded mode that still offers a subset of needed services to the user. Some of the operational modes after a system starts working in a degraded mode could be:

- slow service;
- limited service;
- emergency service;
- etc.

2.2.3. Dependability, security, and their attributes

The definition of dependability is the ability to deliver service that can justifiably be trusted. Dependability is an integrating concept that includes the following attributes:

- availability: readiness for correct service;
- reliability: continuity of correct service;
- safety: absence of catastrophic consequences on the user(s) and the environment;
- integrity: absence of improper system alterations;
- maintainability: ability to undergo modifications and repairs.

2.2.4. Environments of insertion of a system

The development and use environments where a system is designed and used, respectively, may strongly affect the very system, being responsible for faults and malfunction. These both environments are characterized as follows:

Development environment:

- the physical world with its natural phenomena;
- human developers, some possibly lacking competence or having malicious objectives;
- development tools: software and hardware used by the developers to assist them in the development process;
- production and test facilities.

Use environment:

- the physical world with its natural phenomena;
- administrators (including maintainers): entities (humans or other systems) that have the authority to manage, modify, repair and use the system; some authorized humans may lack competence or have malicious objectives;
- users: entities (humans or other systems) that receive service from the system at their use interfaces;
- providers: entities (humans or other systems) that deliver services to the system at its use interfaces;
- the infrastructure: entities that provide specialized services to the system, such as information sources (time, GPS, etc.), communication links, power sources, cooling airflow, etc.;
- intruders: malicious entities (humans and other systems) that attempt to exceed any authority they might have and alter service or halt it, alter the system’s functionality or

performance, or to access confidential information (hackers, vandals, corrupt insiders, agents of hostile governments or organizations, and malicious software).

2.2.5. General classification of faults

Faults can be sorted into eight different basic categories and each one of these has a double sub-classification, as it can be seen from Figure 3:

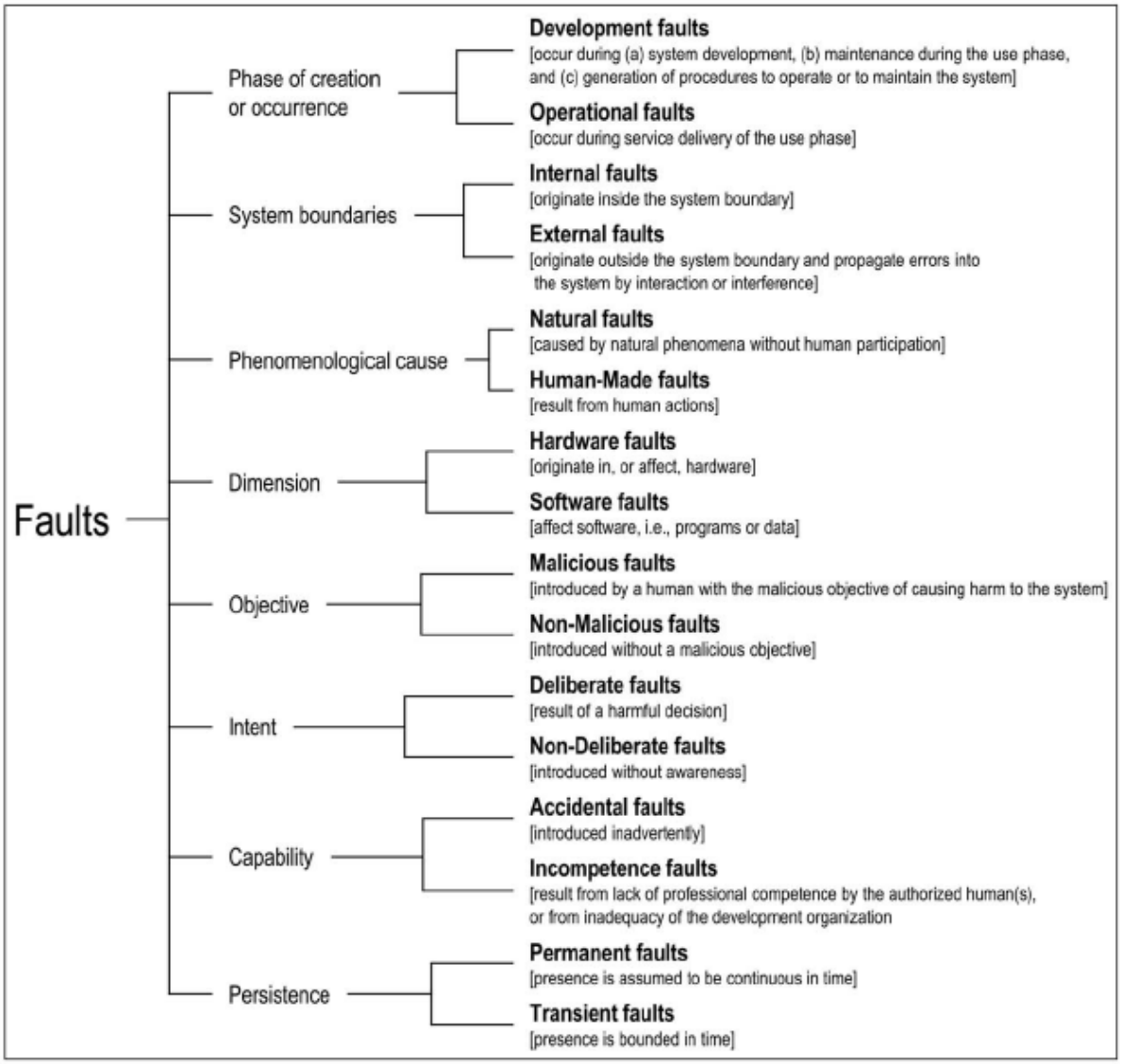


Figure 3 General fault classification [11]

Assuming that all these eight mentioned categories could be combined together to produce diverse kinds of faults, it would exist 256 different possibilities or 256 different combined fault classes (2^8=256). Actually, circa 30 different fault classes were identified up

to now and more combinations may be identified in the future, as further analyses on new or known systems will be made.

2.2.6. The means to attain dependability and security

Over the course of the past 50 years many means have been developed to attain the various attributes of dependability and security. Those means can be grouped into four major categories:

- fault prevention means to prevent the occurrence or introduction of faults;
- fault tolerance means to avoid service failures in the presence of faults;
- fault removal means to reduce the number and severity of faults;
- fault forecasting means to estimate the present number, the future incidence, and the likely consequences of faults.

A way to create more robust systems is to give special attention to the fault tolerance characteristics of these ones. Figure 4 shows some well-known fault tolerance techniques:

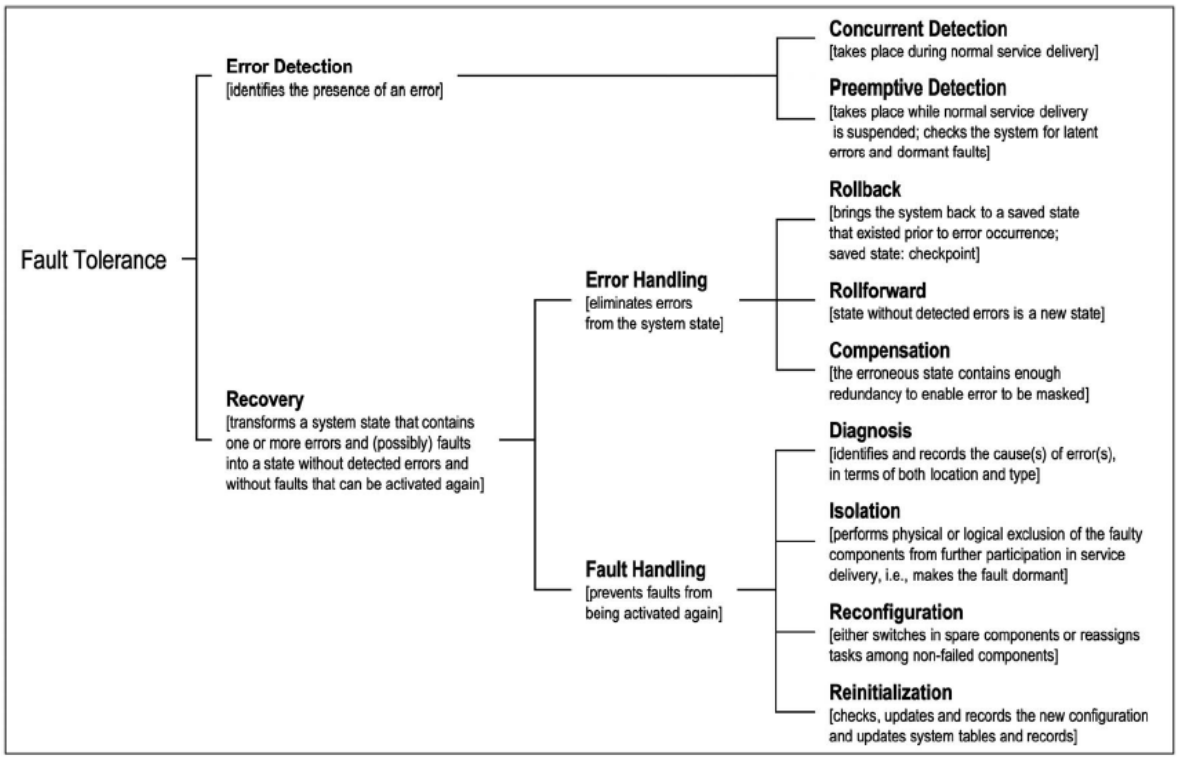


Figure 4 Common fault tolerance techniques [11]

Knowing some techniques of fault tolerance, it is possible to formulate a couple examples of strategies for implementing fault tolerance in any system, as can be seen from Figure 5:

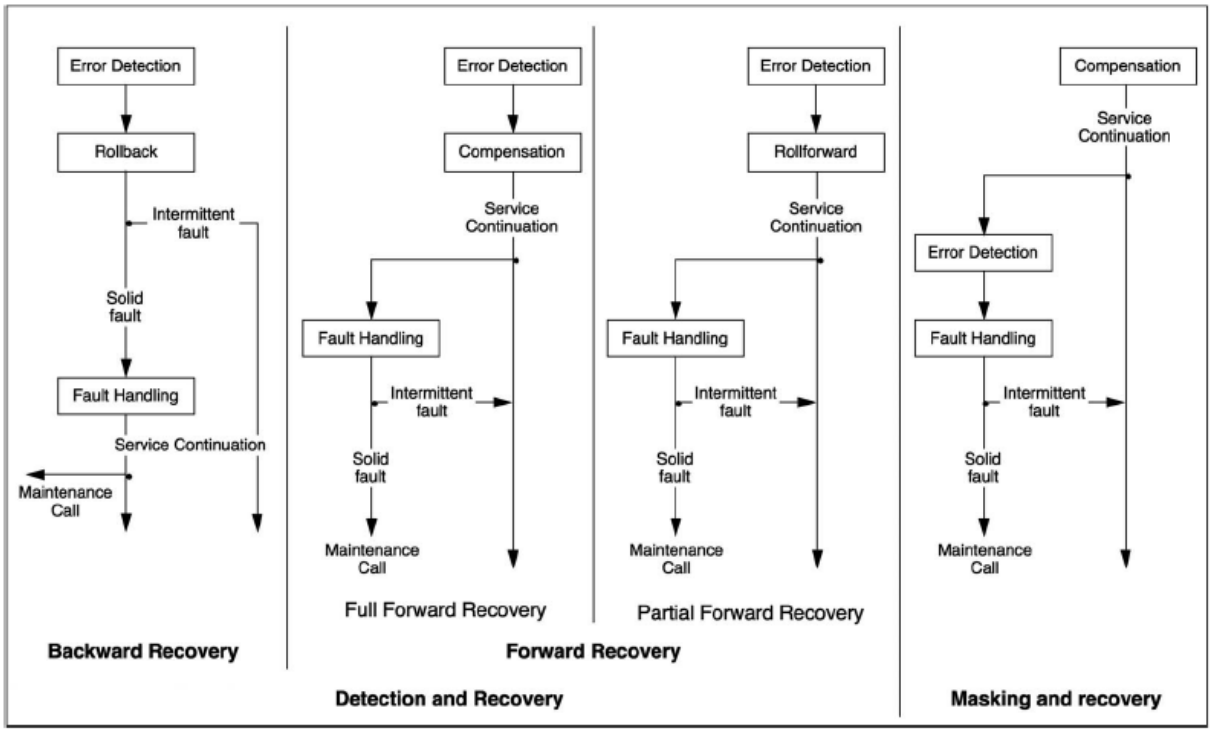


Figure 5 Basic strategies for implementing fault tolerance [11]

2.3. VHDL-AMS [12]

VHDL-AMS (Analog and mixed-signal extensions for VHDL) is a hardware description language which includes the extensions needed to define the behavior of analog and mixed-signal systems. This language derives from the VHDL language (which only defines the behavior of digital systems) and it was created in order to allow designers of analog and mixed-signal integrated circuits to create modules of high-level behavior descriptions of such systems as well as high-level descriptions of whole systems and several components. The difference among the coverage of VHDL and VHDL-AMS is shown in Figure 6:

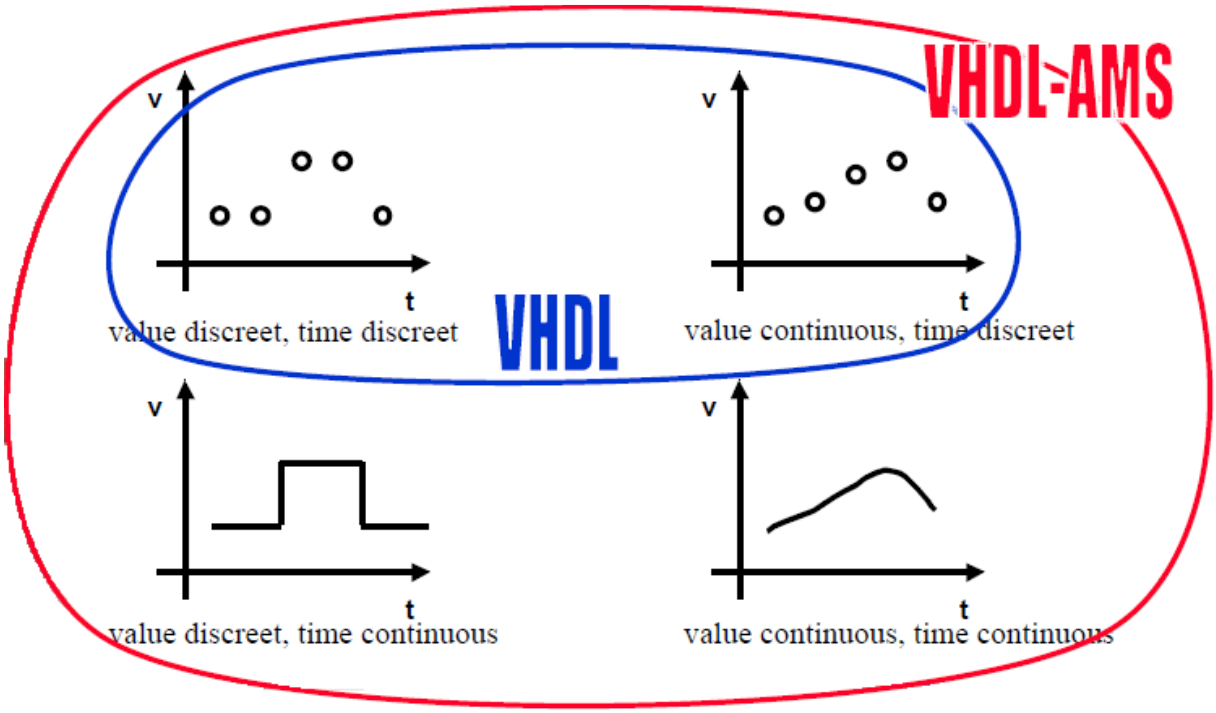


Figure 6 Differences between VHDL and VHDL-AMS couverage [13]

It was originally created to the design of integrated circuits however the possibility of simulation of differential equations and systems with discontinuities made its use possible within the most various physical systems. Nowadays it is widely employed to simulate a varied range of the so-called complex systems.

To accomplish the goal of making a trustworthy analogic simulation of a system, the VHDL-AMS language supports the following set of characteristics [13]:

- continuous models are based on differential algebraic equations;
- differential algebraic equations are solved by a dedicated simulation kernel: the analog solver;
- it supports the handling of initial conditions, piecewise-defined behavior, and discontinuities.

In addition, its semantics are adjusted by respecting the major laws of physics and common system features [13]:

- conservative semantics to model physical systems (e.g. Kirchhoff's law for electrical circuits);
- non-conservative semantics for abstract models (signal-flow descriptions);
- mixed-signal interfaces;
- unified model of time for a consistent synchronization of mixed event-driven/continuous behavior;
- mixed-signal initialization and simulation cycle;
- mixed-signal descriptions of behavior;
- small-signal frequency and noise modeling and simulation.

Information about the language itself and how to code in VHDL-AMS can be found in [12] and [13].

3. PERFORMED WORK AND TECHNICAL DEVELOPMENT

In this section, all the technical work accomplished is presented, going from the VHDL-AMS simulation of a car accident due to a failure of the electronic brake system and its consequences, through the study of the international standard ISO 26262, to the creation and explanation of a table containing a set of possible faults that might occur in an automotive embedded system.

3.1. Failure scenario (motivation): Fault simulation on a car brake system due to the malfunction of the electronic control system (made in VHDL-AMS language)

To illustrate what happens when a fault occurs during the driving and the behavior of the physical system (car with passengers) during this period, a simulation to validate the concern with car security and to quantify the involved physical quantities was performed.

This simulation was elaborated in a VHDL-AMS-based environment using an appropriate compiler: *hAMSter simulation system, Version 2.0 by Ansoft Corporation*.

3.1.1. The model conception

For this simulation, it was collected a set of equations in [14] that define the dynamic behavior of a quarter car and the car tires. The model was constructed using the equations 1 to 6 (presented in the present and next pages). Even if this set of equations is for the dynamic behavior of a car in high velocities (usually more than 1m/s), it can be employed as an approximation for low velocities since it is an application of general purpose.

The first equation is the well-known “Pacejka Magic Formula tire models”:

$$\mu_x = a(1 - e^{-b\lambda} - c\lambda) \quad (1)$$

This equation describes the behavior of the longitudinal friction coefficient μ_x as a function of the wheel slip λ. “a”, “b” and “c” are coefficients related to the road surface. In

the present model these coefficients were defined for a dry asphalt road environment and hence it was applied the following values to them:

$$a = 1.28$$

$$b = 23.99$$

$$c = 0.52$$

Then, it follows a standard set of non-linear equations that describes the dynamic behavior of a given quarter car.

$$J\dot{\omega} = RF_x - sign(\omega)T_b \quad (2)$$

$$m\dot{v} = -F_x \quad (3)$$

$$\lambda = \frac{v - R\omega}{v} \quad (4)$$

$$F_x = F_z\mu_x \quad (5)$$

$$F_z = mg \quad (6)$$

Where:

ω is the angular speed of a wheel [rad/s];

v is the longitudinal speed of the car [m/s];

J is the inertia [kg.m²];

R is the wheel radius [m];

T_b is the brake torque [Nm];

F_x is the longitudinal force [N];

λ is the longitudinal wheel slip [%];

F_z is the vertical force [N];

μ_x is the road friction coefficient [%];

m is the car mass [kg];

g is the gravitational force [m/s²].

Some of these quantities have constant values chosen arbitrarily based on common real values:

$J = 1$ [kg.m²];

$R = 0.32$ [m];

$m = 450$ [kg];

$g = 9.81$ [m/s²].

The input is the brake torque and, starting from its value and the values of the initial conditions of the longitudinal and angular speeds, the values of longitudinal force, longitudinal wheel slip, vertical force and road friction coefficient are calculated. The outputs of the system are the longitudinal and angular speeds, nevertheless all the behavior in time of the other quantities can be selected for display.

The range of validity of the set of equations are: $\omega > 0$, $v > 0$ and hence $-1 < \lambda < 1$.

The whole implemented code for executing the simulation is presented in Appendix 2 (A and B). Here, it is divided into small blocks and each of these ones is explained in detail.

The first block is the description, in VHDL-AMS, of the car model. It was created an entity named “quarter_car_model” that grouped together all the parameters and equations of the car model (including the tire model, to facilitate the coding).

Inputs, terminals, quantities, constants and outputs are defined as follows:

VHDL-AMS model of a quart car model:

```
ENTITY quarter_car_model IS
        GENERIC (a:REAL := 1.28;
                 b:REAL := 23.99;
                 c:REAL := 0.52;
                 J:REAL := 1.0;
                 R:REAL := 0.32;
                 m:REAL := 450.0;
                 g:REAL := 9.81);
    PORT (TERMINAL w1,w2:ROTATIONAL_OMEGA;
          TERMINAL v1,v2:KINEMATIC_V;
          QUANTITY Tb: IN REAL;
          QUANTITY Vi: IN REAL;
          QUANTITY Wi: IN REAL;
          QUANTITY mi_x: REAL;
          QUANTITY lambda: REAL;
```

```
                QUANTITY F_x: REAL;
                QUANTITY F_z: REAL;
                QUANTITY Speed: REAL;
                QUANTITY Angular_Velocity: REAL);
END;
```

Then, the behavior of the car is described using the equations 1 to 6:

VHDL-AMS model of a quart car model: (continuation)

```
ARCHITECTURE behavior OF quarter_car_model IS
        QUANTITY w across af THROUGH w1 TO w2;
        QUANTITY v across lf THROUGH v1 TO v2;
BEGIN
        break v=>Vi;
        break w=>Wi;
        mi_x == a * (1.0 - ((math_e)**((-1.0) * b * lambda)) - c * lambda);
        J * w'dot == (R * F_x) - (Tb * sign(w));
        m * v'dot == (-1.0) * F_x;
        lambda == (v - (R * w)) / (v);
        F_x == F_z * mi_x;
        F_z == m * g;
        Speed == v;
        Angular_Velocity == w;
        break v=>0.0 when not v'above(0.0);
        break w=>0.0 when not w'above(0.0);
END ARCHITECTURE behavior;
```

Once the car model is described, stimuli can be applied to the system to simulate its dynamic response in time. The stimuli are applied for the input brake torque and also for the initial conditions for the longitudinal and angular speeds. This block calls the entity created before and pass the stimuli that were chosen. For each scenario, the code is presented.

3.1.2. Simulations

Some different scenarios were simulated in order to find the consequences of the possible different problems caused by a malfunction of the brake system.

3.1.2.1. Scenario 1: soft braking torque applied instantaneously and resultant force applied to the vehicle passengers

In this scenario, the car is placed in motion. The initial conditions are chosen as follow:

$$v = 3.0\frac{m}{s} = 10.8\frac{km}{h}$$

$$\omega = \frac{v}{R} = 9.375\frac{rad}{s}$$

$$T_b = 0\ Nm$$

After 1 second, a significant braking torque is applied to the system.

$$T_b = 80\ Nm$$

The following code represents the implementation of these changes:

Test bench:

```
ENTITY car IS
        GENERIC (Tb,Vi,Wi:REAL);
END;

ARCHITECTURE behavior OF car IS
        TERMINAL w:  ROTATIONAL_OMEGA;
        TERMINAL v:  KINEMATIC_V;
BEGIN
        PROCESS BEGIN
                Tb <= 0.0;
                Vi <= 3.0;
                Wi <= 9.375;
    WAIT FOR 1000ms;
                Tb <= 80.0;
    WAIT;
        END PROCESS;
  Sim1:  ENTITY quarter_car_model (behavior)
                  PORT MAP (w, rotational_omega_ground, v, kinematic_v_ground, Tb, Vi, Wi);
END ARCHITECTURE behavior;
```

On the moment in which the braking torque is applied, the longitudinal force presents a variation in its behavior. The behavior in time of the longitudinal and angular speeds and the force applied to the people inside the vehicle are shown in Figure 7:

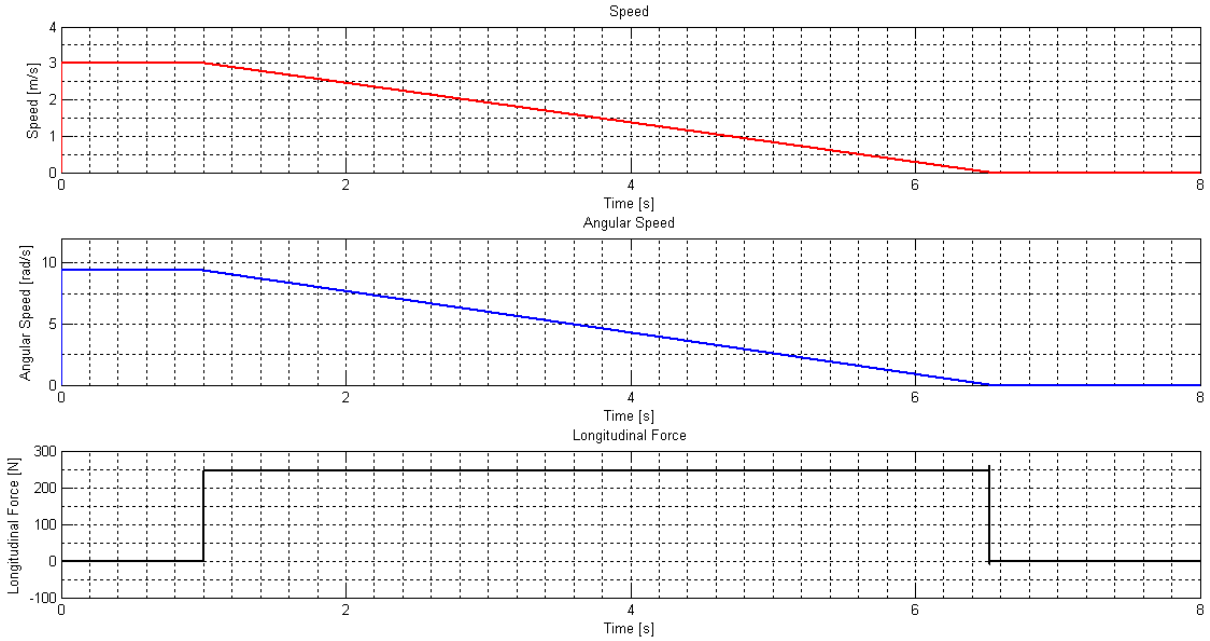


Figure 7 Results from the first simulation

It is possible to verify that on the moment of the beginning and of the end of the braking, a really significant change in the magnitude of the force occurs. It is about a change of 245 N in a period of time tending to zero. This change could cause from health issues to the death of people.

Appendix 2.A shows the whole implemented code for this scenario.

3.1.2.2. Scenario 2: hard braking torque applied instantaneously and the consequences for the car dynamic and the vehicle passengers

Also in this scenario, the car is in motion with:

$$v = 3.0\frac{m}{s} = 10.8\frac{km}{h}$$

$$\omega = \frac{v}{R} = 9.375\frac{rad}{s}$$

$$T_b = 0\ Nm$$

Abruptly, a really high braking torque is applied to the system. Exactly in this moment the longitudinal force presents a peak in its amplitude value. The behavior in time of the longitudinal and angular speeds and the force applied to the passengers are shown in Figure 8:

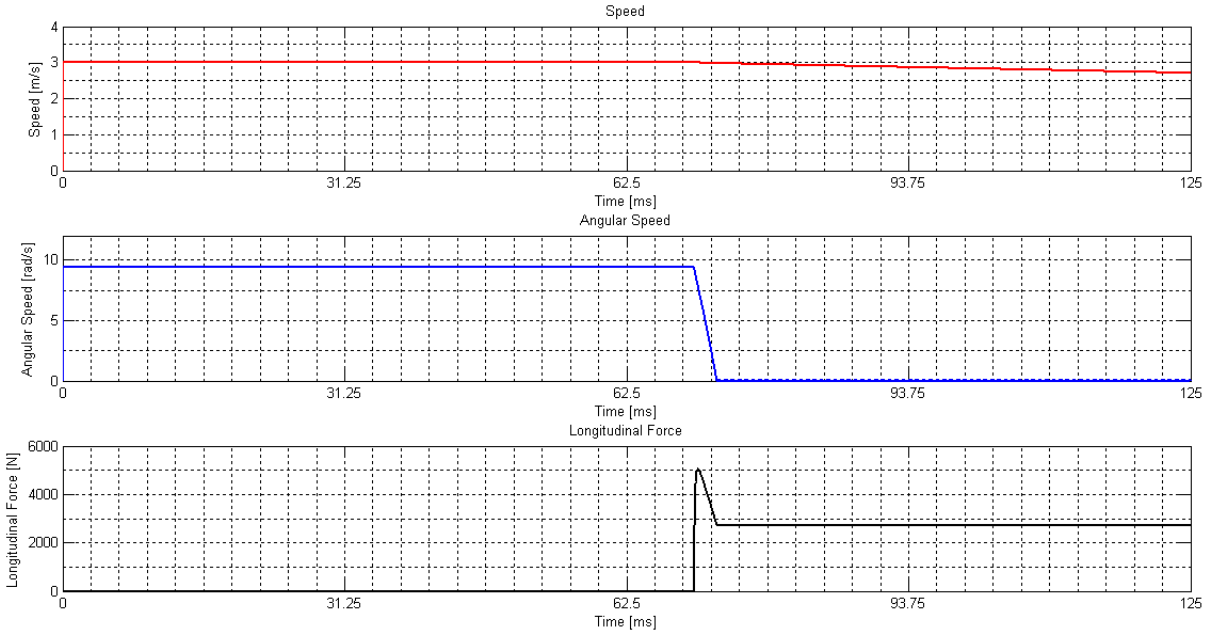


Figure 8 Results from the second simulation

For a better comprehension of the movement during the braking and the consequences of it to the longitudinal force, it is shown in Figure 9 a zoom of the graph region of interest:

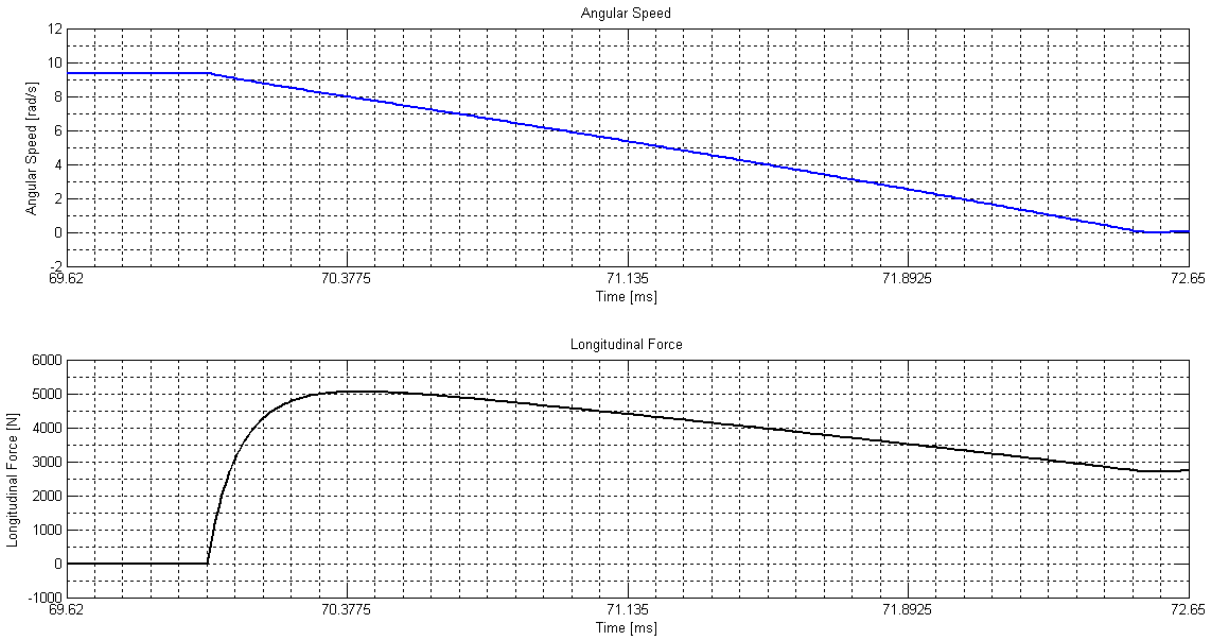


Figure 9 Peak of the longitudinal force during the braking

It can be seen that an expressive variation in the magnitude of the longitudinal force happens. The maximum value is about 5000 N. For a man who weighs 80 kg, it is an applied impact equivalent to about 6.5 times the standard gravity.

The exposure to the fast variation of this quantity is a big issue for people's health, even if it is not easy to determine the limits of this variation [15].

Appendix 2.B shows the whole implemented code for this scenario.

3.2. An ISO 26262-oriented study [16] [17]

To create an ISO 26262-based fault table, it is obviously required from the designer the knowledge about this standard. The first step in doing so is to get the standard and make a research study about it. The following study was totally based on the latest release of ISO 26262.

3.2.1. Category classification

The first step in applying the ISO 26262 is to perform a Hazard analysis and Risk assessment (**H&R**) according to the standard. In order to realize such analysis, it is required to identify the **vehicle-level hazards** that are related to physical injuries or any damage to the health of people who are inside the vehicle or in its neighborhood, as the result of vehicle malfunction and by consequence to failures and unwanted behaviors of any system responsible for a vehicle function.

Once the identification of vehicle-level hazards is done, they can be categorized in accordance with ISO 26262 as **hazardous events**. It means that any vehicle-level hazard classified as a hazardous event is a relevant combination of an operational situation of the vehicle and a risk that may lead to an accident if not handled timely. Figure 10 illustrates the

rapport between a system failure, the corresponding hazard, the hazardous event and the accident:

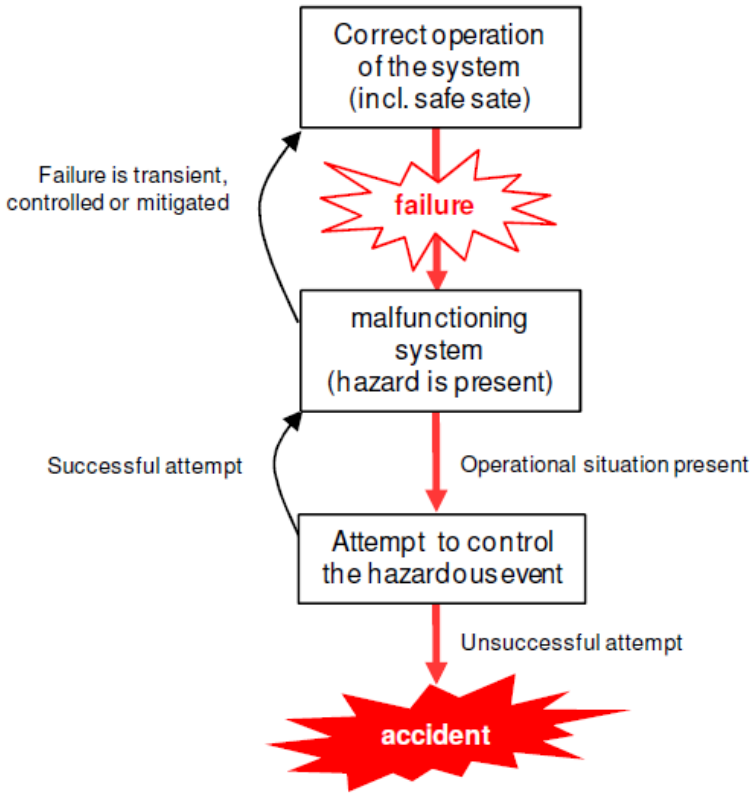


Figure 10 An accident led by a sequence of failure and operational situation not successful attempt [17]

3.2.2. Primary allocation

The way the standard found to quantify these hazardous events was to assign an **ASIL** (Automotive Safety Integrity Level) to each different one. The ASIL classification method determines three parameters of association: the exposure “E”, the controllability “C” and the severity “S”. For each of these categories, there are four classification possibilities: from ASIL D to ASIL A, representing the most demanding regarding a necessary risk reduction and the less demanding, respectively. If an event represents no safety relevance, it is assigned as a “QM” (Quality Management).

The ASIL method is used within the standard to specify the risk reduction measures concerning the design of each vehicle system and of its hardware and software components.

These measures intend to avoid either residual probability of random hardware failures and systematic failures.

“E” is the probability of exposure of a given vehicle to an operational situation which may lead to a hazardous event when a failure of the system happens and it can be reported in terms of frequency of occurrence of an operational situation or its duration.

“C” is a parameter associated to the ability of people to react timely when a hazardous event occurs, in order to avoid any harm originated from it.

“S” characterizes an estimation of the harm to people resulting from a hazardous event when it is not controlled timely.

This way, “S” represents directly the severity to people due to the risk from a hazardous event and “E” and “C” together with the residual failure rate of a system give the likelihood of occurrence of the risk.

The ASIL method natural classification sequence is shown in Figure 11:

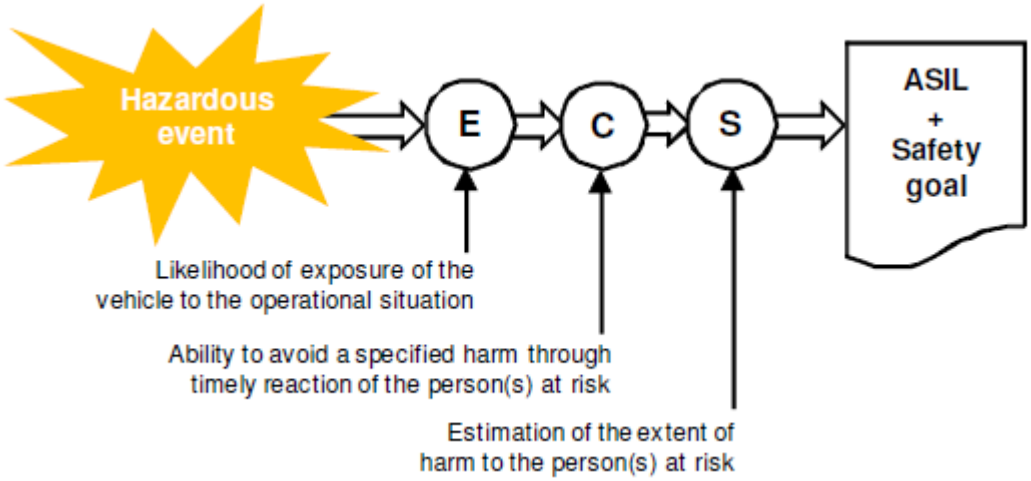


Figure 11 Overview of the ASIL classification method [17]

Discrete scales are assigned to parameters “E”, “C” and “S”, and the final ASIL to the hazardous event takes into consideration the combination of three of them by using a risk graph. Logically, if there is more than one hazardous event related to the same failure, then the highest ASIL may be chosen for characterizing such event.

When an ASIL is assigned to a hazardous event, a **safety goal** is always determined within and it inherits the ASIL of the corresponding hazardous event. The safety goal represents a top-level safety requirement that is assigned to the system as a whole. The rule of choosing the highest ASIL in case of association of various hazardous events may be also applied to the safety goal.

3.2.3. Allocation to components of the system

To start designing the system, after that the H&R is done (usually early in the process of development of a system, during the concept period), a set of safety goals and their corresponding ASIL are applied throughout the system design, the electronic hardware design and the software design.

3.2.4. The means to attain a safety level by using dependable architecture

The following design rule from [17] may be applied from the beginning of a system concept phase:

An initial safety requirement is decomposed to redundant requirements implemented by sufficiently independent architectural components, where each decomposed requirement complies with the initial safety requirement by itself.

An example of ASIL classification for a given set of suggested methods according to each one of the possible ASIL categories [16] can be seen in Table 1. It gathers some hardware integration tests to verify robustness and operation under external stresses. The following classification is used, as well:

- "++" indicates that the method is highly recommended for the identified ASIL;
- "+" indicates that the method is recommended for the identified ASIL;

- “o” indicates that the method has no recommendation for or against its usage for the identified ASIL.

Table 1 Example of ASIL classification of hardware integration tests according to ISO 26262 [16]

Methods	ASIL			
	A	B	C	D
Environmental testing with basic functional verification	++	++	++	++
Expanded functional test	o	+	+	++
Statistical test	o	o	+	++
Worst case test	o	o	o	+
Over limit test	+	+	+	+
Mechanical test	++	++	++	++
Accelerated life test	+	+	++	++
Mechanical Endurance test	++	++	++	++
EMC and ESD test	++	++	++	++
Chemical test	++	++	++	++

3.3. Characterization of possible system failures that might occur on today’s embedded systems (methodology of classification)

The objective of this section is to detail a descriptive table (fully presented in Appendix 1) that was created to put together a really high level description and classification of detected failures acting on modern embedded systems and the mainly employed coverage techniques to detect such failures. These failures were described and classified following the standard ISO 26262. To get started, it is explained each one of the fields (columns) of the table so that the reader is able to understand the chosen arrangement.

First of all, the integrality of an embedded system can be divided into two main categories: hardware and software. The hardware is the material part of a system (microcontrollers or microprocessors, electronic components, interfaces, physical structures,

etc.) and the software is the programmable or virtual part of a system (operational systems, software routines, user interfaces, data processing, etc.).

The whole work is focused on the hardware of the embedded systems. Consequently, all the failures were firstly classified as originated in the hardware part of the embedded system. However, software coverage techniques are an important part of the solution for the problem of detecting failures in such systems and are listed as well.

3.3.1. Primary grouping: Level (in Hardware)

This is the first column of the table and, as the description proposes, it divides widely the level or the layer in hardware in which a given failure can occur. The following levels for this category were created:

- Electrical elements - Actuator Electrical System;
- Electrical elements - Electronic Measurement System;
- Electrical elements - Final elements;
- Electrical elements - General Electrical/Electronic Embedded System;
- General elements - Electrical and Electronic Car System;
- General semiconductor elements - General Electronic Embedded System / Controller;
- General semiconductor elements - Power Supply System;
- Specific semiconductor elements - Communication / General Electronic Embedded System;
- Specific semiconductor elements - Communication / Microprocessors;
- Specific semiconductor elements - Processing Unit / Controller.

3.3.2. Fault location (Element)

According to the preceding classification and going deeper in the hardware level, the elements of the hardware where faults occur were better defined. It gives the possibility of looking for more effective techniques of fault detection. Some examples of categories for this column are:

- Electronic Embedded System;
- CPU;
- Digital I/O;
- Non-volatile memory;
- Etc.

3.3.3. Applicable to following faults

This field is filled with the faults that can occur in the previous hardware elements. The description can be a general or a specified explanation of a set of given failures according to the amount of information found for such elements. For instance, there is for the *Final elements of an electrical system* a general description as *Failures in electromechanical elements* and for *Volatile memory* a specific description as *Unwanted bit-flips due to radiation*.

3.3.4. Mechanism name of diagnostic coverage

Then, mechanisms of diagnostic coverage for fault detection are proposed. This is one of the mainly points that bring us to the creation of the present table. In order to improve the functional safety and security of an automotive embedded system, it is required to implement coverage techniques for fault detection which are strong reliable with respect to the probability of detection of a set of given faults.

In the nearly future, these descriptions are going to be used for researchers and industry to create high level models of each hardware element. Such models may be used to prove that a given hardware is reliable regarding a given level of reliability of, for example, 99.99999% (five nines after the decimal mark or even more), as well as to implement more complex systems from ordinary elements. For high availability systems which must achieve a great rate of near-continuous availability, fault tolerance techniques are substantially required. The concept of fault tolerance is connected to the notion of detection and correction of faults.

As an example of technique for software fault tolerance used mainly in military, nuclear and medical projects which require really reliable software, it can be mentioned a technique called "*N-Version Programming*" (sometimes known as *Dissimilar Software*). It is the software analogy of hardware N-Plexing though it is not as simple as the replication of N-Plexing, since if N copies of the same software were running they would simply contain the same software faults and produce the same software errors N times. So if N units of some software functionality need to run in parallel, they need to be N disparate implementations of that functionality – independently implemented by N separate development teams. It is also applicable to the program that runs in a microprocessor or other element of an embedded system.

3.3.5. Probability of detection of faults (qualitative or quantitative)

For each one of the coverage mechanisms, a probability of fault detection may be calculated or, at least, has its lower boundary estimated. The high level model simulators use this information to determine a global level of reliability of a hardware element that is under the inspection of a given coverage mechanism. This information was collected in the very standard ISO 26262 or other complementary documents, as referenced in the table. Sometimes there is no specific study about the reliability of a given mechanism so a

qualitative classification is done: *high, medium* or *low* probability of detection of faults. This qualitative classification is not formal and it cannot be used to replace a quantitative estimation of the probability, although the ISO 26262 uses it.

3.3.6. Correction (Aim)

One more set of information about the coverage mechanisms is presented in this column. It is a brief description of the aim of each one of the listed mechanisms.

3.3.7. Mechanisms Description

To understand the way a mechanism works, a description of it is done. These descriptions can be used to determine a method to calculate the mechanism probability of detection of faults.

3.3.8. Generalization

When applicable, a generalization of the behavior of the mechanisms between that ones that are listed is done. This is a way to simplify the approach within we attack a probability definition problem.

3.3.9. Source

To support the provided information, for each case the sources are mentioned. It is mainly composed of the fifth chapter of the ISO 26262. Some IEEE papers and others were used as well.

3.4. Explanation of the accomplished work: the case of CPU-located faults and some of the respective coverage mechanisms.

In this section, one of the tables that were constructed is explained in order to elucidate its meaning and to demonstrate the line of reasoning behind its construction. The same sub items of the section *3.3* are used.

As the present objective is to talk about CPUs, to get started, the level in the hardware is defined. A good classification for it would be: *Specific semiconductor elements - Processing Unit / Controller*. The element studied is, as defined before, the *CPU*.

Next, the most common failures that occur in CPUs were listed as follow:

i. *Failures in processing units which lead to incorrect results:*

- *Physical chip damage;*
- *Local damage caused by overheating;*
- *Radiation;*

ii. *Illegal hardware exceptions.*

For each one of the previous failure classification, coverage mechanisms of detection of faults were proposed together with its aim, probability of detection of faults, description and the source from where this information was gathered.

For *i.*, the following mechanisms can be applied:

a. Name of the mechanism: *Self-test by software*.

I. Probability of detection of faults: *Medium (Depends on the quality of the self-test). Up to 96.5% for the opcode-based self-test can be reached.*

II. Correction (Aim): *To detect, as early as possible, failures in the processing unit and other sub-elements consisting of physical storage or functional units, or both, by means of software.*

III. Mechanisms Description: *The failure detection is realized entirely by software which perform self-tests using a data pattern, or set of data patterns, to test the physical storage or the functional units or both.*

IV. Source:

- *SafeTCore Library (Infineon's Website) and Intelligent functional safety concept saves design time and costs by Manfred Choutka, Infineon Technologies;*
- *ISO 26262 - part 5.*

b. Name of the mechanism: *Self-test by software cross exchange between two independent units*.

I. Probability of detection of faults: *Medium (Depends on the quality of the self-test).*

II. Correction (Aim): *To detect, as early as possible, failures in the processing unit consisting of physical storage and functional units.*

III. Mechanisms Description: *The failure detection is realized entirely by means of two or more processing units each executing additional software functions which perform self-tests to test the physical storage (data and address registers) and the functional units. The processing units exchange the results. This test provides very limited or no coverage for soft errors.*

IV. Source: *ISO 26262 - part 5.*

c. Name of the mechanism: *Self-test supported by hardware*.

I. Probability of detection of faults: *Medium (Depends on the quality of the self-test).*

II. Correction (Aim): *To detect, as early as possible, failures in the processing unit and other sub-elements, using special hardware that increases the speed and extends the scope of failure detection.*

III. Mechanisms Description: *Additional special hardware facilities support self-test functions to detect failures in the processing unit and other sub-elements at a gate level. The test can achieve high coverage. Typically only run at the initialization or power-down of the processing unit due to its intrusive nature. Typical usage is for multipoint fault detection.*

IV. Source: *ISO 26262 - part 5.*

Then, for a., b. and c. a generalized description is proposed: *The mechanisms are based on self-tests concepts either by software or hardware and intend to find failures using a self-verification algorithm. The exact probability of finding failures is quite hard to find out but general ideas and a general qualitative description can be done. Usually these self-test have a medium classification in terms of quality of coverage of the given system.*

Following an analogous line of reasoning, the subsequent classification was done for *ii.*:

Name of the mechanism: *Integrated Hardware consistency monitoring*.

I. Probability of detection of faults: *High (Coverage for illegal hardware exceptions only).*

II. Correction (Aim): *To detect, as early as possible, illegal conditions in the processing unit.*

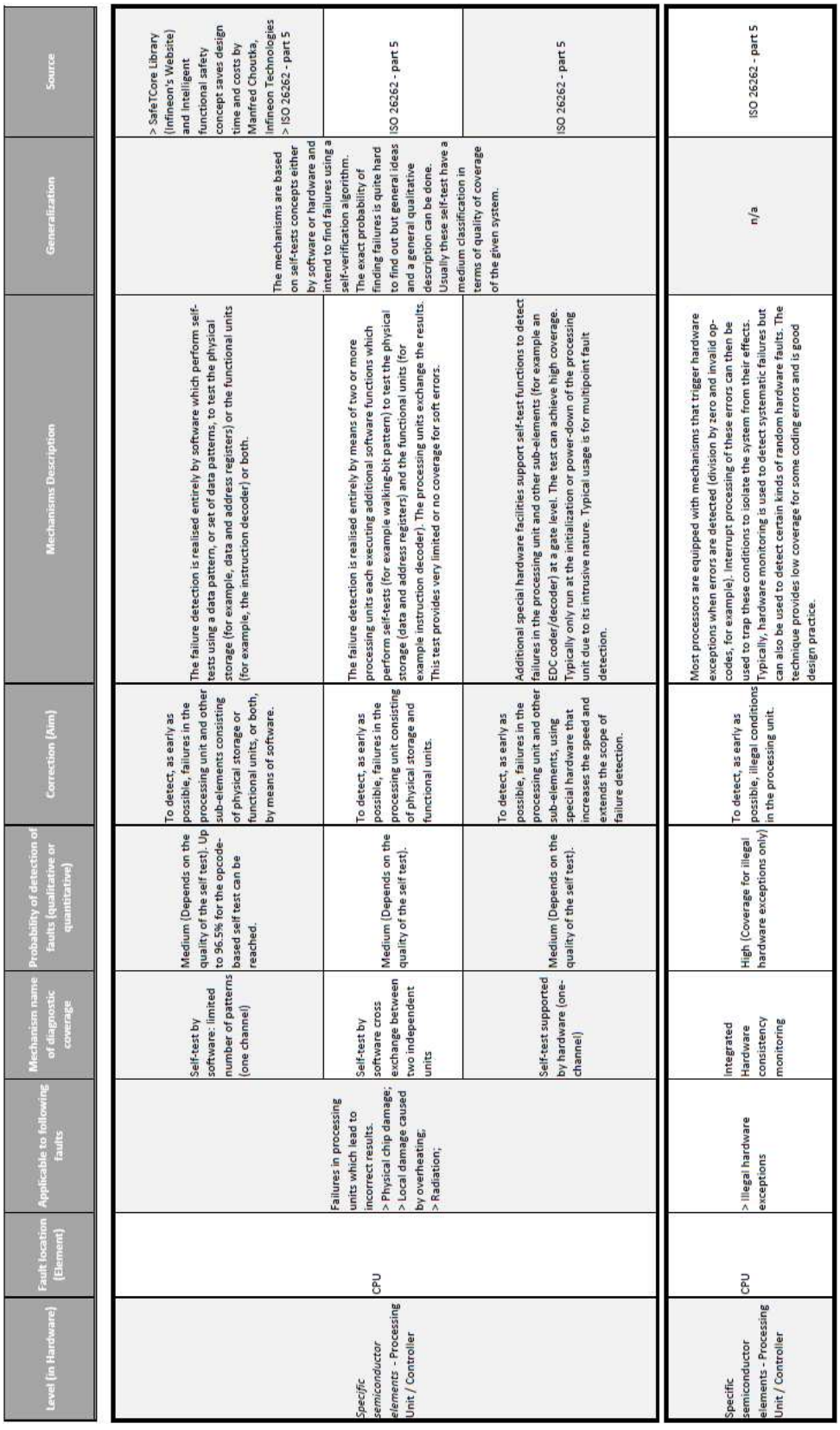
III. Mechanisms Description: *Most processors are equipped with mechanisms that trigger hardware exceptions when errors are detected (division by zero and invalid op-codes, for example). Interrupt*

processing of these errors can then be used to trap these conditions to isolate the system from their effects. Typically, hardware monitoring is used to detect systematic failures but can also be used to detect certain kinds of random hardware faults. The technique provides low coverage for some coding errors and is good design practice.

IV. Source: *ISO 26262 - part 5*.

No generalization was possible to be done for this mechanism, so the corresponding field was filled with an *n/a*.

Figure 12 shows the result of the classification methodology presented previously. For a better view, please consult Appendix 1.



Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
Specific semiconductor elements - Processing Unit / Controller	CPU	Failures in processing units which lead to incorrect results: > Physical chip damage; > Local damage caused by overheating; > Radiation;	Self-test by software: limited number of patterns (one channel)	Medium (Depends on the quality of the self test). Up to 96.5% for the opcode-based self test can be reached.	To detect, as early as possible, failures in the processing unit and other sub-elements consisting of physical storage or functional units, or both, by means of software.	The failure detection is realised entirely by software which perform self-tests using a data pattern, or set of data patterns, to test the physical storage (for example, data and address registers) or the functional units (for example, the instruction decoder) or both.	The mechanisms are based on self-tests concepts either by software or hardware and intend to find failures using a self-verification algorithm. The exact probability of finding failures is quite hard to find out but general ideas and a general qualitative description can be done. Usually these self-test have a medium classification in terms of quality of coverage of the given system.	> SafeTCore Library (Infineon's Website) and Intelligent functional safety concept saves design time and costs by Manfred Choutka, Infineon Technologies > ISO 26262 - part 5
			Self-test by software cross exchange between two independent units	Medium (Depends on the quality of the self test).	To detect, as early as possible, failures in the processing unit consisting of physical storage and functional units.	The failure detection is realised entirely by means of two or more processing units each executing additional software functions which perform self-tests (for example walking-bit pattern) to test the physical storage (data and address registers) and the functional units (for example instruction decoder). The processing units exchange the results. This test provides very limited or no coverage for soft errors.		ISO 26262 - part 5
			Self-test supported by hardware (one-channel)	Medium (Depends on the quality of the self test).	To detect, as early as possible, failures in the processing unit and other sub-elements, using special hardware that increases the speed and extends the scope of failure detection.	Additional special hardware facilities support self-test functions to detect failures in the processing unit and other sub-elements (for example an EDC coder/decoder) at a gate level. The test can achieve high coverage. Typically only run at the initialization or power-down of the processing unit due to its intrusive nature. Typical usage is for multipoint fault detection.		ISO 26262 - part 5
Specific semiconductor elements - Processing Unit / Controller	CPU	> Illegal hardware exceptions	Integrated Hardware consistency monitoring	High (Coverage for illegal hardware exceptions only)	To detect, as early as possible, illegal conditions in the processing unit.	Most processors are equipped with mechanisms that trigger hardware exceptions when errors are detected (division by zero and invalid op-codes, for example). Interrupt processing of these errors can then be used to trap these conditions to isolate the system from their effects. Typically, hardware monitoring is used to detect systematic failures but can also be used to detect certain kinds of random hardware faults. The technique provides low coverage for some coding errors and is good design practice.	n/a	ISO 26262 - part 5

Figure 12 Example of the constructed table

The totality of the table presented in Appendix 1 was filled using an analogous reasoning to the exposed in the present section.

4. CONCLUSIONS

It seems to be quite clear that the automotive sector will depend more and more of the electronic technology. People also may concern themselves about safety and security, not accepting anymore that accidents happen due to the malfunction of electronic parts of the car.

Also, it is quite obvious that there is a more expressive concern about the comfort during the driving. Usually, the way to become a car more comfortable is to add automatic functionalities to the ones already in use that make easier simple actions as open a window or find an address using a GPS. All these functionalities are implemented using electronic systems.

The natural way to ally both requirements is to develop a new methodology of car designing that takes into consideration the reliability of the hardware designed together to its processing power in order to allow the execution of modern electronic systems that are, at the same time, reliable and people security- and safety-concerned.

The automotive industry looks for a new process of developing such reliable and modern cars, not only because it is concerned about safety and security of people but also because the law requires it. This process of developing more reliable vehicles implies an enormous investment. Consequently, industry wants to invest in the right technology that will return the invested money back to it as soon as possible.

The AUTOSAR consortium is an effort to develop together a new methodology to be used in the present case and its methodology seems to be very profitable, since it enables the exchange and reuse of technology from different companies and projects in other companies and future projects. Also, it helps research teams to work together and optimize their researches.

The current phase of development of this whole process is to create high level models for the electronic embedded automotive systems using, to simulate such models, a high level

description and an estimation of the probability that a system failure occurs without its detection and correction.

To construct such models, tables as the one presented in this work may be created and refined, adding important information and more precisely probability equations. Every single electronic element should be characterized using this methodology.

This is an extensive work and it will be necessary lots of man-hours to accomplish the desirable results. However it will contribute significantly to the transformation of the automotive industry and its suitability to the new safety and security rules.

4.1. Further work

A lot of work can be done to refine the one here started. There are many possibilities:

- Refine the tables and complete it with a more extensive set of faults, electronic elements of an electronic embedded system and coverage techniques (research in different sources of information, mainly in IEEE papers);
- For the coverage techniques for fault detection and correction listed in the table, analytical expressions should be constructed.

Some of this proposed work will be continued as part of an internship which will be held at OFFIS[5], in Oldenburg, Germany. This work will be mostly done in the area of conception and design of an electronic car module model using the information collected during the accomplishment of the present work.

[5] OFFIS e.V.: Oldenburger Forschungs- und Entwicklungsinstitut für Informatik-Werkzeuge und -Systeme (Oldenburg Research and Development Institute for Information Technology Tools and Systems)

5. BIBLIOGRAPHY

[1] G. Caruso, "When will World Oil Production Peak?," in *Annual Asia Oil and Gas Conference*, Kuala Lupur, Malasia, 2005.

[2] N. Mutoh and Y. Takahashi, "Front-and-rear-wheel-independent-drive type electric vehicle (FRID EV) with the outstanding driving performance suitable for next-generation adavanced EVs," in *Vehicle Power and Propulsion Conference*, Dearborn, MI, 2009.

[3] E. Armengaud, A. Tengg, M. Driussi, M. Karner, C. Steger and R. Weiss, "Automotive software architecture: Migration challenges from an event-triggered to a time-triggered communication scheme," in *Seventh Workshop on Intelligent solutions in Embedded Systems*, Ancona, 2009.

[4] P. Hansen, "New S-Class Mercedes: Pioneering Electronics," in *The Hansen Report on Automotive Electronics*, 2005, pp. 1-2.

[5] R. Palin, D. Ward, I. Habli and R. Rivett, "ISO 26262 safety cases: Compliance and assurance," in *6th IET International Conference on System Safety*, Birmingham, 2011.

[6] S.-H. Jeon, J.-H. Cho, Y. Jung, S. Park and T.-M. Han, "Automotive hardware development according to ISO 26262," in *13th International Conference on Advanced Communication Technology (ICACT)*, Seoul, 2011.

[7] K. Jaber, B. Ben Saleh, A. Fakhfakh and R. Neji, "Modeling and simulation of electrical vehicle in VHDL-AMS," in *IEEE International Conference on Electronics, Circuits, and Systems*, Yasmine Hammamet, 2009.

[8] ISO, "International Organization for Standardization," ISO, 2012. [Online]. Available: http://www.iso.org/iso/home.html. [Accessed 16 06 2012].

[9] AUTOSAR Partnership, "Official website of the AUTOSAR Partnership," AUTOSAR Partnership, 2012. [Online]. Available: http://www.autosar.org/. [Accessed 16 06 2012].

[10] D. Kum, G.-M. Park, S. Lee and W. Jung, "AUTOSAR migration from existing automotive software," in *International Conference on Control, Automation and Systems*, Seoul, 2008.

[11] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell and Carl Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11-33, Jan.-March 2004.

[12] Ernst Christen and Kenneth Bakalar, "VHDL-AMS-a hardware description language for analog and mixed-signal applications," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 46, no. 10, pp. 1263-1272, Oct 1999.

[13] Dolphin, "VHDL-AMS Introduction," [Online]. Available: http://www.dolphin.fr/projects/macros/dissemination/pdf/tutorial_vhdl_ams.pdf. [Accessed 20 05 2012].

[14] Goddard Consulting, "Simulink - Slip Control of a Quarter Car Model," Goddard Consulting, 2003-2012. [Online]. Available: http://www.goddardconsulting.ca/simulink-quarter-car-model.html. [Accessed 21 05 2012].

[15] G. Matthes, U. Schmucker, M. Frank, C. Huwer, A. Ekkernkampet und D. Stengel, "Notarztliche Einschatzung der Verletzungsschwere am Unfallort: Diagnostischer Wert technischer Parameter - Ergebnisse einer Pilotstudie," *Unfallchirurg*, Mar 31, 2012.

[16] *ISO 26262 "Road vehicles – Functional safety"*, 2011.

[17] Jean-Paul Blanquart, Jean-Marc Astruc, Philippe Baufreton, Jean-Louis Boulanger, Hervé Delseny, Jean Gassino, Gérard Ladier, Emmanuel Ledinot, Michel Leeman,

Joseph Machrouh, Philippe Quéré, Bertrand Ricque, "Criticality categories across safety standards in different domains," in *ERTS² Embedded Real Time Software and Systems*, Toulouse, 2012.

[18] A. S. Hornby, Oxford Advanced Learner's Dictionary of Current English 7th edition, Oxford: Oxford University Press, 2005.

APPENDIX 1

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
General elements - Electrical and Electronic Car System	Electronic Embedded System	Failures in electromechanical elements	Failure detection by on-line monitoring	Low (Depends on diagnostic coverage of failure detection)	To detect failures by monitoring the behaviour of the system in response to the normal (on-line) operation	Under certain conditions, failures can be detected using information about (for example) the time behaviour of the system. For example, if a switch is normally actuated and does not change state at the expected time, a failure will have been detected. It is not usually possible to localize the failure.	In general, there is no specific hardware element for the realisation of these coverage mechanisms. On-line monitoring detects abnormal behaviour of the system with respect to certain conditions of activation. For example, if such parameter is inverted when the vehicle speed is different from zero, then detection of incoherence between this parameter and vehicle speed leads to failure detection. For the voter, unlike the comparator, the majority voter technique increases the availability by ensuring the functionality of the redundant channel even after the loss of one channel. In this point is still complicated to find a probability within a failure can occur. Next tables go deeper in the hardware level.	ISO 26262 - part 5
			Comparator	High (Depends on the quality of the comparison)	To detect, as early as possible, (non-simultaneous) failures in independent hardware or software	The output signals of independent hardware or output information of independent software, are compared cyclically or continuously by a comparator. Detected differences lead to a failure message. For instance: two processing units exchange data (including results, intermediate results and test data) reciprocally. A comparison of the data is carried out using software in each unit and detected differences lead to a failure message.		
			Majority voter	High (Depends on the quality of the voting)	To detect and mask failures in one of at least three channels	A voting unit using the majority principle (2 out of 3, 3 out of 3, or m out of n) is used to detect and mask failures.		
		Failure in solid-state elements	Dynamic principles	Medium (Depends on diagnostic coverage of failure detection)	To detect static failures by dynamic signal processing	A forced change of otherwise static signals (internally or externally generated) helps to detect static failures in elements. This technique is often associated with electromechanical elements.		
			Analogue signal monitoring in preference to digital on/off states	Low	To improve confidence in measured signals	Wherever there is a choice, analogue signals are used in preference to digital on/off states. For example, trip or safe states are represented by analogue signal levels, usually with signal level tolerance monitoring. In the case of a digital signal, it is possible to monitor it with an analogue input. The technique gives continuity monitoring and a higher level of confidence in the transmitter, reducing the required frequency of the periodic test performed to detect failures of the transmitter sensing function.		
		Failures in processing units which lead to incorrect results.	Self-test by software cross exchange between two independent units	Medium (Depends on the quality of the self test)	To detect, as early as possible, failures in the processing unit consisting of physical storage (for example registers) and functional units (for example, instruction decoder)	The failure detection is realised entirely by means of two or more processing units each executing additional software functions which perform self-tests (for example walking-bit pattern) to test the physical storage (data and address registers) and the functional units (for example instruction decoder). The processing units exchange the results. This test provides very limited or no coverage for soft errors.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
Specific semiconductor elements - Processing Unit / Controller	CPU *(Part 1)*	Failures in processing units which lead to incorrect results. > Physical chip damage; > Local damage caused by overheating; > Radiation;	Self-test by software: limited number of patterns (one channel)	Medium (Depends on the quality of the self test). Up to 96.5% for the opcode-based self test can be reached.	To detect, as early as possible, failures in the processing unit and other sub-elements consisting of physical storage or functional units, or both, by means of software.	The failure detection is realised entirely by software which perform self-tests using a data pattern, or set of data patterns, to test the physical storage (for example, data and address registers) or the functional units (for example, the instruction decoder) or both.	The mechanisms are based on self-tests concepts either by software or hardware and intend to find failures using a self-verification algorithm. The exact probability of finding failures is quite hard to find out but general ideas and a general qualitative description can be done. Usually these self-test have a medium classification in terms of quality of coverage of the given system.	> SafeTCore Library (Infineon's Website) and Intelligent functional safety concept saves design time and costs by Manfred Choutka, Infineon Technologies > ISO 26262 - part 5
			Self-test by software cross exchange between two independent units	Medium (Depends on the quality of the self test).	To detect, as early as possible, failures in the processing unit consisting of physical storage and functional units.	The failure detection is realised entirely by means of two or more processing units each executing additional software functions which perform self-tests (for example walking-bit pattern) to test the physical storage (data and address registers) and the functional units (for example instruction decoder). The processing units exchange the results. This test provides very limited or no coverage for soft errors.		ISO 26262 - part 5
			Self-test supported by hardware (one-channel)	Medium (Depends on the quality of the self test).	To detect, as early as possible, failures in the processing unit and other sub-elements, using special hardware that increases the speed and extends the scope of failure detection.	Additional special hardware facilities support self-test functions to detect failures in the processing unit and other sub-elements (for example an EDC coder/decoder) at a gate level. The test can achieve high coverage. Typically only run at the initialization or power-down of the processing unit due to its intrusive nature. Typical usage is for multipoint fault detection.		ISO 26262 - part 5

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
Specific semiconductor elements - Processing Unit / Controller	CPU *(Part 2)*	Failures in processing units which lead to incorrect results. > Physical chip damage; > Local damage caused by overheating; > Radiation;	Software diversified redundancy (one hardware channel)	High (Depends on the quality of the diversification. Common mode failures can reduce diagnostic coverage)	To detect, as early as possible, failures in the processing unit, by dynamic software comparison.	The design consists of two redundant diverse software implementations in one hardware channel. In some cases, using different hardware resources (e.g. different RAM, ROM memory ranges) can increase the diagnostic coverage. One implementation, referred to as the primary path, is responsible for the calculations that if calculated erroneously can cause a hazard. The second implementation, referred to as the redundant path, is responsible for verifying the primary path's calculations and taking action if a failure is detected. Often the redundant path is implemented using separate algorithm designs and code to provide for software diversity. Once both paths are complete, a comparison of the output data of the two redundant software implementations is carried out. Detected differences lead to a failure message.	In general, these techniques are based on the idea of redundancy, for instance a software technique for fault coverage called “N-Version Programming” or either hardware techniques which duplicate important hardware parts in order to become the hardware more reliable. Usually these techniques are classified as with a high coverage capacity and depend on the quality of the redundancy which depends, on its turn, of the amount spent to develop such solutions. Depending on the quality, the probability of finding failures can be up to 99% with more five nines after the decimal mark.	> Fault Tolerant Design of Embedded Systems, David Kalinsky, Ph.D., Embedded World Conference 2012, Nuremberg DE > ISO 26262 - part 5
			Reciprocal comparison by software	High (Depends on the quality of the comparison)	To detect, as early as possible, failures in the processing unit, by dynamic software comparison.	Two processing units exchange data (including results, intermediate results and test data) reciprocally. A comparison of the data is carried out using software in each unit and detected differences lead to a failure message. This approach allows for hardware and software diversity if different processor types are used as well as separate algorithm designs, code and compilers. The design includes methods to avoid false error detections due to differences between the processors (e.g. loop jitter, communication delays, processor initialization). Paths can be implemented using separate cores of a dual core processor. In this case, the method includes analysis to understand common cause failures modes, due to the shared die and package, of the two cores.		ISO 26262 - part 5
			Hardware redundancy (e.g. Dual Core Lockstep, asymmetric redundancy, coded processing)	High (It depends on the quality of redundancy. Common mode failures can reduce diagnostic coverage)	To detect, as early as possible, failures in the processing unit, by step-by-step comparison of internal or external results or both produced by two processing units operating in lockstep.	In one version of this type of diagnostic technique, the Dual Core Lockstep, two symmetrical processing units are contained on one die. The processing units run duplicate operations in lockstep (or delayed by a fixed period) and the results are compared. Any mismatch results in an error condition and usually a reset condition. Other types of Hardware redundancies are possible, such as asymmetric redundancy. In those architectures, a diverse and dedicated processing unit is tightly coupled with the main processing units by means of an interface enabling a step-by-step comparison of internal and external results. Coded processing is also possible: processing units can be designed with special failure-recognising or failure correcting circuit techniques.		> CAN in Automation (CiA), Nuernberg, Germany > ISO 26262 - part 5

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
Specific semiconductor elements - Processing Unit / Controller	CPU *(Part 3)*	Failures in processing units which lead to incorrect results. > Physical chip damage; > Local damage caused by overheating; > Radiation;	Stack over/under flow Detection	Low (Stack boundary test only)	To detect, as early as possible, stack over or under flows.	The boundaries of the stack in volatile memory are loaded with predefined values. Periodically, the values are checked and if they have changed, an over or under flow is detected. A test is not needed if writes outside stack boundaries are controlled by a memory management unit.	n/a	ISO 26262 - part 5
		> Illegal hardware exceptions	Integrated Hardware consistency monitoring	High (Coverage for illegal hardware exceptions only)	To detect, as early as possible, illegal conditions in the processing unit.	Most processors are equipped with mechanisms that trigger hardware exceptions when errors are detected (division by zero and invalid op-codes, for example). Interrupt processing of these errors can then be used to trap these conditions to isolate the system from their effects. Typically, hardware monitoring is used to detect systematic failures but can also be used to detect certain kinds of random hardware faults. The technique provides low coverage for some coding errors and is good design practice.		
Specific semiconductor elements - Processing Unit / Controller	Control logic (Sequencer, coding and execution logic including flag registers and stack control)	Failures in processing units which lead to incorrect results. > Wrong coding, wrong or no execution > Execution out of order > Execution too fast or too slow > Stack overflow/underflow	Self-test by software	Medium	To detect, as early as possible, failures in the processing unit and other sub-elements consisting of physical storage (for example, registers) or functional units, or both, by means of software.	The failure detection is realised entirely by software which perform self-tests using a data pattern, or set of data patterns, to test the physical storage (for example, data and address registers) or the functional units (for example, the instruction decoder) or both.	For the control logic, as in general for CPU failures, the most common way to realize coverage is to execute self-tests either by software and hardware. Results are more effectively if used hardware approach.	ISO 26262 - part 5
			Self-test supported by hardware (one-channel)	High (Effectiveness depends on the type of self-test. Gate level is an appropriate level for this test)	To detect, as early as possible, failures in the processing unit and other sub-elements, using special hardware that increases the speed and extends the scope of failure detection.	Additional special hardware facilities support self-test functions to detect failures in the processing unit and other sub-elements (for example an EDC coder/decoder) at a gate level. The test can achieve high coverage. Typically only run at the initialization or power-down of the processing unit due to its intrusive nature. Typical usage is for multipoint fault detection.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
General semiconductor elements - General Electronic Embedded System / Controller	Digital I/O	Failures in input and output units and sending of inadmissible outputs to the process. > Failures concerning Stuck-at (including signal lines outside of the microcontroller) (see notes) > Failures concerning d.c. fault model (including signal lines outside of the microcontroller) (see notes) > Drift and oscillation	Failure detection by on-line monitoring (Digital I/O)	Low (Depends on diagnostic coverage of failure detection)	To detect failures by monitoring the behaviour of the system in response to the normal (on-line) operation.	Under certain conditions, failures can be detected using information about (for example) the time behavior of the system. If a device is normally actuated and for any reason it does not change state at the expected time, a failure will have been detected. It is not usually possible to localize the failure. Digital I/O can be periodic, so an algorithm can be implemented in order to detect some failures according to the malfunction of I/O gates. Just the easiest failures can be detected because algorithms cannot predict the complete behavior of a complex I/O gate.	For I/O digital gates, usually on-line monitoring for the expected data flow or a comparison with a pattern are used. Other better solutions can be applied to ensure a great consistency. For random hardware failures, a parallel output can be used. For a really high level of reliability, voters with a big number of voting units should be used. The many voters are used the greater will be the level of trustworthiness of the coverage method.	ISO 26262 - part 5
			Test pattern	High (Depends on type of pattern)	To detect static failures (stuck-at failures) and cross-talk.	This is a dataflow-independent cyclical test of input and output units. It uses a defined test pattern to compare observations with the corresponding expected values. Test coverage is dependent on the degree of independence between the test pattern information, the test pattern reception, and the test pattern evaluation. In a good design, the functional behaviour of the system is not unacceptably influenced by the test pattern.		
			Code protection for digital I/O	Medium (Depends on type of coding)	To detect random hardware and systematic failures in the input/output dataflow.	This procedure protects the input and output information from both systematic and random hardware failures. Code protection gives dataflow-dependent failure detection of the input and output units, based on information redundancy, or time redundancy, or both. Typically, redundant information is superimposed on input data, or output data, or both. This gives a means to monitor the correct operation of the input or output circuits.		
			Multi-channel parallel output	High	To detect random hardware failures (stuck-at failures), failures caused by external influences, timing failures, addressing failures, drift failures and transient failures.	This is a dataflow-dependent multi-channel parallel output with independent outputs for the detection of random hardware failures. Failure detection is carried out via external comparators. If a failure occurs, the system can possibly be switched off directly. This measure is only effective if the dataflow changes during the diagnostic test interval.		
			Monitored outputs	High (Only if dataflow changes within diagnostic test interval)	To detect individual failures, failures caused by external influences, timing failures, addressing failures and transient failures.	This is a dataflow-dependent comparison of outputs with independent inputs to ensure compliance with a defined tolerance range (time, value). A detected failure cannot always be related to the defective output. This measure is only effective if the dataflow changes during the diagnostic test interval.		
			Input comparison/voting (1oo2, 2oo3 or better redundancy)	High (Only if dataflow changes within diagnostic test interval)		This is a dataflow-dependent comparison of independent inputs to ensure compliance with a defined tolerance range (time, value). There will be 1 out of 2, 2 out of 3 or better redundancy. This measure is only effective if the dataflow changes during the diagnostic test interval.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
General semiconductor elements - General Electronic Embedded System / Controller	Analogue I/O	Failures in input and output units and sending of inadmissible outputs to the process. > Failures concerning Stuck-at (including signal lines outside of the microcontroller) (see notes) > Failures concerning d.c. fault model (including signal lines outside of the microcontroller) (see notes) > Drift and oscillation	Test pattern	High (Depends on type of pattern)	To detect static failures (stuck-at failures) and cross-talk.	This is a dataflow-independent cyclical test of input and output units. It uses a defined test pattern to compare observations with the corresponding expected values. Test coverage is dependent on the degree of independence between the test pattern information, the test pattern reception, and the test pattern evaluation. In a good design, the functional behaviour of the system is not unacceptably influenced by the test pattern.	For I/O analogic gates, on-line monitoring a test pattern can be used. Also voters are used in order to increase the reliability. Probability of detecting failures is variable and depends on the method of each technique.	ISO 26262 - part 5
			Multi-channel parallel output	High	To detect random hardware failures (stuck-at failures), failures caused by external influences, timing failures, addressing failures, drift failures and transient failures.	This is a dataflow-dependent multi-channel parallel output with independent outputs for the detection of random hardware failures. Failure detection is carried out via external comparators. If a failure occurs, the system can possibly be switched off directly. This measure is only effective if the dataflow changes during the diagnostic test interval.		
			Monitored outputs	High (Only if dataflow changes within diagnostic test interval)	To detect individual failures, failures caused by external influences, timing failures, addressing failures, drift failures and transient failures.	This is a dataflow-dependent comparison of outputs with independent inputs to ensure compliance with a defined tolerance range (time, value). A detected failure cannot always be related to the defective output. This measure is only effective if the dataflow changes during the diagnostic test interval.		
			Input comparison/voting (1oo2, 2oo3 or better redundancy)	High (Only if dataflow changes within diagnostic test interval)	To detect individual failures, failures caused by external influences, timing failures, addressing failures, drift failures and transient failures.	This is a dataflow-dependent comparison of independent inputs to ensure compliance with a defined tolerance range (time, value). There will be 1 out of 2, 2 out of 3 or better redundancy. This measure is only effective if the dataflow changes during the diagnostic test interval.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
General semiconductor elements - General Electronic Embedded System / Controller	Non-volatile memory *(Part 1)*	Failures concerning information corruption in the non-volatile memory > Unwanted bit-flips due to radiation and others issues > Single-bit failure > Two-bit-failure > Three-bit-failure > Multi-bit-failure (>3) > Failures concerning stuck-at for data and addresses and control interface, lines and logic (see notes) > Failures concerning d.c. fault model for data, addresses (includes address lines within same block) and control interface, lines and logic (see notes)	Memory monitoring using error-detection-correction codes (EDC). Often referred to as ECC (Error Correcting Code).	> High (The effectiveness depends on the number of redundant bits. Can be used to correct errors) > single-bit failure (100%)	To detect each single-bit failure, each two-bit failure, some three-bit failures, and some all-bit failures in a word (typically 32, 64 or 128 bits).	Every word of memory is extended by several redundant bits to produce a modified Hamming code with a Hamming distance of at least 4. Every time a word is read, checking of the redundant bits can determine whether or not a corruption has taken place. If a difference is found, a failure message is produced.	Each of these mechanisms is based on information redundancy in the memory. CRCs copy part of the information, block replication doubles all the information. The generalized mechanism is therefore memory information redundancy. The degree of information redundancy can be expressed in one single value which can be used to calculate the probability of failing detection.	ISO 26262 - part 5
			Hamming codes with additional parity (SECDED)	Better than a simple EDC		Memory extended parity. Hamming codes extended by an extra parity bit to allow the detector to distinguish between single bit errors and two-bit errors in order to allow the detector corrects a single error and additionaly detect one error (if it does not correct an error, it can detect three bit failures in the same time).		
			Modified checksum	> Low (Depends on the number and location of bit errors within test area) > single-bit failure (100%) > two-bit failure (1 - 1/2^(sizeofchecksum))		A checksum is created by a suitable algorithm which uses each of the words in a block of memory. The checksum can be stored as an additional word in ROM, or an additional word can be added to the memory block to ensure that the checksum algorithm produces a predetermined value. In a later memory test, a checksum is created again using the same algorithm, and the result is compared with the stored or defined value. If a difference is found, a failure message is produced. The probability of a missed detection is 1/(2^size of checksum) if a random result is returned. If certain data disturbances are more probable, some checksums can provide a better detection ratio than the one for random results.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
General semiconductor elements - General Electronic Embedded System / Controller	Non-volatile memory *(Part 2)*	Failures concerning information corruption in the non-volatile memory > Unwanted bit-flips due to radiation and others issues > Single-bit failure > Two-bit-failure > Three-bit-failure > Multi-bit-failure (>3) > Failures concerning stuck-at for data and addresses and control interface, lines and logic (see notes) > Failures concerning d.c. fault model for data, addresses (includes address lines within same block) and control interface, lines and logic (see notes)	Memory Signature	> High > Single-bit failure (100%)	To detect each single-bit failure, each two-bit failure, some three-bit failures, and some all-bit failures in a word (typically 32, 64 or 128 bits).	The contents of a memory block are compressed (using either hardware or software) into one or more bytes using, for example, a cyclic redundancy check (CRC) algorithm. A typical CRC algorithm treats the whole contents of the block as byte-serial or bit-serial data flow, on which a continuous polynomial division is carried out using a polynomial generator. The remainder of the division represents the compressed memory contents – it is the "signature" of the memory – and is stored. The signature is computed once again in later tests and compared with one already stored. A failure message is produced if there is a difference. CRCs are particularly effective in detecting burst errors. The effectiveness of the signature depends on the polynomial in relation to the block length of the information to be protected. The probability of a missed detection is 1/(2^size of checksum) if a random result is returned. Use of an 8 bit CRC is not generally considered the state of the art for memory sizes above 4k.	Each of these mechanisms is based on information redundancy in the memory. CRCs copy part of the information, block replication doubles all the information. The generalized mechanism is therefore memory information redundancy. The degree of information redundancy can be expressed in one single value which can be used to calculate the probability of failing detection.	ISO 26262 - part 5
			Block replication (for example double memory with hardware or software comparison)	> High > Any kind of bit failures (100%) > Same faults in all replicated blocks		The address space is duplicated in two memories. The first memory is operated in the normal manner. The second memory contains the same information and is accessed in parallel to the first. The outputs are compared and a failure message is produced if a difference is detected. Dependent on memory subsystem design, storage of inverse data in one of the two memories can enhance diagnostic coverage. Coverage can be reduced if failure modes (such as common address lines, write-enables) exist that are common to both blocks or if physical placement of memory cells makes logically distant cells physical neighbours.		
			Parity bit	Low (can detect some errors but simple parity checking cannot correct them)	To detect a single corrupted bit or an odd number of corrupted bits failures in a word (typically 8 bits, 16 bits, 32 bits, 64 bits or 128 bits).	Parity checking can verify memory-errors by storing of a redundant parity bit representing the parity (odd or even) to detect whether a data error has occurred. Every word of the memory is extended by one bit (the parity bit) which completes each word to an even or odd number of logical 1s. The parity of the data word is checked each time it is read. If the wrong number of 1s is found, a failure message is produced. The choice of even or odd parity ought to be made such that, whichever of the zero word (nothing but 0s) or the one word (nothing but 1s) is the more unfavourable in the event of a failure, then that word is not a valid code.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
General semiconductor elements - General Electronic Embedded System / Controller	Volatile memory *(Part 1)*	Failures during addressing, writing, storing and reading > Unwanted bit-flips due to radiation and others issues > Failures concerning stuck-at for data and addresses and control interface, lines and logic (see notes) > Failures concerning d.c. fault model for data, addresses (includes address lines within same block and inability to write to cell) and control interface, lines and logic (see notes) > Soft error model for bit cells (see notes)	RAM pattern test	Medium (High coverage for stuck-at failures. No coverage for linked failures. Can be appropriate to run under interrupt protection)	To detect predominantly static bit failures.	A bit pattern followed by the complement of that bit pattern is written into the cells of memory. RAM locations are generally tested individually. The cell content is stored and then all 0s are written to the cell. The cell contents are then verified by a read back of the 0 values. The procedure is repeated by writing all 1s to the cell and reading the contents back. If a transition failure from 1 to 0 is a failure mode of concern, an additional write and read of 0s can be performed. Finally, original contents of the cell are restored. The test is effective at detecting stuck-at and transition failures but cannot detect most soft errors, addressing faults and linked cell faults. The test is often implemented in the background with interrupt suppression during the test of each individual location. Because the implementation includes a read of a just written value, optimizing compilers have a tendency to optimize out the test. If an optimizing compiler is used, good design practice is to verify the test code by an assembler-level code inspection. Some RAMs can fail such that the last memory access operation is echoed back as a read. If this is a plausible failure mode, the diagnostic can test two locations together, first writing a 0 to 1 and then a 1 to the next and then verifying a 0 is read from the first location.	Pattern tests are applied to the memory in order to detect bit failures. A probability of detecting one, two or more bit failures can be calculated for each technique.	ISO 26262 - part 5
			RAM March test	High (Depends on the write read order for linked cell coverage. Test generally not appropriate for run time)	To detect predominantly persistent bit failures, bit transition failures, addressing failures and linked cell failures.	A pattern of 0s and 1s is written into the cells of memory in a specific pattern and verified in a specific order. A March test consists of a finite sequence of March elements; while a March element is a finite sequence of operations applied to every cell in the memory array before proceeding to the next cell. For example, an operation can consist of writing a 0 into a cell, writing a 1 into a cell, reading an expected 0 from a cell, and reading an expected 1 from a cell. A failure is detected if the expected “1” is not read. The coverage level for linked cells depends on the write/read order. March tests are designed to detect various RAM failure modes: stuck-at faults, transition faults (inability to transition from a one to a zero or a zero to a one but not both), address faults and linked cell faults. These types of tests are not effective for soft error detection. These tests can usually only be run at initialization or shutdown.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
General semiconductor elements - General Electronic Embedded System / Controller	Volatile memory *(Part 2)*	Failures during addressing, writing, storing and reading > Unwanted bit-flips due to radiation and others issues > Failures concerning stuck-at for data and addresses and control interface, lines and logic (see notes) > Failures concerning d.c. fault model for data, addresses (includes address lines within same block and inability to write to cell) and control interface, lines and logic (see notes) > Soft error model for bit cells (see notes)	Parity bit	Low (can detect some errors but simple parity checking cannot correct them)	To detect a single corrupted bit or an odd number of corrupted bits failures in a word (typically 8 bits, 16 bits, 32 bits, 64 bits or 128 bits).	Parity checking can verify memory-errors by storing of a redundant parity bit representing the parity (odd or even) to detect whether a data error has occurred. Every word of the memory is extended by one bit (the parity bit) which completes each word to an even or odd number of logical 1s. The parity of the data word is checked each time it is read. If the wrong number of 1s is found, a failure message is produced. The choice of even or odd parity ought to be made such that, whichever of the zero word (nothing but 0s) or the one word (nothing but 1s) is the more unfavourable in the event of a failure, then that word is not a valid code. For RAM cell write-enable failure, parity can detect 50 % of failures if the cell is unable to be initialized. The coverage is 0 % if the write-enable failure affects entire cell after it has been initialized.	n/a	ISO 26262 - part 5
			Memory monitoring using error-detection-correction codes (EDC) (often referred to as ECC (Error Correcting Code))	High (The effectiveness depends on the number of redundant bits. Can be used to correct errors)	To detect each single-bit failure, each two-bit failure, some three-bit failures, and some all-bit failures in a word (typically 32, 64 or 128 bits).	Every word of memory is extended by several redundant bits to produce a modified Hamming code with a Hamming distance of at least 4. Every time a word is read, checking of the redundant bits can determine whether or not a corruption has taken place. If a difference is found, a failure message is produced. For RAM cell write-enable failure, EDC can provide high coverage if the cell cannot be initialized. The coverage is 0 % if the write-enable failure affects the entire cell after it has been initialized.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
General semiconductor elements - General Electronic Embedded System / Controller	Volatile memory *(Part 3)*	Failures during addressing, writing, storing and reading > Unwanted bit-flips due to radiation and others issues > Failures concerning stuck-at for data and addresses and control interface, lines and logic (see notes) > Failures concerning d.c. fault model for data, addresses (includes address lines within same block and inability to write to cell) and control interface, lines and logic (see notes) > Soft error model for bit cells (see notes)	Running checksum / CRC	High. Probability is 1/maximum value of checksum if random pattern is returned.	To detect single bit, and some multiple bit, failures in RAM.	A checksum/CRC is created by a suitable algorithm which uses each of the words in a block of memory. The checksum is stored as an additional word in RAM. As the memory block is updated, the RAM checksum/CRC is also updated by removing the old data value and adding in the new data value to be stored to the memory location. Periodically, a checksum/CRC is calculated for the data block and compared to the stored checksum/CRC. If a difference is found, a failure message is produced. The probability of a missed detection is 1/size of checksum/CRC if a random result is returned. DC can be reduced as memory size increases. The effectiveness of the signature depends on the polynomial in relation to the block length of the information to be protected. Care needs to be taken so that values used to determine checksum are not changed during checksum calculation.	n/a	ISO 26262 - part 5
			Block replication (for example double memory with hardware or software comparison)	High (Common failure modes can reduce diagnostic coverage)	To detect each bit failure.	The address space is duplicated in two memories. The first memory is operated in the normal manner. The second memory contains the same information and is accessed in parallel to the first. The outputs are compared and a failure message is produced if a difference is detected. Dependent on memory subsystem design, storage of inverse data in one of the two memories can enhance diagnostic coverage. Coverage can be reduced if failure modes (such as common address lines, write-enables) exist that are common to both blocks or if physical placement of memory cells makes logically distant cells physical neighbours.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
Specific semiconductor elements - Communication / Microprocessors	On-chip communication including bus-arbitration	Failures in the information transfer. > Concerning stuck-at (data, control, address and arbitration signals) (see notes) > Concerning d.c. fault model (data, control, address and arbitration signals) (see notes) > Time out > No or continuous or wrong arbitration > Soft errors (for sequential part)	One-bit hardware redundancy	Low	To detect each odd-bit failure, i.e. 50 % of all the possible bit failures in the data stream.	The communication bus is extended by one line (bit) and this additional line (bit) is used to detect failures by parity checking.	Redundancy techniques are most used. The probability may be calculated depending on the technique employed. Depending on the number of the redounded bits the probability of detection of a failure changes.	ISO 26262 - part 5
			Multi-bit hardware redundancy	Medium (Multi-bit redundancy can achieve high coverage by proper interleaving of data, address and control lines, and if combined with some complete redundancy, e.g. for the arbiter.)	To detect failures during the communication on a bus and in serial transmission links.	The communication bus is extended by two or more lines and these additional lines are used in order to detect failures by using Hamming code techniques.		
			Complete hardware redundancy	High (Common failure modes can reduce diagnostic coverage)	To detect failures during the communication by comparing the signals on two buses.	The bus is duplicated and the additional lines are used to detect failures.		
		Failures in input and output units (digital, analogue) and sending of inadmissible outputs to the process.	Test pattern	High (Depends on type of pattern)	To detect static failures (stuck-at failures) and cross-talk.	This is a dataflow-independent cyclical test of input and output units. It uses a defined test pattern to compare observations with the corresponding expected values. Test coverage is dependent on the degree of independence between the test pattern information, the test pattern reception, and the test pattern evaluation. In a good design, the functional behaviour of the system is not unacceptably influenced by the test pattern.	n/a	

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
Specific semiconductor elements - Communication / General Electronic Embedded System	Data trans-mission *(Part 1)*	Failures in the information transfer. > Failure of communication peer > Message corruption > Message delay > Message loss > Unintended message repetition > Resequencing > Insertion of message > Masquerading	One-bit hardware redundancy	Low	To detect each odd-bit failure, i.e. 50 % of all the possible bit failures in the data stream.	The communication bus is extended by one line (bit) and this additional line (bit) is used to detect failures by parity checking.	Redundancy techniques are most used. The probability may be calculated depending on the technique employed. Depending on the number of the redounded bits the probability of detection of a failure changes.	ISO 26262 - part 5
			Multi-bit hardware redundancy	Medium	To detect failures during the communication on a bus and in serial transmission links.	The communication bus is extended by two or more lines and these additional lines are used in order to detect failures by using Hamming code techniques.		
			Complete hardware redundancy	High (Common mode failures can reduce diagnostic coverage)	To detect failures during the communication by comparing the signals on two buses.	The bus is duplicated and the additional lines are used to detect failures.		
			Transmission redundancy	Medium (Depends on type of redundancy. Effective only against transient faults)	To detect transient failures in bus communication.	The information is transferred several times in sequence. The technique is only effective in detecting transient failures.		
			Information redundancy	Medium (Depends on type of redundancy)	To detect failures in bus communication.	Data is transmitted in blocks, together with a calculated checksum or CRC (cyclic redundancy check) for each block. The receiver then re-calculates the checksum of the received data and compares the result with the received checksum. For CRC coverage depends on the length of the data to be covered, the size of the CRC (number of bits) and the polynomial. The CRC can be designed to address the more probable communication failure modes of the underlying hardware (for example burst errors). The message ID can be included in the checksum/CRC calculation to provide coverage for corruptions in this part of the message (masquerading). High coverage can be reached concerning data and ID corruption, however, overall high coverage cannot be reached by checking only the coherence of the data and the ID with a signature, whatever the efficiency of the signature. Specifically, a signature does not cover the message loss or the unintended message repetition. If a checksum algorithm has a Hamming distance of less than 3, a high coverage concerning data and ID corruption can still be claimed if supported by a proper rationale.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
Specific semiconductor elements - Communication / General Electronic Embedded System	Data trans-mission *(Part 2)*	Failures in the information transfer. > Failure of communication peer > Message corruption > Message delay > Message loss > Unintended message repetition > Resequencing > Insertion of message > Masquerading	Inspection using test patterns	High	To detect static failures (stuck-at failure) and cross-talk.	This is a dataflow-independent cyclical test of data paths. It uses a defined test pattern to compare observations with the corresponding expected values. Test coverage is dependent on the degree of independence between the test pattern information, the test pattern reception, and the test pattern evaluation. In a good design, the functional behaviour of the system is not unacceptably influenced by the test pattern.	n/a	ISO 26262 - part 5
			Read back of sent message	Medium	To detect failures in bus communication.	The transmitter reads back its sent message from the bus and compares it with the original message. This safety mechanism is used by CAN. High coverage can be reached concerning data and ID corruption, however, overall high coverage cannot be reached by checking only the coherence of the data and the ID. Other failure modes like the unintended message repetition are not necessarily covered by this safety mechanism.		
			Frame counter	Medium	To detect frame losses. A frame is a coherent set of data sent from one controller to other controller(s). The unique frame is identified by a message ID.	Each unique safety-related frame includes a counter as part of the message which is transmitted on the bus. The counter is incremented (with roll-over) during the creation of each successive frame transmitted. The receiver is then able to detect any frame loss or non-refreshment by verifying that the counter is incrementing by one. A special version of the frame counter would be to include separate signal counters tied to the refreshment of safety-related data. In this situation, if a frame contained more than one piece of safety-related data, an individual counter for each piece of safety-related data is provided.		
			Timeout monitoring	Medium	To detect loss of data between the sending node and the receiving node.	The receiver monitors each expected safety-related message ID for time between the receipt of valid frames with this message ID. A failure would be indicated by too long a period elapsing between messages. This is intended to detect continuous loss of a communications channel or the continuous loss of one a specific message (no frames received for a specific message ID).		
			Combination of information redundancy, frame counter and timeout monitoring	High (For systems without hardware redundancy or test patterns, high coverage can be claimed for the combination of these safety mechanisms)	It brings together the aims of all techniques concerned	This technique employs the mechanisms described earlier in the present table.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
General semiconductor elements - General Electronic Embedded System	Clock	Failures due to the processing in the wrong sequence or period of time of individual elements of a program (for example, software modules, subprograms or commands), or when the clock of the processor is faulty. > Stuck-at (see notes) > d.c. fault mode (see notes) > Incorrect frequency > Period jitter	Watchdog with separate time base without time-window	Low	To monitor the behaviour and the plausibility of the program sequence.	External timing elements with a separate time base (for example, watchdog timers) are periodically triggered to monitor the processor's behaviour and the plausibility of the program sequence. It is important that the triggering points are correctly placed in the program. The watchdog is not triggered at a fixed period, but a maximum interval is specified.	Use if separate time base technique. Probability depends on the use or not of a time window.	ISO 26262 - part 5
			Watchdog with separate time base and time-window	Medium (Depends on time restriction for the time-window)	To monitor the behaviour and the plausibility of the program sequence.	External timing elements with a separate time base (for example watchdog timers) are periodically triggered to monitor the processor's behaviour and the plausibility of the program sequence. It is important that the triggering points are correctly placed in the program (e.g. not in an interrupt service routine). A lower and upper limit is given for the watchdog timer. If the program sequence takes a longer or shorter time than expected, action is taken.		
			Logical monitoring of program sequence	Medium (Only effective against clock failures if external temporal events influence the logical program flow)	To monitor the correct sequence of the individual program sections.	The correct sequence of the individual program sections is monitored using software (counting procedure, key procedure) or using external monitoring facilities. It is important that the checking points are placed in the program so that paths which can result in a hazard if they fail to complete or execute out of sequence, due to a single or multiple-point fault, are monitored. The sequences can be updated between each function call or more tightly integrated into the program execution. Provides coverage for internal hardware failures (such as interrupt frequency errors) that can cause the software to run out of sequence.	n/a	
			Combination of temporal and logical monitoring of program sequence	High	To monitor the behaviour and the correct sequence of the individual program sections.	A temporal facility (for example a watchdog timer) monitoring the program sequence is retriggered only if the sequence of the program sections is also executed correctly. This is a combination of the previous techniques.	Probability can be calculated using probabilities of each combined technique	
			Combination of temporal and logical monitoring of program sequences with time dependency	High	To monitor the behaviour, correct sequencing and the execution time interval of the individual program sections.	A Program Flow Monitoring strategy is implemented where software update points are expected to occur within a relative time window. The PFM sequence result and time calculation are monitored by external monitoring facilities. Provides coverage for internal hardware failures that can cause the software to run out of sequence. When implemented with asymmetrical designs, provides coverage regarding communication sequence between main and monitoring device. This method is to be designed to account for execution jitter from interrupts, CPU loading, etc.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
Electrical elements - Electronic Measurement System	Sensors including signal switches *(Part 1)*	Failures in the sensors of the system. > Out-of-range > Offsets > Stuck in range > Oscillations	Failure detection by on-line monitoring	Low (Depends on diagnostic coverage of failure detection)	To detect failures by monitoring the behaviour of the system in response to the normal (on-line) operation.	Under certain conditions, failures can be detected using information about (for example) the time behaviour of the system. For example, if a switch is normally actuated and does not change state at the expected time, a failure will have been detected. It is not usually possible to localize the failure.	n/a	ISO 26262 - part 5
			Test pattern	High	To detect static failures (stuck-at failures) and cross-talk.	This is a dataflow-independent cyclical test of input and output units. It uses a defined test pattern to compare observations with the corresponding expected values. Test coverage is dependent on the degree of independence between the test pattern information, the test pattern reception, and the test pattern evaluation. In a good design, the functional behaviour of the system is not unacceptably influenced by the test pattern.		
			Input comparison/voting (1oo2, 2oo3 or better redundancy)	High (Only if dataflow changes within diagnostic test interval)	To detect individual failures, failures caused by external influences, timing failures, addressing failures, drift failures (for analogue signals) and transient failures.	This is a dataflow-dependent comparison of independent inputs to ensure compliance with a defined tolerance range (time, value). There will be 1 out of 2, 2 out of 3 or better redundancy. This measure is only effective if the dataflow changes during the diagnostic test interval.		
			Sensor valid range	Low (Detects shorts to ground or power and some open circuits)	To detect sensor shorts to ground or power and some open circuits.	Limit valid reading to the middle part of the sensor electrical range. If a sensor reading is in an invalid region, this indicates an electrical problem with the sensor such as a short to power or to ground. Typically used with sensors read by the ECU using ADCs.		
			Sensor correlation	High (Detects in range failures)	To detect sensor-in-range drifts, offsets or other errors using a redundant sensor.	Comparison of two identical or similar sensors to detect in-range failures such as drifts, offsets or stuck-at failures. Typically used with sensors read by the ECU using ADCs. Sensors would be converted to equal slope and compared to agree within a threshold. The threshold is selected taking into account the ADC tolerance and the variation in the electrical elements. Both sensors are sampled by the ECU at as close to the same time as possible to avoid false failures due to the sensor readings dynamically changing. Equal slope sensor based diagnostics do not detect situations where the two sensors are shorted together yielding correlated readings at the crossing point or common cause failures where a single component, e.g. the ADC, corrupts both sensor results in a similar way.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
Electrical elements - Electronic Measurement System	Sensors including signal switches *(Part 2)*	Failures in the sensors of the system. > Out-of-range > Offsets > Stuck in range > Oscillations	Sensor rationality check	Medium	To detect sensor-in-range drifts, offsets or other errors using multiple diverse sensors.	Comparison of two (or more) sensors measuring different properties to detect in-range failures such as drifts, offsets or stuck-at failures. The sensor measurements are converted to equivalent values using a model to provide values that can be compared. For instance: The comparison of gasoline engine throttle position, manifold pressure and mass air flow sensors after each is converted to an air flow reading. The usage of diverse sensors reduces the problem of systematic faults.	n/a	ISO 26262 - part 5
Electrical elements - Final elements	> actuators > lamps > buzzer > screen > etc.	Failures in electromechanical elements.	Failure detection by on-line monitoring	Low (Depends on diagnostic coverage of failure detection)	To detect failures by monitoring the behaviour of the system in response to the normal (on-line) operation.	Under certain conditions, failures can be detected using information about (for example) the time behaviour of the system.	n/a	ISO 26262 - part 5
		Failures in input and output units (digital, analogue) and sending of inadmissible outputs to the process.	Test pattern	High	To detect static failures (stuck-at failures) and cross-talk.	This is a dataflow-independent cyclical test of input and output units. It uses a defined test pattern to compare observations with the corresponding expected values. Test coverage is dependent on the degree of independence between the test pattern information, the test pattern reception, and the test pattern evaluation. In a good design, the functional behaviour of the system is not unacceptably influenced by the test pattern.		
		Failures in the final elements of the system.	Monitoring (i.e. coherence control)	High (Depends on diagnostic coverage of failure detection)	To detect the incorrect operation of an actuator.	The operation of the actuator is monitored. Monitoring can be done at the actuator level by physical parameter measurements (which can have high coverage) but also at the system level regarding the actuator failure effect.		
General semiconductor elements - Power Supply System	Power supply	> Drift > Under and over Voltage	Voltage or current control (input)	Low	To detect as soon as possible wrong behaviour of input current or voltage values.	Monitoring of input voltage or current.	Generally, the way to control and react over a power supply failure is monitoring the voltage and/or current of the given power supply and as soon as they go out of the specified limits, it should be acted quickly on the supplier device.	ISO 26262 - part 5
		> Out-of-range > Offsets > Stuck in range > Oscillations	Voltage or current control (output)	High	To detect as soon as possible wrong behaviour of output current or voltage values.	Monitoring of output voltage or current.		

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
Electrical elements - Actuator Electrical System	Relays	Failures in electromechanical elements. > Does not energize or de-energize > Individual contacts welded	Failure detection by on-line monitoring	60%, 90% or 99% / Considered High (Depends on diagnostic coverage of failure detection)	To detect failures by monitoring the behaviour of the system in response to the normal (on-line) operation.	Under certain conditions, failures can be detected using information about (for example) the time behaviour of the system.	n/a	ISO 26262 - part 5
Electrical elements - General Electrical/Electronic Embedded System	Harnesses including splice and connectors	Failures in electromechanical elements. > Open Circuit > Contact Resistance > Short Circuit to Ground (d.c Coupled) > Short Circuit to Vbat > Short Circuit between neighbouring pins > Resistive drift between pins	Failure detection by on-line monitoring	Ability to cover the listed faults to achieve 60%, 90% or 99% of diagnostic coverage of the element depending on the quality of the monitor	To detect failures by monitoring the behaviour of the system in response to the normal (on-line) operation.	Under certain conditions, failures can be detected using information about (for example) the time behaviour of the system.	n/a	ISO 26262 - part 5
Specific semiconductor elements - Processing Unit / Controller	ALU - Data Path	> Failures concerning stuck-at at gate level (see notes) > Failures concerning d.c. fault model (see notes) > Failures concerning soft error model (see notes)	*Information or model required*	Ability to cover the listed faults to achieve 60%, 90% or 99% of diagnostic coverage of the element depending on the quality of the technique	n/a	n/a	n/a	ISO 26262 - part 5
Specific semiconductor elements - Processing Unit / Controller	Configuration Registers	Failures concerning: > Stuck-at wrong value (see notes) > Corruption of registers (soft errors) > Stuck-at fault model (see notes)	*Information or model required*	Ability to cover the listed faults to achieve 90% or 99% of diagnostic coverage of the element depending on the employed technique	n/a	n/a	n/a	ISO 26262 - part 5

Level (in Hardware)	Fault location (Element)	Applicable to following faults	Mechanism name of diagnostic coverage	Probability of detection of faults (qualitative or quantitative)	Correction (Aim)	Mechanisms Description	Generalization	Source
Specific semiconductor elements - Processing Unit / Controller	> Registers (general purpose registers bank, DMA transfer registers, etc.) > Internal RAM	Failures concerning: > Stuck-at at gate level (see notes) > d.c. fault model including no, wrong or multiple addressing of registers (see notes) > Soft error model (see notes)	Configuration Register Test	High (Configuration registers only)	To detect, as early as possible, failures in the configuration resisters of a processing unit. Failures can be hardware related (stuck values or soft errors induced bit flips) or software related (incorrect value stored or register corrupted by software error).	Configuration register settings are read and then compared to an encoding of the expected settings (e.g. a mask). If the settings do not match, the registers are reloaded with their intended value. If the error persists for a pre-determined number of checks the fault condition is reported.	n/a	ISO 26262 - part 5
Specific semiconductor elements - Processing Unit / Controller	Address calculation (Load/Store Unit, DMA addressing logic, memory and bus interfaces)	Failures concerning: > Stuck-at at gate level (see notes) > d.c. fault model including no, wrong or multiple addressing (see notes) > Soft error model (for sequential parts) (see notes)	*Information or model required*	Ability to cover the listed faults to achieve 60%, 90% or 99% of diagnostic coverage of the element depending on the employed technique	n/a	n/a	n/a	ISO 26262 - part 5
Specific semiconductor elements - Processing Unit / Controller	Interrupt handling	> Omission of or continuous interrupts > Incorrect interrupt executed > Wrong priority > Slow or interfered interrupt handling causing missed or delayed interrupts service	*Information or model required*	Ability to cover the listed faults to achieve 60%, 90% or 99% of diagnostic coverage of the element depending on the technique used	n/a	n/a	n/a	ISO 26262 - part 5
Specific semiconductor elements - Processing Unit / Controller	Other sub-elements not belonging to previous classes	Failures concerning: > Stuck-at at gate level (see notes) > d.c. fault model (see notes) > Soft error model (for sequential part) (see notes)	*Information or model required*	Ability to cover the listed faults to achieve 60%, 90% or 99% of diagnostic coverage of the element depending on the used technique	n/a	n/a	n/a	ISO 26262 - part 5

APPENDIX 2

```
--------------------------------------------------------------------------------------------
--     Title: simulation of a failure in the brake electronic system of a quarter car.     --
--                    First scenario: consequences of a soft braking                       --
--          Project: electrical engineering graduation course conclusion project           --
--                          Document type: source code in VHDL-AMS                         --
--                                 Source code version: 1.0                                --
--                                   Date: 10th June 2012                                  --
--                                Author: Guilherme BAUMGARTEN                             --
--                                       Inspected: N/A                                    --
--                                       Approved: N/A                                     --
--------------------------------------------------------------------------------------------

--------------------------------------------------------------------------------------------
--                           Project description (block diagram)                          --
--                                                                                        --
--                                      ->                                                --
--                                  - Fz                                                  --
--                                  -                                                     --
--                                  -                                                     --
--                                  -                                                     --
--                                  -                                                     --
--                                  -                                                     --
--                                -   -                                                   --
--                                 - -                                                    --
--                                  -                                                     --
--              ------------------------------------------                                --
--              -                                        -                                --
--              -                                        -             --                 --
--              -                                        -              --                --
--              -                   car                  -   -------------                --
--              -                                        -             --   -> ->         --
--              -                                        -            --    Vx,Fx         --
--              -                                        -                                --
--              ------------------------------------------                                --
--                        --                    --                                      --
--                      -  -                  -  -                                      --
--                    -    -                -    -                                      --
--                      -  -                  -  -                                      --
--                       --                    --                                       --
--                  -                                                                     --
--                ------------                 ->                                       --
--                 -                           w                                        --
--                    -> ->                                                               --
--                    µx,Tb                                                               --
--                                                                                        --
--------------------------------------------------------------------------------------------

--------------------------------------------------------------------------------------------
--                           VHDL-AMS model of a quart car model:                         --
--------------------------------------------------------------------------------------------
LIBRARY IEEE;                                                       --  Libraries declaration.
LIBRARY DISCIPLINES;                                                                       --
                                                                                           --
USE IEEE.MATH_REAL.ALL;                                                --  Used packages.
USE DISCIPLINES.KINEMATIC_SYSTEM.ALL;                                                      --
USE DISCIPLINES.ROTATIONAL_SYSTEM.ALL;                                                     --
                                                                                           --
ENTITY quarter_car_model IS --  Quantitative definitions of the quarter car model parameters:
    GENERIC (a:REAL := 1.28;                    --  For a vehicle being driven on dry asphalt
             b:REAL := 23.99;                     --  and hence the coefficients are
             c:REAL := 0.52;                    --  a = 1.28, b = 23.99 and c = 0.52
             J:REAL := 1.0;                                --  J: inertia [Kg * m²]
             R:REAL := 0.32;                               --  R: whell radius [m]
             m:REAL := 450.0;                          --  m: quarter vehicle mass [kg]
             g:REAL := 9.81);                        --  g: gravitational force [m * s-²]
    PORT (TERMINAL w1,w2:ROTATIONAL_OMEGA;                                                 --
          TERMINAL v1,v2:KINEMATIC_V;                    --  Declaration of block terminals.
          QUANTITY Tb: IN REAL;                                                            --
          QUANTITY Vi: IN REAL;                                                            --
          QUANTITY Wi: IN REAL;                                                            --
          QUANTITY mi_x: REAL;                                                             --
          QUANTITY lambda: REAL;                                                           --
          QUANTITY F_x: REAL;                                                              --
          QUANTITY F_z: REAL;                                                              --
          QUANTITY Speed: REAL;                                                            --
          QUANTITY Angular_Velocity: REAL);                                                --
END;                                                                                       --
```

```
                                                                                --
ARCHITECTURE behavior OF quarter_car_model IS                                   --
    QUANTITY w across af THROUGH w1 TO w2;                                      --
    QUANTITY v across lf THROUGH v1 TO v2;                                      --
BEGIN                                                                           --
    break v=>Vi;                                                                --
    break w=>Wi;                                                                --
    mi_x == a * (1.0 - ((math_e)**((-1.0) * b * lambda)) - c * lambda);       --  The Pacejka
    J * w'dot == (R * F_x) - (Tb * sign(w));                                --  "Magic Formula"
    m * v'dot == (-1.0) * F_x;                                                  --  tire models
    lambda == (v - (R * w)) / (v);                                              --
    F_x == F_z * mi_x;                                                          --
    F_z == m * g;                                                               --
    Speed == v;                                                                 --
    Angular_Velocity == w;                                                      --
    break v=>0.0 when not v'above(0.0);                                 --  Points of singularity
    break w=>0.0 when not w'above(0.0);                                         --
END ARCHITECTURE behavior;                                                      --
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
--                                Test bench                                  --
--------------------------------------------------------------------------------
LIBRARY IEEE;                                                           --  Libraries declaration.
LIBRARY DISCIPLINES;                                                            --
                                                                                --
USE IEEE.MATH_REAL.ALL;                                                 --  Used packages.
USE DISCIPLINES.KINEMATIC_SYSTEM.ALL;                                           --
USE DISCIPLINES.ROTATIONAL_SYSTEM.ALL;                                          --
                                                                                --
ENTITY car IS                                                                   --
    GENERIC (Tb,Vi,Wi:REAL);                                                    --
END;                                                                            --
                                                                                --
ARCHITECTURE behavior OF car IS                                                 --
    TERMINAL w:  ROTATIONAL_OMEGA;                                              --
    TERMINAL v:  KINEMATIC_V;                                                   --
BEGIN                                                                           --
    PROCESS BEGIN                                                       --  Initial conditions:
        Tb <= 0.0;                                                      --  brake torque = 0 Nm;
        Vi <= 3.0;                                                          --  speed = 3 m/s;
        Wi <= 9.375;                                                --  angular velocity = 9.375 rad/s
        WAIT FOR 1000ms;                                                        --  After 1s:
        Tb <= 80.0;                                                     --  brake torque = 80 Nm.
        WAIT;                                                                   --
    END PROCESS;                                                                --
    Sim1:  ENTITY quarter_car_model (behavior)                                  --
           PORT MAP (w, rotational_omega_ground, v, kinematic_v_ground, Tb, Vi, Wi);     --
                                --  Connections between all the blocks of the designed system
                                        --  (as described in the beginning of this file,
                                --  in the "Project description (block diagram)" section).
END ARCHITECTURE behavior;                                                      --
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
--                                END OF FILE                                 --
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
--     Title: simulation of a failure in the brake electronic system of a quarter car.     --
--                  Second scenario: consequences of a hard braking                       --
--          Project: electrical engineering graduation course conclusion project          --
--                          Document type: source code in VHDL-AMS                        --
--                                 Source code version: 1.0                               --
--                                   Date: 10th June 2012                                 --
--                                Author: Guilherme BAUMGARTEN                            --
--                                       Inspected: N/A                                   --
--                                       Approved: N/A                                    --
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
--                           Project description (block diagram)                      --
--                                                                                    --
--                                      ->                                            --
--                                    - Fz                                            --
--                                    -                                               --
--                                    -                                               --
--                                    -                                               --
--                                    -                                               --
--                                    -                                               --
--                                  -   -                                             --
--                                  - -                                               --
--                                    -                                               --
--              ------------------------------------------                            --
--              -                                        -                            --
--              -                                        -             --             --
--              -                                        -              --            --
--              -                   car                  -   -------------            --
--              -                                        -              --   -> ->     --
--              -                                        -             --    Vx,Fx     --
--              -                                        -                            --
--              ------------------------------------------                            --
--                       --                    --                                     --
--                     -  -                  -  -                                     --
--                    -    -                -    -                                    --
--                     -  -                  -  -                                     --
--                      --                    --                                      --
--                   -                                                                --
--                  ------------              ->                                    --
--                   -                        w                                     --
--                      -> ->                                                         --
--                      µx,Tb                                                         --
--                                                                                    --
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
--                           VHDL-AMS model of a quart car model:                     --
--------------------------------------------------------------------------------
LIBRARY IEEE;                                                  --  Libraries declaration.
LIBRARY DISCIPLINES;                                                                --
                                                                                    --
USE IEEE.MATH_REAL.ALL;                                                --  Used packages.
USE DISCIPLINES.KINEMATIC_SYSTEM.ALL;                                               --
USE DISCIPLINES.ROTATIONAL_SYSTEM.ALL;                                              --
                                                                                    --
ENTITY quarter_car_model IS --  Quantitative definitions of the quarter car model parameters:
    GENERIC (a:REAL := 1.28;                    --  For a vehicle being driven on dry asphalt
             b:REAL := 23.99;                        --  and hence the coefficients are
             c:REAL := 0.52;                      --  a = 1.28, b = 23.99 and c = 0.52
             J:REAL := 1.0;                                --  J: inertia [Kg * m²]
             R:REAL := 0.32;                                --  R: whell radius [m]
             m:REAL := 450.0;                          --  m: quarter vehicle mass [kg]
             g:REAL := 9.81);                       --  g: gravitational force [m * s-²]
    PORT (TERMINAL w1,w2:ROTATIONAL_OMEGA;                                          --
          TERMINAL v1,v2:KINEMATIC_V;                --  Declaration of block terminals.
          QUANTITY Tb: IN REAL;                                                     --
          QUANTITY Vi: IN REAL;                                                     --
          QUANTITY Wi: IN REAL;                                                     --
          QUANTITY mi_x: REAL;                                                      --
          QUANTITY lambda: REAL;                                                    --
          QUANTITY F_x: REAL;                                                       --
          QUANTITY F_z: REAL;                                                       --
          QUANTITY Speed: REAL;                                                     --
          QUANTITY Angular_Velocity: REAL);                                         --
END;                                                                                --
```

```
                                                                                --
ARCHITECTURE behavior OF quarter_car_model IS                                   --
    QUANTITY w across af THROUGH w1 TO w2;                                      --
    QUANTITY v across lf THROUGH v1 TO v2;                                      --
BEGIN                                                                           --
    break v=>Vi;                                                                --
    break w=>Wi;                                                                --
    mi_x == a * (1.0 - ((math_e)**((-1.0) * b * lambda)) - c * lambda);       --  The Pacejka
    J * w'dot == (R * F_x) - (Tb * sign(w));                    -- "Magic Formula" tire models
    m * v'dot == (-1.0) * F_x;                                                  --
    lambda == (v - (R * w)) / (v);                                              --
    F_x == F_z * mi_x;                                                          --
    F_z == m * g;                                                               --
    Speed == v;                                                                 --
    Angular_Velocity == w;                                                      --
    break v=>0.0 when not v'above(0.0);                          --  Points of singularity
    break w=>0.0 when not w'above(0.0);                                         --
END ARCHITECTURE behavior;                                                      --
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
--                                Test bench                                  --
--------------------------------------------------------------------------------
LIBRARY IEEE;                                                 --  Libraries declaration.
LIBRARY DISCIPLINES;                                                            --
                                                                                --
USE IEEE.MATH_REAL.ALL;                                          --  Used packages.
USE DISCIPLINES.KINEMATIC_SYSTEM.ALL;                                           --
USE DISCIPLINES.ROTATIONAL_SYSTEM.ALL;                                          --
                                                                                --
ENTITY car IS                                                                   --
    GENERIC (Tb,Vi,Wi:REAL);                                                    --
END;                                                                            --
                                                                                --
ARCHITECTURE behavior OF car IS                                                 --
    TERMINAL w:  ROTATIONAL_OMEGA;                                              --
    TERMINAL v:  KINEMATIC_V;                                                   --
BEGIN                                                                           --
    PROCESS BEGIN                                                --  Initial conditions:
        Tb <= 0.0;                                               --  brake torque = 0 Nm;
        Vi <= 3.0;                                                  --  speed = 3 m/s;
        Wi <= 9.375;                                    --  angular velocity = 9.375 rad/s
        WAIT FOR 70ms;                                                  --  After 0.07s:
        Tb <= 5000.0;                                      --  brake torque = 5000 Nm.
        WAIT;                                                                   --
    END PROCESS;                                                                --
    Sim1:  ENTITY quarter_car_model (behavior)                                  --
           PORT MAP (w, rotational_omega_ground, v, kinematic_v_ground, Tb, Vi, Wi);       --
                              --  Connections between all the blocks of the designed system
                                       --  (as described in the beginning of this file,
                                 --  in the "Project description (block diagram)" section).
END ARCHITECTURE behavior;                                                      --
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
--                                END OF FILE                                 --
--------------------------------------------------------------------------------
```