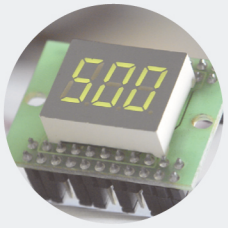
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Evaluation Methodologies of Dynamically Reconfigurable Systems in the Automotive Industry

RÚBEN DIAMANTINO GARCIA BERTELO
novembro de 2019

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Evaluation Methodologies of Dynamically Reconfigurable Systems in the Automotive Industry

Masters em Electrical and Computers Engineering
Branch Automation and Systems

Ruben Bertelo

No. 1120803

Supervisors

Manuel G. Gericota, PhD
Instituto Superior de Engenharia do Porto, Porto, Portugal

Florian Oszwald
BMW AG, Munich, Germany

November 12, 2019

Instituto Superior de Engenharia do Porto
Departamento de Engenharia Eletrotécnica
Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto

Abstract

Classical decentralized architectures based on large networks of microprocessor-based Electronic Control Units (ECU), namely those used in self-driving cars and other highly-automated applications used in the automotive industry, are becoming more and more complex. These new, high computational power demand applications are constrained by limits on energy consumption, weight, and size of the embedded components. The adoption of new centralized Electric/Electronic (E/E) architectures based on dynamically reconfigurable hardware represents a new possibility to tackle these challenges. However, they also raise concerns and questions about their safety. Hence, an appropriate evaluation must be performed to guarantee that safety requirements resulting from an Automotive Safety Integrity Level (ASIL) according to the standard ISO 26262 are met.

In this work, a methodology for the evaluation of dynamically reconfigurable systems based on centralized architectures is presented. The aim is to evaluate the reliability and probability of failure while exploring the design space and identify key aspects for continuous improvement. The methodology is divided into three stages. In the first stage, the system is decomposed, and its sub-systems are isolated before applying a Fault Tree Analysis on the elements of each sub-system. The mathematical stochastic model of Markov Chain is used in the second stage to obtain the reliability function and the quantification of the Mean Time to Failure of the system in Failure In Time format. Finally, the model is submitted to stress test by introducing uncertainties into the model and compute them through a Monte Carlo Simulation. Combined with a statistical analysis it is possible to obtain insights regarding key aspects of the model.

Applying this method it is possible to evaluate each sub-system independently and obtain the respective ASIL decomposition of the overall system. With the application of this methodology, we can evaluate the reliability and performance of dynamically reconfigurable systems and define new E/E automotive architectures and scenarios.

Contents

List of Figures

List of Tables

Glossary

Abbreviation	Description
AI	*Artificial Intelligence*
AIB	*AXI Isolation Blocks*
APU	*Application Processing Unit*
ARINC	*Avionics Application Software Standard Interface*
ARP	*Address Resolution Protocol*
ASIL	*Automotive Safety Integrity Level*
AT	*Acceptance Test*
AUTOSAR	*AUTomotive Open System ARchitecture*
AXI	*Advanced eXtensible Interface*
BIST	*Built-In Self-Test*
BPD	*Battery power domain*
BRAM	*Block RAM*
CAN	*Controller Area Network*
CCF	*Common Cause Failures*
CPS	*Cyber-Physical Systems*
CPU	*Central Processing Unit*
CRAM	*Configuration RAM*
CSU	*Configuration Security Unit*
DAL	*Design Assurance Levels*
DC	*Direct Current*
DMAC	*Direct Memory Access Controllers*
DNN	*Deep Neural Network*
DO-178C	*Software Considerations in Airborne Systems and Equipment Certification*
DRS	*Dynamic Reconfigurable System*
DSA	*Dynamic Simplex Architecture*

Abbreviation	Description
DSE	*Design Space Exploration*
DSP	*Digital Signal Processing*
ECU	*Electronic Control Unit*
ECC	*Error Correction Code*
EDC	*Error Detection Code*
E/E	*Electric/Electronic*
EN	*European Standard*
FIT	*Failures In Time*
FF	*Flip-Flop*
FMECA	*Failure Mode Effect and Criticality Analysis*
FMEDA	*Failure Modes, Effect and Diagnostics Analysis*
FPD	*Full-Power Domain*
FPGA	*Field Programmable Gate Array*
FTA	*Fault Tree Analysis*
GIC	*Generic Interrupt Controller*
GP	*General Port*
HARA	*Hazard And Risk Analysis*
HDO	*High Demand mode of Operation*
HDS	*Hardware Dependant E/E System*
HWA	*Hardware Abstraction*
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electrical and Electronics Engineers*
INTC	*INTerrupt Controller*
I/O	*Input/Output*
IP	*Internet Protocol*
IPC	*Inter-Process Communication*
ISO	*International Organization for Standardization*
IT	*Information Technology*
LDO	*Low Demand mode of Operation*
LFM	*Latent Fault Metric*
LPD	*Low-Power Domain*
LUT	*Lookup Tables*
MB	*MicroBlaze*
MCS	*Monte Carlo Simulation*
MCU	*Multiple-Cell Upsets*
MMU	*Memory Management Units*
MoA	*Model of Architecture*

Abbreviation	Description
MoC	*Model of Computation*
MPSoC	*Multi-Processor System on Chip*
MTBF	*Mean Time Between Failures*
MTTF	*Mean Time to Failure*
MTTR	*Mean Time to Repair*
NMR	*N-Modular Redundancy*
NoC	*Network-on-Chip*
OCM	*On-Chip Memory*
OS	*Operating System*
PCI	*Peripheral Component Interconnect Express*
PL	*Programmable Logic*
PLPD	*PL Power Domain*
PMFH	*Probabilistic Metric of Random Hardware Faults*
PMU	*Platform Management Unit*
POSIX	*Portable Operating System Interface*
PR	*Partial Reconfiguration*
PS	*Processing System*
RAM	*Random Access Memories*
RDB	*Reliability Block Diagrams*
RPU	*Real-Time Processing Unit*
RTC	*Real-Time Clock*
RTCA	*Radio Technical Commission for Aeronautics*
RTOS	*Real-Time-Operating-Sytem*
RTS	*Real-Time System*
SAE	*Society of Automotive Engineers*
SEU	*Single Event Upsets*
SEFI	*Single Event Functional Interrupts*
SIL	*Safety Integrity Level*
SOA	*Service Oriented Architecture*
SOME/IP	*Scalable Service-Oriented Middleware over IP*
SoC	*System-on-Chip*
SPFM	*Single-Point Fault Metric*
SRAM	*Static RAM*
TCM	*Tightly-Coupled Memory*
TMR	*Triple Modular Redundancy*
URAM	*Ultra RAM*
VC	*Vehicle Computer*

Abbreviation	Description
VM	*Virtual Machine*
WCET	*Worst Case Execution Time*
WHO	*World Health Organization*
XPPU	*Xilinx Peripheral Protection Unit*
ZynqMP	*Xilinx Zynq Ultrascale + MPSoC*

Acknowledgements

The realization of this thesis would not be possible without the contribution of several persons. To all of them my profound gratitude.

I would like to sincerely thank my supervisor Professor Manuel Gericota, for its guidance, support and tireless availability to successfully conclude this thesis.

I would also like to specially thank my supervisor Florian Oszwald for enabling this opportunity and provide the basis for this thesis. All the support and brainstorming sessions were fundamental to the definition of this thesis.

To the LT3 and my colleagues, that together with me faced the challenge of the master thesis, an enormous thank you for the companionship, water breaks and for facilitating my integration in Germany.

To my fellow friends of the former *Quem quer ser mestre* group, thank you for the friendship, motivation and knowledge sharing that allowed us to reach higher goals.

To serendipity of life and friends, thanks for the encouragement, optimism, discoveries and inspiration throughout my life.

Finally, I want thank to my family for the strength and determination to surpass obstacles. As someone would say: *E vou! E vou! E vou!*.

Chapter 1

Introduction

One of the big challenges for the automotive industry is the realization of highly or fully automated driving vehicles. Furthermore, future automotive systems may not be limited to the physical boundaries of modern vehicles but are rather interacting beyond this point with the cloud. [22] These challenges stretch the boundaries of the traditional design and implementation on the embedded systems to adopt and include other architectures such as Service Oriented Architecture (SOA).

Subsequently, the embedded systems complexity increases dramatically to encompass the increasing complexity of hardware and software. The increasing intelligent functionalities and their components cannot be efficiently deployed into the traditional way into small Electronic Control Units (ECU). They have to be integrated on platforms with higher computational capacity, by using for example function specific hardware accelerators. [13] The dependency of automotive architectures on computing systems makes their reliability imperative. But statistically speaking, the probability of errors in a complex system makes the accuracy of the response of the final outputs uncertain. It is necessary to have mechanisms to control and manage errors so the computing systems are able to execute their functionalities correctly regardless of hardware and software failures. [7]

As state of the art on automotive context, Electric/Electronic (E/E) architecture is static in the sense that it is deployed once the car is built. To provide high intelligent functionalities, the E/E architecture has to be able to dynamically change during runtime. [2] The same approach can be used to ensure redundancy, reliability and safety. This is were Dynamically Reconfigurable Systems have a role to play in order to achieve fail-operational systems.

On this paradigm, it is imperative to predict the biggest challenges of ex-

ploding design complexity with a special focus on the centralized architectures and the problems of reliability and fault tolerance. [15] Being so, we need evaluation methods, criteria and tools in order to evaluate and assess the increasing complexity of and reliability on Dynamically Reconfigurable Systems.

1.1 Context

Autonomous Driving has a major impact on future E/E-architecture in the automotive industry. The way towards autonomous driving is described in the levels of driving automation of the standard Society of Automotive Engineers (SAE) J3016. This standard describes six levels reaching from level zero to level five, where the latter contains the driver less driving or the fully automated driving as represented in Fig. 1.1. [1]

Figure 1.1: SAE J3016 Levels of Driving Automation. [1]

Current automotive architectures feature only partially automated driving (level two of the SAE scale) where the driver is always in charge of monitoring the system and must react in case of a failure. With the release of any future vehicle function that are included on the highly and fully automated driving scenarios (levels four and five of the SAE scale), the automotive industry needs to transit from a fail-safe to a fail-operational mode. [2]

This thesis framework is related to the AutoKonf project aiming to achieve the fail-operational state by implementing a novel reconfigurable E/E architec-

ture consisting in three ECUs. The functions for braking and steering are implemented separately in two ECUs. The dynamic reconfiguration of these ECUs is realized by a third ECU that can be dynamically activated from a hot standby mode in order to serve as a fallback instance either for the brake or steering control function. An overview of the principal targeted E/E-architecture is shown in Fig. 1.2. [2]

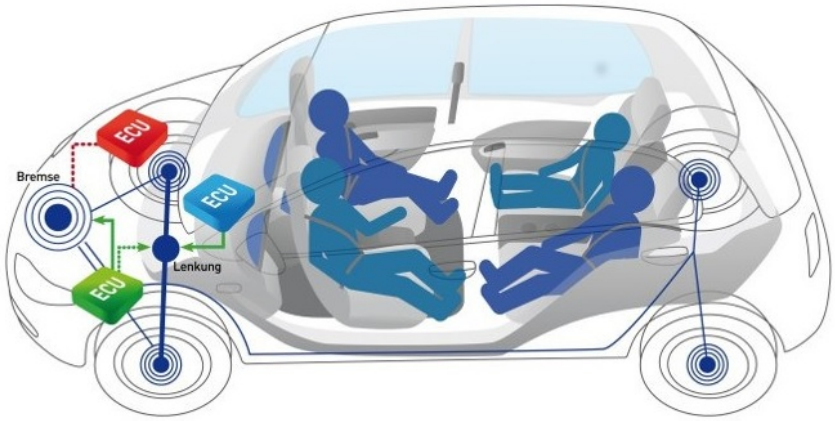


Figure 1.2: Autokonf Architecture Overview. [2]

1.2 Motivation

It is noticed in the last two decades that the number of road accidents has increased significantly and as an outcome, a massive number of casualties have occurred. According to the global status report on road safety released by the World Health Organization (WHO), over 1.2 million people die each year on the world's roads and between 20 and 50 million suffer non-fatal injuries. [23]

Reliability of vehicle architectures and their components are one of the major factors that have a direct effect on the safety of road users. Hence, as the technology for autonomous driving and other highly automated applications are becoming more advanced and excitement grows, so do concerns and questions about their safety. Current standards are strict and comprehensive resulting in high complex systems not only in terms of number of components/elements to achieve redundancy but also their verification and validation. On the counterpart, the field of automotive electronics is characterized by an ever-increasing demand for performance. At the same time, the energy consumption, the weight, and the size of embedded components is heavily constrained. Furthermore, ac-

cording to current technological previsions, a major threat is the imperfection of nano-electronics on the next years. [15]

The combination of these factors makes of the utmost importance the analysis of a complex system through modelling, partitioning, and evaluation of its subsystems in order to achieve intelligent functionalities.

1.3 Objectives

Dynamically reconfigurable systems are not established in the automotive domain. Centralized architectures are becoming essential in the future due to energy and computing power. The acceptance and employment of these type of systems is new in the automotive domain since it has not been demonstrated yet, how to certify such systems according to ISO 26262. Hence, in the scope of this work to develop a suitable evaluation methodology it is necessary to:

- Review the concepts regarding safety, dependability, redundancy, reconfiguration and define concepts that will be used across this work;
- Review the safety standards for safety-critical systems and automotive industry on the E/E centralized architectures perspective;
- Review State of Art regarding automotive E/E architectures and dynamically reconfigurable systems;
- Research the evaluation methods towards standards certification of centralized E/E architectures and dynamic reconfigurable systems;
- Development and implementation an evaluation methodology;
- Define a case study and scenarios to apply the evaluation methodology.

1.4 Structure

This dissertation is structured as follows:

In Chapter 2 it exposes the background and foundation concepts of safety critical systems, dependability, redundancy and reconfiguration.

In Chapter 3 it presents the State of Art regarding E/E architectures in the automotive industry and autonomous driving, standards and regulations, and finally evaluation methodologies.

In Chapter 4 it describes the case study where the proposed dynamically reconfigurable system is characterized and the proposed evaluation methodology is presented.

The Chapter 5 refers to the definition of the scenarios and evaluation constraints.

In Chapter 6 the results are presented and discussed.

Finally, Chapter 7 presents the conclusions regarding this work and the proposed evaluation methodology.

Chapter 2

Foundations

This chapter has no pretension to document the state of the art regarding the implementation of dependability, redundancy and reconfiguration but instead give an exposure of the groundings of this work and the different redundant and reconfigurable schemes.

The concepts of redundancy, reconfiguration and fault tolerance are deeply interconnected and is often difficult to isolate each concept without entering the field of the other concept. Clarifying these concepts is surprisingly difficult when the topic of discussion are complex systems in which uncertainty exists about its sub-system boundaries and how these concepts are applied. Furthermore, the very complexity of systems is often a major challenge, the determination of possible causes or consequences of failure can be a very subtle process, and therefore the provisions for preventing faults from causing failures accompanies this complexity. [3] Having so, in this chapter it will be given an introduction on the safety-critical systems aspects, followed by the a more thoroughly exposure of dependability, redundancy and reconfiguration.

2.1 Safety-critical systems

Safety-critical systems are systems whose failure could result in loss of life, significant property damage or damage to the environment. There are many well-known examples in application areas such as medical devices, aircraft flight control, military and nuclear systems. [24] As for all safety-related systems, all aspects of reliability, availability, maintainability and safety have to be considered. To meet safety requirements, special procedures and standards were developed in different technical disciplines like aircraft, space, military, nuclear and, later, automotive systems. [25]

Real-time safety critical systems can be coarsely classified into *fail-safe* and *fail-operational* systems. In a *fail-safe system*, the controlled object has to transit to a *safe state* when a failure on the controlling computer system occurs and the controlled object needs has to be able to stay in this safe state without the help of the computing system. [26] Many medical systems fall into this category, when a failure occurs e.g on a infusion pump, it will stay blocked in safe state until human assistance is provided. The safe state is a state of the controlled object in which no human life is at risk. This state can be entered in response to faults and ensures that the system continues to satisfy its safety requirements. [16]

On the *fail-operational* context, the system does not have a safe state in which the controlled object can stay without the help of the controlling computer system. For example, a fly-by-wire system without a backup system does not have a safe state. However, a fly-by-wire system with a mechanical backup system can be viewed as a fail-safe system: the controlled object includes the mechanical backup system which allows it to stay in a safe state even when the controlling computing system has failed. [26] A fail-operational definition can be extended to a *component/system that continues operational after the occurrence of a failure if a fail-safe does not exist*. [24] A fail-operational system needs to maintain a certain minimum level of functionality, even when it is subject to a certain number of faults. Depending on the safety requirements of the system, a degraded functionality might be sufficient. [16]

The degraded functionality represents a smooth change of some system functionality to a lower state as a response to an error. The *graceful degradation* has been used in a number of domains to allow systems to maintain basic (degraded) functionalities while reconfiguration or repair is performed. It makes use of hardware and software redundancy, and one example of implementation is the checkpoint and restart that will be explained in more detail in section 2.3. [27]

2.2 Dependability

In practice, all electronic components are at risk of experiencing anomalies that can lead to situations in which a system is unable to fulfil its desired function. Such a condition is called a failure and might, in particular, impair the safety of the considered system. [16] Computer system dependability *is the quality of the delivered service such that reliance can justifiably be placed on this service*. [28] On a broader sense, dependability is the ability of a system to deliver its intended level of service to either another system or end user. Along with cost and performance, dependability is the third main criteria upon which critical system-related decisions are made. Dependability evaluation is important, because it helps identify elements and specificities of the system which are critical for its purpose. [4]

Dependability *attributes* describe the properties which are required of a system. Dependability *threats* express the causes for a system to cease to perform its function or, in other words, the threats to dependability. Dependability *means* are the methods and techniques to enable the development of a dependable system, such as fault prevention, fault tolerance, fault removal, and fault forecasting. [4] The dependability tree is summarized in Fig. 2.1. [3]

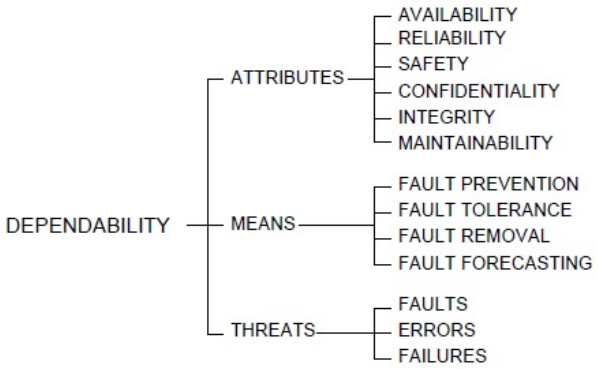


Figure 2.1: The dependability tree. [3]

2.2.1 Reliability, Availability and Safety

The attributes of dependability are the properties that are expected from a system. Three primary attributes are *reliability*, *availability*, and *safety*. [4] Depending on the application, one or more of these attributes may be needed to appropriately evaluate a system behaviour. As shown in Fig. 2.1 the attributes can be extended to [3] :

- *Confidentiality*: absence of unauthorized disclosure of information;
- *Integrity*: absence of improper system state alterations;
- *Maintainability*: ability to undergo repairs and modifications;
- *Security*.

Apart from the maintainability, these extended attributes are secondary to dependability but primary regarding *security* attribute. Security has a vast field of specification and validation and in [3] it is given a broader explanation. This work will focus on the primary attributes for dependability.

2.2.1.1 Reliability

Reliability is a measure of the continuous delivery of correct service. High reliability is required in situations when a system is expected to operate without interruptions, as in the case of a heart pacemaker, or when maintenance cannot be performed because a system cannot be accessed, as in the case of deep-space applications. [24] It can be defined as the *probability that the system remains operational over an observation period. It is an appropriate measure for evaluating the effectiveness of systems where no down time is tolerated*. [7]

Reliability $R(t)$ of a system at time t is the probability that the system operates without a failure in the interval $[0, t]$, given that the system was performing correctly at $t = 0$. Unreliability $Q(t)$ expresses the probability of failure under the same conditions. The reliability $R(t)$ and the unreliability $Q(t)$ are related as [19] :

$$Q(t) = 1 - R(t)$$

2.2.1.2 Availability

Availability is typically used as a figure of merit in systems in which a function or service can be delayed or denied for short periods without serious consequences. [7] In more detail, availability $A(t)$ of a system at time t is the probability that the system is functioning correctly at the instant t. $A(t)$ is also referred as point availability, or instantaneous availability. It is defined by[4]:

$$A(T) = \frac{1}{T} \int_0^T A(t)\mathrm{d}t.$$

$A(T)$ is the value of the point availability averaged over some interval of time T . This interval might be the life-time of a system or the time to accomplish some particular task. [4]

2.2.1.3 Safety

Safety is required in safety-critical applications where a failure may result in human injury, loss of life, or environmental disaster. Safety can be considered as an extension of reliability, namely reliability with respect to failures that may create safety hazards. As an example, consider an alarm system. The alarm may either fail to function correctly even though a danger exists, or it may give a false alarm when no danger is present. The former is classified as a fail-unsafe failure. The latter is considered a fail-safe one. [19]

Safety $S(t)$ of a system at time t is the probability that the system either performs its function correctly or discontinues its operation in a fail-safe manner in the interval $[0, t]$, given that the system was operating correctly at $t = 0$. [19]

2.2.1.4 Modelling Dependability

Modelling and analysis methods are used to predict the frequency of failure and to estimate the likelihood of violating a safety goal due to failures. System analysis methods can be classified into two generic categories: *inductive* methods and *deductive* methods. [29] In an *inductive* system analysis, it is postulated a particular fault or initiating event and attempt to find out the effect of this fault on the entire system failure. In other words, the inductive methods are applied to determine what system states (usually fail-states) are possible. Examples of inductive system analysis include Failure Modes, Effect and Diagnostics Analysis (FMEDA) and Failure Mode Effect and Criticality Analysis (FMECA) normally in the form of tables where are described the function/system or subsystem failure in an individual component or program module, failure detection methods, their compensating provisions and at greater extent the probability of failure by severity class ranking. [30]

In a *deductive* system analysis, it is postulated a system failure and attempt to find out what modes of system or component behaviour contribute to the system failure. One example is the Fault Tree Analysis (FTA) where a failure scenario is considered, and decomposed into its possible causes normally using a tree structured presentation. [30] Each possible cause is then investigated and further refined until the basic causes of the failure are understood. Similar to FTA, we can also use Reliability Block Diagrams (RDB) and Reliability Graphs to specify various combinations of component failures that lead to a specific state or performance level of a system (combinatorial models). [29]

Dynamic fault trees extend traditional FTA to include dynamic system behaviour such as sequence dependence and shared pool of resources (stochastic models). This particularity allows to incorporate the dynamic behaviour into Markov chains, which are used for the reliability analysis. The two main concepts in the Markov model are system states and state transitions. For representing the system reliability, each state of the Markov model generally represents a distinct combination of faulty and fault free components. As time passes and failures occur, the system goes from one state to another until the fail-state (usually the system failure) is reached. The state transitions are characterized by parameters such as failure rates (λ), fault coverage factors, and repair rates (μ). [29]

The Fig. 2.2 shows the Markov reliability model of a *1 out of 2* redundant system with three operational and one fail-state (4). States and transitions between states are first recognized, and the rates λ and μ of transitions are allocated.

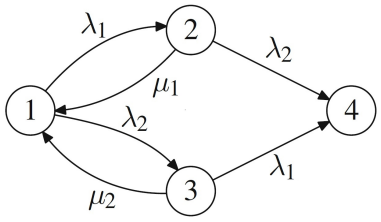


Figure 2.2: Markov chain of a single-component system.[4]

2.2.1.5 Evaluation of Dependability

The goal of Markov chain analysis is to calculate the probability of the system to be in state i at time t, $Pi(t)$. Taking the example in Fig. 2.2 and assuming that at $t = 0$ all the system components are operational the probability of state 1 is $P1(0) = 1$ [19].

To determine the following probabilities $Pi(t)$, it is required to derive a set of differential equations, one for each state i of the system. These equations are called state transition equations which are determined in terms of the failure rate λ. The state transition matrix M representing the different states equation for the above example is [19]:

$$M = \begin{bmatrix} -\lambda_1 & \mu_1 & 0 & 0 \\ \lambda_1 & -\lambda_2 - \mu_1 & 0 & 0 \\ \lambda_2 & 0 & -\lambda_1 - \mu_2 & 0 \\ 0 & \lambda_2 & \lambda_1 & 0 \end{bmatrix}$$

Using state transition matrix, state transition equations are derived as follows. Let $P(t)$ be a vector whose *ith* element is the probability $Pi(t)$ that the system is in state i at time t. The matrix of a system of state transition can be represented by the derived equations given as follows [19]:

$$\frac{d}{dt}P(t) = M \times P(t)$$

Once the system of equations is solved and the probabilities $Pi(t)$ are known, system reliability can be computed as a sum of probabilities [19]:

$$R(t) = \sum_{i \in O} P_i(t)$$

The Unreliability $Q(t)$ is related to the *failure rate* which is defined as the expected number of failures per unit time. This definition has been broadly used to demonstrate a typical evolution of a system during its *useful life*. It is assumed to have a constant value λ. Since $Q(t) = 1 - R(t)$ and according to the *exponential failure law*, the reliability can be represented by the [31]:

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

Mean Time to Failure (MTTF) of a system is the expected time of the occurrence of the first system failure. MTTF is related to the reliability as follows [19]:

$MTTF = \int_0^{+\infty} R(t)\mathrm{d}t.$

So if the reliability function obeys the exponential failure law, then:

$MTTF = \int_0^{+\infty} e^{-\lambda t}\mathrm{d}t = \frac{1}{\lambda}.$

It is common to present MTTF in Failures In Time (FIT) format, which shows how many failures can be expected from one billion hours of operation. If MTTF is expressed in hours, then [19]

$FIT = \frac{10^9}{MTTF}$

Other measures for dependability are the *Mean Time to Repair* (MTTR) and *Mean Time Between Failures* (MTBF). MTTR of a system is the average time required to repair the system and MTBF is the average time between failures of the system. Having so, the MTBF is represented by the sum of MTTF and MTTR. [19]

$MTBF = MTTF + MTTR.$

2.2.2 Fault, Error and Failure

Failures are caused by errors and errors are caused by faults. The causality relationship of a fault, error and failure is summarized in Fig. 2.3. A fault is active when it produces an error otherwise is dormant. After the activation, the fault produces an error which can be propagated by successively being transformed in other errors. Lastly it will cause a failure on the system and this failure can cause an external fault to other subsystem. [3]

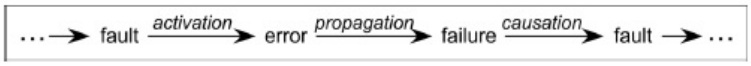


Figure 2.3: The fault, error and failure causality. [3]

A *fault* is the adjudged or hypothesized cause of an error. [3] It can be a physical defect, imperfection, or flaw that occurs in some hardware or software component. [4] Faults whose activation is reproducible are called solid, or hard faults, whereas faults whose activation is not systematically reproducible are elusive or soft faults. [3]

An *error* is a deviation from correctness or accuracy in computation, which occurs as a result of a fault and are usually associated with incorrect values in the system state. For example, a circuit or a program computed an incorrect value, or incorrect information was received while transmitting data. [4] An error is detected, if the system is aware of its presence. It is latent if it is present, but

not detected, yet. Errors produced by intermittent faults are usually termed soft errors. [3]

A *failure* is a non-performance of some functionality which is due or expected and the service it delivers is deviated from the system specification for a specified period of time. A system may fail either because it does not act in accordance with its specification, or because the specification did not adequately describe its function. [4]

2.2.2.1 Fault Origins and Types

Faults are, in turn, caused by numerous problems occurring at the specification, implementation, or fabrication stages of the design process either software or hardware. Faults due to *Incorrect specification* are common and result from incorrect algorithms, architectures, or requirements. For example, the lack of information detail from the vendor regarding the specification of a certain Intellectual Property (IP) during its integration on the development of a System-on-Chip (SoC) can lead to faults. [4]

Incorrect implementation, usually referred to as *design faults* or *development faults* [3], occur when the system implementation does not adequately implement the specification. In hardware, these include poor component selection, logical mistakes, poor timing, or poor synchronization. In software, examples of incorrect implementation are bugs in the program code and incorrect software component reuse. Additionally, the software execution depends according to its operating environment and faults are likely to occur when these differences are not taken into account. [4]

Many hardware faults are due to *physical defects* and *defective components*. These include manufacturing imperfections and random device defects. Fabrication defects were the primary reason for applying fault-tolerance techniques to early computing systems, due to the low reliability of components. [4] Other cause is the degrading hardware such as decreased maximum operation frequency of a power-aware system due to degraded thermal performance. [32]

The fourth cause of faults are *external factors*, which arise from outside the system boundary: environmental disturbances, either accidental or deliberate human actions. External factors include phenomena that directly affect the operation of the system, such as temperature, vibration, electrostatic discharge, and nuclear or electromagnetic radiation that affect the inputs provided to the system. [4]

A *common-mode fault* is a fault which occurs simultaneously in two or more redundant components. It can occur due to a fault on a component that will create dependencies between the redundant units which cause them to fail simultaneously such as common communication buses, single source of power

and Input/Output (I/O) bus. [4] On the class of human-made faults are the *operational faults* and *configuration faults*. Among others, these errors normally result from wrong setting of parameters that can affect security, networking and storage. Such faults can occur during configuration changes performed during upgrades, changes or maintenance performed during system operation. [3]

2.2.2.2 Hardware Faults

Hardware faults are classified with respect to fault duration into permanent, transient, and intermittent faults. A *permanent fault* remains active until a corrective action is taken. These faults are usually caused by some physical defects in the hardware, such as shorts in a circuit, broken interconnections, or stuck cells in a memory. [24]

A *transient fault* remains active for a short period of time. Because of their short duration, transient faults are often only detected through the errors that result from their propagation. Transient faults are the dominant type of faults in today's integrated circuits. For example, about 98% of Random Access Memories (RAM) faults are Single Event Upsets (SEUs) that affect a bit of memory and Multiple-Cell Upsets (MCU) which corrupt several bits. The causes of transient faults are mostly environmental, such as alpha particles, atmospheric neutrons, electrostatic discharge, electrical power drops, or overheating. Single Event Functional Interrupts (SEFI) are faults caused by heavy-ion strikes in the control circuitry that can corrupt the memory device. [33]

A transient fault that becomes active periodically is an *intermittent fault*. Intermittent faults are due to implementation flaws, ageing and to unexpected operating conditions. [4]

2.2.2.3 Software Faults

Software eliminates many of the physical constraints of hardware, for example, it does not suffer from random fabrication defects and its performance will not degrade over time. Software fault modelling is very difficult compared to the hardware fault modelling since it is more complex in logic and it is diverse in its development methods. Hence the software must be designed in a way to tolerate the faults to achieve the design goal and to be able to use it in safety critical systems. In contrast to hardware failures, software failures can be caused by [34]:

- Design faults such as incorrect algorithms, incorrect implementation of algorithms, incorrect software documentation, which will lead to incorrect user actions;
- Input data which is processed by the software;

- Temporary failures of hardware which occur under external factors (ionizing radiation, temperature, humidity or another factor);
- Coding faults such as logical errors within calculations, stack overflow or underflows and bad use of variables;
- Feature upgrades/software releases and poor development processes.

2.2.3 Means for Dependability

The purpose of the *means* is the improvement of our confidence in the dependable system. It relies on methods and techniques during different development phases. It is strongly related with the impairments and attributes of dependability exposed before.

2.2.3.1 Fault Avoidance and Error Removal

Fault avoidance techniques are employed during the development phase, in order to avoid the introduction of development faults into the system. [35] It is achieved by quality control techniques during the specification, implementation, and fabrication stages of the design process. [4] In large systems it is very likely that some development faults cannot be avoided, no matter how carefully fault avoidance techniques are applied and therefore, error removal techniques are used.

Fault removal during the development phase of a system life-cycle consists of three steps: verification, diagnosis, and correction. Verification, is the process of checking whether the system adheres to a set of predefined conditions. If it does not, the other two steps have to be undertaken: diagnosing the fault that prevented the verification conditions from being fulfilled, and then performing the necessary corrections. After correction, the verification process should be repeated in order to check that fault removal had no undesired consequences. Checking the specification is usually referred to as validation. [3]

2.2.3.2 Error-Forecasting

Fault forecasting is a set of techniques in order to estimate how many faults are present in the system, possible future occurrences of faults, and the consequences of faults. Fault forecasting is done by performing an evaluation of the system behaviour with respect to fault occurrences or activation. [4] Fault forecasting is conducted by performing an evaluation of the system behaviour with respect to fault occurrence or activation. Evaluation has two aspects [3]:

- *Qualitative*, which aims to identify, classify, and rank the failure modes, or the event combinations (component failures or environmental conditions) that would lead to system failures;

- *Quantitative* which aims to evaluate in terms of probabilities the extent to which some of the attributes are satisfied; those attributes are then viewed as measures. The two main approaches to probabilistic fault-forecasting, aimed to derive probabilistic estimates, are modelling and (evaluation) testing.

2.2.3.3 Fault Tolerance

No matter how much effort is spend during the development phase on fault avoidance and error removal, there is a very high probability that faults remain undiscovered until the system is in operation. When the dormant faults and faults related to the physical degradation become active during the operational phase, then an error occurs in the system. [35]

Fault tolerance is the ability of a system to continue to perform its task after the occurrence of faults. The ultimate goal of fault tolerance is to prevent system failures from occurring and so achieve a dependable system. [35] In a broad sense, fault tolerance is associated with reliability, with successful operation with the absence of breakdowns. Furthermore, it is also related to availability in the sense that the system should be able to handle faults in individual hardware or software components, power failures, or other kinds of unexpected problems and still meet its specification. [4]

There are various approaches to achieving fault tolerance. Common to all these approaches is a certain amount of redundancy. [4] The most relevant techniques will be presented on the following section.

2.3 Redundancy

In a fault tolerant design, redundancy is used to provide the resources and methods needed to negate the effects of failures. The redundancy can be either extra time or extra components or both. [7] Since redundancy is the provision of functional capabilities that would be unnecessary in a fault-free environment, redundancy increases the complexity of a system beyond the requirement of achieving the system objective. [4]

Time redundancy is usually provided by software and involves extra execution of the same calculation and in some cases using different methods. Comparisons or other operations on the multiple results provide the basis for subsequent action. Component redundancy is the use of extra gates, memory cells,

bus lines, functional modules, and so forth. [7] There are several ways to implement redundancy either on software, hardware or both on complex systems. For latter case, since the separation would require deeper analysis on their own and includes concepts of time and information redundancy which are predominantly present on software redundancy, it will be presented an holistic point of view, as *System/Process Redundancy*, where it will be briefly described some fault detection techniques, their implementation and the correspondent redundancy.

2.3.1 Hardware Redundancy

Hardware redundancy is achieved by providing two or more physical copies of a hardware component. [7] Hardware redundancy may be the only way to improve the dependability of a system. For example, in situations in long life systems, e.g., communication satellites, redundant components allow uninterrupted operating time to be prolonged. As the reliability of basic components improved, redundancy was shifted to higher levels. Larger components, such as memories or processor units, became replicated. [25]

The overall increase in complexity caused by redundancy can be quite severe. It may diminish the dependability improvement, unless redundant resources are allocated in a proper way. A careful analysis has to be performed to show that a more dependable system is obtained at the end. A number of possibilities have to be examined to determine at which level it is best to provide redundancy and which components should be made redundant. In high-level redundancy, the entire system in duplicated, as shown in Fig. 2.4 a). In low-level redundancy, the n-duplication takes place at component level, as shown in Fig. 2.4 b). For this particular comparison, despite having the same number of components and assuming that the component failures are mutually independent, low-level redundancy yields a higher reliability than high-level redundancy. [4]

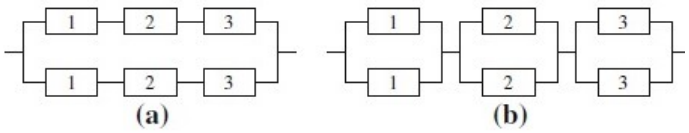


Figure 2.4: a) High-level and b) low-level redundancy.[4]

There are three types of hardware redundancy: passive, active, and hybrid. *Passive* redundancy achieves fault tolerance by masking the faults that occur without requiring any action from the system. It is usually used in high-reliability applications in which even short interruptions of system operation are unacceptable, or it is not possible to repair the system. Examples of such

applications include aircraft flight control systems, embedded medical devices such as heart pacemakers and deep-space electronics. [4]

The most common form of passive hardware redundancy is Triple Modular Redundancy (TMR). The N-Modular Redundancy (NMR) approach is based on the same principle as TMR, but it uses n modules instead of three. The components are replicated to perform the same computation in parallel. Majority voting is used to determine the correct result as shown in Fig. 2.5. If one of the modules fails, the majority voter masks the fault by recognizing as correct the result of the remaining two fault-free modules. A TMR system can mask only one module fault. A failure in either of the remaining modules would cause the voter to produce an erroneous result. [24] [5]

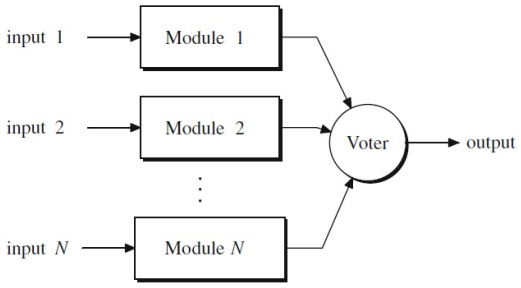


Figure 2.5: N-modular Redundancy.[5]

Related to the TMR and NMR is the *fault masking* technique which is used to prevent generation of erroneous results. [7] This is done either by isolating or correcting the fault before it affects the system's output. However, fault masking techniques do not provide fault detection, instead the effects of faults are automatically neutralized without notification of their presence. [31]

The fact that a (TMR) system can mask one module fault does not immediately imply that the reliability of a (TMR) system is higher than the reliability of a non-redundant system. In this example the voter is a *single point of failure* which is any component within a system whose failure leads to the failure of the system. One possibility is to triplicate voters as shown in Fig. 2.6a) and also apply multi stage (TMR) systems as shown in Fig. 2.6b). Such a structure avoids the single point of failure, but requires consensus to be established among three voters. Such a technique is used, for example, in Boeing 777 aircraft to protect all essential hardware resources, including computing systems, electrical power, hydraulic power, and communication paths. [4]

With *Active* redundancy, after the detection of the fault, the actions of location, containment and recovery are performed to remove/replace the faulty component and return the system back to an operational state. Active tech-

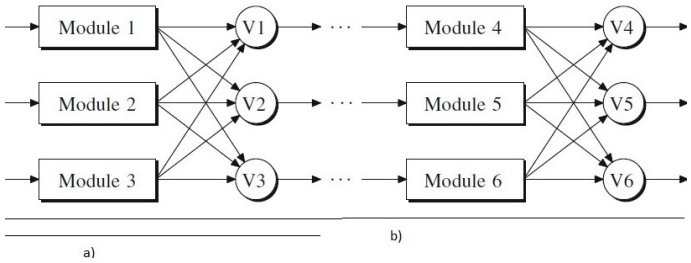


Figure 2.6: a) (TMR) with 3 voters b) Multiple-stage (TMR) system.[4]

niques require that a system is stopped and reconfigured to tolerate faults. It is typically used in applications requiring high availability, such as time-shared computing systems or transaction processing systems, where temporary erroneous results are preferable to the high degree of redundancy required for fault masking. One technique is the *duplication with comparison* shown in Fig. 2.7. Two identical modules operate in parallel. Their results are compared using a comparator. If the results disagree, an error signal is generated. [4]

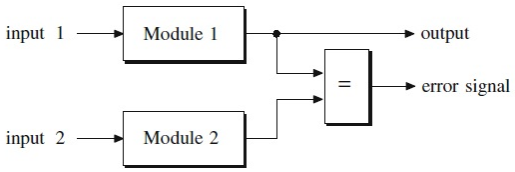


Figure 2.7: Duplication with comparison.[4]

Standby redundancy is another technique for active hardware redundancy. The basic configuration is shown in Fig. 2.8 where one of the n modules is active. The remaining $n-1$ modules serve as back-ups. A switch monitors the active module and switches operation to a back-up if an error is detected by a fault-detection unit. [4]

There are two types of standby redundancy: hot standby and cold standby. In the *hot standby*, both operational and back-up modules are powered up. This minimizes the downtime of the system due to reconfiguration. In the *cold standby*, the back-ups are powered down until they are needed to replace a faulty module. This increases reconfiguration time by the amount of time required to power and initialize a spare. However, since spares do not consume power while in the standby mode, such a trade-off is preferable for applications where power consumption is critical such as satellite systems. [4]

Hybrid redundancy combines advantages of passive (fault masking) and active (fault detection, location and recovery) approaches to reconfigure a system

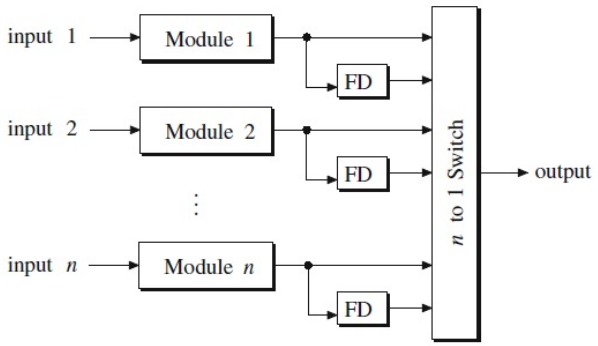


Figure 2.8: Standby redundancy.[4]

when a fault occurs. An example is the *Self-purging redundancy* consisting of n identical modules which performs the same computation in parallel and actively participate in voting is presented in Fig. 2.9. The output of the voter is compared to the outputs of each individual module to detect disagreement. If a disagreement occurs, the switch opens and removes, or purges, the faulty module from the system. [4]

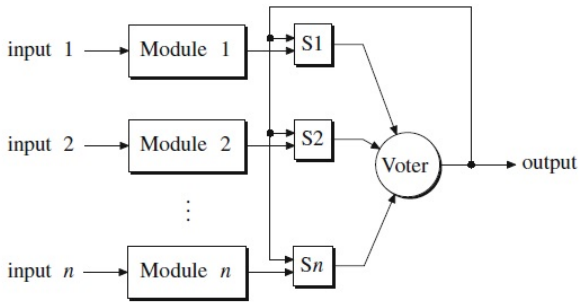


Figure 2.9: Self-purging redundancy. [4]

2.3.2 Time Redundancy

In some applications, it is preferable to use extra time rather than extra hardware to tolerate faults. The fundamental concept of time redundancy is to perform the same computation or data transmission multiple times by using the same hardware components. The results obtained are either compared with each other or will undergo an acceptance test. The intention of time redundancy is the detection of temporary faults. The detection of permanent faults is not possible by using the simple concept of re-execution, because the computations are performed on the same piece of hardware. [35]

If a fault is transient, then stored results differ from the recomputed one. If the repetition is done twice, as shown in Fig. 2.10 , a fault can be detected. If the repetition is done three times, as shown in Fig. 2.11, a fault can be corrected. Time redundancy is useful to differentiate transient from permanent faults. If a fault disappears after re-computation, we can assume that it is transient. [4]

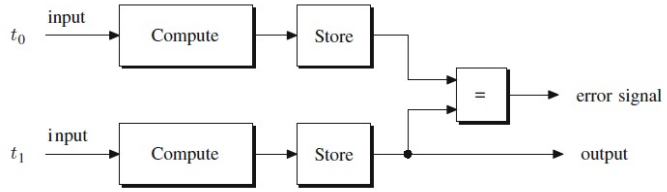


Figure 2.10: Time redundancy for transient fault detection. [4]

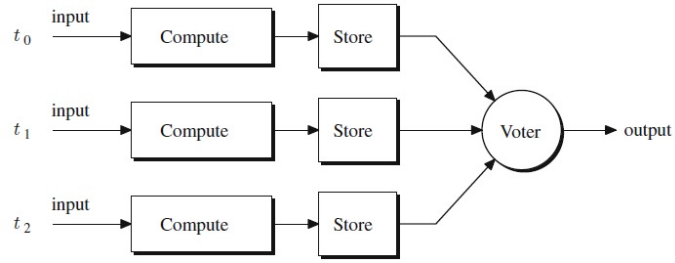


Figure 2.11: Time redundancy for transient fault correction. [4]

2.3.3 Information Redundancy

Information redundancy is based on redundancy applied to data. Commonly it is used to detect errors that occur during transmitting or storing data. In its simplest form, data is just replicated in the system. [35]

The most widely used information redundancy technique is *error control coding*. In general error control coding is the mapping of the original data word into a code word by adding redundancy. Therefore, the binary code word has usually more bits than the original binary data word. After transmitting or storing a code word, the original data word is reconstructed from the received word by a decoding process. When this decoding process detects and corrects errors in the received word, then the code is called an *Error Correction Code* (ECC). [35]

When errors can be detected, but not corrected, then the code is called an *Error Detection Code* (EDC). In general, the error detection is based on checking whether the received word is a code word or not. Therefore, errors that corrupt a

code word in such a way that another code word originates cannot be detected. The usage of ECCs and EDCs for protecting memories and processor registers against transient faults are state of the art in current designs of server processors or embedded processors for avionics. [35]

Coding checks are applicable to programs whose data can be encoded. For example, a cyclic code such as *cyclic-redundancy-check* can be used if the information is merely transported from one module to another without modifying its content. An *arithmetic code* can be used to detect errors in arithmetic operations. In some programs, it is possible to reverse the output values and to compute the corresponding input values. In this case, *reversal checks* can be applied. A reversal check compares the actual inputs of a program with the computed ones. A disagreement indicates a fault. *Reasonableness checks* use semantic properties of data to detect faults. For example, a range of data can be examined for overflow or underflow to indicate a deviation from the systems requirements. Structural checks are based on known properties of data structures. For example, a number of elements in a list can be counted, or links and pointers can be verified. [4]

2.3.4 Software Redundancy

Whereas we have to cope with ageing hardware, software obviously does not age in the same manner. This opportunity might be exploited by applying software redundancy measures such as spatial redundancy by code replication or temporal redundancy by re-computation to obtain a higher level of reliability of the system. [15]

In addition to the techniques used in the Information and Time Redundancy that is implemented through software, single-version techniques comprises two more techniques: fault containment and fault recovery. [36] *Fault containment* in software can be achieved by modifying the structure of the system and by imposing a set of restrictions defining which actions are permitted within the system [4]:

- *Modularization* by attempting to prevent the propagation of faults by decomposing a system into modules, eliminating shared resources, monitor messages and limit the amount of communications between modules;
- *Partitioning* is the isolation between functionally independent modules and can be achieved by adopting a modular hierarchy of a software system;
- *System closure*: This technique is based on the principle that no action is permitted unless explicitly authorized. Any component of a system is granted only the minimal capability to perform its function;

- *Atomic actions* are activities in which the components interact exclusively with each other. There is no interaction with the rest of the system for the duration of the activity.

On Software redundancy, *single-version* techniques aim to improve the fault tolerance of a software component by adding to it mechanisms for fault detection, containment, and recovery. *Multi-version* techniques use redundant software components which are developed following design diversity rules. For example, different teams, different coding languages, or different algorithms can be used to maximize the probability that different versions do not have common faults. The most critical issue in multi-version techniques is assuring independence between the different versions of a software module through design diversity. *Design diversity* aims to protect multiple versions of a module from common design faults. Software systems are vulnerable to common design faults if they implement the same algorithm, use the same program language, or if they are developed by the same design team and tested using the same technique. [4]

A common *Fault recovery* mechanism for single-version software fault tolerance is the *checkpoint and restart*. A typical mechanism for initiation of fault recovery in software is the exception handling. There are two types of checkpoints: static and dynamic. A *static* checkpoint takes a single snapshot of the system state at the beginning of the program execution and stores it in the memory. If a fault is detected, the system returns to this state and starts the execution from the beginning. *Dynamic* checkpoints are created dynamically at various points during the execution. If a fault is detected, the system returns to the last checkpoint and continues the execution. Fault-detection checks need to be embedded in the code and executed before the checkpoints are created. [4]

The *recovery block* technique can be applied as a multi-version technique by making use of checkpoint and restart on multiple versions of a software module. Its configuration is shown in Fig. 2.12 where N different versions of a program and a single centralized Acceptance Test (AT) are available. A program is only invoked, if the AT of the previously invoked programs has failed so the result of the first program whose output is accepted is used. Checkpoints are created every time before a new version executes. Various checks are used for acceptance testing of the active version of the module. [36]

In *N-version* programming N different versions of the program are implemented and executed concurrently on the same module. Their outputs are compared and the result is selected by a majority vote by a selection algorithm. The block diagram is shown in Fig. 2.13. [4] The selection algorithm is usually implemented as a generic voter. This is an advantage over the recovery block technique, which requires an application-dependent AT. [36]

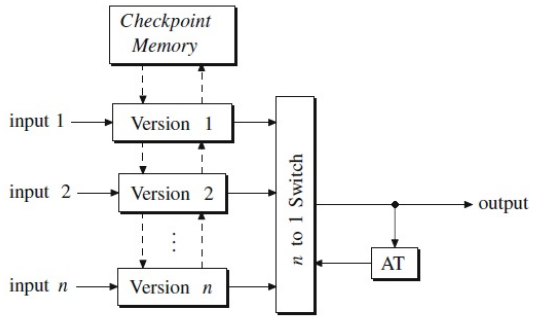


Figure 2.12: Recovery block technique. [4]

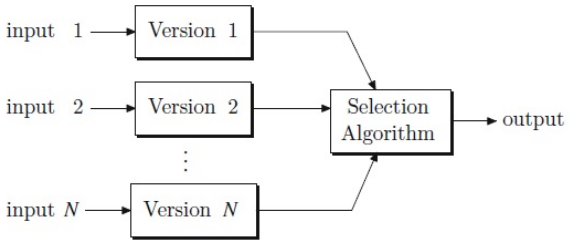


Figure 2.13: N-version programming. [4]

N self-checking programming combines recovery blocks and N version programming. The checking is performed either by using ATs, or by comparing pairs of modules. One example of application is the Airbus A-340 flight control computer. The structure of an N self-checking programming system using ATs is shown in Fig. 2.14. Each of them is capable of performing an AT. The voter only votes on accepted outputs. The execution of each version can be done either serially, or concurrently. In both cases, the output is taken from the highest-ranking version which passes its AT. The structure of an N self-checking programming system using comparison is shown in Fig. 2.15. [4] An advantage over the N self-checking programming using ATs is that an application-independent decision algorithm is used for fault detection. [36]

2.3.5 System/Process Redundancy

System/Process redundancy is achieved by using the redundant hardware resources and control them by software. This task is normally within the responsibility of a control system such a Real-Time System (RTS). One example is the hardware redundancy that is inherently available in processors. However,

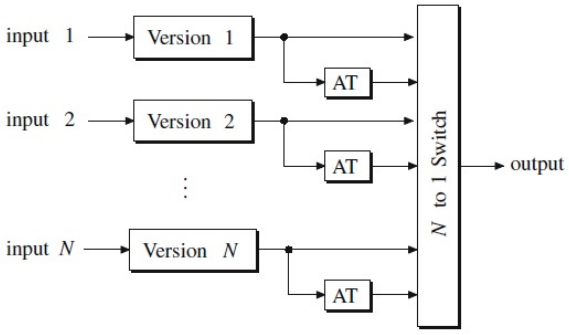


Figure 2.14: *N* self-checking programming using acceptance tests. [4]

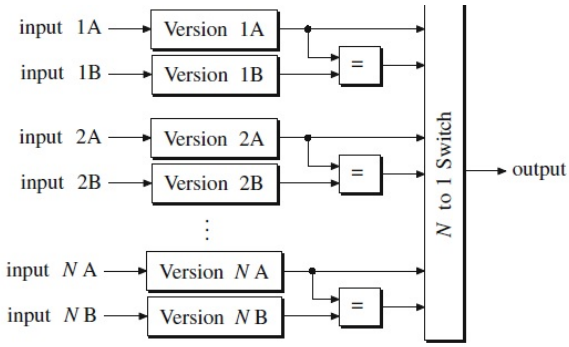


Figure 2.15: *N* self-checking programming using comparison. [4]

the redundancy in processors is typically used for improving the performance by performing computations in parallel. This inherently available redundancy can be also be used for making non-fault tolerant processors fault tolerant. [35] Combined with software and other fault tolerance techniques a system should be capable of recovering processors from transient and permanent faults. The *process* and *thread management* is responsible for process creation, process termination, scheduling, dispatching, context switching and other related activities. If a system does not implement process management, a new task/activity may not be able to use the processor or other system resources as a result of malicious or careless execution of other tasks and may result in miss of deadline and consequent failure. [37]

A fault-tolerant RTS should provide a mechanism that whenever an error occurs, a notification is sent to an agent. This agent, called supervisor has to perform some types of error recovery operations. Fault tolerance techniques should include support for event logging mechanism for improved detection of errors, protection against improper system calls and prevention of spread of

faults to the system. [37]

The RTS implements redundancy through the *task scheduling* which is the process to decide when and in which processor the given task should be executed. Scheduling algorithms can be *preemptive* where any task can be interrupted during its execution and *non-preemptive* that does not allow a running task to be interrupted. N-version programming is normally applied on the scheduling algorithms. [37] Additionally, for keeping track of proper execution, *watchdog timers* and *timeouts* are employed. A timer is maintained as a process separate from the one it checks. If the timer is not reset before it expires, the corresponding process has failed in some way. Timing checks are applicable to programs whose specification includes timing constraints. [7] [4]

Redundancy is also applied on the memory management either by software or hardware. Apart of the redundancy of data and state duplication on different memory units, most of the systems using a real time system already have features out-of-the-box such as memory management unit, ECC and dynamic storage allocation to ensure memory and data integrity. [37]

I/O management, is another important feature, where the possible use of fault tolerance techniques should be taken into account. The system should be able to manage the order of I/O accesses, without interrupting or prevent the tasks to meet their timing constraints. I/O Replication is a common fault tolerance technique that can be used, by replicating I/O devices. Backup devices can continue the unfinished process of the primary I/O device. [37] In order to preserve networking functionalities, upon detection of errors such as a link transmission failure, new routes of messages are determined and instantiated. In case of a node failure, software tasks are also migrated to other nodes. [15]

2.3.6 Static vs Dynamic Redundancy

In addition to the different redundancy techniques, it is also compelling to differentiate redundancy in another classification: static or dynamic. *Static* redundancy can be defined as continuous availability of a component, service or resource regardless whether the faults are present or not. It is related to the passive redundancy and one example of static redundancy is the fault masking technique. The hardware modules must be available at all time to ensure redundancy. The Fig. 2.16 a) shows a scheme for static redundancy. It uses three or more parallel modules which have the same input signal and are all active. Their outputs are connected to a voter who compares these signals and decides by majority which signal value is the correct one. [25]

On the other hand, *dynamic* redundancy is related to active and hybrid redundancy and is activated on demand by the system in presence of faults. This can either be by requesting an element (hardware component, system resource,

software, service, etc.) on *standby mode* to become active or to request an element already in use to be reconfigured. [16] Normally dynamic redundancy techniques reconfigure the system components in response to failures to prevent the faults from affecting the system operation. [7] Dynamic redundancy needs less modules on cost of more information processing. This requires a fault detection to observe if the operation modules become faulty. [25]

In the arrangement of Fig 2.16 b) the standby module is continuously operating therefore, it is classified as *hot standby*. Despite the small transfer time it has increased cost of operational wear out and power consumption of the standby module. The dynamic redundancy presented in Fig. 2.16 c) is classified as *cold standby* where the standby system is out of function and does not suffer the same wearing as the previous example. This arrangement needs two more switches at the input and more transfer time due to a start-up procedure. For both schemes the performance of the fault detection is essential. Lastly, the Fig. 2.16 d) the hybrid redundancy [6]

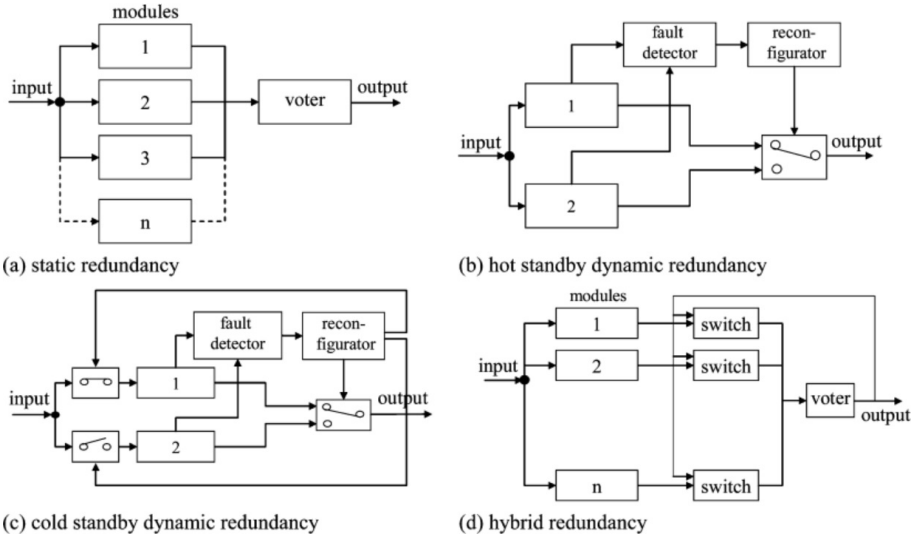


Figure 2.16: a) Static redundancy: NMR (all modules are active); b) Dynamic redundancy: hot standby; c) Dynamic redundancy: cold standby d) Hybrid redundancy . [6]

Dynamic redundancy can also be implemented on software by using standby software with diverse programs such as the recovering blocks technique as exposed before. This means that in addition to the main software module other diverse software modules can coexist. [25]

2.4 Reconfiguration

According to the Oxford dictionary, reconfiguration is *the arrangement of parts or elements in a different form, figure, or combination*. [38] Reconfiguration in the context of modern and complex embedded systems is a method that allows the system to be reconfigured in the event of a subsystem or component failure or in response to changes by installing, updating, and integrating various software entities. [39]

Whereas the redundancy is the actual provisioning of the extra components, software or time, the reconfiguration is the action of controlling the system using the available resources to change its configuration and integration with other components, sub-systems, services and protocols. It is the combination of the redundancy and reconfiguration types that makes a complex dynamical reconfigurable system a possibility to run fail-operational system meeting the increasing constraints in terms of space, energy and computing power efficiently.

On the fail-operational context, reconfiguration makes use of fault tolerance techniques of different redundancy types and so it can be applied to different levels. [40] The example in Fig. 2.17 illustrates a method known as reconfigurable duplication employed in the Voyager spacecraft where a fault detection technique is integrated into the design. only one of the duplicated modules is connected to the system outputs. When a fault is detected in the module by a mismatch, the faulty unit can be found and disconnected from the system. [7]

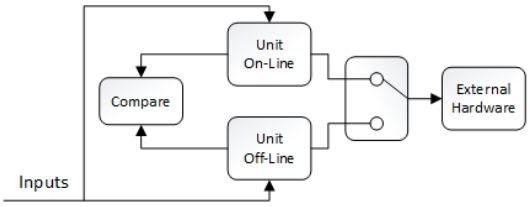


Figure 2.17: Reconfigurable Duplication. (Adapted) [7]

2.4.1 Hardware Level Reconfiguration

The design of reconfigurable hardware varies wildly from system to system, however when the subject is reconfigurable hardware technology Field Programmable Gate Array (FPGA) is the most prevalent on the literature. The use of FPGAs for computation and communication is firmly established. [41] FPGA implementations of applications are now prevalent in signal processing, cryptography, arithmetic, scientific computing, and networking. [42]

Conceptually all FPGA devices can be considered as being composed of two distinct layers: the configuration memory layer and the hardware logic layer.

FPGAs achieve their unique re-programmability and flexibility due to this composition. The hardware logic layer contains the computational hardware resources, including Lookup Tables (LUTs), Flip-Flops (FFs), Digital Signal Processing (DSP) blocks, memory blocks, transceivers and others. [43]

The configuration memory layer stores the FPGA configuration information through a binary file called a configuration file or bitstream. This binary file contains all the information that determines the implemented circuit, such as the values stored in the LUTs, initial set and reset status of flip-flops, initialisation values for memories, voltage standards of the I/O pins, and routing information for the programmable interconnect to enable the resources to form the described circuit. The function implemented by the hardware logic layer is thus wholly determined by the values stored in the configuration memory. [43]

To change the circuit implemented in the FPGA, the contents of the configuration memory are modified by loading a new bitstream. This operation is called FPGA configuration/reconfiguration as shown in Fig. 2.18. Partial Reconfiguration (PR) is also possible and refers to the modification of one or more portions of the FPGA logic while the remaining portions are not altered. [43]

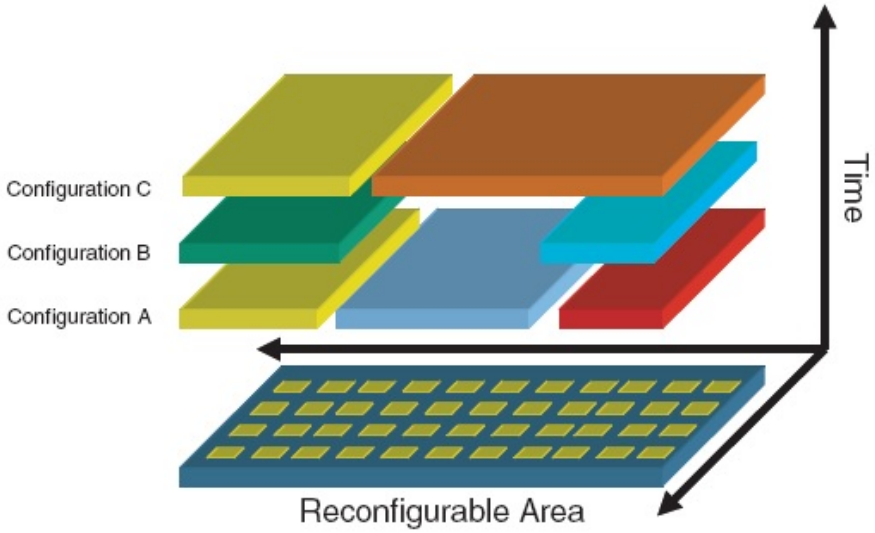


Figure 2.18: FPGA Partial Reconfiguration. [8]

Reconfiguration is also useful in scenarios where an interface is required to persist while functionality changes. Consider an FPGA system interfaced with a host computer via Peripheral Component Interconnect (PCI). A full reconfiguration of the FPGA breaks the communication link, which may even require a host reboot to re-establish. PR allows the link to be maintained by keeping the interface circuitry active while the Programmable Logic (PL) undergoes re-

configuration. PR also has the benefit of reduced external memory footprint for configuration files since partial configuration files are smaller than full configuration files. This can be especially beneficial for embedded systems with constraints on size, cost and power consumption. [43]

In Fig. 2.19 a) it is represented a modern microprocessor based system that allows to run two or more tasks at the same time through parallelism. With runtime reconfigurable hardware, this feature is extended to a new dimension. By exploiting reconfigurability and the configurable chip area as a physical medium for processing tasks at runtime, more processing power, performance and capabilities can be achieved. Fig. 2.19 b) shows the distribution of the tasks to different positions on the chip area where two axes X and Y represents the dimension of the chip area. In the example task 2 requires a larger amount of chip area than task 1. This re-utilization can be used for systems where the processing needs occur on-demand and increases flexibility. One benefit of this approach lies in the reduction of chip size, because only currently required functions are utilizing the reconfigurable area, while idle tasks are loadable on-demand from an external system. [9]

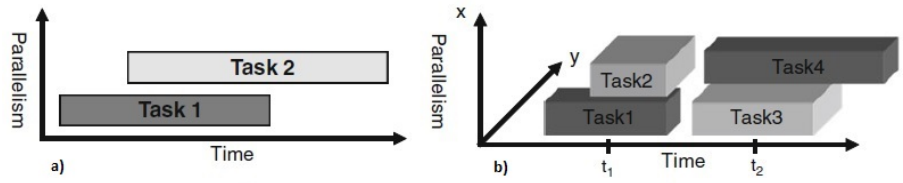


Figure 2.19: a) Parallel task processing in time b) Parallel task processing in time and space. [9]

2.4.2 Software Level Reconfiguration

Generally the software reconfiguration derives of the redundancy applied during the development phase. It is reflected on the system control program and on how it makes use of the available hardware to implement the fault tolerance techniques through reconfiguration. This includes deciding when reconfiguration should happen, which components/modules should be reconfigured, how reconfiguration is achieved, and so on. [43] The N-version programming technique described in redundancy section is itself a software reconfiguration. The software reconfiguration can be analysed in a greater extent when it integrates a more broader system or sub-system such as service-oriented-architecture. These systems normally comprised features to manage the complexity such as virtualization, complex operating systems and network tools.

Run-time reconfiguration can be based upon the concept of *virtual hardware*, which is similar to virtual memory. Here, the physical hardware is much smaller than the sum of the resources required by each of the configurations. Therefore, instead of reducing the number of configurations that are mapped, we instead swap them in and out of the actual hardware as they are needed. Because run-time reconfiguration allows more sections of an application to be mapped into hardware than can be fit in a non-run-time reconfigurable system. [40]

A packet processing system such as Field Programmable Port Extender is able to dynamically reprogram hardware modules and route individual traffic flows in network applications. The reconfigurable virtual network combines software virtual routers with several partially-reconfigurable hardware virtual routers, that are configured using dynamic reconfiguration. Functions such as header verification, checksum verification, Internet Protocol (IP) lookup, Address Resolution Protocol (ARP) lookup, and time to live updates, are implemented in PR regions and loaded as needed. The forwarding table for the virtual router can also be updated via the PCI bus. [43]

There has been work on extend Operating Systems (OS) such as Linux to support run-time PR. A number of new system calls, such as module-request, and module-release, are implemented to enable the OS to manage hardware modules similar to software processes. Also different caching and allocation policies influences the decision of how a PR region should be handled once the allocated module finishes execution and to map new module requests to available regions. For low-level reconfiguration operations, dedicated device drivers are integrated with these system calls and user libraries. [43]

2.4.3 Static vs Dynamic Reconfiguration

The configuration of the hardware, software, system, network and services can be done at the system initialization and maintained with no alterations. This suggest a *static* configuration and in order to perform reconfiguration, the system or sub-system has to be re-initialized in order to changes to take effect. For instance, on hardware static reconfiguration using a non-volatile technology based FPGA, while the entire configuration memory is reloaded the FPGA has to remain inactive/inaccessible (reset state) during this period. [43] This means that the FPGA application has to be stopped in order to include the new functionality on-chip. [8]

On the other hand, *dynamic* reconfiguration means that the initial configuration of the system can be altered or reconfigured during run time. Taking the same example of the FPGA but this time using a subset of Static Random Access Memories (SRAM) based FPGA, the reconfiguration can be dynamically executed with the configuration bitstream being loaded to the FPGA memory

during run-time. Run-time in this context means that FPGA does not have to be stopped (reset state) during the reconfiguration phase. [8]

Although the terms static and dynamic reconfiguration seems to be differentiative, on a complex system, they are interchangeable. Taking again the static FPGA example, while the static reconfiguration occurs, the system where the FPGA is integrated might continue running with no necessity to reset. The FPGA reconfiguration is static but the operation is dynamic. [43] The same can be applied to a Network-on-Chip (NoC), where a route switching context due to service unavailability can be static in the sense of the configuration required on the network control system to update its configuration, while the overall application/functionality continues running. [14] This particularity is intrinsically connected with the concept of time redundancy, in this respect, meaning that MTTR is an important aspect when evaluating a dynamically reconfigurable system on the context of safety-critical systems. [44]

Another type of reconfiguration may occur at processor level using the system and process redundancy as explained in 2.3.5. Scheduling policies are used to assign operations from a single instruction stream to a particular computation domain. In this sense, *dynamic scheduling* and *static scheduling* are available. [35]

When dynamic scheduling is used, then the processor dynamically assigns each operation to a free computation domain during the execution of the application. The software has no control about the computation domain that is used for executing a particular operation. On the occurrence of a defective computation domain then the administration of the redundant domains must be done in hardware by the processor itself. This fault avoidance mechanism includes static hardware reconfiguration. [35]

For instance, at the task (module) level on the software side, communicating processes/ threads are bound to one or multiple processors on which they must be prioritized and scheduled depending on the scheduling algorithm. [15] Task scheduling for these systems can also consider the assignment of tasks to either hardware or software based on available resources and latency requirements. The operating system support for dynamically placing portions of an application in FPGA logic appeared later. The FPGA is viewed as a co-processor in a microprocessor-based system and its configuration was scheduled considering multiple software threads of execution. [45]

Chapter 3

State of the Art

The empirical Moore law does not only describe the increasing density of transistors permitted by technological advances. It also imposes new requirements and challenges. Systems complexity increases at the same speed. New architectures are and must be continuously conceived. [46] One case of this increased complexity is the appearance of the concept of Cyber-Physical Systems (CPS) as a modern embedded system on the automotive industry.

A look at the history of car electronics can be helpful to understand this increasing complexity. As shown in Fig. 3.1 the number of ECUs increases dramatically as new functions are integrated into vehicle architecture. The high number of ECUs also demonstrate a trend for the architecture to become much more complex than it is required (accidental complexity) to be achieved by the functionalities (essential complexity) mainly due to redundancy measures. Therefore, only a substantial revision of the architecture enforced by a disruptive technology leap can bring the overhead complexity back down to its essential level. [10]

In this chapter it will be exposed the state of the art regarding the recent CPS application development and automotive systems. Secondly it will be briefly addressed the current standards and regulations regarding safety critical applications. Finally it will be presented the state of the art regarding system evaluation of safety critical applications.

3.1 From Embedded Systems to Cyber Physical System

The concept of CPS appeared due to the increasing amount of applications that makes use of networked embedded devices on the most varied industries. On the contrary of its pre-cursor embedded systems where the emphasis tends

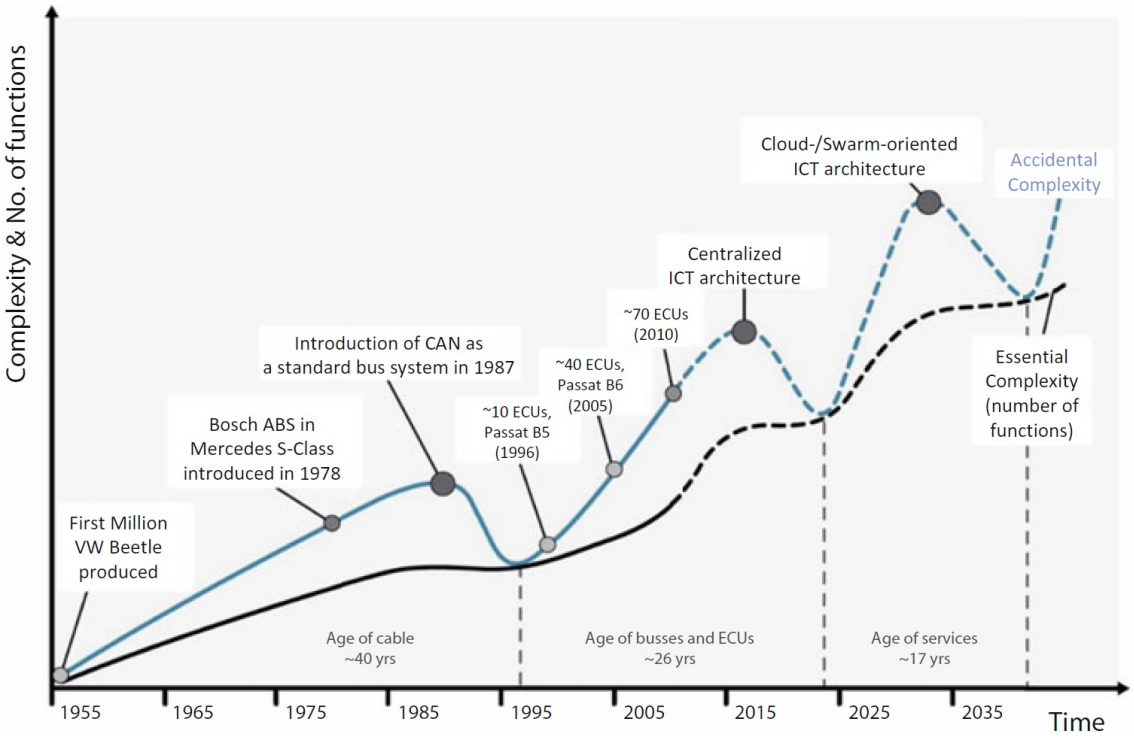


Figure 3.1: Past and expected future growth of essential and accidental complexity due to increasing functionalities. [10]

to be more on the computational elements, the CPS put more emphasis on an intense link between the computational and physical elements. [15]

A CPS is a complex system integrating sensing, computing, communication, networking, and control based on the environmental perception. It makes use of embedded sensing devices to acquire data, transfer data via the connected network systems, store and process data in cyberspace, and make decisions according to the results of information processed. This enables an new ability of the CPS to take action in the core functions of a system based on the external information provided by the communication layer. Therefore, CPS represents a bridge for a intricate interaction between cyberspace and physical space. [11] One example of such interaction on the automotive context is the ability of vehicles to move under computer control in platoon or reconfigure the system dynamics according to environmental changes. [47]

For the example of platooning depicted in Fig. 3.2, the perceptual layer represents the embedded sensing and in the service layer, the system has access to traffic information, and other rich information. The communication layer subsystem performs the functions of data acquisition, transmission, and com-

munication in CPS. The computing subsystem completes the storage, analysis, and processing of various data. The control subsystem determines the control strategy for the physical world and coordination of the various actuators of the physical world object operation. [11]

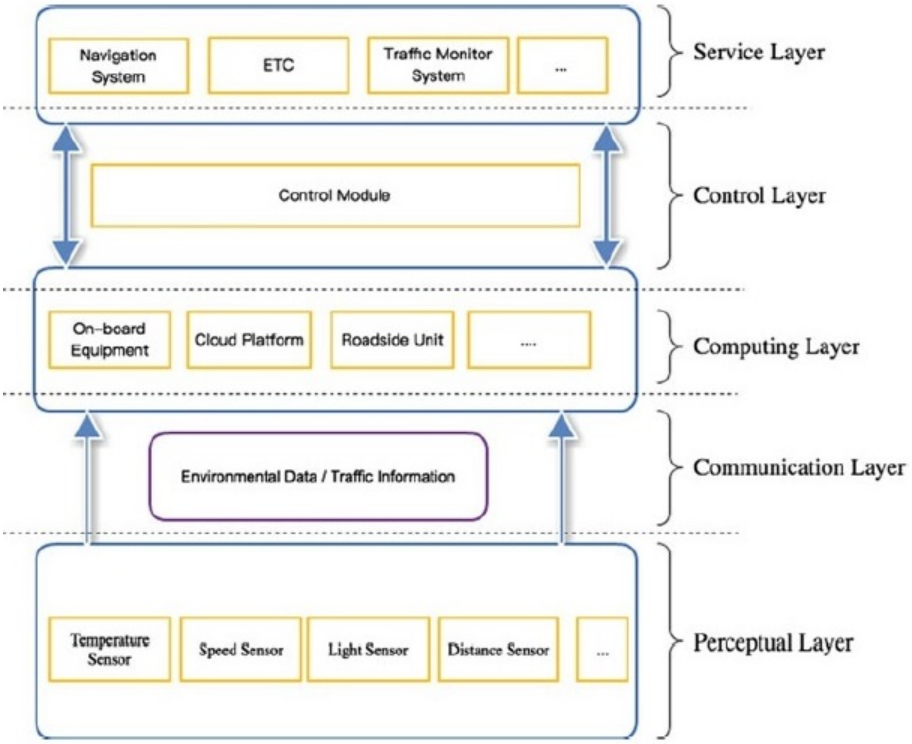


Figure 3.2: Function layers of transportation cyber-physical systems.[11]

At the same time that increased outsourcing resources opens the potential for new applications it also increases the possibility of losing system control and reduces system stability. Hence, all CPS elements must be designed to operate under the assumption that other parts of the system can fail. This implies redundant and fail-operational capabilities. The disconnection of some resources may require mechanisms to mask their unavailability to the rest of the system. Similarly to timing constraints that the current automotive embedded systems needs to comply, the CPS must dynamically respond to or reconfigure to changes within it or its environment and still meet these timing requirements. [22] CPSs have obvious inherent risks for both safety and security, therefore the development of appropriate design and evaluation methodologies is vital. [47]

3.1.1 Modern Automotive Systems

In the automotive industry, new technologies have been developed to increase the autonomy of the vehicles resulting in Advanced Driver-Assistance Systems (ADAS). Applications such as the adaptive cruise control, advanced emergency braking system, and lane keeping assist are already available in modern cars controlling part of the vehicles. [1] The domain of automated driving requires a much higher number of sensors based on different physical technologies to ensure a proper sensing of the environment outside and inside the vehicles. An unpredictable amount of new algorithms for sensor fusion and data analysis will influence the software driven vehicle behaviour. These algorithms are partly based on Artificial Intelligence (AI) technologies like Deep Neural Networks (DNN), which require high amount of computing resources and can only be achieved by the usage of high performance computation nodes within vehicles. [13]

A typical modern vehicle uses up to 120 decentralized, highly specialized ECUs, often with similar functions resulting in redundancies. The complexity of the distributed architecture is an obstacle and increasing centralization of E/E architectures is a key point to overcome this technical challenges. [48] A middleware based centralized E/E architecture can reduce drastically the overhead complexity by allowing new functions to be integrated not in individual ECUs but instead, relying even more on software. The virtualization of the required total system of hardware and software into a service oriented architecture allows not only to implement features, such as fault tolerance mechanisms and communication but also to be possible to distribute functions as required. The vehicle would thus become quite naturally part of a larger system as a CPS application and namely as a Vehicle Computer (VC). [10]

Furthermore, these centralized E/E architecture by making use of high performance domains such multiprocessor and hardware accelerators with operating systems coming from the Information Technology (IT) domain introduce new approach paradigms either in software and hardware into the automotive industry. This approach allows a flexible and dynamic function configuration in the vehicle by providing mechanisms already in middleware for easy update and upgrade ability. Vehicle connectivity is also implemented in the same way by service-oriented communication. [13]

3.1.1.1 AUTOSAR

From the point of view of the developer on the automotive domain, one obstacle is the lack of standards and methods to describe and integrate subsystems developed by different stakeholders. For example, integration of ECUs usually starts on a test board that connects different subsystems. Heavy testing scenar-

ios are then applied to perform functional analysis and detect potential errors often without the necessary knowledge of the software implemented in each subsystem. AUTomotive Open System ARchitecture (AUTOSAR) is an open and standardized automotive software architecture, jointly developed by automotive manufacturers, suppliers, and tool developers. One of its major goals is to facilitate the exchange and update of software and hardware over the service life of the vehicle. [15]

AUTOSAR provides implementation and standardization of basic system functions including, bus technologies, operating systems, communication layer, hardware abstraction layer, memory services, mode management, middleware, interfaces, and standard library functions as well as integration of functional modules from multiple suppliers. AUTOSAR defines several safety and security features such as memory partitioning, protection against unauthorized use software modules, protection of safety related data against corruption, end-to-end communication protection mechanisms, program monitoring to check the correct execution of software, provision of synchronized time bases, and fault detection mechanism. [49]

Many extensions of AUTOSAR have been presented that allows to specify reconfiguration aspects at the architectural level and to automatically derive the needed reconfiguration functionality based on the architectural information. These reconfiguration capabilities make automotive systems more flexible and robust as well as able to deal with failures of sensors or attacks. [49] One example is the AUTOSAR Adaptive architecture which supports new possibilities such as firmware updates Over-the-Air. [48]

3.1.1.2 Hypervision

Hypervisor-based virtualization is a technology to concurrently run various embedded real-time applications on a single multicore hardware. In Fig. 3.3 a) depicts a traditional architecture where the tasks are allocated directly on the ECUs. In Fig. 3.3 b), an OS hypervisor runs between hardware and operating systems and virtualizes hardware. As a result, tasks and operating systems can be executed in a hardware-independent way. It provides spatial as well as temporal separation of different applications allocated to one hardware platform. A type-1 OS hypervisor runs baremetal applications directly on hardware while a type-2 OS hypervisor runs on a host operating system and supports other guest operating systems. OS hypervision provide Virtual Machines (VM) that represent duplicates of the real hardware. These VMs allow to run various systems spatial and temporal separated on a single hardware platform providing high flexibility and isolation. [50] [12]

Flexible multicore and multiprocessor platforms use hypervision to achieve

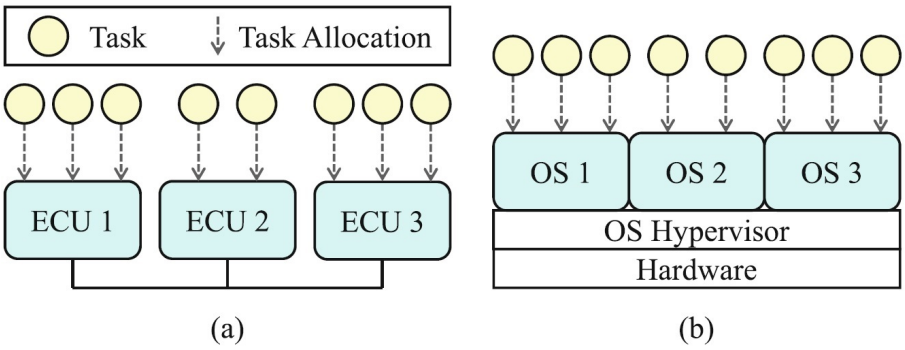


Figure 3.3: **a)** A traditional architecture, and **b)** an architecture supported by an OS hypervisor. [12]

isolation of several different safety-critical level applications. Hypervisor technologies allow the usage of different OSs on the same VC, providing a flexible framework, which allows a flexible configuration and isolation of applications on VC. One example is the Real-Time-Application Vehicle Run Time Environment (RTA-VRTE). It is an AUTOSAR Adaptive conform middleware that provides relevant platform features related to functional safety and security requirements. It provides easy approach for configuration of multi-partition solutions on vehicle computer. This evolution is demonstrated in Fig. 3.4, where on the left side is represented a classic approach with the different severity level applications not divided in comparison with the combined approach of classic and adaptive platform including the concept of Hardware Abstraction (HWA) on the right. [13]

3.1.1.3 Reconfiguration: A new approach

Currently automotive E/E architectures can be considered static in the sense that the placement of software components providing services is fixed to often purpose-built ECUs. Modern automotive systems will more and more rely on cloud-provisioned services to handle the increasing computational demands of intelligent vehicle functions. [22]

Especially challenging in this context is the relation between autonomous driving and vehicle functionalities with the inconsistent state of their environment. Depending on the operational situation, the vehicle architecture can be dynamically reconfigured as required, causing a rapid alteration of driving scenarios which need to be handled in different ways. In this way, the ability to dynamically adapt the deployment of services, either they run within the vehicle or outside in a cloud based provider is a tremendous advantage. [1] In [14] it is presented three different approaches, which can be seen complementary

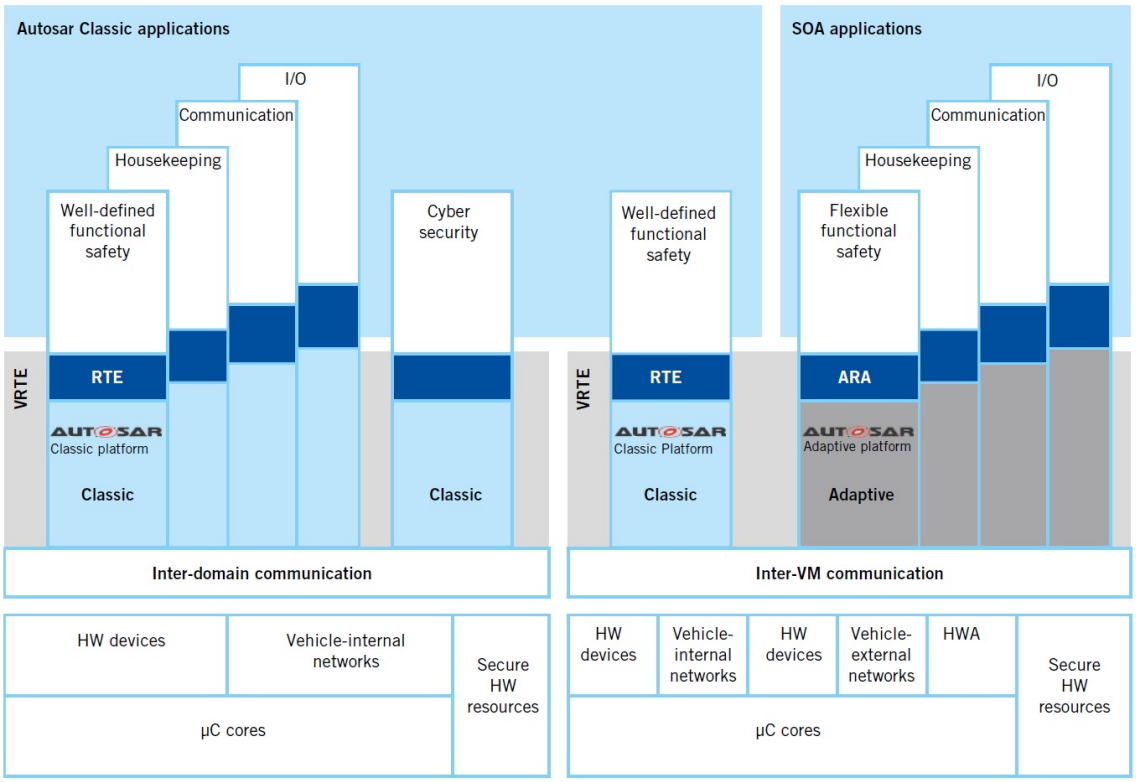


Figure 3.4: Basic structure of software for vehicle computers with AUTOSAR Classic and AUTOSAR Adaptive components. [13]

rather than as mutually exclusive:

- *Platform based* approach is where service instances can be deployed to redundant ECUs or to a cloud infrastructure. Their priorities are carefully evaluated and defined at design time. Therefore, only a very limited number of operational situations can be considered.

- Similarly, to the platform-based, the *controller-based* system is manually designed at development time and executed at runtime where following a rule-based approach the system executes a strategy according to the operational scenario such as reconfiguration in case of hardware fault.

- *AI-based* approach provides a mean to achieve the desired flexibility by applying neural network-based solutions and machine learning methods to situations that are not considered on the possibilities above. This allows to predict situations by using a richer knowledge repository formed by other vehicles. [1]

The correspondent reconfiguring architecture layers of the aforementioned concept is depicted in Fig. 3.5. If the platform-based reflex layer with real-time capabilities has no pre-configured reconfiguration rule, then the task is forwarded to the controller-based strategy layer. Again, if the operational situation cannot be handled by the controller, the AI-based strategy layer is consulted. This provides flexibility in handling new situations, however, the lack of predictability and computational requirements of advanced AI and DNN might not be able to fulfil the timing and safety constraints. [14]

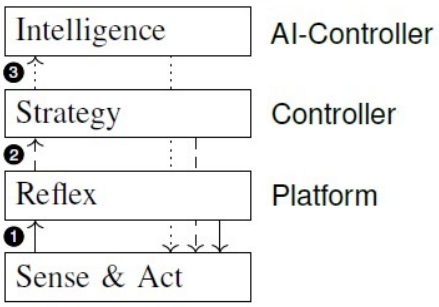


Figure 3.5: CPS decision paths: 1- platform-based (reflex), 2 - controller-based (strategy), 3 - AI-based (intelligence). [14]

3.1.1.4 Challenges and Requirements

With the emergence of the new software systems that can be dynamically adjusted at runtime, the hardware and software architecture design is at risk considering the demand for computational power and handling the increasing data volume and streams resulting from sensor fusion. [1] Being so it is possible to enumerate specific characteristics to encompass the increasing demanding functionalities:

- High-performance computing applications, such as image processing are still limited due to insufficient *processing power*. High-volume and dynamic input data has a significant impact on the design of the computation platform. [12] In former times, the approach was to increase the clock rate of a processor. Nowadays, the approach has shifted toward increasing the number of processors while keeping the clock rate stable or even reducing it. [51] Other approach is to include in the design architecture powerful specialized processing units like GPUs, hardware accelerators such as FPGAs and Tensor Processing Units (TPU). [1]
- Most algorithms used in high-performance computing have a high inherent *parallelism*, which can be exploited by such a multiprocessor system.

The disadvantage is that the hardware architecture of the multiprocessor system is fixed at design and at runtime. [51] However, many embedded applications require sequential interaction between different components. Increasing system performance is not reached by parallelization of dedicated software but rather by running various applications on one multicore platform concurrently. Virtualization provides means to separate various applications. Multicore architectures and virtualization are recognized as symbiotic technologies. [50]

- Powerful and reliable *network interconnection*. For ADAS and autonomous functions, an advanced lidar can have input data rate that is up to 100 Mbps, which far exceeds the capacity of currently prevalent in-vehicle bus protocol such as the Controller Area Network (CAN). [12] The NoC architecture represent a clear trend for interconnection used by safety critical applications. [44] In order to provide fault-tolerance for critical connections and meet tight timing constraints, the NoC typically encompass several mechanisms. [52]

- *Highly configurable* at the interconnections level as well as at the individual computational elements, thus allowing them to be customized to specific functions and overall application requirements. [53]

- The principle of a *high availability* system is to share resources, so an occasional loss of a architecture component can be tolerated. [7]

- *Redundancy* needs to be provided not just to increase reliability and ensure safety, but also to increase the availability in order to achieve desired functionalities. [1]

- Systems using the characteristics above are widely used on *Long life systems* such as unmanned spacecraft that cannot be manually maintained for five or more years. STAR and Voyager are examples of long life spacecrafts. [7]

3.1.1.5 MPSoC

Looking at the aforementioned characteristics, the multi-core architectures are an appealing design choice for the next generation safety-critical embedded systems. Multi-core devices have been on the market for decades now and currently it is common to have tens of cores, several accelerators, alongside with a set of peripherals, integrated on a same Multi-Processor System on Chip (MPSoC) device. [52] [46] MPSoC device devices normally make use of NoC interconnection. The NoC integrates routers and links to exchange the IP cores data encapsulated as packets. One example of such a complex MPSoC target

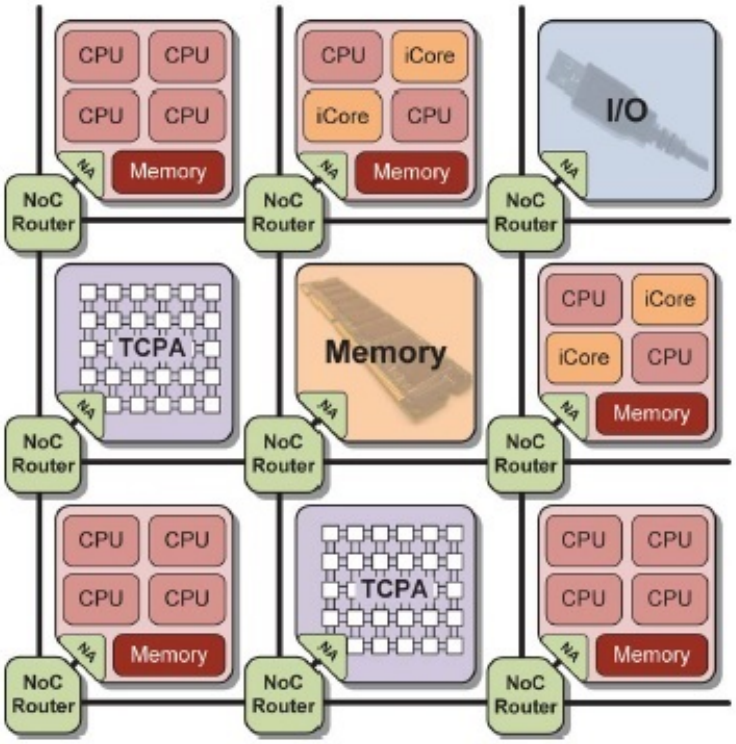


Figure 3.6: Example of MPSoC including several Central Processing Units (CPU), tightly couple processor arrays, memory I/O interconnected by a NoC. [15]

architecture is shown in Fig. 3.6. The communication of tasks may thus involve several hops and require routing. [15]

The cutting edge MPSoCs are able to provide a virtually unbounded scalability, flexibility and computational power which has turned MPSoCs into a perfect solution for many critical applications and development off cutting edge technology. [54] The combination of MPSoC and reconfigurable architectures such as Xilinx FPGAs can be exploited for runtime reconfiguration systems (hardware and software) design. [9]

Contemporary FPGA devices allow MPSoCs to achieve better performance than conventional processor systems as they can directly implement in hardware specific high-level operations. Reconfiguration is also seen as a key technique to mitigate, or even eliminate, issues with reliability and intermittent hardware faults in large systems. [53] FPGAs are increasingly demanded by spacecraft electronic designers because of their high flexibility. In particular, SRAM-based FPGAs are very valuable for remote missions because of the possibility of being reprogrammed by as many times as necessary in short periods.

[55] SRAM-based FPGAs have been the preferred choice, for instance, in space missions, like MARS Lander and Rover vehicles, where they were exposed to extremely harsh conditions. [41]

One example of such MPSoC is the Zynq UltraScale+ MPSoC from Xilinx. This device combines a block of hard-wired components, such as a dual-core Cortex-R5 from Arm, and an FPGA on a single chip. UltraScale+ devices provide 64-bit processor scalability while combining real-time control hard engines for graphics, video, waveform, advanced analytics, and packet processing capabilities in the programmable logic for task acceleration. The reconfigurable fabric of the Zynq uses the 7-series FPGA architecture which can also be partially reconfigured. [17]

Despite their technical advantages of MPSoCs, they are not yet fully adopted in safety-critical applications not only due to the potential interference on shared resources such as interconnect and memory, but also the complexity and cost to ensure the absence interference between different criticality applications. [52] [44]

3.2 Standards and Regulations

The design of a safety-critical system has to follow a set of standards to be eligible for certification. Commonly, the certification standards classify applications into different criticality levels, according to the severity of the catastrophic consequences that applications can provoke in case of failure and also other aspects. [52]

International Electrotechnical Commission (IEC) 61508 is an international standard consisting of methods on how to apply, design, deploy and maintain safety-related systems. It is titled Functional Safety of Electrical/Electronic Programmable Electronic Safety-related Systems and has different derivations according to specific domains. This standard defines Safety Integrity Level (SIL). The SIL is the level of certification attributed to a given module of the system. The most known standards are International Organization for Standardization (ISO) 26262 for automotive, Software Considerations in Airborne Systems and Equipment Certification (DO-178C) for avionics and European Standard (EN) 50128 for railway. [52]

To face cyber-security challenges, related standards such as ISO/IEC 15408 and ISO/IEC 27034 need to be integrated with recent standards for Wireless Access in Vehicular Environments namely the Institute of Electrical and Electronics Engineers (IEEE) 1609 family of standards. Such standards, aims at defining an architecture and a complementary, standardized set of services and interfaces that collectively enable secure vehicle-to-vehicle and vehicle-to-infrastructure

wireless communications. Together, these standards are designed to provide the foundation for a broad range of applications in the transportation environment, including vehicle safety, enhanced navigation and traffic management. [49]

ISO 26262 is an international standard deriving from the IEC 61508 for Automotive E/E Systems. It addresses functional safety features where the risk of hazardous operational situations is qualitatively assessed and safety measures are defined to avoid, detect and control systematic in order to mitigate their effects. The ISO 26262 standard [49]:

- Provides automotive safety guidelines and support for the design of hardware and software and the necessary activities during these life cycle phases. It covers functional safety aspects of the entire development process (management, development, production, operation, service, de-commissioning);
- Provides an automotive-specific risk-based approach for determining risk classes;
- Uses Automotive Safety Integrity Level (ASIL) as an adaptation from IEC 61508 SIL for specifying the item's necessary safety requirements for achieving an acceptable residual risk;
- Provides requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety is being achieved.

The avionics industry adopted two standards deriving from the core concepts defined by IEC 61508, the Radio Technical Commission for Aeronautics (RTCA) DO-178C and DO-254, the first targeting the software development and the second the hardware development process. Design Assurance Levels (DAL), defined for the avionics, can be roughly equivalent to the SILs. The spatial and temporal partitioning to achieve a proper system module isolation is defined by Avionics Application Software Standard Interface (ARINC) 653. [52]

3.3 Evaluation Parameters and Methods

Reflecting the tremendous increase of the the complexity which resulting from the combination of diverse hardware functionalities, is the complex process of architecture requirements specification, development, validation and solution certification. [53] Standards like IEC 61508, DO-178C or ISO 26262 increasingly emphasize ensuring software safety. They require to identify functional and non-functional hazards and to demonstrate that the system does not violate the relevant safety goals. Classical software validation methods like code review and testing with debugging cannot really guarantee the absence of errors. Formal verification methods such as static and dynamic analysis provide

an alternative to ensure the absence of violation of timing constraints or run-time errors. [56]

It is required some sort of flexibility when assessing a complex system such as a dynamic reconfigurable system. For example, in case of a FPGA implementation, the evaluation parameters could be the number of logic gates, flip flops, and block RAM combined with performance objectives such as throughput and clock rate of the synthesized system. In terms of timing analysis, an important parameter to evaluate is the Worst Case Execution Time (WCET) estimation of a task when implemented to a certain computing resource. [15] Additionally, stochastic models such as Markov processes can be used for the proper analysis of dependability as presented on the section 2.2.1.

An overview map of the potential parameters and evaluation methods is shown in Fig. 3.7 and reflects the research made. Aligned with the standards and architectures previously exposed it can be highlighted three main aspects of evaluation that are deeply interconnected and influence each others. First the *dependability* encompassing the safety, reliability and predictability. Secondly the *timing* including different types of latency and WCET and lastly the *performance* ranging from the used hardware to overhead analysis.

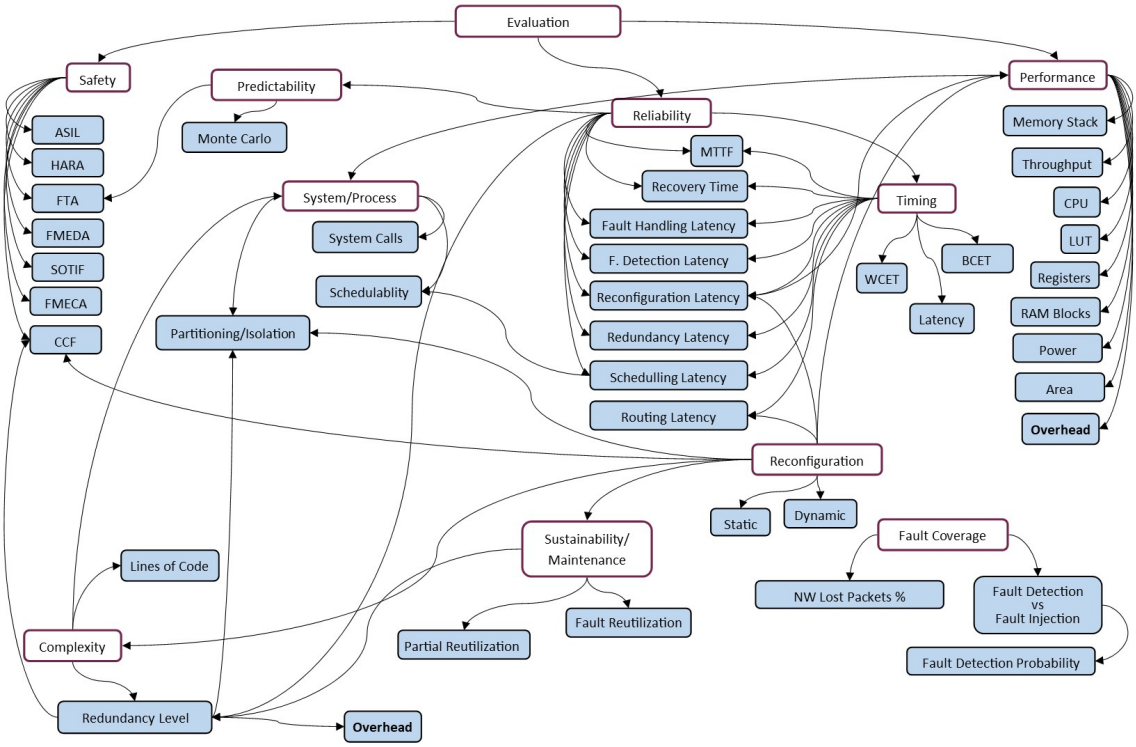


Figure 3.7: Evaluation Map.

3.3.1 Dependability Evaluation Methods

As discussed on section 2.2 dependability analysis inductive methods are applied to determine what system states (usually failed states) are possible and the deductive methods are applied to determine how a particular system state can occur. [29]

Inductive methods include the aforementioned FMECA, FMEDA and Hazard And Risk Analysis (HARA). These methods are used for SIL calculations, performed before and updated during and after the development phase. FMECA is a inductive method that analyses the failure modes that each component may produce, the impact of each failure mode on the system, and the criticality of the failure. [57]

The HARA process is difficult and complex and tries to identify and classify all hazardous events according to three main criteria: probability of occurrence, severity and controllability. This process complexity is due to the increased state space but also because the classification is often not based on historical data but on expert opinions. While a HARA is top-down and includes environmental and operator states, the FMEDA analyses the effects of a failing component on the correct functioning of the system in terms of the potential hazards. Failures are categorized according to their origin. [20]

The most common deductive methods are FTA and RBD and the fundamental difference between them is that an RBD is a success-oriented model, while a FTA is failure-oriented. For most safety critical applications, it is recommended to start by constructing a FTA instead of an RBD because thinking in terms of failures will often reveal more potential failure causes than thinking from the function point of view. [29]

Both FTA and RBD are evolutionary in nature, meaning that their modelling capabilities are enhanced as needed to support a wide range of scenarios. For example, introducing new gates, the FTA is enhanced to support sequence dependent failures. However, RBDs are not enhanced to support failure modelling features. [29]

3.3.1.1 Fault Tree Analysis

The deductive diagnosis via FTA corresponds to a top-down approach which illustrates the failure logic of a system, and shows all combinations and sequences of failures that can lead to a system failure. The FTA is used to establish a link between causes and effects of failures. The FTA is proved to be a powerful tool in system failure diagnosis. [58]

The main elements of a FTA include [29]:

- A *top event* which represents the undesired event, usually the system failure or accident.

- *Basic events* which represent basic causes for the undesired event, usually the failures of components that constitute the system, human errors, or environmental stresses. No further development of failure causes is required for basic events.

- *Undeveloped events* which represent fault events that are not examined further because information is unavailable or because its consequence is insignificant.

- *Gates* are outcomes of one or a combination of basic events or other gates. The gate events ('OR' and 'AND') are also referred to as intermediate events and are related to the faults.

FTA helps determine the combinations of hardware component failures and errors that result in the occurrence of undesired events. FTA calculates the probability of the top event based on the probabilistic-based information. It assumes that a probability distribution of each basic event is known and complete. Such information can be derived from either hardware component data, mathematical models or expert knowledge. [59] By underlying safety evaluation, it represents a relevant method for failure modelling used in a comprehensive range of industries from nuclear power, process industry to automotive and aerospace. [60]

3.3.1.2 Extended Dependability Evaluation

As exposed on section 2.2.1.4, the dynamic FTAs can be translated to stochastic processes such as Markov Chains model. The two main concepts in the Markov model are system states and state transitions. The state of a system represents a specific combination of system parameters that describe the system at any given time. For representing the system reliability, each state of the Markov model generally represents a distinct combination of faulty and fault free components. The state transitions represent the changes of a state that occur within a system. As time passes and failures occur, the system goes from one state to another until one of the final states (usually the system failure states) is reached. [19]

Common Cause Failures (CCF) are multiple dependent component failures within a system that are a direct result of a shared root cause such as power outage, sudden changes in environment or human errors. They are defined as *a subset of dependent events in which two or more component fault states exist at the same*

time, or in a short time interval, and are direct results of a shared cause. [29] CCF typically occur in systems designed with redundancy techniques, especially when using *n*-identical components. It is critical to consider CCF in the system reliability analysis since it can diminish dramatically the reliability. On the opposite, isolation techniques aim to ensure the minimization of CCF probability. [29]

The Monte Carlo Simulation (MCS) method, also known as random sampling method, is based on probability theory, mathematical statistics and the use of random numbers for statistical tests to solve the problems that can be described directly or indirectly with a random process. Its basic characteristic is that the statistic characteristic of the estimated parameter is obtained by using the method of repeated sampling according to the distribution of the various underlying events, and the estimated value of the higher parameter is determined. With the rapid development of computer technology, the use of FTA and Monte Carlo method for comprehensive analysis is possible by using software programming simulation to obtain numerical reliability analysis for complex systems. [57]

3.3.2 Worst Case Execution Time

One fundamental challenge in building a safety-critical system with hard real–time characteristics is guaranteeing that it will perform its required functionality within specified time constraints. Determining WCET is key to predictability, that is to ensure that temporal behaviour of the system is correct and meets timing requirements. The WCET of a computational task is the maximum length of time the task could take to execute on a specific hardware platform. [61]

The WCET of a certain task running on certain hardware can hardly be calculated exactly. Usually it is approximated by upper bounds, either calculated in a static fashion or estimated by execution on the target platform. The estimated WCETs of all tasks are then used as input for schedulability analysis. The correct estimation of WCET is difficult to obtain since components such as buses, global memory do not follow the same increased performance of the multicore-processors. [62]

It is often assumed that the WCET is a fixed value during the whole system life. However, for some long-lived systems, the WCET is not constant but may be gradually increasing with system lifetime. One reason is that many real-world applications are highly data-dependent, while the size of input data naturally grows up with time. Another cause of increased worst-case execution times is gradually degrading hardware. The influence of these effects could be minimal in a short period, but if examined in a large time-scale, the impact on task execution times can be observable. If one WCET has a trend that would

potentially cause a timing fault in the future, it should be addressed earlier to make the system achieve a graceful degradation. [32]

Literature distinguishes between several fundamentally different approaches for WCET estimation [62]:

- Static analysis;
- Dynamic estimation;
- Hybrid.

Static analysis purely rely on hardware models to simulate hardware behaviour caused by certain tasks so comprehensive knowledge of the hardware is required. Using the information returned from cache analysis a worst case path and in consequence a WCET can be derived. All basic blocks or subtasks within the worst case path are considered and counted. For every basic block the WCET can be estimated and summed to obtain an WCET upper bound. [62] From a methodological point of view, static analyses can be seen as equivalent to testing with full coverage has to meet the standards testing requirements. [56]

As opposed to this, *dynamic estimation* does not need hardware models since it uses real hardware to estimate upper bounds of execution times. Since tasks run directly on hardware no complex models are needed and hardware timings are guaranteed to be correct. Hence, a task is run several times, each time with different input data. Two different ways to derive upper bounds to a WCET are used in dynamic analysis, the measurement approach and probabilistic approach. [62]

The *measurement* approach is preferable to not too critical real time systems since it rely purely on measurements and depending on complexity of a task, multiple paths of execution are possible. Hence, an accurate WCET estimation by trying all possible input values is not feasible. However, it can be extended to another approach of *probabilistic* analysis of the execution behaviour. A timing profile is derived by running a task multiple times and predict a WCET through extreme value theories. [62]

The static branch prediction has limitations since it is performed at compile-time and does not adapt to the run-time behaviour. On the other hand, dynamic branch prediction records the behaviour of branches at run-time and uses that information to predict future behaviour. However, dynamic branch prediction is notoriously difficult to model for WCET analysis. Due to its performance benefits, dynamic branch prediction is commonly found in modern processors and achieve better precision than static predictors. [61]

Hybrid techniques use advantages of both approaches. Normally, static analysis is used to find worse execution paths while execution profiling is done on

real hardware or simulators. Furthermore, statically analysis is performed on the source code. Program paths are decomposed into sub-paths and submitted to dynamic analysis. Additionally full range test cases aids finding worst case run-times in small sub-paths. At the end, a static calculation combines the results obtained by dynamic analysis to calculate the WCET upper bound. [62]

3.3.3 CoDesign and Design Space Exploration

Designing hardware and software separately from each other may lead to under designed system implementations not meeting all non functional properties such as timing, cost, power consumption, reliability and safety. Design Space Exploration (DSE) is the task to explore the wide set of feasible implementations efficiently and select the optimal solution. The DSE approach is summarized in Fig. 3.8. By performing DSE, it is possible to find a final implementation based on Model of Computation (MoC) and a Model of Architecture (MoA). The next step of the DSE is to explore the design space of implementation candidates. At this point DSE refers to systematic analysis and discard unwanted design points based on parameters of interest. Each synthesis candidate is evaluated according to typically multiple objectives that are implemented by evaluation functions. [15]

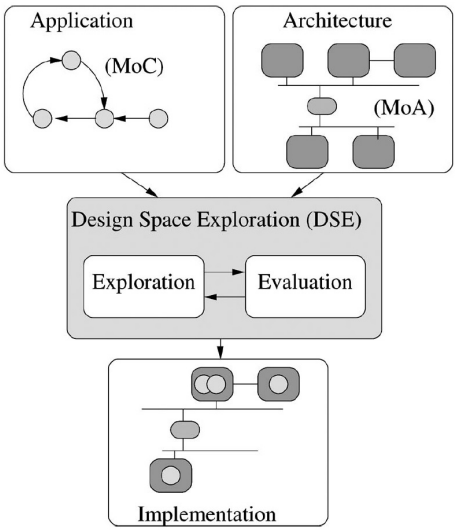


Figure 3.8: Design Space Exploration workflow. [15]

Given the complex specification of electronic systems and the wide range of design choices (from the selection of components, number of components, operating modes, redundancy schemes, reliability constraints, connections and algorithms) the exploration process is complex. The same functionality can be implemented on a variety of ways, either in hardware and software. A trade-off

analysis between each of the implementation option based on a certain parameter of interest forms the basis of DSE. The parameters of interest vary according to each systems, but the commonly used parameters are power, performance, and cost. More specifically for architecture such as MPSoC that includes FPGA based processors, the throughput, latency, number of required FFs, LUTs and block RAMs (area occupied in the FPGA) are taken into account. [15]

Chapter 4

Case Study: System Description and Evaluation Methodology

On this chapter is presented the system proposed for evaluation according to its layers and focus points such as redundancy and reconfiguration schemes. The system described is modelled from the reliability and safety point of view. This includes the FTA and modelling through Markov Chain. Finally further evaluation is performed in terms of CCF, ASIL decomposition and statistical analysis of the reliability function through MCS.

4.1 Dynamic Reconfigurable System

The Dynamic Reconfigurable System (DRS) proposed as case study is composed by three layers as shown in Fig. 4.1. On the lowest layer are the modules for the brake and the steering represented by switches and brushless Direct Current (DC) motors that interact with the steering and braking sub-systems that are controlled by the *Hardware Dependant E/E System* (HDS). The reconfiguration logic is implemented on the middle layer into three sub-levels/sub-system with their own redundancy and reconfiguration methods. The lowest sub-level is composed by the HDS composed by three ECUs that connect with the upper sub-level, the *Dynamic Simplex Architecture* (DSA), through CAN protocol. The *Service Oriented E/E System* represents a Service Oriented Architecture (SOA) that connects to its respective DSA and to the homologous SOA through Ethernet based protocol in the form of the Scalable Service-Oriented Middleware over IP (SOME/IP) which is in line with the Portable Operating System Interface (POSIX) based AUTOSAR Adaptive OS. This connection in addition to the CAN bus and data sharing allow both DSAs to maintain congruent states be-

tween DSAs and the HDS. The upper layer is where the monitoring component is presented. It is necessary in order to detect the faults and perform control operations. The functionalities are also extended for visualisation and analysis functions.[2]

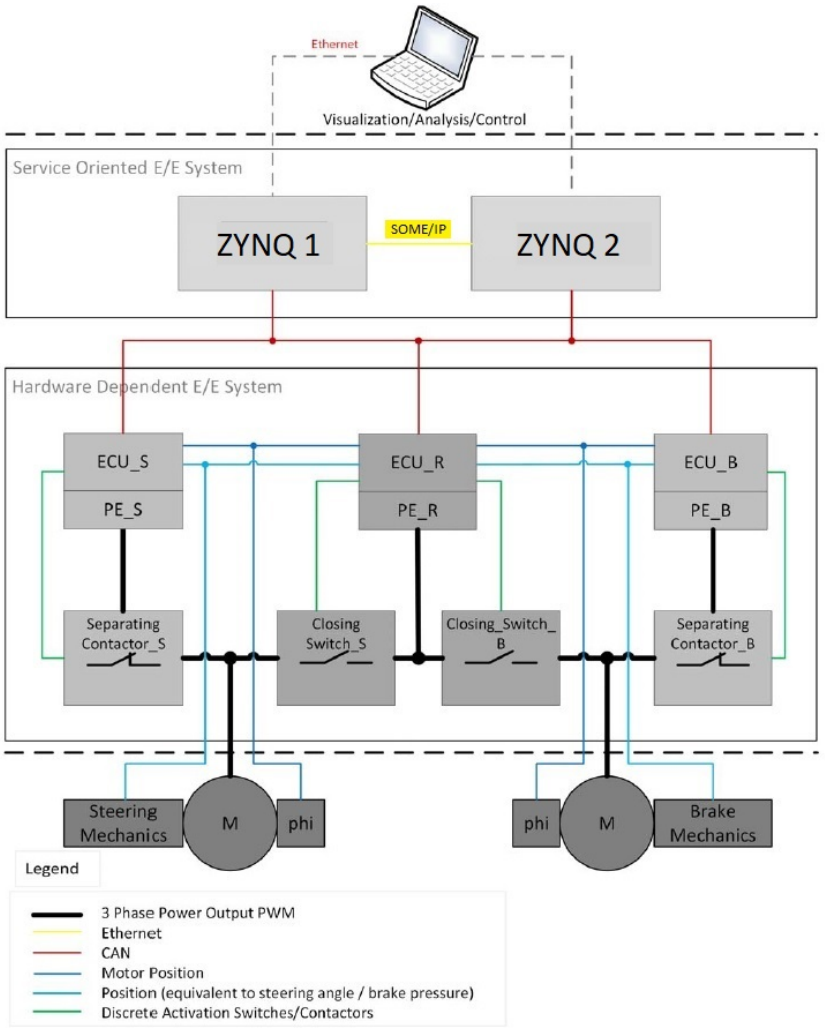


Figure 4.1: E/E Architecture Overview. [2]

4.1.1 Hardware Dependant System

The most relevant aspect of the HDS to be considered in this study, is that its architecture represents an application of an *hot standby dynamic redundant* system as exposed on the section 2.3.6. As shown in Fig. 4.1, the ECU_S is responsible for the control of the steering mechanism and the ECU_B the braking mechanism. In case of a failure on each of these ECUs the reconfiguration is

made to the redundant ECU_R which overtakes the functionality of controlling either the braking or the steering system. The reconfiguration is controlled by the entity *Complex System* of the DSA. It is capable of detecting faults on the ECUs and provide reconfiguration in case of need between ECUs of the HDS. The machine states for the correct reconfiguration on the HDS system are also managed on the *State Transfer Entity* of the DSA to ensure coherent states during reconfiguration.

4.1.2 Dynamic Simplex Architecture

The goal of the proposed DSA concept is to achieve a processor-based fail-operational system behaviour and is related to the work of [16]. In Fig. 4.2 is shown a block schematic from a logical perspective of this concept. The Complex System ($m_{complex}$) is able to detect faults within the DSA and HDS and notify the Control Entity ($m_{control}$) about their occurrence. The DSA defines the mechanism that is triggered after a fault is detected. In this case, the $m_{control}$ disables the $m_{complex}$ and enables the fallback module ($m_{fallback}$). The latter is considerably less complex since it focuses mainly on meeting the system safety requirements. A set of application-specific slave modules, $S = s1, ..., sl$, represent the modules that both the $m_{complex}$ and the $m_{fallback}$ need to interact with. They can be represented as the ECUs of the HDS. [16]

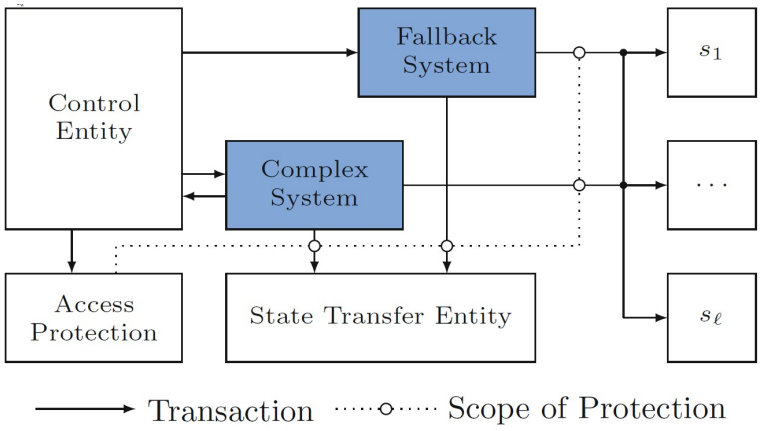


Figure 4.2: Logical view of the dynamic simplex architecture. [16]

The dynamic approach of this concept resides on the functionality of the fallback system, that needs to be implemented on a processor with reconfiguration capabilities. At any point in time, the overall system is in one of two contexts: either in $c_{complex}$ or in $c_{fallback}$ that makes use of the $m_{complex}$ and $m_{fallback}$ respectively where the disabled system is isolated. The $m_{fallback}$ consists of a soft-core processor and occupies the dynamic portion of the FPGA if and only if $c_{fallback}$

is active. If this is not the case, the dynamic portion can be utilized for other purposes. It could, for instance, be used to implement hardware accelerators that perform non-safety-relevant tasks. [16]

A switch back to $c_{complex}$ is possible if the faults are no longer present. Context switches are orchestrated by the $m_{control}$. Adherence to the access permissions is enforced by the access protection module ($S_{protection}$). The state transfer entity (S_{state}) provides a certain amount of buffered memory to transfer consistent snapshots of internal state variables. [16]

The above concept can be implemented on the Xilinx Zynq Ultrascale + MPSoC (ZynqMP) whose block diagram is depicted in Fig. 4.3 showing a combination of a Processing System (PS) and Programmable Logic (PL). The ZynqMP also comprises four main power domains individually isolated reducing the prone for CCF and including units such as:

- Low-Power Domain (LPD): Configuration Security Unit (CSU), Platform Management Unit (PMU) and Real-Time Processing Unit (RPU);
- Full-Power Domain (FPD): Application Processing Unit (APU);
- PL Power Domain (PLPD): PL, General Port (GP) I/O;
- Battery Power Domain (BPD): Real-Time Clock (RTC), Battery-Backed RAM.

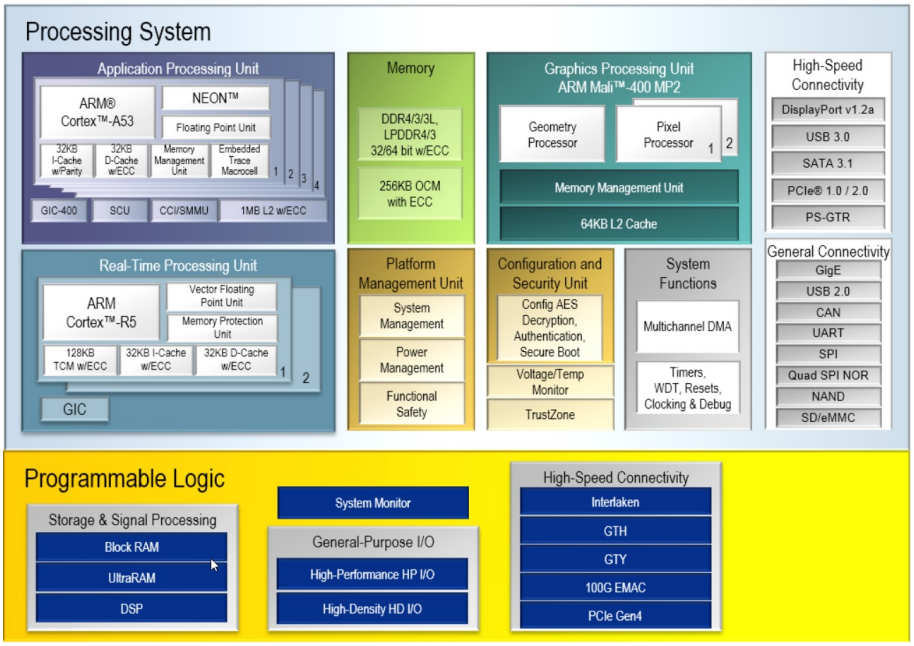


Figure 4.3: ZynqMP Block Diagram. [17]

The *complex system* is implemented on the RPU combined with its Generic Interrupt Controller (GIC), and its Tightly-Coupled Memory (TCM) ports for low-latency and deterministic accesses to local RAM. A Real-Time-Operating-Sytem (RTOS) runs on the the RPU's pair of Cortex-R5 (Arm v7-R 32-bit architecture) cores operating in lockstep mode as a redundant CPU configuration. The context switch to $c_{fallback}$ can be triggered whenever the RPU detects a lockstep error. Apart from the hardware redundancy provided by the lockstep mode, the RPU has other safety features such as, EDC, ECC, Built-In Self-Test (BIST) to detect random faults in hardware and watchdog to detect both systematic and random failures. If required by a particular use case, more sophisticated fault detection techniques may be applied. [17]

The *fallback system* consists of a MicroBlaze (MB) soft-core processor, its local memory and an INTerrupt Controller (INTC). The MB is an FPGA based soft-core processor required to address the dynamical requirements of the concept and is under the PLPD. [17]

The *Control System* entity is implemented on the PMU which provides fault tolerance by applying a triple-redundant processor and ECC on the RAM interface. In response to lockstep error notifications from the RPU, it performs a context switch from $c_{complex}$ to $c_{fallback}$. Following this, the $m_{control}$ resets the RPU and, in case of a transient fault, performs a controlled context switch back to $c_{complex}$. This is possible since the PMU is responsible for handling the primary pre-boot tasks and management of the PS hardware for reliable power up/power down of system resources and system error management. [16]

The *access protection* described in the concept is performed by the Xilinx Peripheral Protection Unit (XPPU). This module is part of the PS and provides detailed control over accesses to the I/O peripherals and the LPD units which are used as application-specific slave modules and have specific access permissions depending on the context. During a context switch, the $m_{control}$ uses the permission definitions to reconfigure the XPPU. The *State Transfer Entity* is implemented in two dedicated PL areas as two entities, the state manager S_{state} which implements the state transfer entity during the context switch and the S_{atom} which stores the state. [16]

The concept described above and shown in Fig. 4.2 can be implemented into the ZynqMP which is implemented according to the block diagram presented in Fig. 4.4.

This concept is characterized by many degrees of freedom. Designs can differ, for instance, on the configuration of the MB, the size of its local memory, the capacity of the state transfer entity, the number of contexts, the PL clock frequency, and the region of the PL that the dynamic portion is constrained to. It is

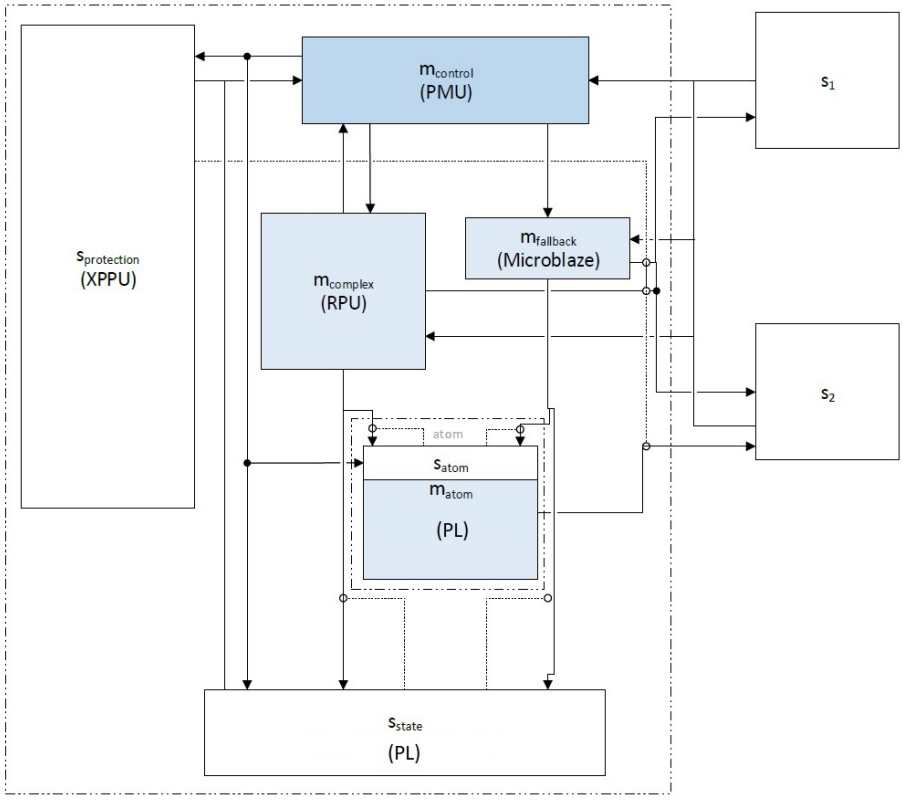


Figure 4.4: Dynamic Simplex Architecture implementation.

therefore imperative to evaluate the system and assess each configuration possibilities to obtain maximum performance while meeting the safety requirements.

4.1.3 Service Oriented Architecture/System

The SOA system has as principal functionality to provide redundancy in terms of managing and control the context switch between two DSAs. Its functionalities can be extended to provide services for each DSA or even allow the implementation of new functionalities during run-time through reconfiguration. The APU, the main unit in this sub-system, is already available on the ZynqMP board. It is integrated on the FPD domain and therefore isolated from the rest of the ZynqMP units in the perspective of safety.

The APU runs a Linux based OS that uses Inter-Process Communication (IPC) to communicate with the correspondent DSA units (RPU, PMU and MB) through CAN bus and with the redundant SOA system through Ethernet. This management is provided by the SOME/IP protocol that routes the information between the application to the redundant APU and the proprietary IPC Can

Provider that communicates with the DSA. The block diagram scheme for this approach is exposed in Fig. 4.5. Each ZynqMP board comprises the DSA and SOA systems and are identified as Zynq1 and Zynq2.

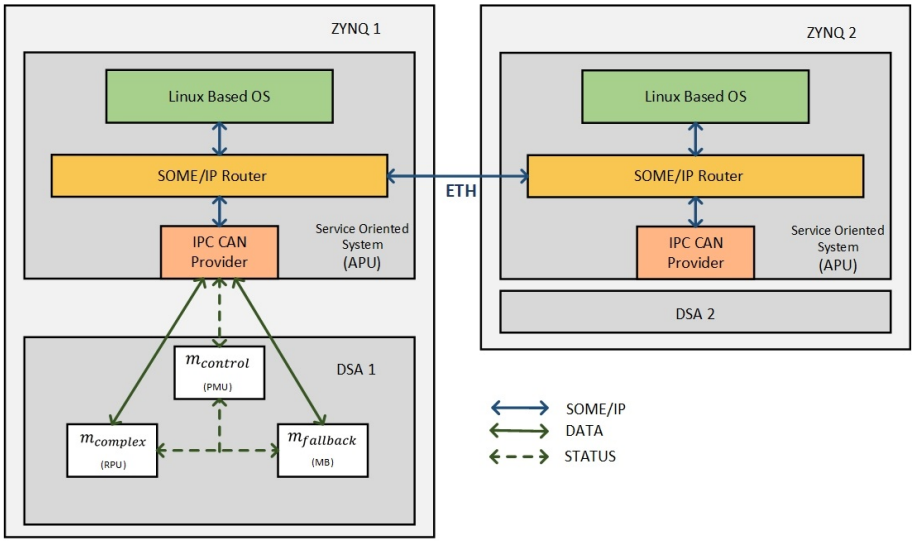


Figure 4.5: Service Oriented Architecture/System.

The APU from each ZynqMP maintain connection in order to share and replicate data on both systems. This allows the DRS system to maintain coherent and reliable information from the HDS, DSA and APU based SOA unit states, so the required reconfiguration on each level is controlled and performed with safety.

4.2 System Modelling

The presented stochastic method used for modelling and reliability analysis are applied considering the following assumptions:

- Fault detection unit is perfect;
- Fault coverage equal to 1;
- Repair rates not considered and therefore only irreparable faults.

These assumptions are made since the values of FIT rates include the FIT for the detection mechanisms to simplify the model. The same approach is used for every block on each subsystem where each block encompass different hardware elements including redundant and fault detection elements. This approach

allows to perform smart-slicing and partitioning of a complex system model in smaller sub-system models enabling the re-usability the model of each sub-system. [63]

In order to simplify the system modelling the fault coverage is considered 100% since the methods and tests used for failure detection and FIT determination rely on approximations, different confidence levels and past history data. [64] Evidently these aspects introduce uncertainties on the model and its results. In order to be possible to integrate the model within certification frameworks, it is possible to include: (a) combination of random simulation and statistical model checking; (b) model-based pattern identification in order prune results and state space. [63] Monte Carlo simulation is included on the evaluation methodology to identify potential impacts and deviations resulting from these uncertainties.

4.2.1 Hardware Dependant System Model

As exposed before the HDS is composed by three ECUs and the CAN bus. Each ECU functional block FIT rate includes not only the ECU itself and underlying software and hardware redundancy but also the interfaces to breaking and steering systems.

The HDS can be analysed using the deductive method of FTA through a top-down approach. The *top* event represents the total loss function of steering and breaking. The basic events are represented by the combinations of failures either the ECUs and the CAN (interfaces and bus). This aggregation is a result of the previous FMEDA and FTA analysis made on each ECU. Having so, the correspondent FTA maintaining the CAN bus isolated from the ECUS of the HDS is depicted in Fig. 4.6.

The FTA with the CAN aggregated to each ECU can be translated to the Markov Chain model. The possible states are described in Table 4.1.

Table 4.1: States of the Markov chain for HDS.

State	Description
1	The three ECUs are operational
2	Two ECUs are operational (Braking ECU failed)
3	Two ECUs are operational (Redundant ECU failed)
4	Two ECUs are operational (Steering ECU failed)
5	The System Failed (Two ECUs failed)

The resulting Markov chain is shown in Fig. 4.7. The states are labelled according to Table 4.1. States 1 to 4 are operational states. State 5 is a failed state.

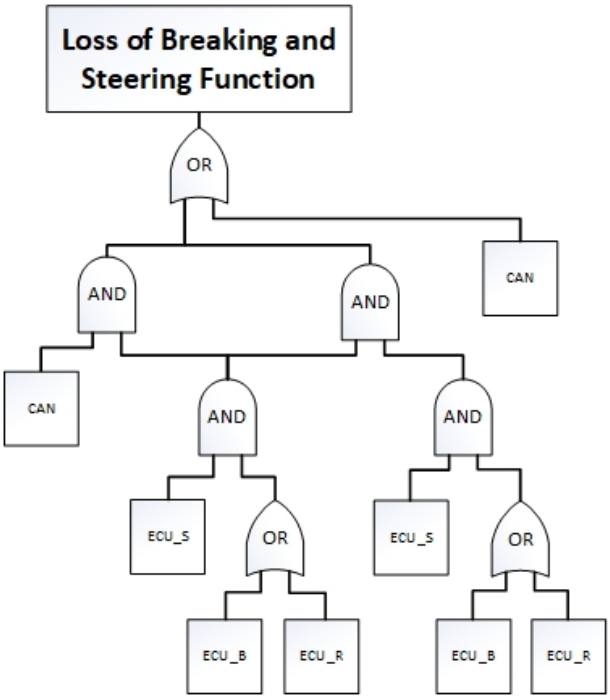


Figure 4.6: Fault Tree Analysis of HDS.

In a *hot standby with one spare configuration*, it is required at least two modules for the system to be operational. Thus, when the third component fails, the system fails. In this reliability evaluation, the system cannot be repaired from a failed state. The failure rates λ_B, λ_S, λ_R are, respectively, the failure rates for the breaking, steering and redundant ECUs.

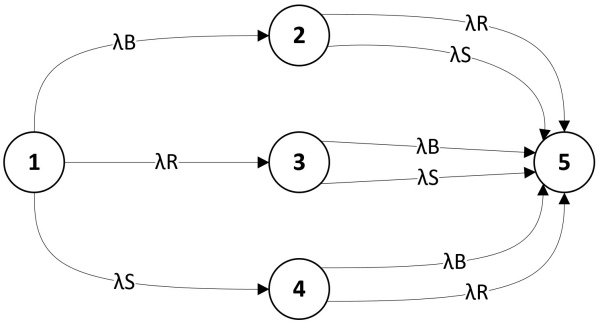


Figure 4.7: Markov chain of of HDS.

As exposed on section 2.2.1.5 of this work Markov chains can be expressed by a state transition matrix. The state transition matrix for the HDS Markov

chain exposed in Fig. 4.7 and Table 4.1 is given by:

$$M = \begin{bmatrix} -(\lambda_B + \lambda_S + \lambda_R) & 0 & 0 & 0 & 0 \\ \lambda_B & -(\lambda_R + \lambda_S) & 0 & 0 & 0 \\ \lambda_R & 0 & -(\lambda_B + \lambda_S) & 0 & 0 \\ \lambda_S & 0 & 0 & -(\lambda_R + \lambda_S) & 0 \\ 0 & \lambda_R + \lambda_S & \lambda_B + \lambda_S & \lambda_R + \lambda_S & 0 \end{bmatrix}$$

Following the transition matrix M, the following differential equations can be derived:

$$\dot{P_1} = -(\lambda_B + \lambda_S + \lambda_R)P_1$$

$$\dot{P_2} = \lambda_B P_1 - (\lambda_R + \lambda_S)P_2$$

$$\dot{P_3} = \lambda_R P_1 - (\lambda_B + \lambda_S)P_3$$

$$\dot{P_4} = \lambda_S P_1 - (\lambda_B + \lambda_R)P_4$$

$$\dot{P_5} = (\lambda_R + \lambda_S)P_2 + (\lambda_B + \lambda_S)P_3 + (\lambda_R + \lambda_S)P_4$$

Assuming the initial state as faultless the following conditions are applied:

$P_1(t = 0) = 1;$

$P_2(t = 0) = P_3(t = 0) = P_4(t = 0) = P_5(t = 0) = 0;$

With these initial conditions is now possible to solve the differential equations and determine the resulting state probabilities for each state. These probabilities can be found in the digital annex A. It is possible now to find the system reliability $R(t)$ by computing the sum of probabilities taken over all the operational states (P_1 to P_4):

$$R(t) = \sum_{i=1}^{4} P_i(t)$$

or alternatively:

$$R(t) = 1 - P_5(t)$$

The resulting reliability function for the HDS, $R_{HDS}(t)$, is presented in the digital annex A.

4.2.2 Dynamic Simplex Architecture Model

The DSA is implemented on the Zynq board LPD and PLPD domains and isolated from the SOA system despite implemented in the same board. The possible states are described in Table 4.2. The λ_{RPU} and λ_{MB} are the failure rates for the RPU ($m_{complex}$) and Microblaze ($m_{fallback}$) respectively. The λ_K comprises the failure rates of the modules (K-modules) implemented on the PLPD (m_{atom}, S_{atom}, S_{state} and respective interconnect), XPPU ($S_{protection}$), PMU ($m_{control}$), NOC

module (representing interconnections, I/0 between LPD and PLPD) and out-of-the-box modules that have specific functions such as the CSU for secure booting purposes.

Table 4.2: States of the Markov chain for the DSA.

State	Description
1	All modules are operational.
2	Failure on the Microblaze. RPU and K-modules operational.
3	Failure on the RPU. Microblaze and K-modules operational.
4	DSA system failure. K-modules failed or both RPU and MB failed.

The same approach used for the HDS is applied to the DSA and its FTA. The Markov chain for the DSA is presented in Fig. 4.8.

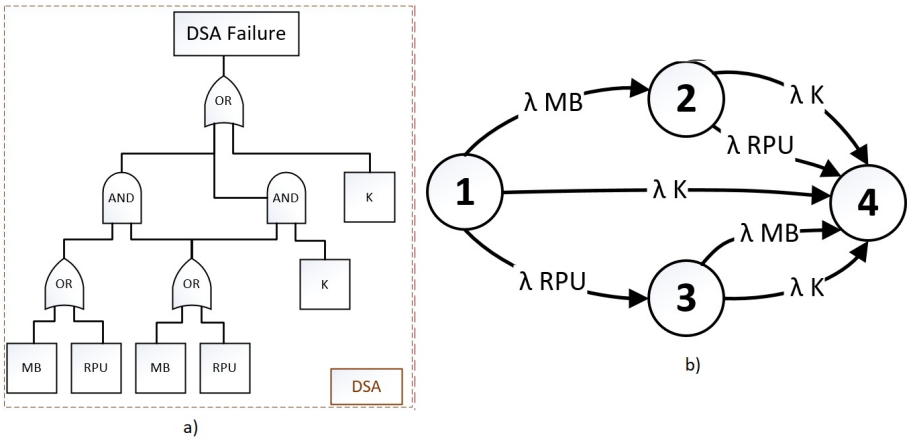


Figure 4.8: DSA: a) FTA b) Markov Chain.

The state transition matrix for the HDS Markov chain exposed in Fig. 4.8 and Table 4.2 is given by:

$$M = \begin{bmatrix} -(\lambda_{MB} + \lambda_K + \lambda_{RPU}) & 0 & 0 & 0 \\ \lambda_{MB} & -(\lambda_K + \lambda_{RPU}) & 0 & 0 \\ \lambda_{RPU} & 0 & -(\lambda_{MB} + \lambda_K) & 0 \\ \lambda_K & (\lambda_K + \lambda_{RPU}) & (\lambda_{MB} + \lambda_K) & 0 \end{bmatrix}$$

Following the transition matrix M, the following differential equations can be derived:

$$\dot{P_1} = -(\lambda_{MB} + \lambda_K + \lambda_{RPU})P_1$$

$$\dot{P_2} = \lambda_{MB}P_1 - (\lambda_K + \lambda_{RPU})P_2$$

$\dot{P}_3 = \lambda_{RPU} P_1 - (\lambda_{MB} + \lambda_K) P_3$

$\dot{P}_4 = \lambda_K P_1 (\lambda_K + \lambda_{RPU}) P_2 (\lambda_{MB} + \lambda_K) P_3$

Assuming the initial state as faultless the following conditions are applied:

$P_1(t = 0) = 1;$

$P_2(t = 0) = P_3(t = 0) = P_4(t = 0) = 0;$

With these initial conditions is now possible to solve the differential equations and determine the resulting state probabilities for each state. These equations can be found in the digital annex A. It is possible now to find the system reliability $R(t)$by computing the sum of probabilities taken over all the operational states (P_1 to P_4):

$R(t) = \sum_{i=1}^{3} P_i(t)$

or alternatively:

$R(t) = 1 - P_4(t)$

The resulting reliability function for the DSA, $R_{DSA}(t)$, is presented in the digital annex A.

4.2.3 Dynamic Redundant System

As mentioned before, the DRS comprises two ZynqMP boards and they can be seen as a sub-system in the sense that they are symmetric in order to achieve both DSA and SOA redundancy. Therefore, it is possible to consider each DSA+SOA as a subsystem (A and B) on the FTA. It is important to note that each APU keeps tracking of the status of their respective DSA modules and communicates with the other APU to control the feasibility of a context switch between a faulty DSA and the redundant DSA. In case of failure on one APU, the other APU can acknowledge this failure and perform the context switch between DSAs. This approach is analysed on the following subsection considering the whole DRS (HDS, DSA and SOA). The resulting possible states are described in Table 4.3.

The DRS FTA is shown in Fig. 4.9. It encompasses the previous systems and organized them in terms of a top-down analysis. The *top* event is represented by the complete loss of function and similarly as the HDS, it results in loss of steering and breaking. However, differently from the HDS, the top event of DRS also includes the failure of the DSA and other less critical functionalities that might be included but that are not evaluated on the scope of this work.

The same approach as for DSA and HDS is applied to the DRS. The FTA can be translated to the Markov chain as shown in Fig. 4.10.

Same Assumptions are made as before so that:

Table 4.3: States of the Markov chain for the DRS.

State	Description
1	All systems are operational.
2	Failure on the DSA A. Remaining systems operational.
3	Failure on the SOA A. Remaining systems operational.
4	Failure on the SOA B. Remaining systems operational.
5	Failure on the DSA B. Remaining systems operational.
6	Failure of the sub-system A (DSA A + SOA A). Sub-system B Operational.
7	Failure of the sub-system B (DSA B + SOA B). Sub-system A Operational.
8	The system failed.

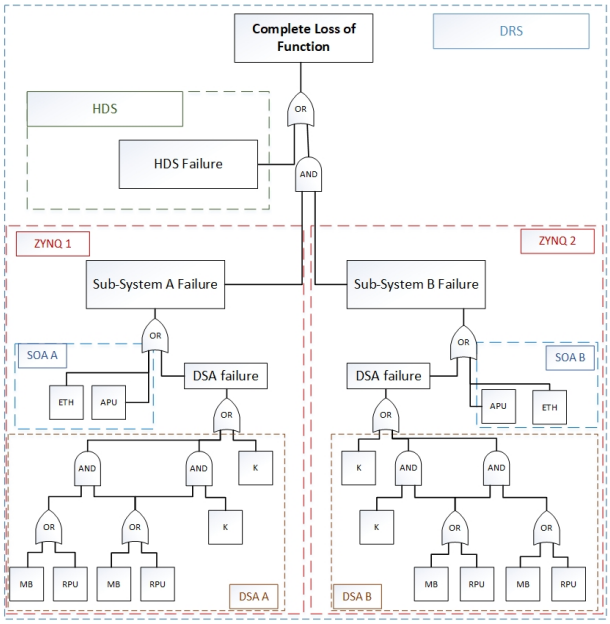


Figure 4.9: Dynamic Reconfigurable System FTA.

$P_1(t=0) = 1$;

$P_2(t=0) = P_3(t=0) = P_4(t=0) = P_5(t=0) = P_6(t=0) = P_7(t=0) = P_8(t=0) = 0$;

With these initial conditions is now possible to solve the differential equations and determine the resulting state probabilities for each state. These equations can be found in the digital annex A. It is possible now to find the system reliability $R(t)$by computing the sum of probabilities taken over all the operational states (P_1 to P_4):

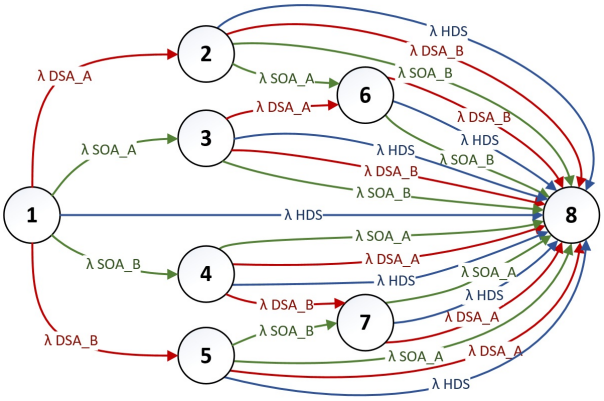


Figure 4.10: DRS Markov of Chain.

$$R(t) = \sum_{i=1}^{7} P_i(t)$$

or alternatively:

$$R(t) = 1 - P_8(t)$$

The resulting reliability function for the DSA, $R_{DRS}(t)$, is presented in the digital annex A.

4.3 Extended Evaluation

4.3.1 Safety Integrity Level

Safety is a critical property that encompass three elements. The first element starts with the principle that safety is never absolute. Hence it considers the likelihood of a failure and the severity of the consequences, the second element. A third element is controllability. The combination of these three factors is used to determine a required SIL or Safety Integrity Level, categorized in 4 levels, SIL-1 being the lowest and SIL-4 being the highest. The levels can then also classified in categories as shown in table 4.4. [20]

The SIL level is used as a directive to guide selecting the required architectural support and development process requirements. For example SIL-4 imposes high redundancy and makes the use of formal methods as highly recommended. Table 4.5 shows an approximate mapping between IEC 61508, ISO 26262 and DO-178C standards. The respective SILs are defined as DAL for aviation and ASIL for automotive. For instance, a DAL-A application might cause the aircraft to crash, while a DAL-D application, in the worst case, can just result in minor issues. [52] These levels require corresponding Risk Reduction Factors

Table 4.4: Categorization of SIL. [20]

Category	Typical SIL	Consequence upon failure
Catastrophic	4	Loss of multiple lives
Critical	3	Loss of a single life
Marginal	2	Major injuries to one or more persons
Negligible	1	Minor injuries at worst or material damage only

that depend on the operation scenario. Risk Reduction Factors are vastly different as well and comprises different fail-safe modes. For example while a train can be stopped if a failure is detected, a plane must at all cost be kept in the air in a state that allows it still to land safely. [20]

Table 4.5: Aproximate comparison between IEC-61508, ISO-26262 and DO-178C/254 in term of safety levels. [20]

Domain	Domain specific SILs				
IEC-61508	SIL-0	SIL-1	SIL-2	SIL-3	SIL-4
ISO-26262	ASIL-A	ASIL-B	ASIL-C	ASIL-D	-
DO-178C/254	DAL-E	DAL-D	DAL-C	DAL-B	DAL-A

4.3.1.1 SIL Determination

The SIL levels are normally determined after the system analysis through inductive methods such as HARA and FMEDA executed before and updated during and after the development phase. According to the severity and probability of each SIL, it is determined the requirements in terms of failure rate as a quantification of the Probability of Failure per Hour. This value is also referred as Probabilistic Metric of Random Hardware Faults (PMFH). Table 4.6 summarises the required failure rates for IEC 61508. The Low Demand mode of Operation (LDO) represents the average probability of failure to perform its design on demand and the High Demand mode of Operation (HDO) represents probability of one dangerous failure per hour on a continuous mode of operation. [21]

Despite ASIL B and ASIL C have the same FIT range, they differ on the other metrics of Single-Point Fault Metric (SPFM) and Latent Fault Metric (LFM). The SPFM metric reflects the robustness of an element/component to the single-point faults (faults that are not covered by a safety mechanism that lead directly to the violation of a safety goal). The LFM reflects the robustness of an element/component against latent faults (whose presence are not detected by a

Table 4.6: SIL and ASIL failure rate comparison. [20] [21]

SIL	ASIL	LDO	HDO	Failure Rate (HDO)
1	-	10^{-2} to 10^{-1}	10^{-6} to 10^{-5}	1000 to 10000 FIT
2	A	10^{-3} to 10^{-2}	10^{-7} to 10^{-6}	100 to 1000 FIT
3	B	10^{-4} to 10^{-3}	10^{-8} to 10^{-7}	10 to 100 FIT
-	C	10^{-4} to 10^{-3}	10^{-8} to 10^{-7}	10 to 100 FIT
4	D	10^{-5} to 10^{-4}	10^{-9} to 10^{-8}	1 to 10 FIT

safety mechanism or the driver). The SPFM and LFM are normally used for elements that will be integrated in a broader system, i.e sensors. Therefore for the analysis of the DRS it is used the PFMH normally used for integrated systems. [21]

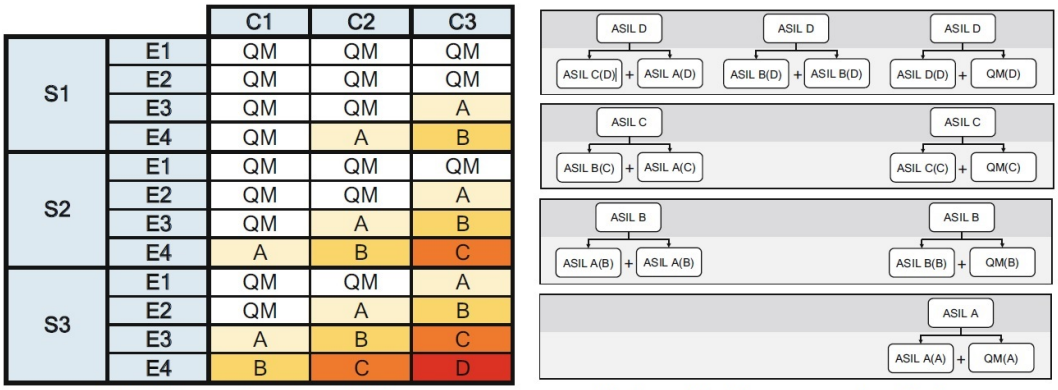
4.3.1.2 ASIL

ASILs are the key component of ISO 26262, used to represent the severity of safety requirements. From the previously SIL levels, another level is added so that there are five levels (QM, A, B, C, D) from the least strict ASIL A to the strictest ASIL D, where QM (Quality Management) means no safety requirements. According to ISO 26262 ASILs are assigned integer values as: ASIL QM = 0, ASIL A = 1, ASIL B = 2, ASIL C = 3, and ASIL D = 4. ASILs are allocated to hazardous components based on the severity of the hazard caused by the failure of that component. *ASILs decomposition* concept allows the ASIL to be decomposed over components that together provide the same hazard. ASILs allocation is a hard, complex problem of finding the most appropriate allocation of safety requirements to the components of the automotive system. An appropriate ASILs allocation to components and subsystems must guarantee the fulfilment of least-risk safety requirement with the least development cost. [23]

The Fig. 4.11 a) shows how the ASIL values are calculated based on the three risk parameters. The highest level D corresponds to all the risk parameters being at their maximum: S3 corresponds to life-threatening or fatal injuries, E4 to a high probability of exposure and C3 to a difficult to control or uncontrollable risk. [18]

Moreover, the standard uses the FIT as a measure of the failure rate and as a measure of the probability of failure per hour. This includes the calculation of the failure amount in hours. In this context, the ASILs require different FIT rates which are listed in Table 4.6. [31]

Designing ASIL D compliant devices and software is a difficult task, and often the highest safety level can be achieved only by exploiting the knowledge of



a) Asil Determination Table b) Possible Decompostion Scheme

Figure 4.11: ASIL Calculation table. [18]

the architecture and feature to increase system reliability. Decomposing the system into less-critical elements may be the most cost-efficient solution. Fig. 4.11 b) shows the acceptable ASIL decomposition schemes defined by the standard, which follow the rule of the equation below [18]:

$$ASIL_{orig} \leq \sum ASIL_{decomp}$$

The standard uses the notation $ASIL_{decomp}(ASIL_{orig})$ to mark which elements have been decomposed and trace the original requirement. To validate ASIL D decomposition it is necessary further analysis. When a requirement is decomposed into redundant elements, it is necessary to establish independence between them for the original safety requirement to be correctly satisfied meaning that they cannot have CCFs that could result in a system failure. [18]

4.3.1.3 DRS ASIL Decomposition

Being the goal to achieve ASIL D, one possible ASIL decomposition for the DRS is presented in Fig. 4.12. The ASIL value attributed to each block in the scheme is based on the vendor assessment according to proprietary FMEDA tools and safety analysis. According to this information, this diagram represents a possible decomposition at an early stage of specification phase. On the levels highlighted in red (level 1 and 2), it is subjective that the ASIL D is already achieved at level 2, however as mentioned before, to validate ASIL D decomposition it is necessary further analysis as described on the following sections.

These ASIL estimations are based on general safety evaluation and do not represent the real dependency between the functionalities deployed in each block. Ultimately, the DRS Markov Chain model can be replaced with the FIT rates for each block and a more accurate results can be achieved as it will be described

on chapter 6. Furthermore, the CCF and single point fault analysis should be taken into consideration and therefore the overhead on the ASIL determination for each block might be necessary to validate the ASIL decomposition or identify key aspects to define a new architecture.

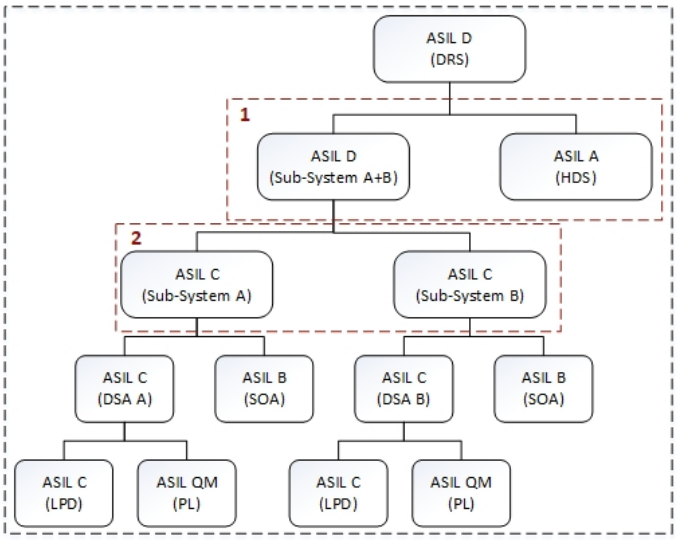


Figure 4.12: Possible ASIL Decomposition for DRS.

4.3.2 Common Cause Failure

As perceived in Fig. 4.1 at the beginning of this chapter, the HDS system is isolated from the DSA and SOA sub-systems either by power and computing resources. The DSA and SOA subsystems are also isolated through the power domain ZynqMP isolation mechanisms. Furthermore, the ZynqMP provides further memory and peripherals isolation.

The ZynqMP has four fundamental memory regions. These memory regions are the Double Data Rate (DDR) memory, On-Chip Memory (OCM), TCM, and Advanced eXtensible Interface (AXI) block RAM in the PLPD. Access to memory is controlled by the memory controllers, Direct Memory Access Controllers (DMAC), and Memory Management Units (MMU). Access to the peripherals can be dedicated or shared. Isolation of the peripherals is provided using the XPPU. Additionally AXI Isolation Blocks (AIBs) can be used to enhance isolation. These blocks are spread throughout the entire PS of the device. [65] On the PLPD the ZynqMP has a set of constraints to control routing and floorplanning to avoid overlapping and improve isolation within the FPGA. [66]

These features provide some confidence while implementing the isolation of a sub-system or specific function and prevent CCF at the element/function level.

However, when translating the CCF analysis to the system modelling, further evaluation must be carried out. Remembering the CCF definition, they result from a single causing fault on different (normally redundant) components/subsystem. The CCF analysis is similar to the example presented on section 2.2.1.4 and can be included in the Markov Chain model. However, it must be segregated from the block failure rate. The CCF acts as a new element resulting in a new system state in the system's reliability structure with a failure rate equal to the occurrence rate δ_C of the CCF and repair equal to the repair rate μ_C of the CCF. Fig. 4.13 reflects this approach based on the example of *1 out of 2* redundant system provided on the dependability section 2.2.1.5 (Fig. 2.2). [19]

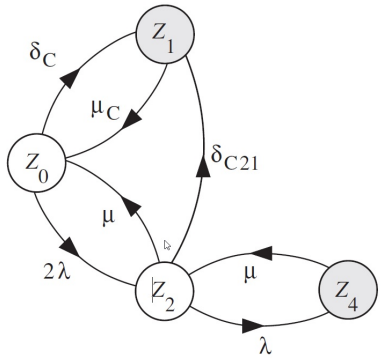


Figure 4.13: Markov chain CCF analysis of example 2.2.1.5 with constant failure ($\lambda_1 = \lambda_2$) and repair rates (μ, and μ_C - CCF repair rate), constant occurrence rates for CCF (δ_{C21}).[4] [19]

Taking into consideration the isolation features described, the fact that the DRS implementation is carried out on a development board and that CCF analysis increases the model complexity, CCF incorporation on the Markov model was excluded in the scope of this work and it may be a topic for future work. However, there are some considerations that are important to note. When looking to the Markov Chain of the DSA in Fig. 4.8, we can identify the λ_K as a CCF in the sense that, when a fault occurs on the K-modules it leads to the impossibility of the RPU and MB to fulfill their functionalities. This occurence is partially the definition of CCF described on the section 3.3.1.2. Similarly when analysing the DRS we can identify the HDS has a possible CCF for the sub-systems of DSA and SOA. This approach allows to identify design weaknesses which are a form of CCF. [19]

4.3.3 Monte Carlo Simulation

As explained on the system modelling introduction section 4.2, the FIT rates have associated uncertainties resulting from the methods used for its determina-

tion which include measurement errors, statistical uncertainty and transformation uncertainty. Understandably it is difficult to characterize and include these uncertainties on the Markov Chain model. The purpose of the MCS is to have an approach that considers FIT rate fluctuations and consequently reduces the system reliability confidence level. MCS is an important computational technique to improve the statistical tests and handling .

Fig. 4.14 shows the flow chart for the implementation of the MCS with MATLAB. The implementation consists in:

1. Determining the uncertainties in FIT rates through probability distribution for each block;
2. Generating the $n \times N$ matrix (n =number of different blocks, N =number of samples) of random samples according to the distribution defined on the previous point;
3. Establishing Reliability Probabilistic Model R(t) for HDS, DSA, SOA and DRS through Markov Chain;
4. Calculating N time the FIT rate and MTTF for HDS, DSA, SOA and DRS by substituting random generated FIT rates;
5. Performing statistical analysis of the simulation results to find correlations and identify key indicators and parameters;
6. Determining optimizations and new scenarios according to the insights on the previous point.

The MCS generates a data set containing the failure rates for each block of the DRS combined with the calculated MTTF and FIT for each sub-system. This data set is afterwards submitted to statistical analysis. By performing statistical analysis it is possible to identify key aspects of the architecture and correlations between blocks, sub-systems and overall subsystem. This step finalizes the evaluation methodology where all the results of each phase of the evaluation methodology are gathered and analysed in order to identify new scenarios, approaches and key points of improvement. The next chapter describes the implementation of this methodology on the case study presented.

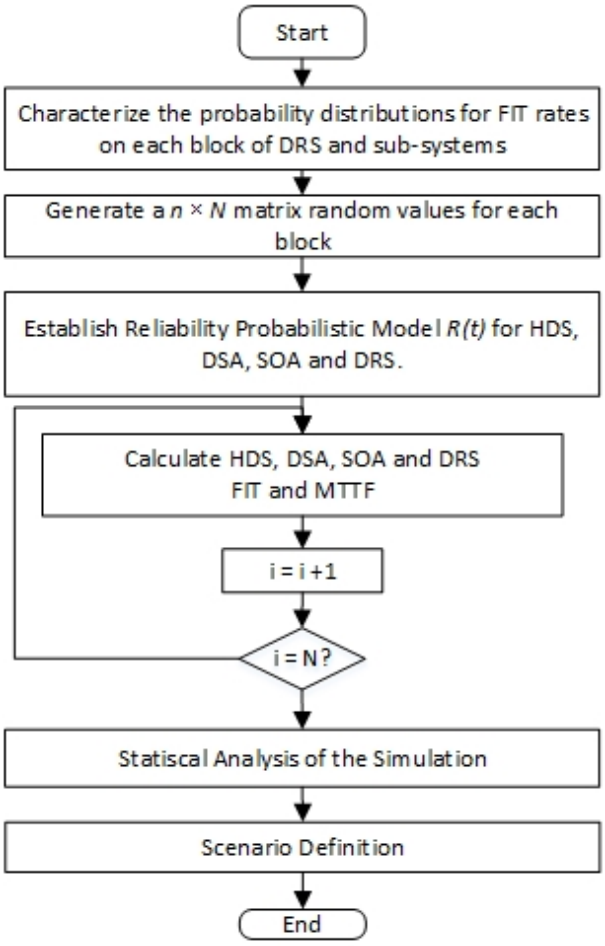


Figure 4.14: Implementation flow chart for MCS.

Chapter 5

Implementation and Simulation

The implementation of the evaluation method is separated in two distinct parts. The first part is related to the system modelling and find the reference scenario. Afterwards, the MCS and statistical analysis is performed to obtain more insight and define new scenarios and optimizations. The implementation of the system modelling evaluation requires firstly the determination of the constraints of analysis in terms of failure rate of each block. The FIT rates are result of complex processes and dependant of the type of hardware under analysis. Afterwards these initial values are used as reference for the first reliability model results.

5.1 Reliability Constraints

The FIT rates for each system block are determined by proprietary FMEDA tools, synthesis and simulation reports which are based on the quantitative analysis of all the electronic components in terms of permanent and transient hardware faults. Additionally the FMEDA analysis is based on the expected vehicle electronics lifespan considered as 15 years and 8000 hours of operation per year.

The reference values for the HDS are obtained through in-company FMEDA analysis and comprises not only the ECU with its in-built redundant cores but also its interfaces, peripherals and memory. The same approach is used on the SOA to find the FIT rate for the APU and its associated components such as Ethernet interfaces within FPD. Finally for the DSA the FIT rate determination is more complex since it is composed by two isolated power domains, the LPD and PLPD. In this respect the RPU FIT rate encompass the Arm cortex-R5 in lockstep mode including the TCM and the INTC. The remaining components such as PMU and CSU and I/O and interconnect are included on the K-modules.

Similarly, on the PLPD it is isolated the implementation of the MB represented by the utilized area of the PL used for its implementation (logic circuits and memory - Block RAM (BRAM), Configuration RAM (CRAM) and Ultra RAM (URAM)), AXI interconnection and INTC. The remaining modules and PL features are grouped on the K-modules such as the Atom, State, reset system and AXI INTC for these modules.

The statistical distributions chosen for the MCS are Normal (N) distribution for values within the range QM to ASIL C so the reference values are centered and characterized by the mean ($\bar{x}$) and the uncertainties of the model are characterized by the standard deviation (σ). For low FIT rate values ($FIT \leqslant 1$) the chosen statistical distribution is the Beta distribution $B(a, b)$ with the uncertainties being characterized by the interval of the curve defined by a and b with the values centred on the reference values.

5.2 Scenarios

5.2.1 Reference Scenario

The reference values summarized in Table 5.1 reflects the block FIT rates for hardware permanent faults. This means that only irreparable scenarios for each block and sub-system are considered. This scenario represents the reference and is directly related to the implementation of the DRS into the evaluation method.

Table 5.1: Reference Scenario and Values

Scenario	1						
Description	Reference Scenario						
Block	ECU_B	ECU_S	ECU_S	K-Modules	MB	RPU	APU
FIT	700	700	700	16.97	0.0357	0.13	50
MCS	Samples: 1000000						
Distribution	N	N	N	N	B	B	N
Parameters	$\bar{x} = 700$ $\sigma = 140$	$\bar{x} = 700$ $\sigma = 140$	$\bar{x} = 700$ $\sigma = 140$	$\bar{x} = 16.97$ $\sigma = 3.4$	$a = 81$ $b = 2265$	$a = 81$ $b = 619$	$\bar{x} = 50$ $\sigma = 10$

5.2.2 Alternative Scenarios

The different modelling scenarios reflect an incremental approach. In the first scenario mentioned before, the reference values were obtained and a preliminary results analysis done. The following scenarios were derived from the first assuming a pessimistic approach considering new optimizations with the

objective to improve the reliability. For these scenario deviations formulation it was taken into account the Monte Carlo results, CCF analysis and DSE trade-off.

The first alternative scenario is intended to approach the transient faults for each block of DSA as permanent faults since the DSA is one key point of possible future implementations/optimizations and comprises the most critical reconfiguration features of this DRS concept. This approach increases dramatically the FIT rate and so it is considerer to be a pessimistic approach since by definition, transient faults are normally corrected or minimized through redundancy implemented on each block (such as out-of-the-box PLPD TMR). This scenario could represent for instance the occurrence of a transient fault where the fault handling was poorly designed and result in a failure of a software modules or a inconsistent state. Furthermore, the complexity of the model would have to be redesigned to include recovery rates for this case study implementation as exposed in section 2.2.1.5 and 4.3.2. Therefore, this pessimistic approach represents the case where the recovery rates are not sufficient at the same instant according to a fault and the respective redundancy or recovery mechanism will always fail. The values used are referenced in Table 5.2.

Table 5.2: Scenario 2: Pessimistic approach on DSA.

Scenario	2						
Description	Pessimistic approach on DSA						
FIT	ECU_B	ECU_S	ECU_S	K-Modules	MB	RPU	APU
Value	700	700	700	55.41	4.79	0.885	50
MCS	Samples: 1000000						
Distribution	N	N	N	N	N	B	N
Parameters	$\bar{x} = 700$ $\sigma = 140$	$\bar{x} = 700$ $\sigma = 140$	$\bar{x} = 700$ $\sigma = 140$	$\bar{x} = 55.41$ $\sigma = 11$	$\bar{x} = 4.79$ $\sigma = 0,9$	$a = 44$ $a = 50$	$\bar{x} = 50$ $\sigma = 10$

The third scenario considered is the optimization of the HDS system and consists in lowering the FIT rate of each ECU block. This scenario is based on a preliminary analysis where the results suggested the HDS to be a key point on the overall DRS MTTF and therefore FIT and ASIL value. The scenario obeys to the custom architecture and model where the HDS ECUs FIT rates shown in Table 5.3 are optimized for the overall reliability.

The fourth scenario consists in lowering the FIT rate of the implementation of the SOA system. This scenario derives from the results of MCS correlation after the optimization of the HDS. Table 5.4 presents the values for this scenario.

The fifth scenario derives of the possibility to optimize the blocks that are

Table 5.3: Scenario 3: Optimization for the HDS system

Scenario	3						
Description	Optimization of the HDS system						
FIT	ECU_B	ECU_S	ECU_S	K-Modules	MB	RPU	APU
Value	50	50	50	16.97	0.0357	0.13	50
MCS	Samples: 1000000						
Distribution	N	N	N	N	B	B	N
Parameters	$\bar{x} = 70$ $\sigma = 10$	$\bar{x} = 70$ $\sigma = 10$	$\bar{x} = 70$ $\sigma = 10$	$\bar{x} = 16.97$ $\sigma = 3.4$	$a = 81$ $b = 2265$	$a = 81$ $b = 619$	$\bar{x} = 50$ $\sigma = 10$

Table 5.4: Scenario 4: Optimization of SOA and HDS sub-system.

Scenario	4						
Description	Optimization of SOA						
FIT	ECU_B	ECU_S	ECU_S	K-Modules	MB	RPU	APU
Value	50	50	50	16.97	0.0357	0.13	7
MCS	Samples: 1000000						
Distribution	N	N	N	N	B	B	N
Parameters	$\bar{x} = 50$ $\sigma = 10$	$\bar{x} = 50$ $\sigma = 10$	$\bar{x} = 50$ $\sigma = 10$	$\bar{x} = 16.97$ $\sigma = 3.4$	$a = 81$ $b = 2265$	$a = 81$ $b = 619$	$\bar{x} = 7$ $\sigma = 1.4$

implemented on the PL since the FIT rate of each block is dependant on the area they occupy in PL. The Vivado Design Suite provides optimization options during synthesis and implementation phase. Table 5.5 presents the values for the PL optimization (5.1) and the clock optimization (5.2).

Table 5.5: Scenario 5: DSE Optimizations.

Scenario	5						
Description	Optimization for Area (5.1) and Clock (5.2)						
FIT	ECU_B	ECU_S	ECU_S	K-Modules	MB	RPU	APU
Value (5.1)	700	700	700	16.971	0.0334	0.135	50
Value (5.2)	700	700	700	16.972	0.0336	0.135	50

Chapter 6

Results

6.1 Markov of Chain Model and Monte Carlo Simulation

6.1.1 Scenario 1: Reference

6.1.1.1 HDS

With the reliability function $R_{HDS}(t)$ obtained from the model it is possible to compute the values for the HDS by replacing the values for the reference FIT rates of ECU_B, ECU_S, ECU_R according to the scenario 1 presented in Table 5.1. The left graph in Fig. 6.1 shows a reliability function $R_{HDS}(t)$ exponentially decreasing as expected. The right graph shows the reliability after 15 years of continuous operation ($t = [0, 120000]h$) of 98%. From the results summarized in Table 6.1 it is inferred that HDS as a system is rated as ASIL A.

Table 6.1: Scenario 1: quantitative results for HDS sub-system.

Scenario	1 - Reference Scenario - HDS				
Parameters	MTTF	$R(15years)$	λ_{HDS}	FIT	ASIL
Values	$1.19 \times 10^6 h$	0.98	8.4×10^{-7}	840	A

6.1.1.2 DSA

The left graph in Fig. 6.2 shows the reliability function $R_{DSA}(t)$ exponentially decreasing as expected. The right graph shows a reliability after 15 years of continuous operation ($t = [0, 120000]h$) of 99%. From the results summarized in Table 6.2, it is inferred that DSA is rated as ASIL B/C according to the FIT rate for this reference scenario. The resulting FIT is close to the FIT of K-modules

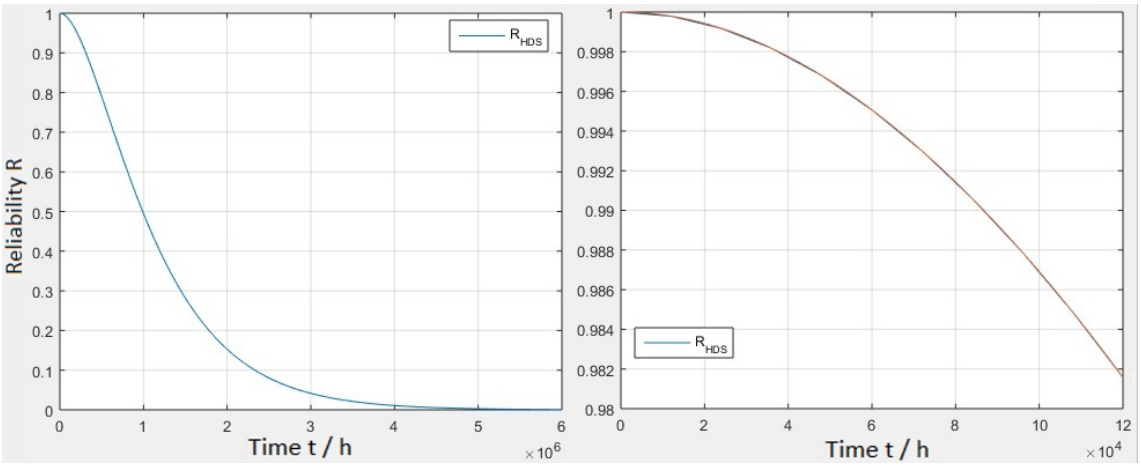


Figure 6.1: HDS reliability function.

and suggests a potential CCF as exposed on section 4.3.2. However this fact cannot be fully considered as a design weakness in the sense that among other features, the K-modules also encompass elements used to achieve redundancy and represent a mean to achieve flexibility to deploy safely several functions in the PL by using its powerful interconnections and isolation mechanisms.

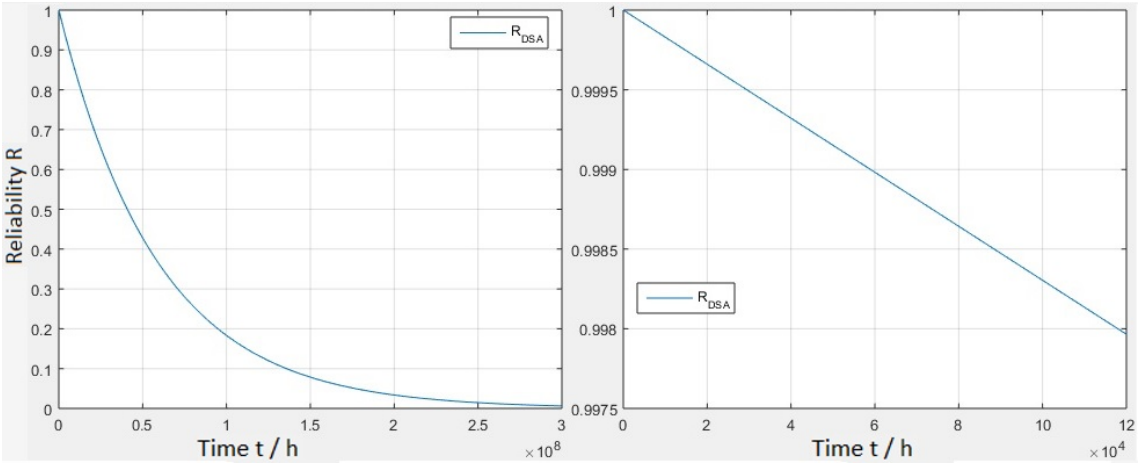


Figure 6.2: DSA reliability function.

6.1.1.3 DRS

The left graph in Fig. 6.2 shows the reliability function $R_{DRS}(t)$ exponentially decreasing as expected. The right graph shows a reliability after 15 years of continuous operation ($t = [0, 120000]h$) of 90.5%. From the results summarized

Table 6.2: Scenario 1: quantitative results for DSA sub-system.

Scenario	1 - Reference Scenario - DSA				
Parameters	MTTF	$R(15years)$	λ_{HDS}	FIT	ASIL
Values	$5.89 \times 10^7 h$	0.99	1.697×10^{-8}	16.97	B/C

in Table 6.3, it is inferred that DRS as a system is rated as ASIL A according to the FIT rate for this scenario.

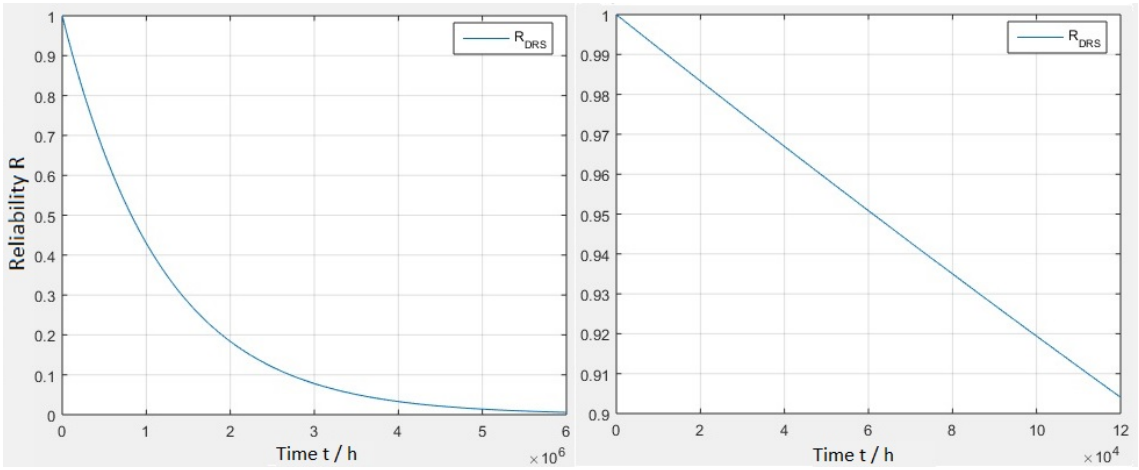


Figure 6.3: DRS reliability function.

Table 6.3: Scenario 1: quantitative results for DRS sub-system.

Scenario	1 - Reference Scenario - DRS				
Parameters	MTTF	$R(15years)$	λ_{HDS}	FIT	ASIL
Values	$1.18 \times 10^6 h$	0.905	8.48×10^{-7}	848.61	A

The MCS for this scenario shows a Normal distribution for the data considering the uncertainties on the methods used to obtain block FIT rate. The distribution is presented in Fig. 6.4. The median and mean values are respectively $\tilde{x} = \bar{x} = 842$ and the standard deviation $\sigma = 100$ with a 95% confidence interval. The data dispersion for Q_1 and Q_3 shows that the overall FIT rate interval between the [775, 910] interval.

By performing Pearson Correlation test on the results of MCS we can infer that the most relevant parameter for the DRS FIT rate is the HDS as shown in Fig. 6.5. This finding is also backed up by the results in Table 6.3 showing a very close value of the FIT from DRS to the FIT of HDS. Additionally these findings

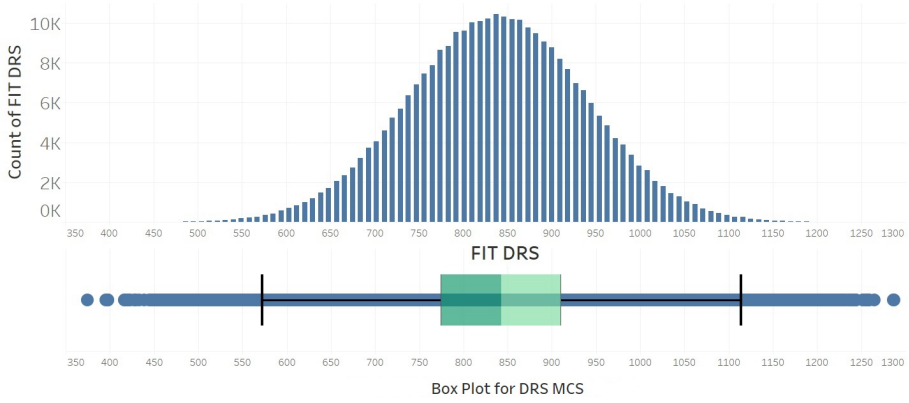


Figure 6.4: MCS histogram distribution and Box plot for DRS.

suggest that this custom DRS architecture might have a potential CCF on the HDS as exposed on section 4.3.2. Improvements on the DRS can be achieved by improving the HDS FIT rate by lowering its elements FIT rate. This could be achieved by either duplicate the HDS system under each DSA as traditional approach or transfer controlling functionalities of the HDS to the PL as an approach towards centralized architecture such as the approach used on the DSA presented in [16].

6.1.2 Scenario 2: Pessimistic Approach on DSA

The resulting reliability function $R_{DSA_2}(t)$ and $R_{DRS_2}(t)$ for this scenario are shown in Fig. 6.6. Both graphs are compared with the reference scenario for DSA R_{DSA_1} and DRS R_{DRS_1}. Through visual inspection it is possible to infer a deterioration on DSA reliability. However the impact is not significant on the DRS when comparing to the reference scenario.

The results for this scenario are summarized in Table 6.4 where the scenario 2.1 presents the results for DSA and 2.2 for the DRS. We can infer an increase of $\approx$ 180% of the DSA FIT rate whereas the FIT for DRS only increases by $\approx$ 1.3%. Both systems maintain their ASIL levels and reliability at 15 years which in turn devalue the importance of MTTF on this scenario.

6.1.3 Scenario 3: Optimization for the HDS system

The resulting reliability for R_{HDS_2} and R_{DRS_2} for the third scenario are presented in Fig. 6.7 and compared with the reference scenario of R_{HDS_1} and DRS R_{DRS_1}. As expected the improvement of the HDS to ASIL B/C values improves dramatically the reliability in both systems model as perceived on the visual inspection of both graphs.

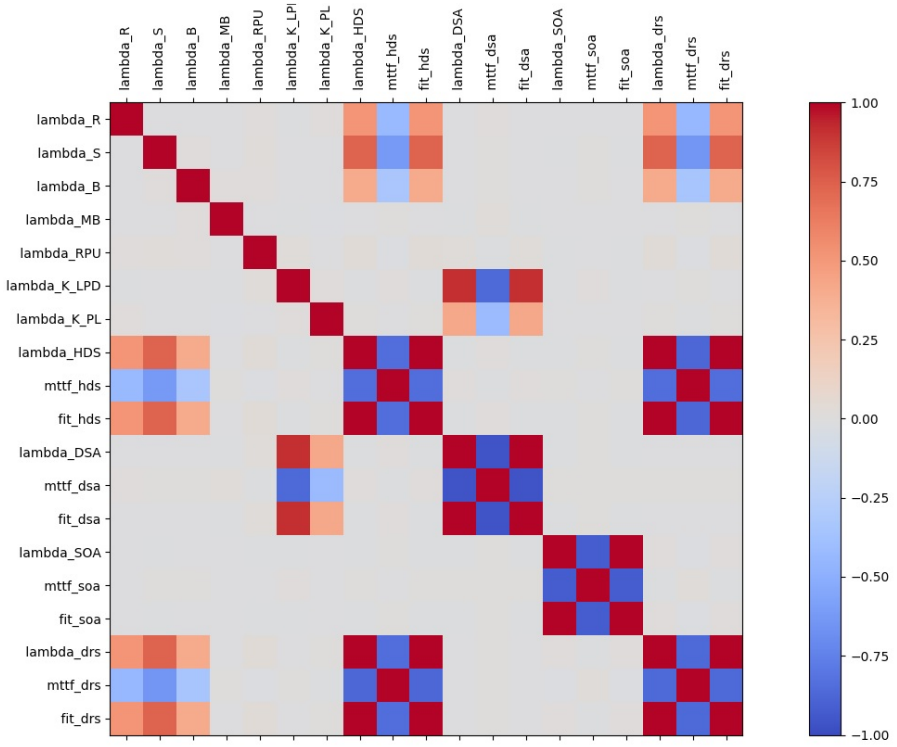


Figure 6.5: MCS correlation graph.

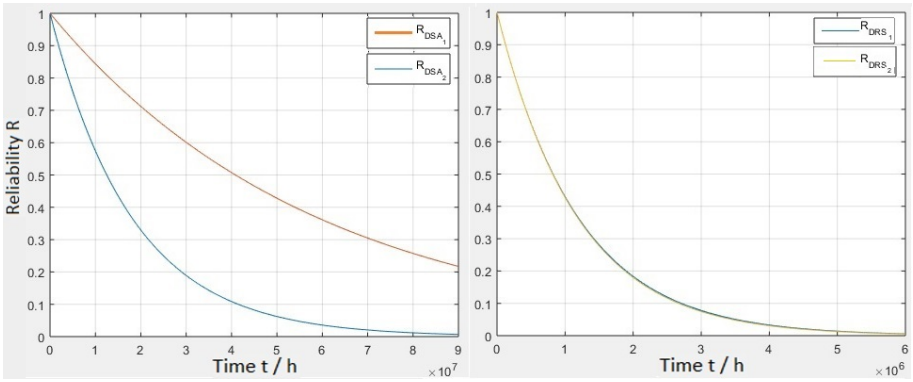


Figure 6.6: a) DSA reliability b) DRS reliability.

The results for this scenario are summarized in Table 6.5. The results for the HDS sub-system are evidently improved in all aspects being the most relevant the improvement on the FIT rate and therefore the improvement from ASIL A to ASIL B/C. Despite these improvements the reliability for 15 years only increases by 1% when comparing with reference values of the HDS. However, the

Table 6.4: Scenario 2: quantitative results for DSA and DRS System.

Scenario	2.1 - Pessimistic Approach - DSA				
Parameters	MTTF	$R(15years)$	λ_{DSA}	FIT	ASIL
Values	$1.80 \times 10^7 h$	0.995	5.55×10^{-8}	55.54	B/C
Scenario	2.2 - Pessimistic Approach - DRS				
Parameters	MTTF	$R(15years)$	λ_{DRS}	FIT	ASIL
Values	$1.16 \times 10^6 h$	0.905	8.59×10^{-7}	859.26	A

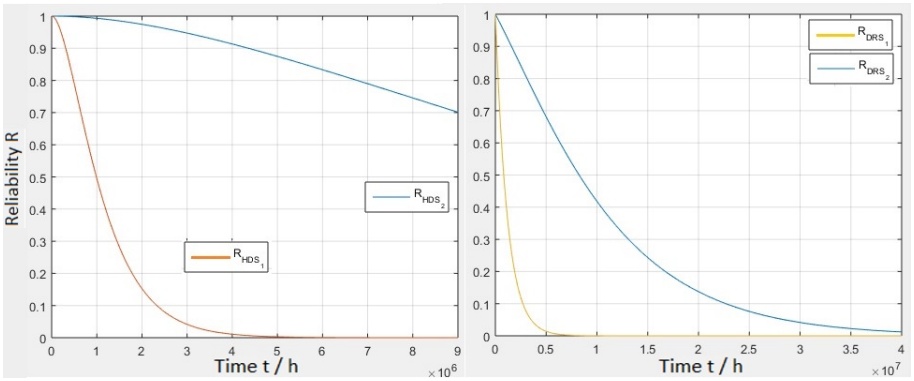


Figure 6.7: a) HDS reliability b) DRS reliability.

bigger improvements occur on the DRS where additionally to the ASIL and FIT improvement is the increase of the $R(15years)$ to 99% from 90% of the reference scenario.

Table 6.5: Scenario 3: quantitative results for HDS and DRS System.

Scenario	3.1 HDS Optimization - HDS				
Parameters	MTTF	$R(15years)$	λ_{DSA}	FIT	ASIL
Values	$1.66 \times 10^7 h$	0.99	6.00×10^{-8}	60	B/C
Scenario	3.2 HDS Optimization - DRS				
Parameters	MTTF	$R(15years)$	λ_{DRS}	FIT	ASIL
Values	$1.06 \times 10^7 h$	0.993	9.44×10^{-8}	94.37	B/C

The MCS for this scenario shows a Normal distribution for the data considering the uncertainties on the methods used to obtain block FIT rate. The distribution is presented in Fig. 6.8. The median and mean values are respectively $\tilde{x} \approx \bar{x} = 94$ and the standard deviation $\sigma = 9.13$ with a 95% confidence interval. The data dispersion for Q_1 and Q_3 shows that the overall FIT rate is

located on the interval $[87, 100]$.

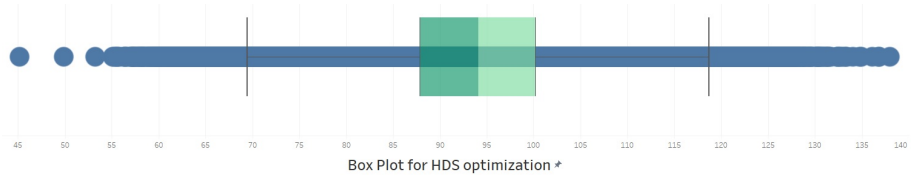


Figure 6.8: MCS histogram distribution and box plot for DRS.

When using the Pearson correlation to analyse the MCS for this scenario, it is possible to infer the SOA has the most significant importance to the DRS FIT as shown in Figure 6.9 despite higher HDS FIT. This suggests the SOA sub-system as possible improvement key point leading to the next scenario.

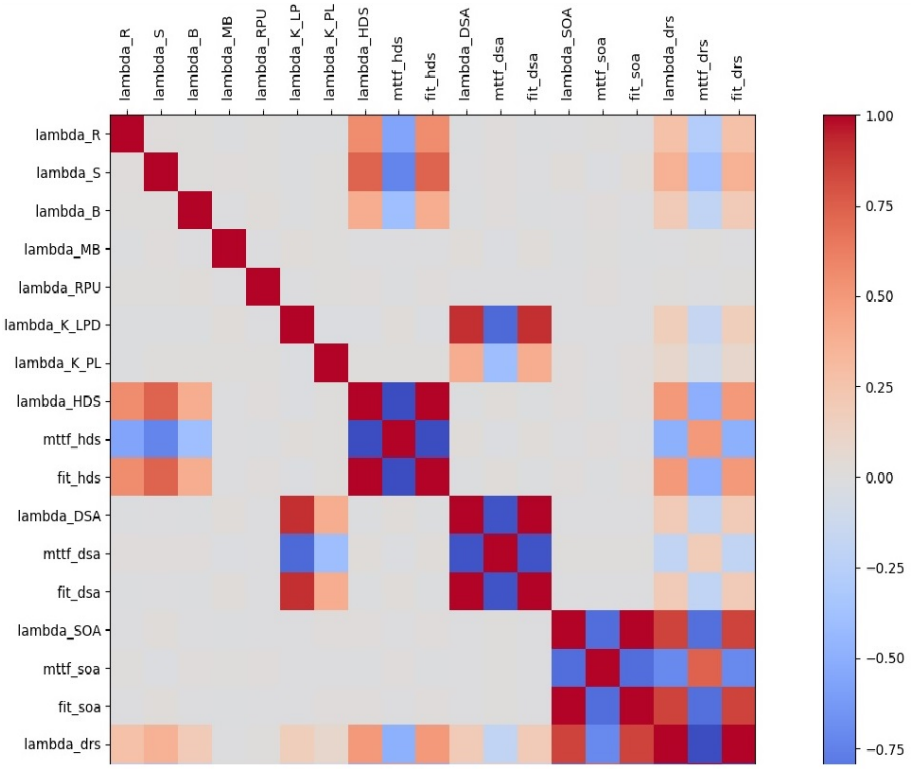


Figure 6.9: MCS correlation graph.

6.1.4 Scenario 4: Optimization for the SOA system

The resulting reliability of R_{DRS_2} for this scenario is presented in Fig. 6.10. The results for the DRS are summarised in Table 6.6. It improves the DRS FIT

rate by $\approx$ 20% when comparing to the previous scenario as presented in Table 6.5-3.2. Despite the improvement of SOA system to ASIL D (FIT rate improvement from 50 to 7), it does not have the desired impact on the overall DRS system of reducing from ASIL B to ASIL D as suggested on ASIL decomposition proposal in section 4.3.1.3. Further scenarios, optimizations, redundancy schemes must be considered in order to achieve ASIL D on the DRS perspective.

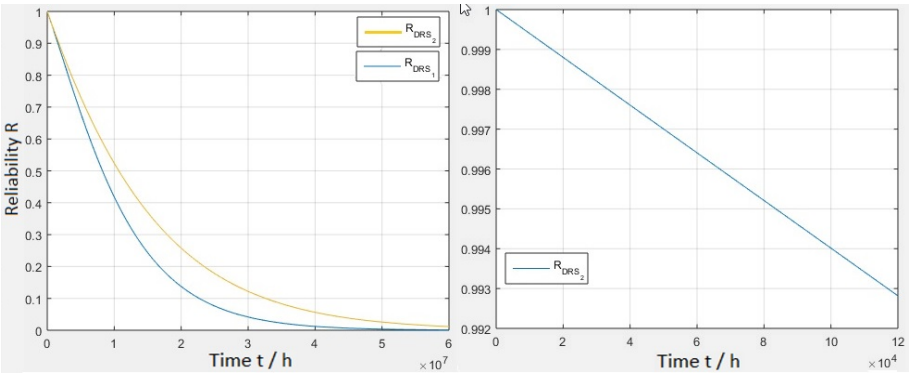


Figure 6.10: a) HDS Reliability b) DRS Reliability.

Table 6.6: Scenario 4: quantitative results for DRS System for SOA and HDS Optimization.

Scenario	4 SOA and HDS Optimization				
Parameters	MTTF	$R(15years)$	λ_{DSA}	FIT	ASIL
Values	$1.45 \times 10^7 h$	0.99	6.87×10^{-8}	68.71	B/C

6.1.5 Scenario 5: Optimization for PL area and Clock

The results for this optimization do not present any relevance on the improvement of the FIT of DSA or DRS. This is due to the low expression of the implementation ($\approx$ 2% of the PL is used). Therefore the overall improvement is also low ($\approx$ 1%) when comparing both optimizations in terms of used area on the PL. However the optimization scenario might be considered if the implementation is expanded to others functionalities or further redundant functions are integrated on the PL where the unused area of the PL might need to be considered as well.

6.2 Discussion of Results

From the results we can infer that the HDS is the system with more importance on the overall DRS system FIT and ASIL quantification. Furthermore, the quantitative findings from the Markov chain model on the reference scenario combined with the MCS Pearson correlation analysis impose some questions regarding the DRS custom architecture hierarchy. The DRS is a custom architecture encompassing high and low-level redundancy as exposed in Fig. 2.4 on section hardware redundancy section 2.3.1 and further scenarios and alternative designs can be derived from this analysis. Additionally, the findings suggest that CCF analysis proposed in 4.3.2 should be performed in order to evaluate the importance of the HDS on this analysis mode and rule out design weaknesses.

In comparison with other architectures such as the *K-out-of-N* presented in [31], the DRS can be altered to adopt a symmetric redundancy on the HDS point of view and lower the HDS on the FTA hierarchy. This could be achieved by either duplicate the HDS system under each DSA as traditional approach or transfer controlling functionalities of the HDS to the PL as an approach towards centralized architecture such as the approach used on the DSA presented in [16].

From scenario 2 we can infer that the pessimistic approach on the DSA do not have impact on the overall DRS reliability. Combining this finding with results from scenario 5, it reinforces the robustness of the DSA in terms of FIT rate and possibilities to encompass further redundancy and other functionalities safely. The trade off between PL usage area for deploying functionalities directly on the hardware can be very valuable being the HDS control functionalities a relevant candidate.

On optimization scenarios (3 and 4) the reliability for 15 years ($R(15 years)$ presents promising results showing values above 98%. However, the scenario 4 of the SOA optimization did not produced the expected improvements in terms of FIT rate.

Apart from the MCS, Person Correlation and Markov model methods used to derive alternative scenarios, it is also used considerations regarding CCF and ASIL decomposition as seen through the text. The findings suggest that ASIL decomposition is clearly insufficient to design ASIL D compliant applications.

The data dispersion for the MCS simulation shows that the uncertainties do not have significant influence on the overall FIT rate of the DRS on thethe proposed scenarios. However, scenario 3 shows a Q_3 near 100 which is the FIT border between ASIL B/C and ASIL A.

Chapter 7

Conclusion and Future Work

In this work, it is presented a methodology to evaluate dynamically reconfigurable systems according to the ISO 26262. The methodology consists of three stages: The Failure Tree Analysis, the Markov chain, and the Monte Carlo simulation combined with statistical analysis. With this model, we extend the traditional approach of FMEDA to a dynamic modeling and safety analysis using the Markov chain.

With the proposed evaluation methodology, it is possible to describe and assess quantitatively a dynamically reconfigurable system in terms of reliability and obtain its respective mathematical model. The evaluation method represents an incremental approach to system design allowing to be incorporated into CoDesign and Agile methodologies as well as on traditional approaches. It is possible to modularize and re-use each sub-system model into new designs allowing the recombination of the system architecture. This represents an advantage when assessing trade-offs between the safety and placement of different functionalities into different hardware components (e.g, implement functionalities in the PL rather than through software or different computational units) with performance and timing constraints.

The MCS allows us to quantify and incorporate uncertainties resulting from the different E/E suppliers and FMEDA methods into the model. The findings obtained through Pearson correlation are useful to get insights regarding key indicators related to the functional safety. In ASIL A (100 to 1000 FIT) the MCS might not be of interest but at a smaller scale at ASIL D (1 to 10 FIT), where the system is increasingly complex, the MCS combined with the statistical data distribution and dispersion evaluation can be valuable to get insights of key elements to improve and where redundancy must be applied. Regarding the case study scenarios, a future improvement is to include an alternative architecture

implementation (e.g study the feasibility to include control functions into the PL and assess the reliability through the area utilization, leaving the HDS system only with hardware interfaces).

As future improvements, it is also recommended to include evaluation in terms of SPFM and LFM in order to differentiate ASIL B and C and deepen the CCF analysis as exposed in section 4.3.2. Incorporation of WCET explained in section 3.3.2 into the evaluation is especially useful to asses optimizations on the PL or further design changes in terms of architecture to certify that timing constraints are met. Related to timing evaluation it is also advantageous to include into the model the repair rates and perform evaluation in terms of reconfiguration latency and MTTR.

Overall, the automotive case study presented was successfully applied to the proposed evaluation methodology. An ASIL A level was obtained for the case study, possible design weaknesses identified and improving points were proposed. The main conclusion is that the presented methodology is suitable for dynamically reconfigurable systems and can be extended to other approaches as well due to its dynamic and iterative characteristic. Finally, to achieve ASIL D compliant systems is very demanding, and continuous improvements and implementations need to be considered.

References

[1] P. Mallozzi, P. Pelliccione, A. Knauss, C. Berger, and N. Mohammadiha, "Autonomous vehicles: State of the art, future trends, and challenges," in *Automotive systems and software engineering* (Y. Dajsuren and M. van den Brand, eds.), vol. 57, pp. 347–367, New York NY: Springer Berlin Heidelberg, 2019. [Quoted on p. v, 2, 38, 40, 41, 42, 43]

[2] F. Oszwald, J. Becker, P. Obergfell, and M. Traub, "Dynamic reconfiguration for real-time automotive embedded systems in fail-operational context," in *2018 IEEE 32nd International Parallel and Distributed Processing Symposium Workshops*, (Los Alamitos, California), pp. 206–209, IEEE Computer Society, Conference Publishing Services, 2018. [Quoted on p. v, vi, 1, 2, 3, 56]

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004. [Quoted on p. v, 7, 9, 13, 14, 15, 16]

[4] E. Dubrova, *Fault-Tolerant Design*. New York, NY: Springer New York, 2013. [Quoted on p. v, vi, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 73]

[5] C. H. Yi, K. Kwon, and J. W. Jeon, "Method of improved hardware redundancy for automotive system," in *2014 14th International Symposium on Communications and Information Technologies (ISCIT)*, (Piscataway, NJ), pp. 204–207, IEEE, 2014. [Quoted on p. v, 19]

[6] M. H. Kim, S. Lee, and K. C. Lee, "Kalman predictive redundancy system for fault tolerance of safety-critical systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 1, pp. 46–53, 2010. [Quoted on p. v, 28]

[7] V. B. Prasad, "Fault tolerant digital systems," *IEEE Potentials*, vol. 8, no. 1, pp. 17–21, 1989. [Quoted on p. v, 1, 10, 17, 18, 19, 27, 28, 29, 43]

[8] J. Becker and M. Hübner, "Dynamic reconfiguration," in *Designing Embedded Processors* (J. Henkel and S. Parameswaran, eds.), pp. 503–512, Dordrecht: Springer Netherlands, 2007. [Quoted on p. v, 30, 32, 33]

[9] D. Göhringer, M. Hübner, and J. Becker, "Adaptive multiprocessor system-on-chip architecture: New degrees of freedom in system design and runtime support," in *Multiprocessor system-on-chip* (M. Hübner and J. Becker, eds.), vol. 4, pp. 127–151, New York and London: Springer, 2011. [Quoted on p. v, 31, 44]

[10] A. Knoll, C. Buckl, K.-J. Kuhn, and G. Spiegelberg, "The race project: An informatics-driven greenfield approach to future e/e architectures for cars," in *Automotive systems and software engineering* (Y. Dajsuren and M. van den Brand, eds.), pp. 171–195, New York NY: Springer Berlin Heidelberg, 2019. [Quoted on p. v, 35, 36, 38]

[11] L. Wu and Y. Sun, "Guaranteed security and trustworthiness in transportation cyber-physical systems," in *Secure and trustworthy transportation cyber-physical systems* (Y. Sun and H. Song, eds.), vol. 26 of *SpringerBriefs in Computer Science*, pp. 3–22, Singapore: Springer, 2017. [Quoted on p. v, 36, 37]

[12] C.-W. Lin, B. Zheng, H. Liang, and Q. Zhu, "Platform-based design for automotive and transportation cyber-physical systems," in *Design Automation of Cyber-Physical Systems* (M. A. A. Faruque and A. Canedo, eds.), vol. 65, pp. 21–40, Cham, Switzerland: Springer, 2019. [Quoted on p. vi, 39, 40, 42, 43]

[13] D. Zerfowski and A. Lock, "Functional architecture and e/e-architecture – a challenge for the automotive industry," in *19. Internationales Stuttgarter Symposium* (M. Bargende, H.-C. Reuss, and J. Wiedemann, eds.), Proceedings, pp. 909–920, Wiesbaden: Springer Fachmedien Wiesbaden GmbH and Springer Vieweg, 2019. [Quoted on p. vi, 1, 38, 40, 41]

[14] Stefan Kugele, David Hettler, Sina Shafaei, "Elastic service provision for intelligent vehicle functions," *2018 IEEE Intelligent Transportation Systems Conference*, 2018. [Quoted on p. vi, 33, 40, 42]

[15] J. Teich, "Hardware/software codesign: The past, the present, and predicting the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1411–1430, 2012. [Quoted on p. vi, 2, 4, 23, 27, 33, 36, 39, 44, 47, 52, 53]

[16] T. Dörr, T. Sandmann, F. Schade, F. K. Bapp, and J. Becker, "Leveraging the partial reconfiguration capability of fpgas for processor-based fail-operational systems," in *Applied reconfigurable computing* (C. Hochberger, ed.), vol. 11444 of *LNCS sublibrary: SL1 - Theoretical computer science and*

general issues, pp. 96–111, Cham, Switzerland: Springer, 2019. [Quoted on p. vi, 8, 28, 57, 58, 59, 84, 89]

[17] Xilinx, *Zynq UltraScale+ Device Technical Reference Manual (UG1085)*. January 17, 2019. [Quoted on p. vi, 45, 58, 59]

[18] A. Frigerio, B. Vermeulen, and K. Goossens, "A generic method for a bottom-up asil decomposition," in *Developments in language theory* (M. Hoshi and S. Seki, eds.), vol. 11088 of *LNCS sublibrary. SL 1, Theoretical computer science and general issues*, pp. 12–26, Cham, Switzerland: Springer, 2018. [Quoted on p. vi, 70, 71]

[19] A. Birolini, *Reliability Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. [Quoted on p. vi, 10, 12, 13, 49, 73]

[20] E. Verhulst, B. Sputh, and V. de Florio, "From safety integrity level to assured reliability and resilience level for compositional safety critical systems," *ICSSEA 2013 - International Conference on Simulation Software Engineering and Applications*, 2013. [Quoted on p. ix, 48, 68, 69, 70]

[21] W. Granig, D. Hammerschmidt, and H. Zangl, "Calculation of failure detection probability on safety mechanisms of correlated sensor signals according to iso 26262," *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, vol. 10, no. 1, pp. 144–155, 2017. [Quoted on p. ix, 69, 70]

[22] I. Krüger, "Rich services — a soa pattern for dynamic change in cyber-physical systems," *it - Information Technology*, vol. 55, no. 1, pp. 10–18, 2013. [Quoted on p. 1, 37, 40]

[23] Y. Gheraibia, S. Kabir, K. Djafri, and H. Krimou, "An overview of the approaches for automotive safety integrity levels allocation," *Journal of Failure Analysis and Prevention*, vol. 18, no. 3, pp. 707–720, 2018. [Quoted on p. 3, 70]

[24] M. S. Farias, N. Nedjah, and P. V. R. de Carvalho, "Resilient hardware design for critical systems," in *2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS)* (R. S. Murphy, ed.), ([Piscataway, New Jersey]), pp. 237–240, IEEE, 2019. [Quoted on p. 7, 8, 10, 15, 19]

[25] R. Isermann, "Fault-tolerant components for automatic driving automobiles – some basic structures and examples," in *Fahrerassistenzsysteme 2016* (R. Isermann, ed.), Proceedings, pp. 209–234, Wiesbaden: Springer Fachmedien Wiesbaden GmbH and Springer Vieweg, 2018. [Quoted on p. 7, 18, 27, 28]

[26] Christof Fetzer, "Fail-awareness: An approach to construct fail-safe systems," *The International Journal of Time-Critical Computing Systems*, no. 2, pp. 203–238, 2003. [Quoted on p. 8]

[27] T. Saridakis, "Design patterns for graceful degradation," in *Transactions on pattern languages of Programming I* (J. Noble and R. E. Johnson, eds.), vol. 5770 of *Lecture notes in computer science. Journal subline*, pp. 67–93, Berlin and London: Springer, 2009. [Quoted on p. 8]

[28] J.-C. Laprie, "Dependable computing and fault tolerance : Concepts and terminology," in *The Twenty-fifth International Symposium on Fault-Tolerant Computing*, (Los Alamitos Calif.), p. 2, IEEE Computer Society Press, 1995. [Quoted on p. 8]

[29] L. Xing and S. V. Amari, "Fault tree analysis," in *Handbook of performability engineering* (K. B. Misra, ed.), vol. 49, pp. 595–620, Berlin and London: Springer, 2008. [Quoted on p. 11, 48, 50]

[30] A. Sundararajan and R. Selvarani, "Case study of failure analysis techniques for safety critical systems," in *Advances in computer science, engineering and applications* (D. C. Wyld, J. Zizka, and D. Nagamalai, eds.), vol. 166 of *Advances in Intelligent and Soft Computing*, pp. 367–377, Berlin and New York: Springer, 2012. [Quoted on p. 11]

[31] A. Kohn, R. Schneider, A. Vilela, U. Dannebaum, and A. Herkersdorf, "Markov chain-based reliability analysis for automotive fail-operational systems," *SAE International Journal of Transportation Safety*, vol. 5, no. 1, pp. 30–38, 2017. [Quoted on p. 12, 19, 70, 89]

[32] J. Blieberger and M. Bader, *Reliable Software Technologies – Ada-Europe 2017*, vol. 10300. Cham: Springer International Publishing, 2017. [Quoted on p. 14, 51]

[33] A. Cobreces, J. Tabero, A. Regadio, A. Sanchez-Macian, P. Reviriego, and J. A. Maestro, "Seu and sefi protection for ddr3 memories in a xilinx zynq-7000 fpga," in *6th IEEE International Conference on Space Mission Challenges for Information Technology*, (Los Alamitos, California), pp. 151–153, IEEE Computer Society, Conference Publishing Services, 2017. [Quoted on p. 15]

[34] B. Volochiy, V. Yakovyna, O. Mulyak, and V. Kharchenko, "Availability model of critical nuclear power plant instrumentation and control system with non-exponential software update distribution," in *Information and Communication Technologies in Education, Research, and Industrial Applications* (N. Bassiliades, V. Ermolayev, H.-G. Fill, V. Yakovyna, H. C. Mayr, M. Nikitchenko, G. Zholtkevych, and A. Spivakovsky, eds.), vol. 826 of *Communications in Computer and Information Science, 1865-0929*, pp. 3–20, Cham: Springer, 2018. [Quoted on p. 15]

[35] Mario Schölzel, *Self-Testing and Self-Repairing Embedded Processors: Techniques for Statically Scheduled Superscalar Architectures*. Doctor rerum naturalium habilitatus, Faculty of Mathematics, Natural Sciences and Computer

Science of the Brandenburg University of Technology Cottbus-Senftenberg, 2014. [Quoted on p. 16, 17, 21, 22, 23, 26, 33]

[36] S. Razvan and S. Csaba, "Software redundancy implementation strategy in reconfigurable hardware framework," in *2019 8th International Conference on Modern Power Systems (MPS)*, pp. 1–6, IEEE, 21/05/2019 - 23/05/2019. [Quoted on p. 23, 24, 25]

[37] E. Ertugrul and O. K. Sahingoz, "Fault tolerance in real-time systems: A review," in *Intelligent systems design and applications* (A. Abraham, P. K. Muhuri, A. K. Muda, and N. Gandhi, eds.), vol. 736 of *Advances in intelligent systems and computing, 2194-5357*, pp. 283–293, Cham, Switzerland: Springer, 2018. [Quoted on p. 26, 27]

[38] Oxford Dictionary, "Definition of reconfiguration in english: reconfiguration." [Quoted on p. 29]

[39] M. D'Souza and R. N. Kashi, "Avionics self-adaptive software: Towards formal verification and validation," in *Distributed computing and internet technology* (G. Fahrnberger, S. Gopinathan, and L. Parida, eds.), vol. 11319 of *LNCS sublibrary. SL 3, Information systems and applications, incl. Internet/web, and HCI*, pp. 3–23, Cham, Switzerland: Springer, 2019. [Quoted on p. 29]

[40] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, 2002. [Quoted on p. 29, 32]

[41] M. G. Gericota, L. F. Lemos, G. R. Alves, M. M. Barbosa, and J. M. Ferreira, "A framework for fault tolerant real time systems based on reconfigurable fpgas," in *2006 IEEE Conference on emerging technologies and factory automation*, (Piscataway NJ), pp. 131–138, IEEE, 2003. [Quoted on p. 29, 45]

[42] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable computing architectures," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 332–354, 2015. [Quoted on p. 29]

[43] K. Vipin and S. A. Fahmy, "Fpga dynamic and partial reconfiguration," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–39, 2018. [Quoted on p. 30, 31, 32, 33]

[44] M. Koenen, N. A. V. Doan, T. Wild, and A. Herkersdorf, "A hybrid noc enabling fail-operational and hard real-time communication in mpsoc," *Lecture Notes in Computer Science*, vol. 11479, pp. 31–44, 2019. [Quoted on p. 33, 43, 45]

[45] M. Platzner, "Reconfigurable hardware operating systems," in *2005 IEEE international conference on field programmable technology*, (New York City NY), p. xxi, IEEE, 2005. [Quoted on p. 33]

[46] L. Torres, P. Benoit, G. Sassatelli, M. Robert, F. Clermidy, and D. Puschini, "An introduction to multi-core system on chip – trends and challenges," in *Multiprocessor system-on-chip* (M. Hübner and J. Becker, eds.), vol. 27, pp. 1–21, New York and London: Springer, 2011. [Quoted on p. 35, 43]

[47] J. Fitzgerald, P. G. Larsen, and M. Verhoef, "From embedded to cyber-physical systems: Challenges and future directions," in *Collaborative design for embedded systems* (J. Fitzgerald, P. G. Larsen, and M. Verhoef, eds.), vol. 138, pp. 293–303, New York: Springer, 2014. [Quoted on p. 36, 37]

[48] D. Zerfowski and J. Crepin, "Vehicle computers - automotive software development rethought," *ATZelectronics worldwide*, vol. 14, no. 7-8, pp. 36–41, 2019. [Quoted on p. 38, 39]

[49] A. Bertolino, A. Calabro', F. Di Giandomenico, G. Lami, F. Lonetti, E. Marchetti, F. Martinelli, I. Matteucci, and P. Mori, "A tour of secure software engineering solutions for connected vehicles," *Software Quality Journal*, vol. 26, no. 4, pp. 1223–1256, 2018. [Quoted on p. 39, 46]

[50] J. Jatzkowski, M. Kreutz, and A. Rettberg, "Hierarchical multicore-scheduling for virtualization of dependent real-time systems," in *System level design from HW* (M. Götz, G. Schirner, M. A. Wehrmeister, M. A. Al Faruque, and A. Rettberg, eds.), vol. 523 of *IFIP advances in information and communication technology, 1868-4238*, pp. 103–115, Cham, Switzerland: Springer, 2017. [Quoted on p. 39, 43]

[51] M. Hübner and J. Becker, eds., *Multiprocessor system-on-chip: Hardware design and tool integration / Michael Hübner, Jürgen Becker, editors*. New York and London: Springer, 2011. [Quoted on p. 42, 43]

[52] S. Avramenko and M. Violante, "Rtos solution for noc-based cots mpsoc usage in mixed-criticality systems," *Journal of Electronic Testing*, vol. 35, no. 1, pp. 29–44, 2019. [Quoted on p. 43, 45, 46, 68]

[53] J. M. P. Cardoso, P. C. Diniz, J. G. de Figueiredo Coutinho, and Z. M. Petrov, *Compilation and synthesis for embedded reconfigurable systems: An aspect-oriented approach*. New York, NY: Springer, 2013. [Quoted on p. 43, 44, 46]

[54] S. P. Azad, G. Jervan, and J. Sepulveda, "Dynamic and distributed security management for noc based mpsocs," in *Computational science – ICCS 2019* (J. M. F. Rodrigues, ed.), vol. 11537 of *LNCS sublibrary: SL1 - Theoretical computer science and general issues*, pp. 649–662, Cham, Switzerland: Springer, 2019. [Quoted on p. 44]

[55] F. Lima, L. Carro, and R. Reis, "Designing fault tolerant systems into sram-based fpgas," in *Proceedings of the 40th design automation conference* (I. Getreu, L. Fix, and L. Lavagno, eds.), (New York, New York, USA), p. 650, ACM Press, 2003. [Quoted on p. 45]

[56] E. Schoitsch, *Computer safety, reliability, and security: 29th international conference, SAFECOMP 2010, Vienna, Austria, September 14-17, 2010, proceedings / [edited by] Erwin Schoitsch*, vol. 6351 of *Lecture notes in computer science, 0302-9743*. Berlin: Springer, 1st ed. ed., 2010. [Quoted on p. 47, 51]

[57] Q. Wang, J. Mao, and H.-y. Wei, "Reliability analysis of multi-rotor uav based on fault tree and monte carlo simulation," in *Advances in Mechanical Design* (J. Tan, F. Gao, and C. Xiang, eds.), vol. 55 of *Mechanisms and Machine Science*, pp. 1525–1534, Singapore: Springer, 2017. [Quoted on p. 48, 50]

[58] Z. Chiremsel, R. Nait Said, and R. Chiremsel, "Probabilistic fault diagnosis of safety instrumented systems based on fault tree analysis and bayesian network," *Journal of Failure Analysis and Prevention*, vol. 16, no. 5, pp. 747–760, 2016. [Quoted on p. 48]

[59] N. Kronprasert and N. Thipnee, "Use of evidence theory in fault tree analysis for road safety inspection," in *Belief functions* (J. Vejnarová and V. Kratochvíl, eds.), vol. 9861 of *LNCS sublibrary. SL 7, Artificial intelligence*, pp. 84–93, Switzerland: Springer, 2016. [Quoted on p. 49]

[60] O. Bäckström, Y. Butkova, H. Hermanns, J. Krčál, and P. Krčál, "Effective static and dynamic fault tree analysis," in *Computer safety, reliability, and security* (A. Skavhaug, J. Guiochet, and F. Bitsch, eds.), vol. 9922 of *LNCS sublibrary. SL 2, Programming and software engineering*, pp. 266–280, Switzerland: Springer, 2016. [Quoted on p. 49]

[61] A. Carminati, R. A. Starke, and R. S. de Oliveira, "On the use of static branch prediction to reduce the worst-case execution time of real-time applications," *Real-Time Systems*, vol. 54, no. 3, pp. 537–561, 2018. [Quoted on p. 50, 51]

[62] C. Zoubek and P. Trommler, "Overview of worst case execution time analysis," *30th GI/ITG International Conference on Architecture of Computing Systems*, 2017. [Quoted on p. 50, 51, 52]

[63] S. Basagiannis and F. Gonzalez-Espin, "Towards verification of multicore motor-drive controllers in aerospace," in *Computer safety, reliability, and security* (F. Koornneef and C. van Gulijk, eds.), vol. 9338 of *LNCS sublibrary. SL 2, Programming and software engineering*, pp. 190–200, Cham: Springer, 2015. [Quoted on p. 62]

[64] Xilinx, *Device Reliability Report - First Half 2019 - UG116 (v10.11)*. September 18, 2019. [Quoted on p. 62]

[65] Xilinx, *Isolation Methods in Zynq UltraScale+ MPSoCs (XAPP1320)*. June 21, 2019. [Quoted on p. 72]

[66] Xilinx, *Vivado Isolation Verifier (UG1291)*. August 10, 2018. [Quoted on p. 72]

Chapter 8

Annex A

$$P\_1(t) = e^{-t\,(\lambda_B+\lambda_R+\lambda_S)}$$

$$P\_2(t) = \frac{e^{-t\,(\lambda_R+\lambda_S)}\,(\lambda_B+\lambda_S)}{\lambda_B} - \frac{\lambda_S\, e^{-t\,(\lambda_R+\lambda_S)}}{\lambda_B} - e^{-t\,(\lambda_B+\lambda_R+\lambda_S)}$$

$$P\_3(t) = e^{-t\,(\lambda_B+\lambda_S)} - e^{-t\,(\lambda_B+\lambda_R+\lambda_S)}$$

$$P\_4(t) = \frac{\lambda_S\, e^{-t\,(\lambda_R+\lambda_S)}}{\lambda_B} - \frac{\lambda_S\, e^{-t\,(\lambda_B+\lambda_R+\lambda_S)}}{\lambda_B}$$

$$P\_5(t) = \frac{e^{-t\,(\lambda_B+\lambda_R+\lambda_S)}\,(\lambda_B+\lambda_S)}{\lambda_B} - e^{-t\,(\lambda_B+\lambda_S)} - \frac{e^{-t\,(\lambda_R+\lambda_S)}\,(\lambda_B+\lambda_S)}{\lambda_B} + 1$$

Figure 8.1: Probabilities functions for HDS.

$$R\_HDS1(t) = e^{-t\,(\lambda_B+\lambda_S)} - e^{-t\,(\lambda_B+\lambda_R+\lambda_S)} - \frac{\lambda_S\, e^{-t\,(\lambda_B+\lambda_R+\lambda_S)}}{\lambda_B} + \frac{e^{-t\,(\lambda_R+\lambda_S)}\,(\lambda_B+\lambda_S)}{\lambda_B}$$

Figure 8.2: Reliability function for HDS.

MTTF =

$$\frac{\sigma_1}{\lambda_B}+\frac{1}{\lambda_B+\lambda_S}+\left(\lim_{t\to\infty}-\frac{e^{-t\lambda_B-t\lambda_S}}{\lambda_B+\lambda_S}-\frac{e^{-t\lambda_B-t\lambda_R-t\lambda_S}\sigma_1}{\lambda_B}-\frac{e^{-t\lambda_R-t\lambda_S}(\lambda_B+\lambda_S)}{\lambda_B(\lambda_R+\lambda_S)}\right)+\frac{\lambda_B+\lambda_S}{\lambda_B(\lambda_R+\lambda_S)}$$

where

$$\sigma_1=\frac{\lambda_R}{\lambda_B+\lambda_R+\lambda_S}-1$$

Figure 8.3: MTTF function for HDS.

$$P\_1(t) = e^{-t(\lambda_K+\lambda_{MB}+\lambda_{RPU})}$$

$$P\_2(t) = e^{-t(\lambda_K+\lambda_{RPU})} - e^{-t(\lambda_K+\lambda_{MB}+\lambda_{RPU})}$$

$$P\_3(t) = e^{-t(\lambda_K+\lambda_{MB})} - e^{-t(\lambda_K+\lambda_{MB}+\lambda_{RPU})}$$

$$P\_4(t) = e^{-t(\lambda_K+\lambda_{MB}+\lambda_{RPU})} - e^{-t(\lambda_K+\lambda_{RPU})} - e^{-t(\lambda_K+\lambda_{MB})} + 1$$

Figure 8.4: Probabilities functions for DSA.

$$R\_DSA1(t) = e^{-t(\lambda_K+\lambda_{MB})} + e^{-t(\lambda_K+\lambda_{RPU})} - e^{-t(\lambda_K+\lambda_{MB}+\lambda_{RPU})}$$

Figure 8.5: Reliability function for DSA.

MTTF =

$$-\frac{1}{\lambda_K+\lambda_{MB}+\lambda_{RPU}}+\left(\lim_{t\to\infty}-\frac{e^{-t\lambda_K}e^{-t\lambda_{MB}}}{\lambda_K+\lambda_{MB}}-\frac{e^{-t\lambda_K}e^{-t\lambda_{RPU}}}{\lambda_K+\lambda_{RPU}}+\frac{e^{-t\lambda_K}e^{-t\lambda_{MB}}e^{-t\lambda_{RPU}}}{\lambda_K+\lambda_{MB}+\lambda_{RPU}}\right)+\frac{1}{\lambda_K+\lambda_{MB}}+\frac{1}{\lambda_K+\lambda_{RPU}}$$

Figure 8.6: MTTF function for DSA.

$$P\_1(t) = e^{-t\,(2\lambda_{DSA}+\lambda_{HDS}+2\lambda_{SOA})}$$

$$P\_2(t) = e^{-t\,(\lambda_{DSA}+\lambda_{HDS}+2\lambda_{SOA})} - e^{-t\,(2\lambda_{DSA}+\lambda_{HDS}+2\lambda_{SOA})}$$

$$P\_3(t) = e^{-t\,(2\lambda_{DSA}+\lambda_{HDS}+\lambda_{SOA})} - e^{-t\,(2\lambda_{DSA}+\lambda_{HDS}+2\lambda_{SOA})}$$

$$P\_4(t) = e^{-t\,(2\lambda_{DSA}+\lambda_{HDS}+\lambda_{SOA})} - e^{-t\,(2\lambda_{DSA}+\lambda_{HDS}+2\lambda_{SOA})}$$

$$P\_5(t) = e^{-t\,(\lambda_{DSA}+\lambda_{HDS}+2\lambda_{SOA})} - e^{-t\,(2\lambda_{DSA}+\lambda_{HDS}+2\lambda_{SOA})}$$

$$P\_6(t) = e^{-t\,(2\lambda_{DSA}+\lambda_{HDS}+2\lambda_{SOA})} - e^{-t\,(2\lambda_{DSA}+\lambda_{HDS}+\lambda_{SOA})} - e^{-t\,(\lambda_{DSA}+\lambda_{HDS}+2\lambda_{SOA})} + e^{-t\,(\lambda_{DSA}+\lambda_{HDS}+\lambda_{SOA})}$$

$$P\_7(t) = e^{-t\,(2\lambda_{DSA}+\lambda_{HDS}+2\lambda_{SOA})} - e^{-t\,(2\lambda_{DSA}+\lambda_{HDS}+\lambda_{SOA})} - e^{-t\,(\lambda_{DSA}+\lambda_{HDS}+2\lambda_{SOA})} + e^{-t\,(\lambda_{DSA}+\lambda_{HDS}+\lambda_{SOA})}$$

$$P\_8(t) =$$

$$\frac{4\lambda_{DSA}^4 + 6\lambda_{DSA}^3\lambda_{HDS} + 12\lambda_{DSA}^3\lambda_{SOA}^2 + 6\lambda_{DSA}^3\lambda_{SOA} + 2\lambda_{DSA}^2\lambda_{HDS}^2 + 8\lambda_{DSA}^2\lambda_{HDS}\lambda_{SOA}^2 + 4\lambda_{DSA}^2\lambda_{HDS}\lambda_{SOA} + 2\lambda_{DSA}^2\lambda_{HDS} + 12\lambda_{DSA}^2\lambda_{SOA}^3 + 4\lambda_{DSA}^2\lambda_{SOA}^2 -}{4\lambda_{DSA}^3 + 8\lambda_{DSA}^2\lambda_{HDS} + 14\lambda_{DSA}^2\lambda_{SOA} + 5\lambda_{DSA}\lambda_{HDS}^2 + 17\lambda_{DSA}\lambda_{HDS}\lambda_{SOA} +}$$

$$\frac{+ 3\lambda_{DSA}\lambda_{HDS}^2 + 4\lambda_{DSA}\lambda_{HDS}\lambda_{SOA}^2 + 5\lambda_{DSA}\lambda_{HDS}\lambda_{SOA} + 6\lambda_{DSA}\lambda_{SOA}^3 + \lambda_{HDS}^3 + 2\lambda_{HDS}^2\lambda_{SOA}^2 + 3\lambda_{HDS}^2\lambda_{SOA} + 6\lambda_{HDS}\lambda_{SOA}^3 + 2\lambda_{HDS}\lambda_{SOA}^2 + 4\lambda_{SOA}^4}{\cdot\ 14\lambda_{DSA}\lambda_{SOA}^2 + \lambda_{HDS}^3 + 5\lambda_{HDS}^2\lambda_{SOA} + 8\lambda_{HDS}\lambda_{SOA}^2 + 4\lambda_{SOA}^3} -$$

$$- 4\lambda_{DSA}\lambda_{SOA}\,e^{-t\,(\lambda_{DSA}+\lambda_{HDS}+\lambda_{SOA})} - \frac{2\,e^{-t\,\sigma_5}\left(\lambda_{DSA}\lambda_{HDS} + \lambda_{DSA}\lambda_{SOA} - \sigma_3 - \sigma_2 + \lambda_{DSA}^2 - 2\lambda_{DSA}\lambda_{HDS}\lambda_{SOA}\right)}{\sigma_5} - \frac{2\,e^{-t\,\sigma_4}\left(\lambda_{DSA}\lambda_{SOA} + \lambda_{HDS}\lambda_{SOA} - \sigma_3 - \sigma_2 + \lambda_{SOA}^2 - 2\lambda_{DSA}\lambda_{HDS}\lambda_{SOA}\right)}{\sigma_4}$$

$$+ \frac{e^{-t\,\sigma_1}\left(2\lambda_{DSA}\lambda_{HDS} - \lambda_{HDS} + 4\lambda_{DSA}\lambda_{SOA} + 2\lambda_{HDS}\lambda_{SOA} - 4\lambda_{DSA}\lambda_{SOA}^2 - 4\lambda_{DSA}^2\lambda_{SOA} + 2\lambda_{DSA}^2 + 2\lambda_{SOA}^2 - 4\lambda_{DSA}\lambda_{HDS}\lambda_{SOA}\right)}{\sigma_1}$$

where $\sigma_1 = 2\lambda_{DSA} + \lambda_{HDS} + 2\lambda_{SOA}$

$\sigma_2 = 2\lambda_{DSA}^2\lambda_{SOA}$ $\sigma_4 = 2\lambda_{DSA} + \lambda_{HDS} + \lambda_{SOA}$

$\sigma_3 = 2\lambda_{DSA}\lambda_{SOA}^2$ $\sigma_5 = \lambda_{DSA} + \lambda_{HDS} + 2\lambda_{SOA}$

Figure 8.7: Probabilities functions for DRS.

$$
\begin{aligned}
R\_DRS(t) = {} & 4\,\lambda_{DSA}\,\lambda_{SOA}\,e^{-t\,(\lambda_{DSA}+\lambda_{HDS}+\lambda_{SOA})} - \frac{4\,\lambda_{DSA}^{4} + 6\,\lambda_{DSA}^{3}\,\lambda_{HDS} + 12\,\lambda_{DSA}^{3}\,\lambda_{SOA}^{2} + 6\,\lambda_{DSA}^{3}\,\lambda_{SOA} + 2\,\lambda_{DSA}^{2}\,\lambda_{HDS}^{2} + 8\,\lambda_{DSA}^{2}\,\lambda_{HDS}\,\lambda_{SOA}^{2} + 4\,\lambda_{DSA}^{2}\,\lambda_{HDS}\,\lambda_{SOA} + 2\,\lambda_{DSA}^{2}\,\lambda_{HDS}^{2}\,\lambda_{SOA}^{3} + 4\,\lambda_{DSA}^{2}\,\lambda_{SOA}^{2}}{4\,\lambda_{DSA}^{3} + 8\,\lambda_{DSA}^{2}\,\lambda_{HDS} + 14\,\lambda_{DSA}^{2}\,\lambda_{SOA} + 5\,\lambda_{DSA}\,\lambda_{HDS}^{2} + 17\,\lambda_{DSA}\,\lambda_{HDS}\,\lambda_{SOA} +} \\
& + \frac{3\,\lambda_{DSA}\,\lambda_{HDS}^{2} + 4\,\lambda_{DSA}\,\lambda_{HDS}\,\lambda_{SOA}^{2} + 5\,\lambda_{DSA}\,\lambda_{HDS}\,\lambda_{SOA} + 6\,\lambda_{DSA}\,\lambda_{SOA}^{3} + \lambda_{HDS}^{3} + 2\,\lambda_{HDS}^{2}\,\lambda_{SOA}^{2} + 3\,\lambda_{HDS}^{2}\,\lambda_{SOA} + 6\,\lambda_{HDS}\,\lambda_{SOA}^{3} + 2\,\lambda_{HDS}\,\lambda_{SOA}^{2} + 4\,\lambda_{SOA}^{4}}{+\,14\,\lambda_{DSA}\,\lambda_{SOA}^{2} + \lambda_{HDS}^{3} + 5\,\lambda_{HDS}^{2}\,\lambda_{SOA} + 8\,\lambda_{HDS}\,\lambda_{SOA}^{2} + 4\,\lambda_{SOA}^{3}} \\
& + \frac{2\,e^{-t\,\sigma_5}\,\left(\lambda_{DSA}\,\lambda_{HDS} + \lambda_{DSA}\,\lambda_{SOA} - \sigma_3 - \sigma_2 + \lambda_{DSA}^{2} - 2\,\lambda_{DSA}\,\lambda_{HDS}\,\lambda_{SOA}\right)}{\sigma_5} + \frac{2\,e^{-t\,\sigma_4}\,\left(\lambda_{DSA}\,\lambda_{SOA} + \lambda_{HDS}\,\lambda_{SOA} - \sigma_3 - \sigma_2 + \lambda_{SOA}^{2} - 2\,\lambda_{DSA}\,\lambda_{HDS}\,\lambda_{SOA}\right)}{\sigma_4} - \\
& - \frac{e^{-t\,\sigma_1}\,\left(2\,\lambda_{DSA}\,\lambda_{HDS} - \lambda_{HDS} + 4\,\lambda_{DSA}\,\lambda_{SOA} + 2\,\lambda_{HDS}\,\lambda_{SOA} - 4\,\lambda_{DSA}\,\lambda_{SOA}^{2} - 4\,\lambda_{DSA}^{2}\,\lambda_{SOA} + 2\,\lambda_{DSA}^{2} + 2\,\lambda_{SOA}^{2} - 4\,\lambda_{DSA}\,\lambda_{HDS}\,\lambda_{SOA}\right)}{\sigma_1} + 1
\end{aligned}
$$

where

$$\sigma_1 = 2\,\lambda_{DSA} + \lambda_{HDS} + 2\,\lambda_{SOA}$$

$$\sigma_2 = 2\,\lambda_{DSA}^{2}\,\lambda_{SOA} \qquad \sigma_4 = 2\,\lambda_{DSA} + \lambda_{HDS} + \lambda_{SOA}$$

$$\sigma_3 = 2\,\lambda_{DSA}\,\lambda_{SOA}^{2} \qquad \sigma_5 = \lambda_{DSA} + \lambda_{HDS} + 2\,\lambda_{SOA}$$

Figure 8.8: Reliability function for DRS.

$$
\begin{aligned}
\text{MTTF2} = \\
&-\frac{2\,\sigma_2}{\sigma_1^{\,2}}+\left(\lim_{t\to\infty}-\frac{t\,\lambda_{DSA}{}^2\,2}{\sigma_1}-\frac{t\,\lambda_{SOA}{}^2\,2}{\sigma_1}+\frac{e^{-t\,\lambda_{DSA}{}^2-t\,\lambda_{HDS}-t\,\lambda_{SOA}{}^2}\,\sigma_2\,2}{\sigma_1^{\,2}}+\frac{t\,\lambda_{DSA}\,2}{\sigma_1}+\frac{t\,\lambda_{SOA}\,2}{\sigma_1}-\frac{t\,\lambda_{DSA}\,\lambda_{SOA}\,4}{\sigma_1}-\frac{\lambda_{DSA}\,\lambda_{SOA}\,e^{-t\,\lambda_{DSA}-t\,\lambda_{HDS}-t\,\lambda_{SOA}}\,4}{\lambda_{DSA}+\lambda_{HDS}+\lambda_{SOA}}+\frac{t\,\lambda_{DSA}\,\lambda_{SOA}{}^2\,4}{\sigma_1}\right. \\
&+\frac{t\,\lambda_{DSA}\,\lambda_{SOA}{}^2\,4}{\sigma_1}+\frac{t\,\lambda_{DSA}{}^2\,\lambda_{SOA}\,4}{\sigma_1}-\frac{\sigma_4\,e^{-t\,\lambda_{HDS}}\,\sigma_3}{\sigma_1}-\frac{\lambda_{DSA}\,\lambda_{SOA}\,\sigma_5\,(2\,\lambda_{DSA}-1)\,2}{\sigma_7^{\,2}}-\frac{\lambda_{DSA}\,\lambda_{SOA}\,\sigma_6\,(2\,\lambda_{SOA}-1)\,2}{\sigma_8^{\,2}}+\frac{\lambda_{DSA}\,\sigma_4\,e^{-t\,\lambda_{HDS}}\,\sigma_3\,2}{\sigma_1}+\frac{\lambda_{SOA}\,\sigma_4\,e^{-t\,\lambda_{HDS}}\,\sigma_3\,2}{\sigma_1} \\
&-\frac{\lambda_{SOA}\,\sigma_5\,(2\,\lambda_{DSA}-1)\,\left(t\,\lambda_{DSA}\,e^{t\,\lambda_{DSA}{}^2+t\,\lambda_{HDS}+t\,\lambda_{SOA}}-1\right)2}{\sigma_7}-\frac{\lambda_{DSA}\,\sigma_6\,(2\,\lambda_{SOA}-1)\,\left(t\,\lambda_{SOA}\,e^{t\,\lambda_{DSA}{}^2+t\,\lambda_{HDS}+t\,\lambda_{SOA}{}^2}-1\right)2}{\sigma_8}-\frac{\lambda_{DSA}\,\lambda_{SOA}\,\sigma_4\,e^{-t\,\lambda_{HDS}}\,\sigma_3\,4}{\sigma_1}\Bigg)-\frac{2\,\lambda_{DSA}}{\sigma_1}-\frac{2\,\lambda_{SOA}}{\sigma_1} \\
&+\frac{1}{\sigma_1}+\frac{4\,\lambda_{DSA}\,\lambda_{SOA}}{\lambda_{DSA}+\lambda_{HDS}+\lambda_{SOA}}-\frac{2\,\lambda_{DSA}\,(2\,\lambda_{SOA}-1)}{\sigma_8}-\frac{2\,\lambda_{SOA}\,(2\,\lambda_{DSA}-1)}{\sigma_7}+\frac{4\,\lambda_{DSA}\,\lambda_{SOA}}{\sigma_1}+\frac{2\,\lambda_{DSA}\,\lambda_{SOA}\,(2\,\lambda_{DSA}-1)}{\sigma_7^{\,2}}+\frac{2\,\lambda_{DSA}\,\lambda_{SOA}\,(2\,\lambda_{SOA}-1)}{\sigma_8^{\,2}}
\end{aligned}
$$

where

$$\sigma_1 = 2\,\lambda_{DSA}+\lambda_{HDS}+2\,\lambda_{SOA}$$

$$\sigma_2 = 2\,\lambda_{DSA}{}^2\,\lambda_{SOA}-\lambda_{DSA}{}^2+2\,\lambda_{DSA}\,\lambda_{SOA}{}^2-2\,\lambda_{DSA}\,\lambda_{SOA}+\lambda_{DSA}-\lambda_{SOA}{}^2+\lambda_{SOA}$$

$$\sigma_3 = e^{-t\,\lambda_{SOA}{}^2}$$

$$\sigma_4 = e^{-t\,\lambda_{DSA}{}^2}$$

$$\sigma_5 = e^{-t\,\lambda_{DSA}{}^2-t\,\lambda_{HDS}-t\,\lambda_{SOA}}$$

$$\sigma_6 = e^{-t\,\lambda_{DSA}-t\,\lambda_{HDS}-t\,\lambda_{SOA}{}^2}$$

$$\sigma_7 = 2\,\lambda_{DSA}+\lambda_{HDS}+\lambda_{SOA}$$

$$\sigma_8 = \lambda_{DSA}+\lambda_{HDS}+2\,\lambda_{SOA}$$

Figure 8.9: MTTF function for DRS.