Development and Certification of Dependable Mixed-Criticality Embedded Systems

DISSERTATION

zur Erlangung des Grades eines Doktors der Ingenieurwissenschaften

vorgelegt von

Dipl.-Ing. Asier Larrucea Ortube

Promotionskommission:
Prof. Dr. Roman Obermaisser
Prof. Dr. Alfons Crespo
Prof. Dr. Christoph Ruland
Prof. Dr. Kristof Van Laerhoven

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen
Siegen 2017

Tag der mündlichen Prüfung:
06 July 2017

Declaration of Authorship

I, Asier Larrucea Ortube, declare that this thesis titled, 'Development and Certification of Dependable Mixed-Criticality Embedded Systems' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: __________________________________

Date: __________________________________

UNIVERSITÄT SIEGEN

Zusammenfassung

Entwicklung und Zertifizierung von zuverlässigen eingebetteten Mixed-Criticality-Systemen

von Asier Larrucea Ortube

Der Übergang von herkömmlichen Verbundarchitekturen zu integrierten Architekturen ermöglicht die Integration von Funktionalitäten mit unterschiedlichen Kritikalitäten in Bezug auf die Betriebs- und Angriffssicherheit sowie Echtzeit fähigkeit in einer einzigen eingebetteten Computer-Plattform. Der Trend zu Multi-core- und Many-Core-Architekturen hat des Weiteren zu dieser Tendenz beigetragen, indem er Vorteile in Hinblick auf die Kosten, Größe und Gewicht liefert. Multi-Core-Architekturen sind derart gestaltet, dass eine maximale durchschnittliche Leistung auf Kosten einer erhöhten Komplexität und Wechselwirkungen zwischen Applikationen geboten wird. Partitionierungslösungen wie Hypervisoren (z.B. XtratuM, PikeOS) werden normalerweise zur Bewältigung von mit diesen Architekturen in Verbindung stehenden Herausforderungen verwendet. Sie begrenzen die Auswirkung von Veränderungen und Störungen auf abgesteckte Bereiche des Systems, auch Partitionen genannt, ermöglichen die Wiederverwendbarkeit und verringern die Komplexität. Derartige Partitionen können individuell und mit unterschiedlichen Kritikalitätslevels gestaltet, entwickelt und zertifiziert werden (z.B. Sicherheitsintegritätslevel (SIL) 1 bis 4 gemäß IEC 61508). Obgleich auch partitionierte Multi-Core-Architekturen die o.g. Vorteile aufweisen können sie viele Probleme im Zusammenhang mit der Zertifizierung mit sich bringen, wie z.B. die Bewertung der zeitlichen Unabhängigkeit, was wiederum zu einer beachtlichen Erhöhung der Engineering- und Zertifizierungskosten führt. Darüber hinaus kann ein eingebettetes System verteilte Subsysteme mit Kommunikationsnetzwerken (wie z.B. EtherCAT) erforderlich machen, um den Rechnerressourcenbedarf zu decken, Fehlertoleranz sicherzustellen und die Installationsanforderungen zu erfüllen. Der allgemeine Trend hin zur Integration von Funktionalitäten mit unterschiedlichen Kritikalitäten in einer einzigen eingebetteten Computer-Plattform erfordert die Implementierung von sicheren und vorhersagbaren Kommunikationssystemen mit einer zeitlichen Trennung zwischen den verschiedenen Kritikalitäten. Aus diesem Grund stellen sich für Kommunikationsnetzwerke besondere

Herausforderungen in Hinblick auf die Zertifizierung, wie z.B. ein Sicherstellen der Non-Interference zwischen sicherheitskritischen und nicht sicherheitskritischen Kommunikationsaktivitäten. In dieser Dissertation werden modulare Sicherheitskonzepte für einen IEC 61508-konformen generischen Hypervisor, die Partition, kommerzielle Mehrkernprozessoren und Netzwerke mit gemischter Kritikalität vorgestellt. In einem modularen Sicherheitsnachweis werden sicherheitsrelevante Argumente und Nachweise definiert, die ein System aufweisen muss, um die Sicherheitsstandards zu erfüllen. Die in dieser Dissertation definierten Sicherheitsnachweise sind von einer Zertifizierungsstelle im Rahmen des Europäischen Forschungsprojekts DREAMS bewertet worden. Darüber hinaus wird in dieser Doktorarbeit eine Verbindungsanalyse für kommerzielle Technologien, wie z.B. den XtratuM-Hypervisor, Zynq-7000, TTEthernet und EtherCAT-Netzwerke durchgeführt. In dieser Analyse wird beschrieben, inwiefern ein kommerzielles System in generischen modularen Sicherheitsnachweisen identifizierte sicherheitsrelevante Anforderungen erfüllt.

Als Ergebnis der Definition von modularen Sicherheitsnachweisen und der dazugehörigen Analyse des IEC 61508 Sicherheitsstandards sind jene Komponenten identifiziert worden, die Herausforderungen für die Entwicklung und Zertifizierung von derzeit vorhandenen eingebetteten Computer-Plattformen mit gemischter Kritikalität aufweisen. Außerdem konnte festgestellt werden, dass die durch den Sicherheitsstandard IEC 61508 empfohlenen Maßnahmen und Diagnosetechniken meist auf Single-Core-Architekturen ausgerichtet sind, bei denen eine Ressource lediglich mit einer Komponente geteilt werden kann. Diese Maßnahmen und Diagnosetechniken sind keineswegs auf heute verfügbare Systeme mit gemischter Kritikalität anwendbar, da bei ihnen sehr häufig eine Ressource mit mehr als einer Komponente geteilt wird. So kann z.B. bei einer Multi-Core-Architektur gleichzeitig mehr als eine Komponente auf einen Speicherbereich zugreifen, was wiederum zu Wechselwirkungen führt, die die Sicherheit des Systems gefährden können. In dem Bestreben, eine Antwort auf diese Herausforderungen zu finden, werden in dieser Dissertation verschiedene allgemeine domänenübergreifende Modelle und Lösungen für bei der Entwicklung von Systemen mit gemischter Kritikalität häufig auftretende Probleme vorgestellt. Diese Modelle werden ausgehend vom DREAMS-Architekturstil untersucht, definiert und letztlich in einer Fallstudie zu einer Windkraftanlage implementiert. Diese Fallstudie liefert ein realistisches Systemszenario, in das die in dieser Dissertation aufgeführten Lösungen integriert und letztlich bewertet werden können.

UNIVERSITY OF SIEGEN

Abstract

Development and Certification of Dependable Mixed-Criticality Embedded Computing Systems

by Asier Larrucea Ortube

The transition from conventional federated architectures to integrated architectures enables the integration of functionalities with different criticality concerning safety, security and real-time on a single embedded computing platform. The trend towards multi-core and many-core architectures has further contributed to this tendency, providing benefits regarding cost-size-weight. Multi-core architectures are designed for offering the maximum average performance at the cost of increasing complexity and interferences. Partitioning solutions such as hypervisors (e.g., XtratuM, PikeOS) are commonly used to tackle the challenges related to these architectures. They limit the impact of changes and faults to reduced areas of the system, also called partitions, enabling reusability and reducing the complexity. Partitions can be designed, developed and certified individually with different levels of criticality (e.g., Safety Integrity Level (SIL) 1 to 4 according to IEC 61508). However, although partitioned multi-core architectures provide the benefits mentioned before, they imply many challenges to certification such as the assessment of the temporal independence, which leads to a significant increase in the engineering and certification cost. Furthermore, an embedded system may require distributed subsystems with communication networks (such as EtherCAT) to satisfy the computational resource demands, ensure fault-tolerance and satisfy the installation requirements. The broad trend of the integration of functionalities with different criticality on a single embedded computing platform involves the implementation of safe and predictable communication systems with temporal segregation between different criticality. Therefore, communication networks represent certification challenges such as guaranteeing the non-interference between safety-critical and non safety-critical communications. This dissertation presents the modular safety concepts for an IEC 61508 compliant generic hypervisor, partition, commercial-off-the-shelf (COTS) multi-core device and mixed-criticality network. A modular safety case (MSC) defines the safety-related arguments and evidences that a system must fulfil in order to be compliant with a safety standard. The MSCs defined throughout this thesis have been assessed by a certification body within the context of the European research project DREAMS. Besides, this dissertation defines the linking analysis for commercial technologies such as XtratuM hypervisor, Zynq-7000 multi-core device and TTEthernet and EtherCAT networks. A linking analysis describes the way in which a commercial system fulfils the safety-related requirements identified in the generic modular safety cases.

As a result of the definition of the modular safety cases and associated analysis of the IEC 61508 safety standard, the remarkable components that imply challenges in the development and certification of today's mixed-criticality embedded computing platforms have been identified. In addition, it is detected that the measures and diagnostic techniques recommended by the IEC 61508 safety standard are mostly geared to single-core architectures where a resource cannot be shared among more than one component. These measures and diagnostic techniques are not at all applicable to today's mixed-criticality systems where sharing a resource among more than one component is a common task. For example, in multi-core architectures a memory area can be accessed simultaneously by more than one component (e.g., CPUs), leading to interferences that may jeopardise the safety of the system. In order to give a solution to those challenges, the dissertation presents several generic cross-domain patterns for commonly occurring problems in the development of mixed-critical systems. These patterns are analysed, defined and implemented in a wind turbine case study based on the DREAMS architecture style. This case study provides a realistic system scenario where the solutions generated in this dissertation are integrated and evaluated.

Acknowledgements

This thesis has been carried out in collaboration with the Electronic department of IK4-Ikerlan research centre (Basque Country, Spain) and Embedded Systems department of University of Siegen (Siegen, Germany).

I would like to thank Prof. Roman Obermaisser for his support during these three years. I really appreciate the time that he has dedicated to discuss the state of my thesis and guide me through the right way. In addition, I would like to thank Dr. Jon Perez for giving me the opportunity to do a PhD under his direction on the topic of safety mixed-critical embedded computing platforms between IK4-Ikerlan and the University of Siegen.

Many thanks to my colleagues from IK4-Ikerlan for enduring me and cheer me up whenever I have needed. Many thanks also to my colleges in the University of Siegen for their hospitality and support during my stages in Germany, you make me feel at home.

I also want to thank the partners with who I have worked in the European research project DREAMS for their support and cooperation.

Last but not least, I heartily thank the compression received from my parents, brother, family and friends. You have patiently endured my frustrations and moodiness during the last three years and more.

Contents

List of Figures

List of Tables

Abbreviations

ACE AXI Coherency Extension

ACP Accelerator Coherency Port

AES Advanced Encryption Standard

AFDX Avionics Full-Duplex Switched Ethernet

AHB Advanced High-Performance Bus

AMBA Advanced Microcontroller Bus Architecture

APB Advanced Peripheral Bus

ARAMIS Automotive, Railway and Avionics Multicore Systems

ASB Advanced System Bus

ASCOS Aviation Safety and Certification of new Operations and Systems

ASIC Application Specific Integrated Circuit

ASIL Automotive Safety Integrity Level

AMP Asymmetric MultiProcessing

AXI Advanced eXtensible Interface

BCET Best Case Execution Time

BE Best-Effort

BRAM Block Random Access Memory (RAM)

CAE Claim, Argument and Evidence

CC Common Criteria

CPLD Complex Programmable Logic Device

CONTREX Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties

COTS Commercial off-the-shelf

CRC Cyclic Redundancy Check

CMUDP *Coherency Management Unit Diagnostic Pattern*

SMDP *Shared Memory Diagnostic Pattern*

CT Critical Traffic

DAS Distributed Application Subsystem

DC Diagnosis Coverage

DDRC DRAM Controller

DSE Design Space Exploration

DIO Digital I/O

DIOS Digital I/O (DIO) Server

DMA Direct Memory Access

DDR Double Data Rate

DECOS Dependable Embedded Components and Systems

DREAMS Distributed REal-time Architecture for Mixed Criticality Systems

ECC Error Correction Code

ECU Electronic Control Unit

EDC Error Detection Correction

E/E/PE Electrical / Electronic / Programmable Electronic

EMC^2 Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments

ET Event-Triggered

EUC Electronic Under Control

FCU Fault Containment Unit

FTU Fault Tolerant Unit

FIQ Fast Interrupt reQuest

FIT Failure In Time

FMEA Failure Mode and Effects Analysis

FMEDA Failure Mode Effects and Diagnostic Analysis

FMECA Failure Modes, Effects and Criticality Analysis

FPGA Field Programmable Gate Array

FSBL First Stage Boot Loader

FSCP Functional Safety Communication Profile

FSM Functional Safety Management

FSoE Functional Safety over EtherCAT

GP Generic Product

GPSC Generic Product Safety Case

GASC Generic Application Safety Case

GA Generic Application

GENESYS GENeric Embedded SYStem

GIC Generic Interrupt Controller

GRM Global Resource Manager

GSN Goal Structuring Notation

GUI Graphical User Interface

HFT HW Fault-Tolerance

HM Health Monitoring

HMAC Hash Message Authentication Code

HMI Human Machine Interface

HP High Performance

HSTL High-Speed Transceiver Logic

HW Hardware

ICAP Internal Configuration Access Port

I/O Input/Output

IDF Isolation Design Flow

IOP I/O Peripheral

IRQ Interrupt Request

ISE Integrated Synthesis Environment

IVT Isolation Verification Tool

IT Information Technology

LPDDR Low-Power DDR

LRM Local Resource Manager

LVCMOS Logic Voltage Complementary Metal-Oxide Semiconductor

M Master

MIO Multiplexed I/O

MMU Memory Management Unit

MPU Memory Protection Unit

MSC Modular Safety Case

MULCORS MULti-core proCessORS in Airbone Systems

NASA National Aeronautics and Space Administration's

MultiPARTES Multi-cores Partitioning for Trusted Embedded Systems

ND Network Description

NCDB Network Configuration database

NC Network Configuration

NoC Networks-on-Chip

OCM On-Chip Memory

QoS Quality-of-Service

PCI Peripheral Component Interconnect

PCIe Peripheral Component Interconnect (PCI) Express

PL Programmable Logic

PLD Programmable Logic Device

PLL Phase Locked Loop

POR Power On Reset

P2P Point-to-Point

PPI Private Peripheral Interrupt

PROXIMA Probabilistic real-time control of mixed-criticality multi-core and manycore systems

PS Processing System

PST Process Safety Time

pSafeCer pilot Safety Certification of Software-Intensive Systems with Reusable Components

RAM Random Access Memory

RAMS Reliability, Availability, Maintainability and Safety

RC Rate Constrained

ReComp Reduced Certification Costs Using Trusted Multi-core Platforms

RSA Rivest, Shamir and Adleman

RTCA Radio Technical Commission for Aeronautics

SA Specific Application

SACM Structured Assurance Case Metamodel

SASC Specific Application Safety Case

SASO Safe Autonomous Systems Operations

SCADA Supervision Control And Data Adquisition

SCL Safety Communication Layer

SCU Snoop Control Unit

SAC SCU Access Control

SEU Single Event Upset

SEM Soft Error Mitigation

SGI SW Generated Interrupt

SPI Shared Peripheral Interrupt

NSAC Non Secure SCU Access Control

SD Secure Digital

SDL Security Development Lifecycle

SDK SW Development Kit

SEooC Safety Element out of Context

SIL Safety Integrity Level

SILCL Safety Integrity Level Claim Level

SM Synchronization Master

SMP Symmetric MultiProcessing

SoC System on a Chip

SoS System of System

SRAC Safety Related Application Conditions

STNoC STmicroelectronics' NoC

SW Software

TDMA Time Division Multiple Access

TERESA Trusted Computing Engineering for Resource Constrained Embedded Systems Applications

TLB Translation Lookaside Buffer

TMR Triple Modular Redundancy

TT Time Triggered

TTA TT Architecture

TTE TTEthernet

TTEL TT Extension Layer

TTNoC TT Network-on-Chip

XADC Xilinx Analog to Digital Converter

XMCF XtratuM Configuration File

VHDL Very High Speed Integrated Circuit

VL Virtual Link

VM Virtual Machine

VMM Virtual Machine Monitor

WCET Worst Case Execution Time

WDT Watch Dog Timer

Nire guraso, anaia, senide, lagun eta nirekin hiru urte hauetan izan diren denei dedikaturik.

Dedicado a mis padres, hermano, familia, amigos y a todos los que han estado conmigo durante estos tres años.

Dedicated to my parents, brother, family, friends and to those who have supported and encouraged me during these three years.

Gewidmet meiner Familie, meinen Bruder, meinem Freunden und den Personen, die mich über diese drei Jahre unterstützt haben.

Chapter 1

Introduction

Embedded systems have commonly followed a federated architecture paradigm in which each Distributed Application Subsystem (DAS) is implemented on its stand-alone distributed HW base with a well-defined functionality. However, the soaring demand for high performance and increasing functionality challenges the viability of this approach, leading to the rising trend of moving towards integrated architectures [OKG04] [Ham03]. As a consequence, system engineers aim at the integration of multiple functionalities with different criticality with respect to safety, security and real-time on the same embedded computing platform. A system that combines functionalities of different criticality is often referred to as a mixed-criticality system [Bau11].

The transition from single-core to multi-core architectures has further contributed to this tendency. Multi-core architectures provide benefits in terms of cost, size and weight reduction as well as improved scalability. However, they imply certification challenges (such as the assessment of the temporal independence) which may lead to a significant and potentially unacceptable increase of engineering and certification cost [DAN+13, RGG+12].

Virtualization mechanisms such as hypervisors are commonly used solutions to tackle challenges related to integrated architectures based on multi-core and many-core processors. These mechanisms limit the impact of changes and faults to reduced areas of the system, enabling in turn reusability and reducing the complexity [Kop08]. A hypervisor is a layer of SW or a combination of SW and HW that enables different independent execution environments on a single computing platform (e.g., XtratuM [Sol14], PikeOS [SYS15]). The execution environments, which are also often referred to as *guest operating systems*, *virtual machines*, *partitions* or *domains*, can be designed, developed and certified individually with different criticality levels (e.g., Safety Integrity Level (SIL) 1 to 4 according to the IEC 61508 safety standard).

The current integration trend of functionalities with different criticality into a single embedded computing platform may use communication networks with support for different criticality for communication purposes. These communication systems, which are usually referred to as mixed-criticality networks, are capable of supporting safe and predictable message exchanges between DAS with different criticality. Mixed-criticality networks are targeted as the natural replacement of legacy buses in modern system architectures due to their low-cost, high-speed and easy integration with existing network infrastructures. They can be divided into on-chip buses (e.g., Advanced High-Performance Bus (AHB), Advanced System Bus (ASB), Advanced Peripheral Bus (APB)) for communication between the components of the integrated circuits such as the cores of a multi-core processor, off-chip buses (e.g., Ethernet) for inter-node communication (e.g., two devices) and local buses (e.g., PCI, PCI Express (PCIe)) for chip-to-chip interconnect. However, the use of mixed-criticality networks may lead to certification challenges related to the increasing safety, security and real-time constraints in demanding application domains such as automotive and railway.

Certification is a third-party attestation related to products, processes, systems or persons [ISO04]. An attestation is the issue of a statement, based on a decision following reviews, where the fulfilment of specified requirements has been demonstrated. In the safety domain, safety certification is an attestation where an authorised organization or a certification body assesses the fulfilment of the safety requirements of a system regarding specified safety requirements or a safety standard (e.g., IEC 61508, ISO 26262). The traditional approach to certification relies on the certification of the whole system, where the change of a safety aspect of the system implies the re-certification of the entire system. Modularity provides a mechanism for managing the complexity of today's mixed-criticality systems, subdividing the system into smaller parts, also called modules, which can be independently certified and re-used to compose a mixed-criticality system. The certification process of a safety-related system is usually carried out by means of safety cases. A safety case is a documented body of evidences that provides convincing and valid arguments that a system is adequately safe for a given application in a given environment (e.g., automotive, railway, lift). Instead, in modular certification, MSCs are used for evidencing the safety of the components that compose a safety-related system. These safety cases take advantage of the modularity in mixed-criticality system design, allowing assurance of the safety of a system that consists of design modules. This approach is supported by different safety-related standards such as IEC 61508 [IEC10a], ISO 26262 [ISO15a], EN 50129 [EN03] and DO-178 [RTC11] and by many research projects such as the European projects Aviation Safety and Certification of new Operations and Systems (ASCOS) [ASC12], Dependable Embedded Components and Systems (DECOS) [SAS^{+}08] and National Aeronautics and Space

Administration's (NASA) Safe Autonomous Systems Operations (SASO) project [DP15] [NAS15].

On the other hand, the measures and diagnostic techniques recommended by today's safety-related standards such as IEC 61508 are geared to single computer systems (see Annex F of IEC 61508-3), where a resource cannot be shared between more than one component at the same time. Instead, multi-core architectures enable sharing a resource between the CPUs at the same time, thus leading to possible interferences in temporal and spatial domains. So, the need for extra measures and diagnostic techniques with support for multi-core devices is identified. However, there are certain exceptions which include non-shared resources of multi-core devices where the measures and diagnostic techniques recommended by those safety standards are applicable (see Tables A.2 to A.14 of IEC 61508-2 [IEC10b]).

1.1 Objective

The objectives of this thesis are the definition of generic MSCs for mixed-criticality systems and the definition and implementation of reusable generic cross-domain patterns to solve the certification challenges of today's mixed-criticality systems. These objectives target the key challenges related to certification and mixed-criticality systems which are identified in the introduction.

- *Challenges related to certification:* Mixed-criticality systems are complex systems that support executing functionalities with different criticality onto the same embedded computing platform. These systems imply several challenges that increase the development and certification cost. In traditional certification, the system is assessed as a whole. Therefore, if a requirement of the system changes, the entire system must be re-certified. This thesis integrates the modularity methodology into the development and certification of mixed-criticality systems to tackle these certification challenges. To that end, this dissertation defines IEC 61508 compliant MSCs for remarkable components of today's mixed-criticality systems such as a hypervisor, a safety partition, a COTS device and a mixed-criticality network.
- *Challenges of mixed-criticality systems:* Today's safety-related standards (e.g., IEC 61508) focus mainly on single computer architectures where a resource is not shared among more than one component. Instead, multi-core architectures enable sharing a resource between more than one component at the same time. For instance, a memory region can be shared between several processing units of a multi-core device at the same time. This sharing of resources may cause interferences in

the temporal and spatial domains hindering the certification of mixed-criticality systems. Therefore, the measures and diagnostic techniques recommended by those standards are only applicable to single computers. For that reason, the need for new measures and diagnostic techniques with support for multi-core mixed-criticality architectures is identified. This thesis presents reusable generic solutions for mixed-criticality systems which are out of the scope of today's safety-related standards.

1.2 Contributions

The contributions of this dissertation consist of:

- Generic IEC 61508 compliant MSCs for components of today's mixed-criticality systems, including a safety hypervisor (see Section 5.1), a safety partition (see Section 5.2), a safety COTS multi-core device (see Section 5.3) and a mixed-criticality network (see Section 5.4). Modularity enables the division of the system into modules which can be independently assessed and re-used for developing a mixed-criticality system.
- Reusable and generic cross-domain patterns for partitioned and networked multi-core mixed-criticality systems. These patterns provide reusable solutions, measures and diagnostic techniques to solve, detect and avoid the key issues of today's mixed-criticality systems which are not considered by current safety-related standards (see Chapter 6).

The MSCs and cross-domain patterns defined throughout this dissertation can be integrated into different domain case studies (e.g., railway, automotive, wind-power, health-care, lift). The solutions presented in this thesis are integrated into a wind turbine case study that complies with the architecture style of the European research project DREAMS (see Chapter 4) and the IEC 61508 safety standard.

1.3 Structure of this Thesis

This thesis is organised as described below and as shown in Figure 1.1.

- Chapter 2 introduces the basic concepts on which the work of this thesis is based.

- Chapter 3 analyses the state-of-the-art of the modularity among different safety standards (e.g., IEC 61508 and ISO 26262) and defines the remarkable safety case notation languages.
- Chapter 4 defines the architecture style of the European research project DREAMS.
- Chapter 5 presents the MSCs for an IEC 61508 compliant generic hypervisor, partition, COTS multi-core processor and mixed-criticality network.
- Chapter 6 analyses the recurring problems in the development and certification process of today's mixed-criticality systems by cross-domain patterns. In addition, this chapter defines several remarkable cross-domain patterns that aim to solve the commonly occurring problems in mixed-criticality systems.
- Chapter 7 presents the wind turbine case study where the main contributions of this thesis are integrated.
- Chapter 8 shows the conclusions and future work.

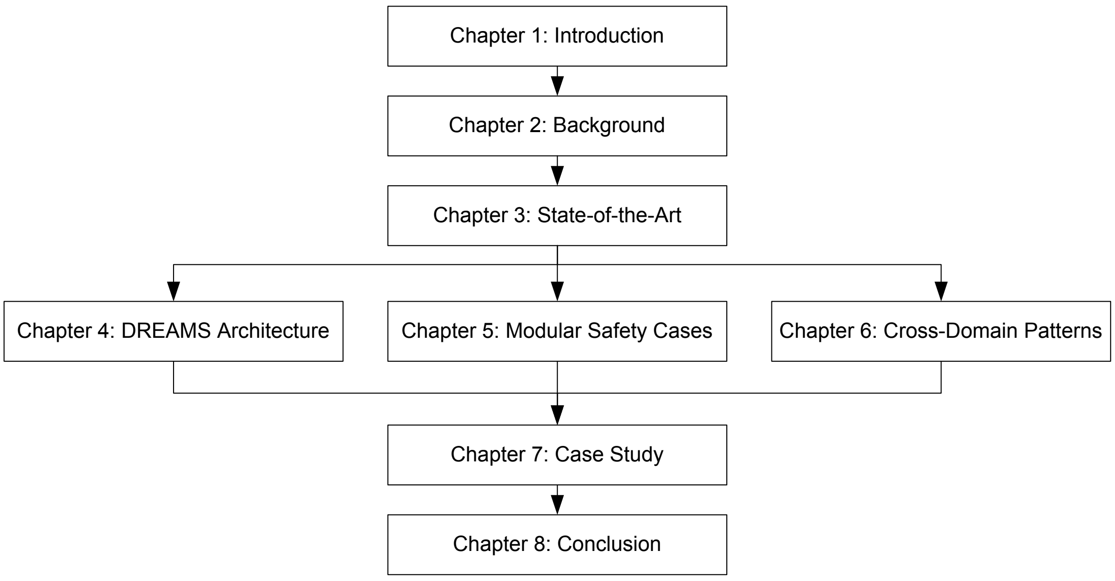
Chapter 1: Introduction
Chapter 2: Background
Chapter 3: State-of-the-Art
Chapter 4: DREAMS Architecture
Chapter 5: Modular Safety Cases
Chapter 6: Cross-Domain Patterns
Chapter 7: Case Study
Chapter 8: Conclusion

FIGURE 1.1: Structure of the thesis.

Chapter 2

Background

This chapter introduces the basic concepts on which the work presented in this thesis is based.

2.1 Mixed Criticality Systems

A mixed-criticality system refers to the integration of HW, operating system, middleware services and software applications (e.g., safety-critical, non safety-critical) on the same embedded computing platform [Com12, WESK10]. This approach enables the reduction of devices, wires, and connectors, thereby improving the overall cost-size-weight factor and scalability. However, mixed-criticality systems, especially multi-core mixed-criticality systems, give rise to challenges related to certification (e.g., the assurance of temporal independence) [ESEH^{+}12, PGT^{+}13, PGN^{+}14].

There are many running or just finished research projects in the field of multi-core and mixed-criticality technology funded by the European Union (e.g., DREAMS [DRE13], Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties (CONTREX) [CON14], Probabilistic real-time control of mixed-criticality multi-core and manycore systems (PROXIMA) [PRO14]), national bodies (e.g., Automotive, Railway and Avionics Multicore Systems (ARAMIS) project [ARA11]) and a mix of European and national funding (e.g., ARTEMIS Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments (EMC^{2}) [EMC14]).

2.2 Certification Standards

Certification is a third-party attestation related to products, processes, systems or persons [ISO04]. An attestation is the issue of a statement, based on a decision following reviews, where the fulfilment of specified requirements has been demonstrated. Safety certification attests that a system is safe enough for its purpose with a given confidence level and in a given environment. For instance, IEC 61508 [IEC10a, IEC10b, IEC10c] is the generic international standard for Electrical / Electronic / Programmable Electronic (E/E/PE) functional safety systems and it is the reference safety standard for different domain-specific safety standards such as machinery, industrial process, automotive and railway (see Figure 2.1). IEC 61508 defines the concept of SIL as a relative level of risk-reduction provided by a safety-related system with values in the range from 1 to 4. SIL1 is the lowest value and SIL4 is the most restrictive one. As a general rule, the highest SIL has the highest certification cost.

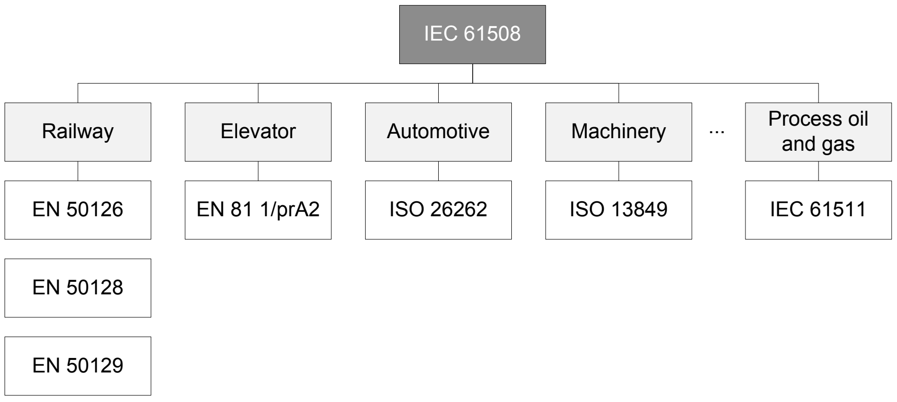

FIGURE 2.1: Safety standards – Hierarchy. (Source IEC 61508-1 [IEC10a])

2.2.1 Fail-Safe and Fail-Operational Systems

Safety-critical systems can be classified as *fail-safe* or *fail-operational* [Kop11]. A system is *fail-safe* when in the event of a failure a safe state can be reached. For example, a train can be stopped in the case of a failure, leading it to a safe state. Conversely, a system that must remain operational even after the occurrence of a fault, i.e., cannot reach a safe state, is called a *fail-operational* system. For example, an aircraft cannot reach a safe state while in flight, it must continue flying although a failure occurs.

2.3 Dependability

Dependability refers to the ability to provide services that can defensibly be trusted at a time. These services can be provided to system users such as human users or computer systems [ALRL04]. Whenever the behaviour of a system deviates from the expected intended usage, it is considered that the system fails. An *error* is part of the system state that may cause a subsequent *failure*. A *fault* is the adjudged or hypothesised cause of an error. Dependability can also be defined as the measure of a system's *reliability*, *safety*, *maintainability*, *availability* and *security* attributes.

- *Reliability (MTTF)* refers to the probability that a system will provide specific services until a given time, considering that the system is operational at the beginning.
- *Safety* is reliability regarding critical failure modes. The failure modes can be *malign* or *benign*. Malign failure modes are concerned to critical failure modes while benign failure modes are considered non-critical failures. For example, the crash of an aeroplane due to a failure in the flight-control system is regarded as a malign failure mode.
- *Maintainability (MTTR)* is a measure of the time interval required to repair a system after the occurrence of a benign failure.
- *Availability (A)* is a measure of the delivery of the correct service given the alternation of correct and incorrect services. Availability is related to the reliability and maintainability attributes as follows:

$$A = MTTF/(MTTF + MTTR)$$

- *Security* is concerned with the authenticity and integrity of information and the ability of a system to prevent unauthorised accesses to information or services.

2.4 Fault Containment

Fault containment refers to design and engineering efforts that ensure that the immediate consequences of a fault are limited to a single Fault Containment Unit (FCU) [Kop11]. An FCU is the specification of the units of failure that can impact on the overall reliability of a system. The quality engineer must ensure that FCUs fail independently. Otherwise, if they fail at the same time, they will have a substantial impact on the overall reliability of the system.

In a distributed system, a HW or a SW component can be considered to form an FCU. For instance, on a multi-core processor, a core that communicates with other cores through messages can be deemed to constitute an FCU. However, since the cores of a multi-core processor are physically close together, share a common power supply and a common timing source, the independence of the failures of the cores cannot be justified. For example, in the E/E/PE safety function or the automotive domain, the compliance to SIL3 or Automotive Safety Integrity Level (ASIL) D means that a maximum of one dangerous error within 10^{-7} operating hours is acceptable [ISO15b].

Fault Tolerant Units (FTUs) tolerate the failures of FCUs. An FTU masks the failure of an FCU inside. To that purpose, an FCU can implement the fail-silent abstraction. A fail-silent FCU consists of a computational subsystem and an error monitor or two FCUs and a result comparator. In addition, if no assumptions about the failure behaviour of the FCU can be made, i.e., Byzantine failures,[1] then the Triple Modular Redundancy (TMR) may be needed. In TMR, an FTU consists of three synchronised replica deterministic FCUs composed of a voter and the computational subsystems which may communicate through a real-time communication network such as shown in Figure 2.2.

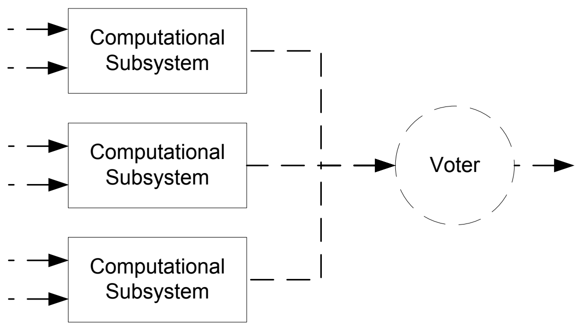

FIGURE 2.2: Triple Modular Redundancy – Overview.

2.5 Complexity Management

As stated in Section 2.1, the integration of applications with different criticality on a single embedded computing platform improves scalability and reduces the number of the system's wires and connectors, thereby improving the cost-size-weight factor. For instance, a system can be composed of tens/hundreds of SW partitions that communicate through an on-chip and/or off-chip mixed-criticality network and execute on the cores in case of the on-chip networks and devices in case of off-chip networks [Kop08].

[1]An error that occurs when a set of receivers observes different values of a real-time entity.

The system architect plays a key role in the specification and design of mixed-criticality systems and the selection of appropriate safety techniques for the system level. In addition, the system architect must provide sufficient evidence to the claim that the overall system is safe to the Reliability, Availability, Maintainability and Safety (RAMS) team and the external certification authorities. The availability of complexity management strategies should ease the development, maintenance and certification of mixed-criticality systems. As stated in [Kop08], four basic simplification strategies can be applied to manage the complexity and overcome human limited cognitive capabilities.

- *Abstraction* refers to the formation of a higher-level concept that captures the essence of the problem-at-hand and reduces the complexity of the scenario by omitting irrelevant details. In a scenario with tens/hundreds of building blocks, which can be custom implementations, COTS or developed by a third party, abstraction becomes a key concern for the system architect.
- *Partitioning* refers to the spatial division of a problem using a *divide and conquer* approach. This is applied, for example using SW partitions.
- *Isolation* refers to the suppression of irrelevant details from the specification of cause and an effect.
- *Segmentation* refers to the temporal decomposition of complex behaviour into smaller parts that can be processed sequentially. This is applied, for example, using SW partition schedules (e.g., periodic cyclic scheduling with pre-assigned time slots).

2.6 Structuring of Mixed-Criticality Embedded Systems

Mixed-criticality embedded systems consist of *physical* and *logical* structures (See Figure 2.3). Physically, a mixed-criticality system consists of a set of *clusters* that contain *nodes*. Each node is a multi-core chip containing *tiles* that is interconnected through a real-time communication network. The tiles provide network interfaces (NI) to the network and offer ports for the transmissions or receptions of the NoC's messages [DRE13].

A tile can be a processor cluster with several processor cores, caches, local memories and I/O resources. Alternatively, a tile can also be a single processor core or an IP core (e.g., memory controller that is accessible using the NoC and shared by several other tiles). Off-chip and on-chip networks are responsible for time and space partitioning between nodes or tiles respectively. They ensure that a node or tile cannot affect the guaranteed timing (e.g., bounded latency and jitter, guaranteed bandwidth) and the

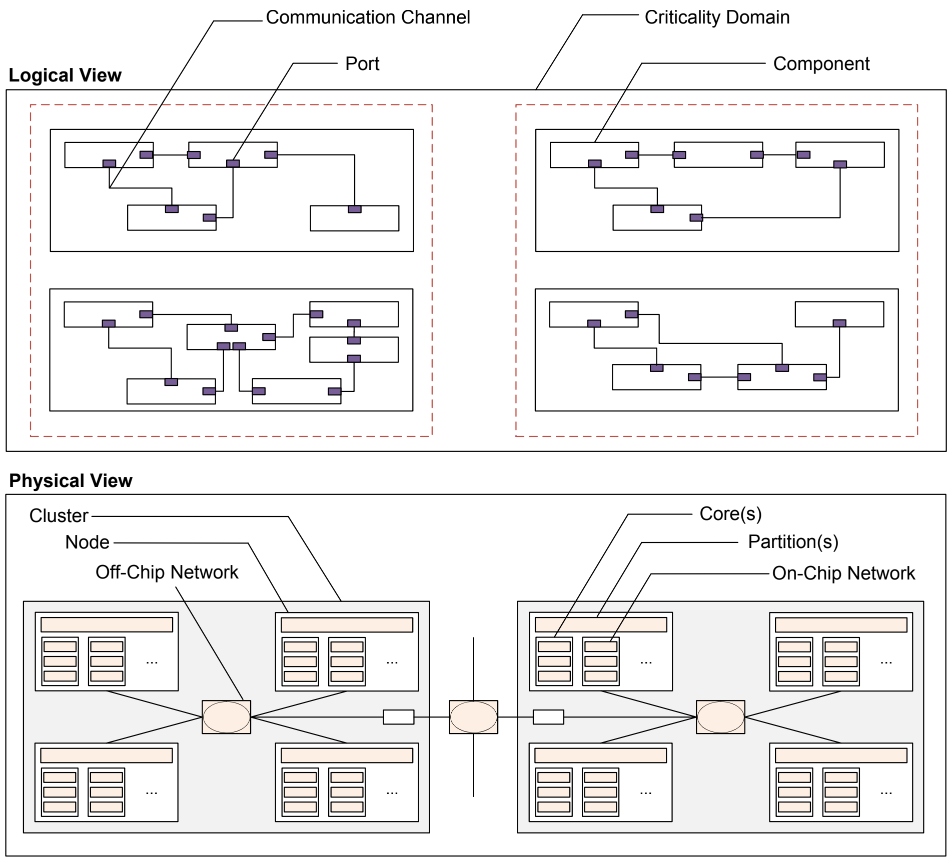

FIGURE 2.3: Logical and Physical structure of a mixed-criticality systems – Overview.

integrity of messages sent by other nodes and tiles. The processor cores within a tile can run a hypervisor (see Section 2.8) that establishes partitions, each of which executes a corresponding SW component. The hypervisor establishes time and space partitioning, thereby ensuring that a SW component cannot affect the availability of the computational resource in other partitions (e.g., time and duration of execution on the processor core, integrity and timing of memory).

On the other hand, the overall logical structure of a mixed-criticality system is structured into criticality levels. Several criticality levels can be distinguished in different application domains such as SIL1 to 4 in IEC 61508, classes A to E in avionics and ASILA to D in automotive. For each criticality level, there can be multiple application subsystems with different safety assurance levels. These subsystems can be further subdivided into components (e.g., processing memory and I/Os). Each component provides services to its environment and interacts by the exchange of messages via ports. TT, Rate Constrained (RC) and Best-Effort (BE) types of messages can be distinguished based on the timing of the messages.

2.7 Partitioning in Time and Value Domains

Partitioning is a requisite for modular certification that ensures the independence regardless of design faults affecting the components of a system. In modular certification, each application subsystem is certified to the respective level of criticality (e.g., SIL1 to 4 according to the IEC 61508 safety standard). The partitions for mixed-criticality systems are usually realised by operating systems and hypervisors (see Section 2.8 with time and space partitioning). In the temporal domain, scheduling mechanisms that assign available resources to tasks based on fixed periodic time slots are applied. On the other hand, spatial partitioning can be achieved by allocating a partition to an address space which is not accessible by other partitions. A partitioned system offers many appealing benefits such as memory protection (space partitioning), execution guarantees (time partitioning) and improved fault containment capabilities.

2.8 Virtualization

A Virtual Machine (VM) is a SW implementation of a machine (computer) that executes programs like a real machine. A hypervisor is a layer of SW or a combination of HW and SW that allows running several independent execution environments on a single computer platform. Several terms are used as synonyms of independent execution environments: *guest operating system*, *virtual machine*, *partition* or *domain*. The key difference between hypervisor technology and other virtualization technologies such as the Java VM are the performance and the portability. An hypervisor is slower than the Java VM but allows to run a wide variety of operating systems, which cannot be done using the Java VM.

Partitioning ensures that the shared computer system provides protection against fault propagation from one partition to another (fault-containment) in the temporal and spatial domains [Rus99]. In real-time embedded applications, the predictability and efficiency are essential requirements. Virtualization mechanisms jointly with real-time constraints must implement strict design methods and efficient solutions to guarantee the safety behaviour of the system [LAN$^+$15]. Consequently, virtualization mechanisms such as hypervisors are promising solutions for developing and certifying safety-critical embedded computing platforms.

Several software execution environments are available on the market reaching from bare metal (type 1) hypervisors to full featured operating systems with the capability of hosting heterogeneous guest operating systems in their partitions. Examples of software execution environments capable of running mixed-criticality applications include XtratuM [Sol14], PikeOS [SYS15] and Wind River Hypervisor [Riv15]. These hypervisors run directly on top of the HW and abstract from it, generating multiple execution environments (partitions) where functionalities with different criticality are hosted.

2.9 COTS Multi-Core Devices

The use of COTS multi-core devices is gaining popularity in different embedded system domains (e.g., automotive, railway, avionic) driven by the demand of low cost, increased complexity and shortened time to market. However, the shift towards these multi-core devices is not straightforward. The development and certification of COTS devices are hindered by numerous drawbacks including temporal interferences. Those interferences may hamper the computation of the Worst Case Execution Times (WCETs) and increase the engineering and certification costs.

COTS multi-core devices are designed with the objective of offering a maximum performance by increasing the complexity of the underlying architecture, which conflicts with the standard practice in safety-critical systems that aims to employ simple, predictable and proven-in-use devices. Examples of COTS multi-core devices with high performance and low temporal predictability are P4080 [Sem10], Zynq [XIL14c], Hercules [Ins13] and MPC5643L [Fre13].

2.10 Mixed-Criticality Networks

Distributed mixed-criticality systems depend on communication networks (e.g., TTEthernet) for the data exchange between distributed computations. The trend of the integration of functionalities with different criticality on a distributed embedded computing platform requires the usage of communication systems supporting the coexistence of different criticalities. These communication systems, which are also called mixed-criticality networks, shall be capable of supporting a safe and a predictable message exchange between DAS with different criticality. Mixed-criticality networks are targeted as the natural replacement of traditional legacy buses due to the increasing amount of data that is required to be exchanged, the decrease of cost, the higher speed and the integration with existing network infrastructures. They can be divided into off-chip and on-chip networks with real-time and non-real-time features, where both can co-exist in one system. Off-chip networks are used for the interconnection of devices while on-chip networks are used for interconnecting the cores within a device. For example, EtherCAT is a real-time industrial Ethernet off-chip network. The use of internal network-on-chip systems shifts the problems associated with traditional networks into the chip.

However, the shift towards mixed-criticality networks poses many challenges related to the increasing demand for real-time, safety and security features in different application domains such as automotive and railway. To cope with those challenges *white channel* and *black channel* network approaches are defined by the IEC 61508-2 safety standard [IEC10b]. *White channel* networks (See Figure 2.4) are designed, implemented and validated according to the IEC 61508 [IEC10b, IEC10c] and IEC 61784-3 [IEC10g] or IEC 62280 [IEC02b] safety standards. In the case of the *black channel* networks (see Figure 2.5) it is assumed that not all parts of the communication channel are designed and validated according to the IEC 61508 safety standard. In that case, the safety-related components that compose the communication channel must implement IEC 61784-3 [IEC10g] or IEC 62280 [IEC02b] compliant measures and diagnostic techniques to ensure the failure performance of the communication process.

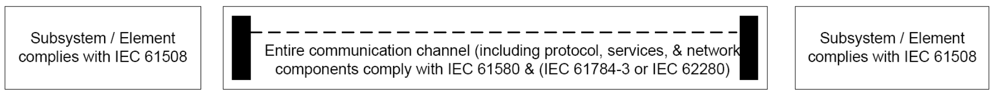
Subsystem / Element complies with IEC 61508
Entire communication channel (including protocol, services, & network components comply with IEC 61580 & (IEC 61784-3 or IEC 62280)
Subsystem / Element complies with IEC 61508

FIGURE 2.4: White channel approach. (Source IEC 61508-2 [IEC10b])

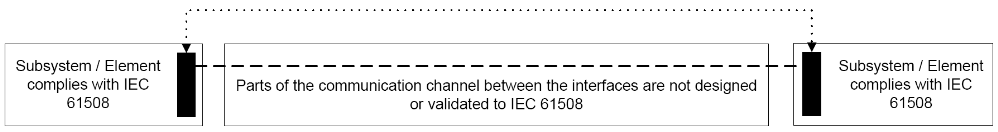
Subsystem / Element complies with IEC 61508
Parts of the communication channel between the interfaces are not designed or validated to IEC 61508
Subsystem / Element complies with IEC 61508

FIGURE 2.5: Black channel approach. (Source IEC 61508-2 [IEC10b])

2.11 Modular Safety Cases

A *safety case* [Kop11] represents an argument supporting the claim that the system is safe for a given application in a given environment [BB98]. It provides I) arguments to demonstrate that safety properties are satisfied and risk has been mitigated, II) a notation mechanism that is often required as a piece of the certification process and III) data interoperability among different standards (e.g., IEC 61508, ISO 26262) and application domains (e.g., avionic, automotive, railway).

A well-partitioned safety case limits the impact of changes to a reduced area of the safety case and enables the reusability of these areas. Partitioning is a complexity management technique [Kop08] that subdivides the system into smaller parts, more commonly referred to as modules, which are independently generated and used to compose a system. The modules can be generated with different levels of criticality such as SIL1 to 4 according to the IEC 61508 safety-related standard. On this basis, the implementation of MSCs enables the reusability of predefined modules, reducing the overall complexity (simplification strategy) and supporting the limitation of change impacts to specific modules.

2.12 Cross-Domain Mixed-Criticality Patterns

Cross-domain patterns are widely used and universal approaches for describing and documenting recurring solutions for design problems. They are used to guide and support engineers towards solutions that solve commonly occurring problems in the development of mixed-criticality systems from design to verification and validation. Two pattern approaches can be identified:

- *Traditional patterns:* Their representation derives from the definition of design patterns which include four essential elements: *name*, *context*, *problems* and *solutions*. In general, the traditional pattern representation consists of the following elements:

  - *Name:* A meaningful name for the pattern.
  - *Context:* Describes the preconditions or the situation in which the pattern can be used to solve a problem.
  - *Problem:* Describes the problem that is indented to be addressed by the pattern.
  - *Solution:* Defines the solution to the problem.

- *Custom patterns:* They are based on traditional patterns and differ from those in terms of different element naming and/or additional elements [HGZ^{+}13, AM95, Rub95, GHJV94, Ada95]. For instance, the *context* element can be subdivided into *Applicability* and *Preconditions* in a custom pattern representation. Most popular works on pattern representation of mixed-criticality systems are defined in [Dou99, Dou02] where the common patterns that deal with building safe and reliable architectures are presented. For example, patterns such as the single protected channel, homogeneous redundancy, triple modular redundancy, heterogeneous redundancy, monitor-actuator, watchdog and safety executive are introduced. Table 2.1 presents an example of a custom cross-domain pattern representation.

Pattern ID	
Pattern Name	
Related Pattern	
Type	
Context	
Problem	
Solution Under Consideration	

Board Name	
Implementation	
Results	
Additional Considerations	
References	

Table 2.1: Custom cross-domain pattern representation template.

Chapter 3

State Of The Art

This chapter analyses modular certification from the perspective of different safety standards and it describes notation languages for representing safety cases. A safety case is a documented body of evidences that provides convincing and valid arguments that a system is adequately safe for a given application in a given environment (e.g., automotive, railway, elevation) [BB98]. Section 3.1 presents the way in which the modularity methodology is supported by the IEC 61508 [IEC10a], ISO 26262 (Automotive) [ISO11], EN 50129 (Railway) [EN03] and DO-178 + IMA (Avionic) [RTC92] safety standards. Section 3.2 gives an overview of the prevalent safety case notation languages.

3.1 Modularity among Standards

3.1.1 IEC 61508

IEC 61508 is the international standard for functional safety of E/E/PE safety-related systems (see also Section 2.2). This standard is intended to be the basis for other domain-specific safety standards such as automotive, railway, machinery and others (see Figure 2.1). These safety standards consider the modularity, such as a method to develop, certify and re-use already certified and qualified components (e.g., actuators, sensors, communication networks, logic units/devices) for developing mixed-criticality systems. Figure 3.1 shows in detail the decomposition considered by this standard where the system can be divided into subsystems and the subsystems can be divided into elements.

At the beginning of this chapter, the concept of a safety case is defined. This definition can be extended to modularity, thus defining the concept of MSC. An MSC is a documented body of evidences that allows the assurance of the safety of a system that is composed of

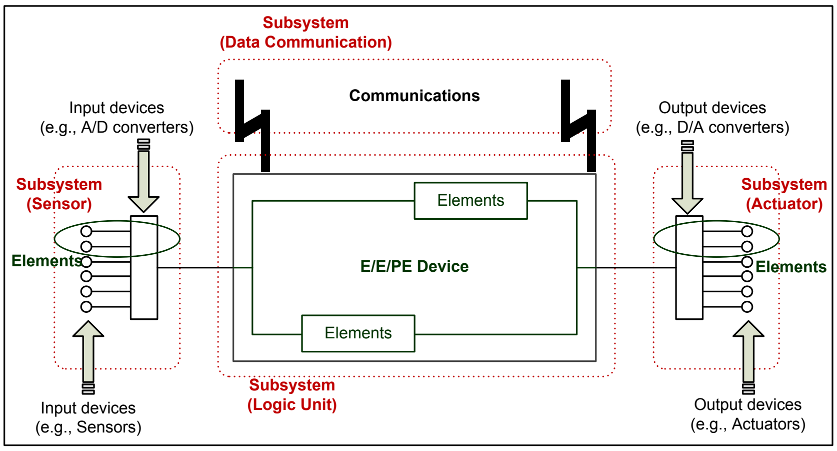

FIGURE 3.1: IEC 61508 – System hierarchy. (Source IEC 61508-1 [IEC10a])

modules with different criticality. According to the IEC 61508 safety standard an MSC shall cover the following aspects:

- Analysis of the system's safety-related needs.
- Strategies adopted to achieve the desirable SILs.
- Measures and diagnostic techniques for random and systematic faults.
- Evidences that demonstrate that the selected measures and diagnostic techniques are sufficient to fulfil safety needs.

Product families are another use case of modular certification. Products of the same product family can be developed using a set of pre-certified components. In addition, in the case that a safety requirement of the product changes, its re-certification according to the IEC 61508 safety standard might cut its cost since the modules affected shall only be certified. Modular certification implies the pre-certification of the modules according to IEC 61508 to achieve a certain Safety Integrity Level Claim Level (SILCL). The HW and SW components shall be developed and certified according to IEC 61508 to fulfil the required SIL.

3.1.2 ISO 26262

ISO 26262 [ISO09] is a functional safety standard for road vehicles. It defines the functional safety for automotive electrical and electronic safety equipment. This standard provides its own component model, where the terms item, system, component, HW part, SW unit and element are respectively defined (see Figures 3.2). An item is a system or an array of systems used to implement a function (e.g., a vehicle system). A system is a set of elements that includes at least a sensor, a controller and an actuator. An element is a system or a part of a system including components, HW, SW, HW parts and SW units. It is considered that an element is non-dissoluble when it includes HW parts or SW units. Instead, a dissoluble element can be labelled as a system, a subsystem or a component. A component is a non-system level element that is logically and technologically separable and is composed of more than one HW parts and/or SW units. A component is part of a system. On the other hand, a HW part is a HW which cannot be subdivided anymore, and a SW unit is an atomic level SW component - one or more SW units - of the SW architecture. This system architecture is shown in Figure 3.3.

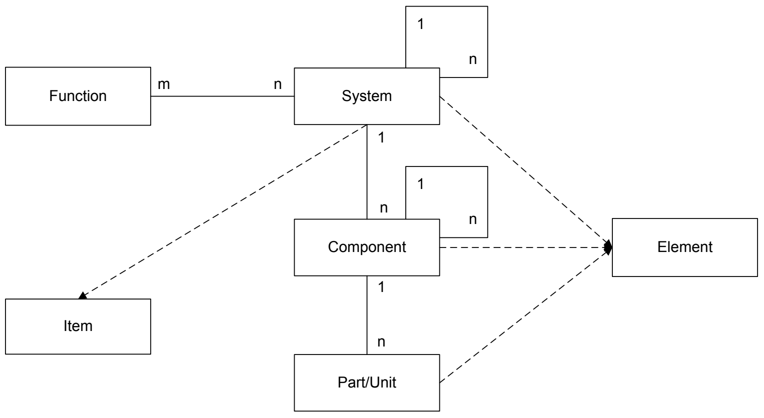

FIGURE 3.2: ISO 26262 – Relationship of items, system components, HW parts, SW units and elements. (Source ISO 26262 [ISO09])

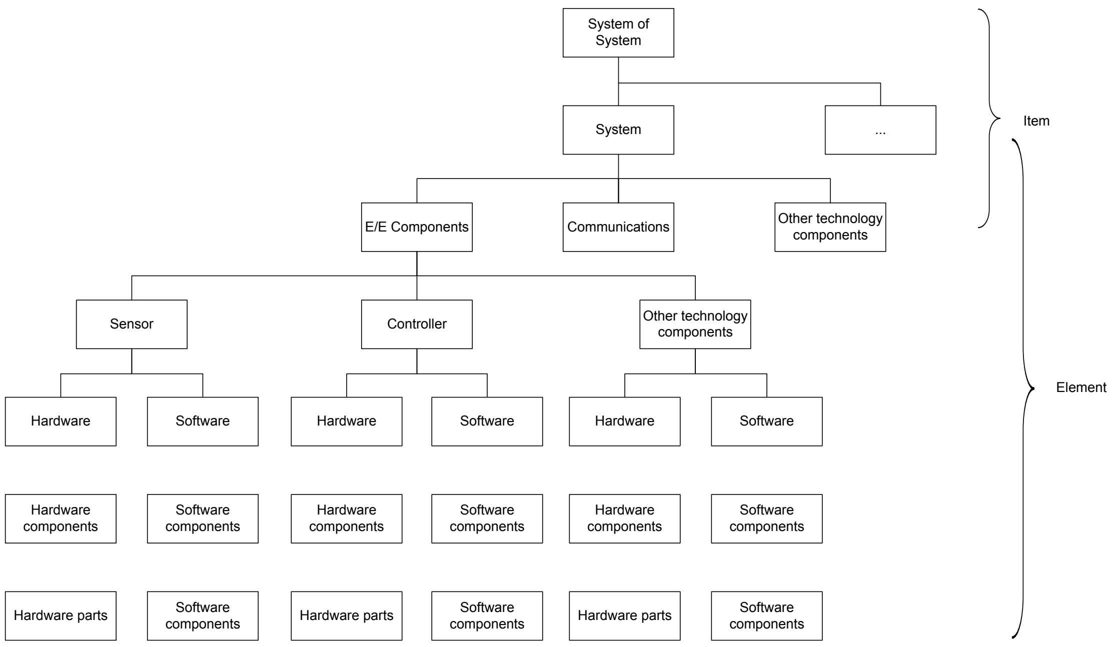

FIGURE 3.3: ISO 26262 – Item dissolution. (Source ISO 26262 [ISO09])

This safety-related standard also defines the term of Safety Element out of Context (SEooC). An SEooC is a safety element for which an item does not exist at the time of the development. An item can be either an element, a SW application or a HW component. The development of an element or a HW component out of context implies the replacement of the prerequisite work products by assumptions on ASIL A, B, C or D capability. A work product is the result of one or more associated requirements of the ISO 26262 standard. In that case, the design specification of the system (clause 7 of ISO 26262-4 [ISO15a]) and the technical safety concept (clause 6 of ISO 26262-4 [ISO15a]) that provide the ASIL attributes are replaced by assumptions. Similarly, the development of a SW out of context can also begin with the SW architectural design (clause 6 of ISO 26262-6 [ISO15a]) or the SW unit design and implementation (clause 8 of ISO 26262-6 [ISO15a]). In that case, the safety requirements of the SW (clause 6 of ISO 26262-6 [ISO15a]) and the architectural design specification (clause 7 of ISO 26262-6 [ISO15a]) can be replaced by assumptions.

3.1.3 EN 50129

EN 50129 [EN03] is the European railway standard that is intended for safety-related railway signalling systems and equipments. This standard defines the conditions and the development process for safety-related electronic railway systems and equipments to be accepted as adequately safe for their intended application. Safety cases evidence the acceptance of these systems and equipments. According to this safety-related standard, a safety case shall contain the definition of the system and the equipment, the quality management, safety management and technical safety reports, the related safety cases and the conclusions. The evidences that demonstrate the compliance of the system or equipment shall be included in the justification documents. These documents contain the following conditions for safety acceptance:

– *Evidence of quality management:*

  The first condition for safety acceptance refers to the quality of the system or equipment which should be controlled by the quality management team in the life-cycle (see EN 50126 [EN99]). These evidences shall be included in the *Quality Management Report*, which is part of the safety case documentation. The purpose of the quality management report is to minimise the number of human errors at each stage of the life-cycle, thus reducing the risk of systematic faults in the systems and equipments.

– *Evidence of safety management:*

  The second condition for safety acceptance is that RAMS manage the safety of the system (see EN 50126 [EN99]). The use of *Safety Management process* is mandatory for SIL1 to 4. Documented evidence to demonstrate the compliance with the safety management process throughout the life-cycle shall be provided in the *Safety Management Report*, which also forms part of the safety case. For example, safety plans for verifying that the phases of the life-cycle satisfies the safety requirements identified and for validating the system, subsystem, equipment against its original safety requirements specification.

– *Evidence of functional and technical safety:*

  In addition to the evidences of quality and safety management, technical evidences for safety design shall be provided in the *Technical Safety Report*. This document forms part of the safety case, and it is mandatory for SIL1 to 4. SIL0 is outside the scope of this safety standard. The headings of the technical safety report include an introduction, the assurance of correct functional operation, including the architecture, interfaces, fulfilment of requirements and assurance of correct HW and SW behaviour, the effects and faults, including random HW faults and systematic faults, the operation with external influences, thus demonstrating the operability and safety of the system or equipment, the safety-related application rules, conditions and constraints and the safety qualification test. For example, the environmental stress testing technique is executed and the results of the tests are verified.

– *Safety Assurance and Approval:*

  This sub-clause defines the safety acceptance and approval procedures for safety-related electronic systems. In addition, it considers the following three categories of safety cases:

  ⋆ *Generic Product Safety Case (GPSC):* A generic product can be re-used for different independent applications, e.g., a balise, a track circuit, a signalling system and a lineside electronic unit (LEU) of a train system.

  ⋆ *Generic Application Safety Case (GASC):* A generic application can be re-used for a range of applications with common functions. Examples include a radio block centre (RBC) and a driver machine interface (DMI) of a train system.

  ⋆ *Specific Application Safety Case (SASC):* A specific application presents the safety properties of a particular combination of products in a given application. An example is a train supervisor. This safety case shall be divided into the *application design* and *physical implementation* safety approvals. The application design safety case contains the safety evidences for the theoretical design of the specific application. On the other hand, the physical implementation safety case is the safety evidence for the physical implementation of the specific application. For example, safety evidences are provided for manufacture, installation, test, facilities for operation and maintenance.

There shall be two safety assessment reports for application design and physical implementation respectively. A safety assessment, in general, must determine whether verification and validation evidence that a system meets the specified requirements, form a judgement as to whether the system is fit for its intended purpose and determine if the components are compliant to the defined SIL.

3.1.4 DO-178 Integrated Modular Avionics (IMA)

DO-178C [RTC11] is an airborne standard approved by Radio Technical Commission for Aeronautics (RTCA), where SW considerations in airborne systems and equipment certification are defined. This standard defines a SW component like an atomic SW element that can be reused or used in conjunction with other components. Ideally, it should work without modifications and without the engineer needing to know the content and internal function of the component. However, the interface, functionality, pre-conditions and post-conditions, performed characteristics and required supported elements must be well known. DO-178C defines two types of components: modifiable and not modifiable components. A modifiable component is part of the SW that is intended to be changed by the user, whereas a non-modifiable component is not intended to be modified by the user. Non-modifiable components should be protected against modifiable components to prevent interferences in the safe operations of non-modifiable components. For example, HW or SW based protection mechanisms can be implemented for that purpose.

In some airborne systems and equipments, optional functions may be selected by SW options rather than by HW. Subsection 5.4.3 of DO-178 [RTC92] defines the considerations for deactivated code. This document also specifies the constraints for using COTS SW in airborne systems or equipments. Figure 3.4 illustrates the DO-178 compliant development process for the components of SW products.

– *Component W* implements a set of system requirements to define a SW design. This design is coded into source code, and then, it is integrated with the HW.

– *Component X* shows the use of previously developed source code in Component W that is employed in a certified aircraft engine.

– *Component Y* illustrates the use of a simple partitioned function that can be coded directly from the SW requirements.

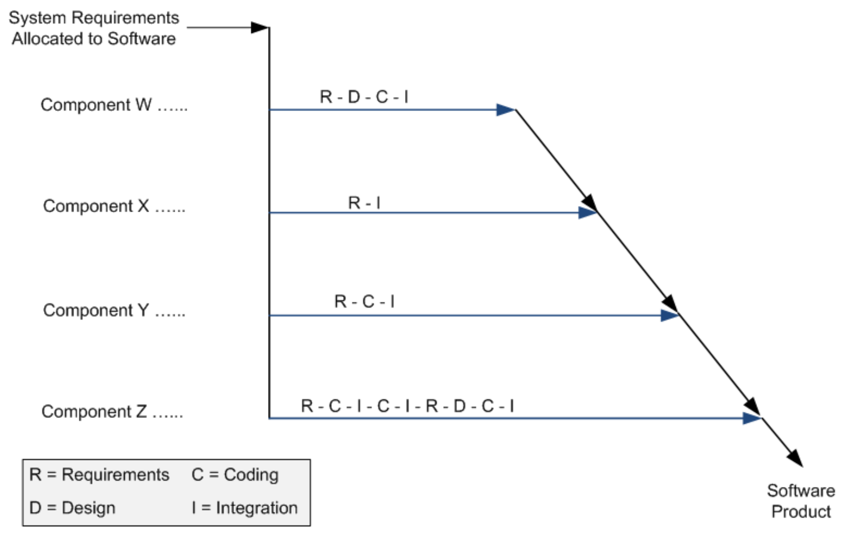

FIGURE 3.4: SW project development sequences. (Source DO-178C [RTC92])

– *Component Z* shows the use of a prototyping strategy. The goals of prototyping are to understand the SW requirements better and to mitigate the development and technical risks, through the continuous evaluation and continuous refinement of the SW project requirements.

3.2 Modular Safety Case Notation Languages

Evidence that a system is suitable for a specific purpose is a complex and essential requirements in today's mixed-criticality systems. Safety cases are usually required by safety standards and regulations as a means to present the safety-related requirements and supporting evidences for safety claims. For instance, safety cases are explicitly required by the IEC 61508 (E/E/PE functional safety systems), the DO-178C (avionic) and the ISO 26262 (road vehicles) industrial standards.

A safety case provides an explicit and definitive set of claims, arguments and evidences with different viewpoints, hierarchies and levels of details. Traditional safety cases are presented through a text notation, where the ways in which the safety requirements have to be interpreted and implemented in the system are specified. These approaches are effective for the notation of simple requirements. However, in the case of complex requirements, they often result in unclear, unstructured and ambiguous safety cases. In 1958 Toulmin [Tou58] developed an approach to overcome the problems that a textual notation presents. This notation language presents a graphical notation language for representing the hierarchical structure of the safety cases (see Figure 3.5). The Toulmin approach consists of the following six parts [Tou14]:

- **Data:** They are evidences used to support the arguments.
- **Claims:** They define the statements to be argued.
- **Warrants:** They are statements to connect the data and the claims.
- **Qualifiers:** They are statements to express the conditions under which the arguments are true or not.
- **Rebuttals:** They are restrictions which may be applied to the claim.
- **Backings:** They are credentials or justifications to certify the statement expressed in the warrants.

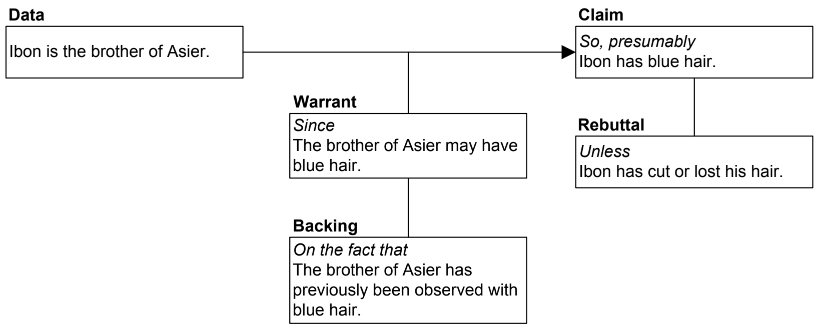
Data
Ibon is the brother of Asier.
Claim
So, presumably
Ibon has blue hair.
Warrant
Since
The brother of Asier may have blue hair.
Rebuttal
Unless
Ibon has cut or lost his hair.
Backing
On the fact that
The brother of Asier has previously been observed with blue hair.

FIGURE 3.5: Toulmin Notation Language – Example.

On the basis of the Toulmin approach, the GSN [Ade98, KW04, Kel07], CAE [Ade98, BB10, EC02] and Structured Assurance Case Metamodel (SACM) [OCM16] notation languages were developed. These notation languages are supported by SW tools like the ASCE [Ade98] that help to create and modify the claim structure, assist in the propagation of changes through the safety case and handle automatic links to the requirements. These tools also enable targeting problems with the Toulmin approach, managing information complexity and linking the arguments and the stakeholders.

3.2.1 Goal Structuring Notation Language (GSN)

GSN is an argumentation notation language developed by University of York in 1990 for the documentation of safety cases. In GSN the safety-related requirements of a system are expressed, as shown in Figure 3.6, by means of goal, strategy, solution, context, assumption and justification elements.

- **Goal:** It is the claim to be supported by sub-claims.
- **Strategy:** It is the nature of the argument that connects the claim and sub-claim.

– **Solution:** It is an evidence asserted to support the truth of the claim.

– **Context:** The context in which the claim should be implemented.

– **Justification:** Justification of claims or strategies.

– **Assumption:** Assumptions to hold valid the claims and strategies.

– **Undeveloped Entity:** It refers to elements requiring further decomposition.

– **Undeveloped Goal:** It refers to elements that require instantiation.

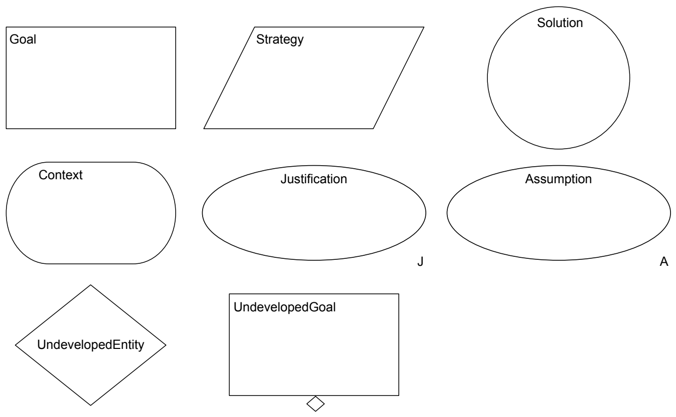

FIGURE 3.6: GSN Elements – Overview.

In addition, the GSN notation language has been adapted to support modular certification as defined in [OBM$^+$14], [Kel07] and [Kel01]. For that purpose, new representation elements such as the module extension and contract elements are included in the GSN representation. The module extension is a package of arguments to provide an abstract view of the argument structure. It is rendered as a rectangle with a second smaller rectangle that represents the reference to a module containing an argument (See Figure 3.7). On the other hand, the contract modules are introduced for describing the relationship between two or more modules, defining how a claim in one supports the argument in the other.

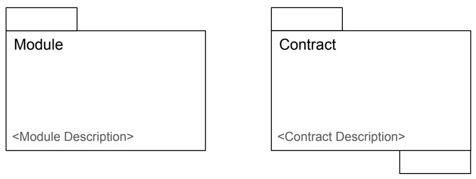

FIGURE 3.7: GSN – Modularity extensions.

3.2.2 Claim, Argument, Evidence Notation Language (CAE)

The CAE notation language has been developed by Adelard as a simple and effective way to define safety cases. These safety cases are composed of claims, arguments and evidences (see Figure 3.8).

- **Claim:** It is the assertion of one or more arguments that can be assessed to be true or not (e.g., the system is adequately safe).
- **Argument:** It describes the content presented to support the claims and links claims to evidences.
- **Evidence:** It is the reference presented in support of claims and arguments (e.g., the Functional Safety Management (FSM) deliverables).

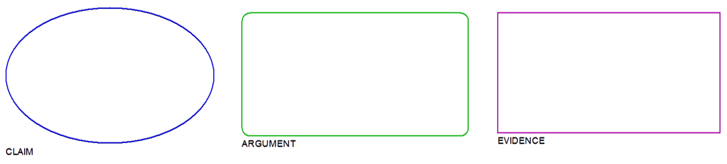

FIGURE 3.8: CAE Elements – Overview.

These arguments and evidences can be enriched with the following custom tags for easing their classification:

Arguments:

- *[Safety – Standard]:* The argument is based on a safety standard.
- *[Safety – Function]:* The argument defines a safety-related function.
- *[Safety – Technique]:* The argument defines a safety technique, e.g. diagnosis technique and system reactions to errors.
- *[Safety – Assumptions]:* The argument defines a set of safety-related assumptions that must be met, e.g. constraints and hypothesis of use.
- *[Recommendations]:* The argument defines a set of recommendations.

Evidences:

- *[Deterministic]:* The evidence is deterministic, e.g. a safety manual is provided.
- *[Probabilistic]:* Quantitative statistical reasoning (numerical level).
- *[Qualitative]:* Compliance with rules (i.e., standards).
- *[Export]:* An external entity provides the evidence (e.g. system architect or third party item provider).

3.2.3 Structured Assurance Case Metamodel (SACM)

SACM [OCM16] combines the Argument Metamodel (ARM) and Software Assurance Evidence Metamodel (SAEM) Object Management Specifications (OMG) specifications. Furthermore, it harmonises the common elements of the GSN and CAE notation languages, resulting in a formal model with structured graph of assertions (claims) ultimately supported by an evidence repository. The evidence repository expresses attributes about the SW artefacts and the relations between them (i.e., containment, the specification of models, library dependencies, versioning).

Chapter 4

DREAMS Architecture Style

The DREAMS research project [DRE13] aims at developing a cross-domain real-time architecture and design tools for complex networked systems where application subsystems of different criticality executing on networked multi-core chips are supported. This project delivers architectural concepts, meta-models, virtualization technologies, model-driven development methods, tools, adaptation strategies and validation, verification and certification methods for the seamless integration of mixed-criticality to establish security, safety and real-time performance as well as data, energy and system integrity. Furthermore, it defines a cross-domain system architecture of a hierarchical distributed platform for mixed-criticality applications combining the logical and physical views (see Figure 4.1). This system design enables tackling the development and certification challenges of today's mixed-criticality systems.

Logically, the architecture style of DREAMS consists of heterogeneous application subsystems with different criticality levels (SIL1 to 4 according to IEC 61508), timing (firm, soft, hard and non-real-time) and computation models such as TT messages, data-flow and shared memory. Application subsystems can have contradicting requirements for the underlying platform such as different trade-offs between predictability, safety and performance in processor cores (i.e., Zynq-7000 processor), hypervisors (i.e., XtratuM hypervisor), operating systems (i.e., Windows CE) and networks (i.e., on-chip and off-chip networks). They can be further split into SW components (e.g., diagnosis partitions and safety protection partitions which are responsible for executing a safety state in the case of a failure).

The platform architecture proposed in DREAMS provides resources such as global and local resource management units (Global Resource Manager (GRM) and Local Resource Manager (LRM)) for executing multiple application subsystems and their components in different execution environments. Blocks highlighted in grey in Figure 4.1 are *core*

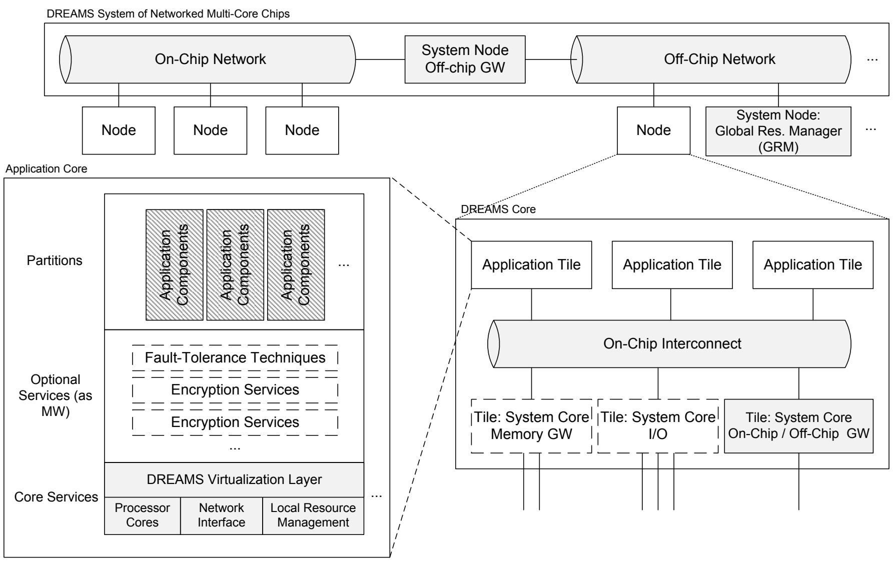

Figure 4.1: DREAMS Architecture – Overview. (Source [DRE13])

platform services, the dotted ones are the *optional platform services* and the block with dark stripes are the *application-related platform services.* Partitioning establishes this system perspective, enabling the decomposition of the system into multiple application subsystems which can be independently certified to the respective level of criticality. Partitioning is a prerequisite for modular certification.

The HW architectural style proposed by the European project DREAMS ensures determinism and temporal independence to simplify the timing and resource analysis. Temporal predictability and low jitter also promote the quality of control of mixed-criticality systems. This project considers the IEC 61508 safety standard for its development, although it also considers the security [ISO15c] and timing standards [USE12]. These standards define different lifecycle development processes for developing safe, secure and real-time mixed-criticality systems. The development processes for secure, and real-time systems are not included in this thesis because they are out of the scope of in this dissertation. The second objective of this project is to reduce the overall cost required for developing and certifying mixed-criticality solutions. This thesis has been developed within the European DREAMS project and therefore, it pursues the same objectives. For that purpose, it implements a general optimisation strategy towards the selection of safety components such as the ones provided by a hypervisor, a COTS multi-core device and the on-chip and off-chip mixed-criticality networks. In addition to the significantly decreased costs in the development lifecycle, deployment and maintenance, the modularity methodology is applied and reusable generic cross-domain patterns are defined and implemented. These patterns allow faster design and higher applicability to different application fields and they guide and support engineers towards solutions that solve commonly occurring problems in the development of mixed-criticality products.

4.1 Safety Development Process for Mixed-Criticality Systems

The engineering of safety-related systems implies strict development processes to attain certification. For example, the IEC 61508 safety standard defines a safety lifecycle development process for E/E/PE functional safety systems [IEC10a]. A safety lifecycle is an engineering process that sets out a set of steps for achieving the required functional safety. It is used to develop and document the safety plan, execute that plan and document its execution. In this section, the IEC 61508 compliant development process is followed as the basis for defining the development process for safety shown in Figure 4.2.

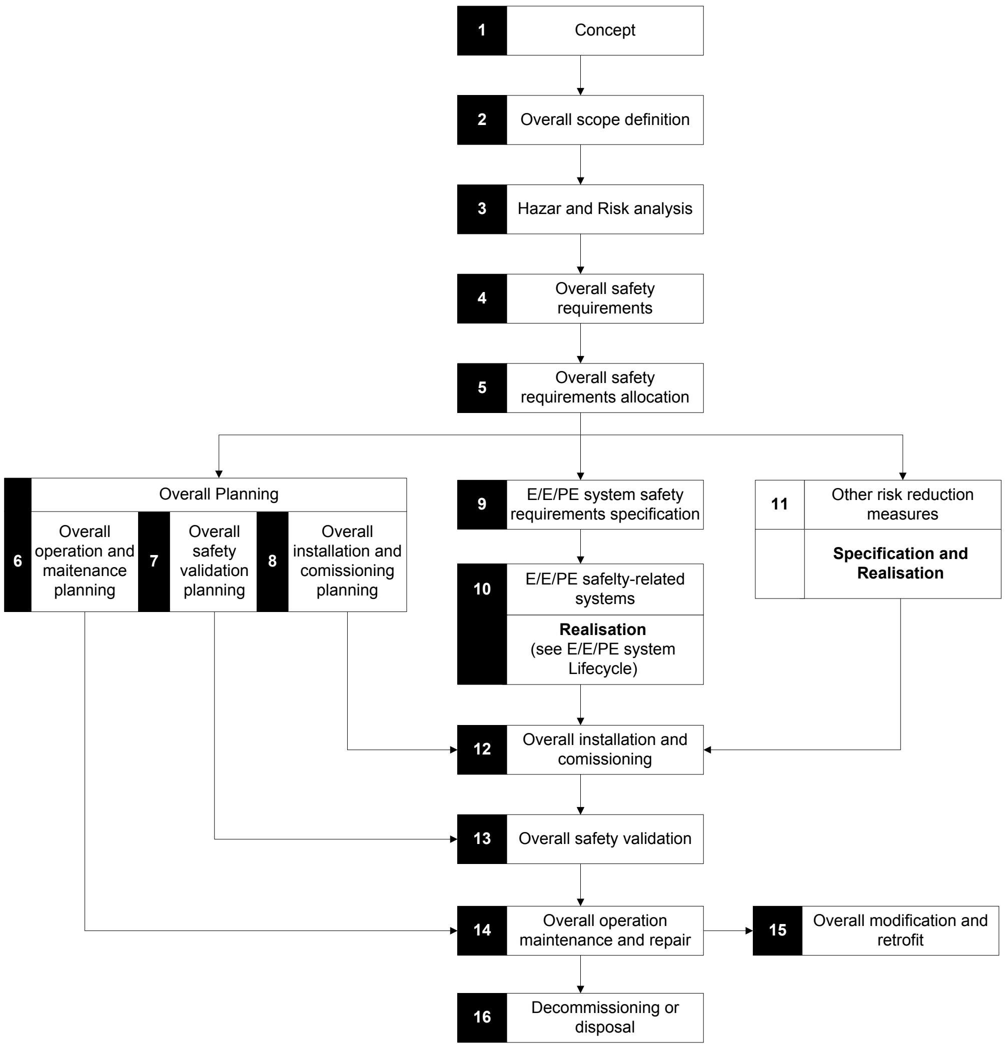
1 Concept
2 Overall scope definition
3 Hazar and Risk analysis
4 Overall safety requirements
5 Overall safety requirements allocation
Overall Planning
6 Overall operation and maitenance planning
7 Overall safety validation planning
8 Overall installation and comissioning planning
9 E/E/PE system safety requirements specification
10 E/E/PE safelty-related systems
Realisation (see E/E/PE system Lifecycle)
11 Other risk reduction measures
Specification and Realisation
12 Overall installation and comissioning
13 Overall safety validation
14 Overall operation maintenance and repair
15 Overall modification and retrofit
16 Decommissioning or disposal

FIGURE 4.2: IEC 61508 – Life cycle. (Source IEC 61508-1 [IEC10a])

Phases 1 "*Concepts*" and 2 "*Overall scope definition*" entail the safety implications of the actions related to the Electronic Under Control (EUC). More specifically, the objective of the first phase is to develop a level of understanding of applications of the EUC and its environment (physical, legislative) to satisfy complete the other safety lifecycle activities. The second phase aims to determine the boundary of the applications of the EUC and the EUC control system and specify the scope of the hazard and risk analysis (e.g., process hazards, environmental hazards), taking as input the information from phase 1. In phase 3 the hazard and risk analyses are performed to determine the hazards, hazardous events and situations related to actions of the EUCs. The defined scope of the hazards and risk analysis are considered as a starting point in this phase. Phase 4 specifies the overall safety requirements regarding safety functions and safety integrity requirements. For that purpose, the information related to the hazard and risk analysis are taken from phase 3. In phase 5 "*Overall safety requirements allocation*", the overall safety requirements of phase 4 are translated to design the safety functions.

The design of safety functions is dependent on the implementation in phases 9, 10 and 11. Phase 9 defines the E/E/PE system safety requirements in terms of the safety function and safety integrity requirements to achieve the required functional safety. Phase 7 provides the inputs for this phase, and its outputs are used for developing the E/E/PE safety-related system in phase 10. Phase 10 defines the life cycle for E/E/PE functional safety systems shown in Figure 4.3. The continuous arrows of the figure define the dependencies between the phases of the development process. At the end of each step, a verification process against the results of the next step (broken vertical arrows) is executed to check the consistency of the steps' I/Os.

This development process is split into a *design branch* and a *testing branch* (green arrows). The design branch covers the development phases related to the definition and design of the HW and/or the SW. Instead, the testing branch covers the phases related to the integration, verification and validation activities of the HW and the SW. The design and testing branches are linked to the test plans, which can differ depending on the implementation in HW or SW. For instance, the development of HW requires the implementation of measures and diagnostic techniques recommended by IEC 61508-2 [IEC10b], whereas the development of SW requires the implementation of measures and diagnostic techniques recommended by IEC 61508-3 [IEC10c]. The solid arrows on the left side of each *test plan* indicate the origin of the test plans while the solid arrows on the right side denote the actions which are required to be performed during the testing. The dashed lines to/from the test plans denote the verification activities between the phases.

Phase 9 defined before comprises the safety-related requirements for the HW and the SW and sets the validation plan that shall be followed in sub-phase 10.8 for validating the entire HW or/and SW system. It helps in the system requirements gathering and classifying. Sub-phase 10.1 defines the HW and SW system modelling and architecture

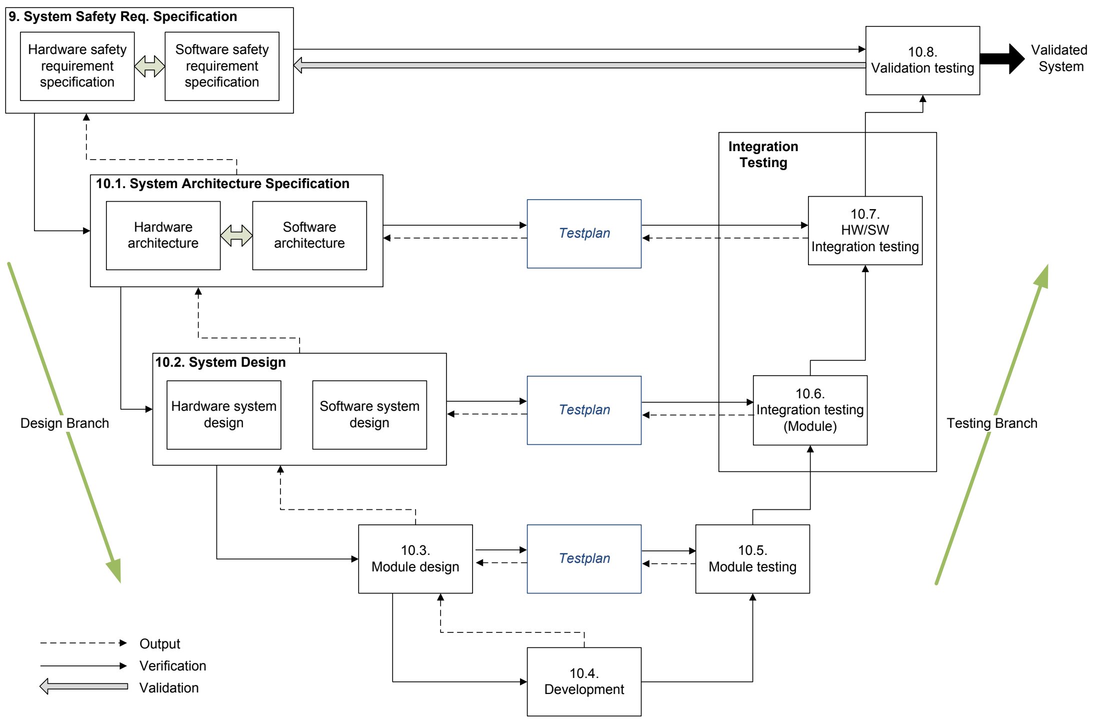

FIGURE 4.3: Safety Life cycle – The V-model.

definition. This sub-phase also defines the integration tests that shall be executed in the HW or/and SW integration testing (Sub-phase 10.7). Sub-phase 10.2 defines the system architecture design and specifies a verification plan for the modules of sub-phase 10.6. In addition, sub-phase 10.3 describes the way in which each component of the HW (module or part) and/or SW system shall be implemented, the communication protocols to be used by the components, configures the SW components associated with the HW architecture and defines the verification plan for testing the modules in sub-phase 10.5. These HW and SW modules, and attached interfaces (application programming interfaces (APIs)) are developed/implemented in sub-phase 10.4. Once the HW and/or SW modules are developed/implemented, the testing branch starts. This branch tests the modules developed/implemented in the design branch. To that end, sub-phase 10.5 uses the test plan defined in the linked sub-phase. Sub-phase 10.6 and 10.7 verify the integration of the modules and the HW and/or the SW using the verification plans established by the sub-phases of the design branch. Finally, in sub-phase 10.8 the entire system is validated according to the validation plan defined in phase 9 to check the system's functionality.

On the other hand, the overall safety context reflected in phases 6, 7 and 8 shall be deemed to ensure a safe development process. Phase 6 develops a plan for operating and maintaining the E/E/PE safety-related system, taking the safety requirements allocation document as its input. Phase 7 defines a plan for the overall safety validation of the E/E/PE safety systems. The inputs of this phase include outputs of phases 4 and 5. Phase 8 "*Overall installation and commissioning planning*" develops a plan for the installation and commissioning of the E/E/PE safety system ensuring the achievement of the required functional safety. This phase takes the overall safety requirements document (phase 5) as an input. Further phases, phases 12-16 are executed when the system has been built. In phase 12 the E/E/PE system is installed and commissioned, whereas, in phase 13, the E/E/PE system is checked to verify that the safety-related requirements are identified and handled during the building and installation before the system is put into operation. Maintenance activities are carried out in phase 14. These activities are usually performed at run-time. It is also foreseen that an E/E/PE system can be modified during run-time, thus requiring the overall modification and retrofit of the safety-related functions. The conditioning and modification processes are carried out in phase 15 of the development process. Finally, in phase 16 "*Decommissioning or Disposal*" the definition of the procedures to assure the adequacy of the E/E/PE system during and after the decommissioning or disposing of the Electronic Control Unit (ECU) is carried out. If another safety lifecycle is required, it should be specified as part of the functional safety management activities, and it should fulfil the requirements defined by the IEC 61508-1-2-3 safety-related standard [IEC10a, IEC10b, IEC10c]. It is assumed that if at any phase of the safety lifecycle, a modification is required, then an impact analysis shall be carried out for defining the areas of impact and which earlier safety lifecycle activities shall be repeated.

The development process based on the IEC 61508 safety standard presented in this chapter is not intended to develop HW in its particular context. Instead, it aims to develop SW. However, this standard also defines that the implementation of communication systems inside a Field Programmable Gate Array (FPGA) (e.g., NoC) can be considered as HW. Therefore, mixed-criticality systems that contain programmable logic applications shall be compatible with both HW and SW developments. For instance, the architecture of the European project DREAMS presented in this chapter shall be consistent with both HW and SW development processes as it uses a standard manufactured multi-core processor, implements a hypervisor for partitioning the cores of the processor and uses a mixed-criticality network for communication between the partitions.

4.1.1 Meet-in-the-Middle Methodology

The meet-in-the-middle methodology is considered as a successive refinement method that combines *top-down* and *bottom-up* methodologies to converge on HW and/or SW solutions. A platform based design is a meet-in-the-middle process emphasising systematic reuse for developing complex products based upon platforms and HW and SW virtual components, intended to reduce development cost and time to market [SVM01]. The meet-in-the-middle approach is not a *top-down* process where the SW is designed in the first place, and the HW is developed in the second place [FJKM11]. Its development strategy offers a middle ground between the *top-down* and *bottom-up* methodologies attempting to take advantages of those approaches while attenuating some of their most notable risks and problems [PvdH06]. For instance, the *top-down* methodology may converge to no concrete solutions, whereas a *bottom-up* method fights against the increase of the simulation time of today's complex models. From a safety perspective, according to the IEC 61508 safety standard, the following two requirements may be implemented to show the absence of systematic faults [IEC10b].

1. *Meet the requirements of Route 2_S or Route 3_S. Route 2_S "Proven-in-use approach"* establishes the compliance with the requirements of proven in use components. It defines that an component shall only be regarded as proven-in-use when it has a clearly restricted and specified functionality and when the absence of systematic faults is demonstrated (see Subsection 7.4.10 of IEC 61508-2 [IEC10b]). On the other hand, *Route 3_S "Pre-existing SW"* establishes the compliance with the requirements of IEC 61508-3, including the requirements for pre-existing and re-used SW.

2. *Provide a safety manual* that includes a precise and complete description of the pre-existent components, enabling the assessment of the integrity of a specific safety function that depends wholly or partly on the pre-existing SW components (see Annex D of IEC 61508-2 [IEC10b] and IEC 61508-3 [IEC10c]).

Figure 4.4 shows the meet-in-the-middle methodology where it applies a top-down design (application design) for a high level of abstraction and implements a bottom-up design for a low level of abstraction (platform design). These two methodologies converge at one

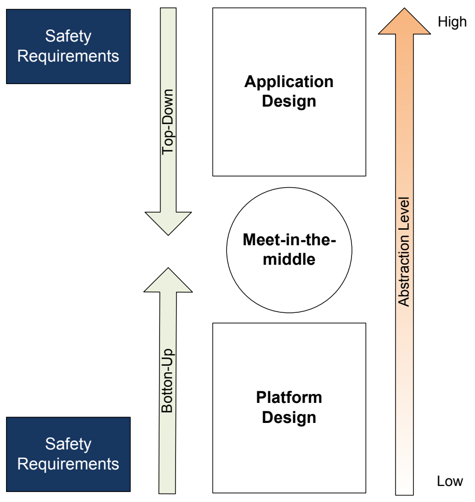

FIGURE 4.4: Safety approach – Meet-in-the-middle methodology.

point, the meet-in-the-middle point, where the platform is ready to host an application and where the application is ready to be hosted on a platform. From a HW perspective, this methodology supports the bottom-up approach (low to high abstraction level), thus enabling the adaptation of HW at design time and run-time using dynamic and partial reconfiguration.

The meet-in-the-middle methodology is applied to mixed-criticality system architectures (e.g., research project CONTREX [TOG+14]) for the integration of the design environment, models, analyses and simulation tools.

4.1.2 Traceability

The traceability is the impact analysis that enables to check that the decisions taken at earlier stages are adequately implemented (forward traceability) and that the decisions made at later stages are required and stated by previous decisions. According to the IEC 61508 safety standard, the following traceability techniques can be implemented during the development of a mixed-criticality system:

– **HW traceability** shall be computer-aided and it shall be based on defined methods (see Table B.6 and Subsection 7.2.2.2 of IEC 61508-2 [IEC10b]). It is placed among the specification, design, circuit and parts phases.

– **SW traceability** is placed among the phases of the development process and shall be compliant to IEC 61508-3 [IEC10c].

A prerequisite for traceability is that the requirements for the HW and the SW must be clearly identified, using unique identifiers.

4.1.3 Modularity

The component based design or modularity enables partitioning the system into smaller comprehensible parts, also called modules. The modules can be independently developed with different criticality levels (e.g., SIL1 to 4 according to the IEC 61508 safety standard) and be re-used for developing different system architectures. This design approach limits the complexity of the system and follows a joint development process (such as the IEC 61508 compliant development process).

Figure 4.5 presents the modular V-model development process based on the meet-in-the-middle methodology. The right branch of the model implements a top-down method (System of System (SoS) layer to component layer) while the left branch implements a bottom-up methodology (component layer to SoS layer). These branches contain different abstraction layers such as the SoS, system, subsystem and component layers, where each layer shall follow its development processes.

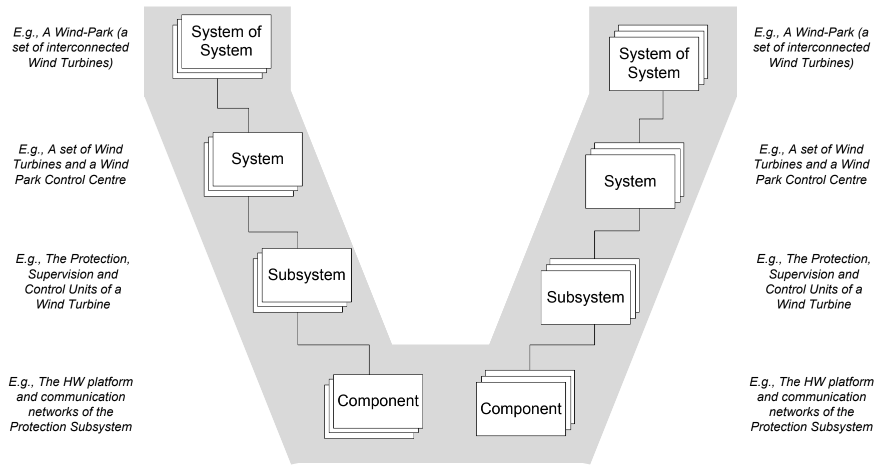
E.g., A Wind-Park (a set of interconnected Wind Turbines)
System of System
E.g., A set of Wind Turbines and a Wind Park Control Centre
System
E.g., The Protection, Supervision and Control Units of a Wind Turbine
Subsystem
E.g., The HW platform and communication networks of the Protection Subsystem
Component
Component
Subsystem
System
System of System
E.g., A Wind-Park (a set of interconnected Wind Turbines)
E.g., A set of Wind Turbines and a Wind Park Control Centre
E.g., The Protection, Supervision and Control Units of a Wind Turbine
E.g., The HW platform and communication networks of the Protection Subsystem

FIGURE 4.5: Safety approach – Modularity.

Modularity methodology is applicable to mixed-criticality systems such as a wind park system based on the DREAMS architecture style. A wind park usually can be split into a set of wind turbines and a wind park control centre, where, a wind turbine contains protection, supervision and control units. Consequently, the wind park can be considered as a SoS while the wind turbines and the control centre are considered such as systems. In addition, the protection, control and supervision units that compose the wind turbine can be regarded as such subsystems. These units can also be decomposed into a set of components such as COTS multi-core processors, virtualization mechanisms, mixed-criticality networks, operating systems and SW. The subsystems and components can be in turn systems and subsystems in their own right. All these abstraction layers shall follow a safety development process such as the development process presented in this chapter to attain certification.

The development of products with the system architecture presented in the Figure 4.6 can be eased by using already certified components, also known as *compliant items*. These items can be reused and integrated for developing different products. They shall be backed by their respective certificates and a set of documents that ensure that the integration of the component is carried out according to the specifications for safety. Figure 4.6 presents the development process based on a compliant item where certified components may be re-used for developing diverse HW and/or SW, thus reducing the development time and cost. For example, an IEC 61508 compliant control system of a wind turbine could be re-used from the compliant item pool to develop a wind park, and a certified COTS multi-core device (such as the Hercules multi-core device [Ins13] or the Zynq-7000 multi-core device [XIL14b]) could be used from the compliant item pool for developing a control system.

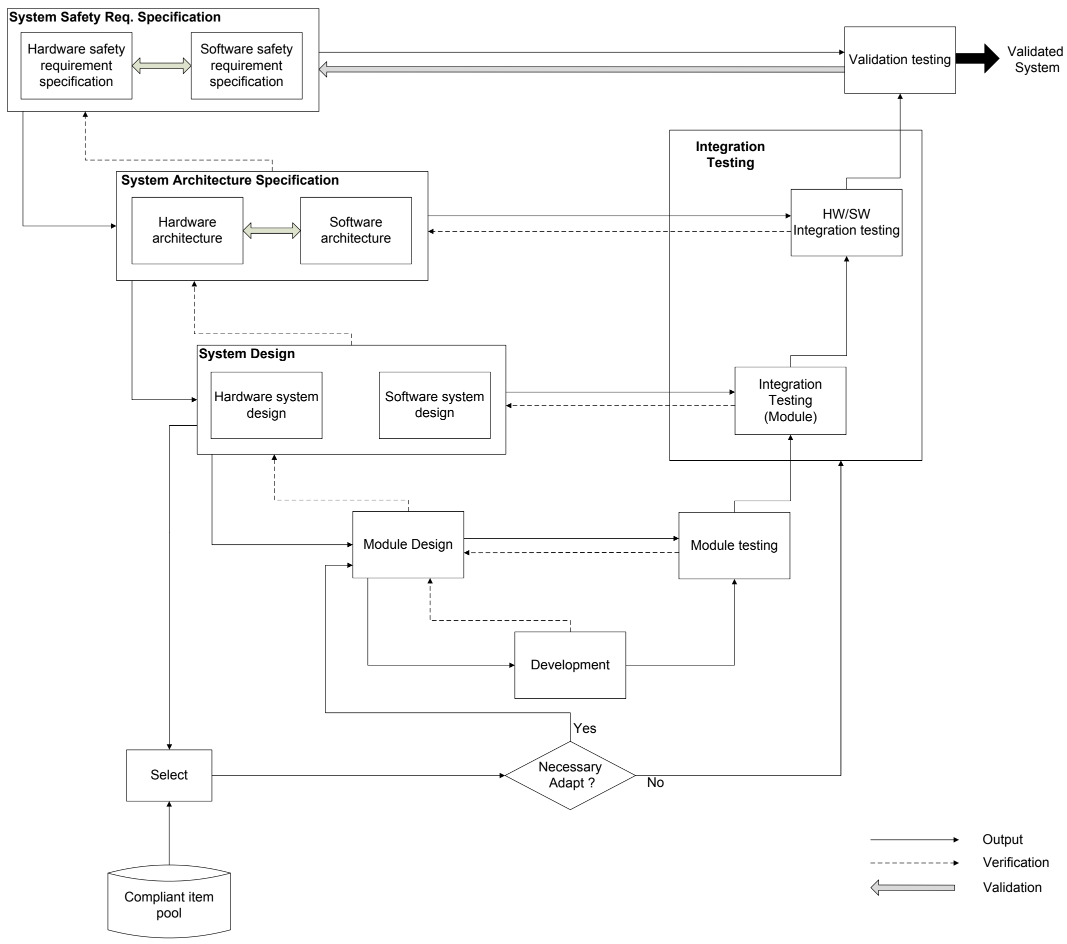

Figure 4.6: Component based development process – Safety approach.

The same figure represents the adaptation process that is required by the compliant items to fulfil the requirements defined at design time. For example, imagine that we re-use a pre-certified COTS multi-core device from the compliant item pool and that we aim to connect it through a PCIe bus to another device. However, imagine that the re-used multi-core device does not implement a PCIe interface, thus precluding the communication between the devices. Consequently, an adaptation strategy shall be executed for achieving the communication between the devices. In this case, a PCIe bus adapter should be required for establishing the communication between both devices.

Chapter 5

Modular Safety Cases

Mixed-criticality systems are complex systems that enable to combine on the same computing platform functions with different criticality. These systems aim to provide high performance and low power consumption, weight, size and cost and improve the reliability and scalability. However, they are commonly subject to internal and external interferences in temporal and spatial domains that significantly increase the development and certification cost. For instance, they suffer temporal interferences due to communication errors and spatial interferences due to resource sharing.

A typical certification process implies the certification of the whole system, where if a safety aspect of the system changes, the entire system shall be re-certified. This traditional certification process, which is also presented in Chapter 3, increases the final cost and time-to-market for developing and certifying mixed-criticality systems. The modularity methodology appears as a way forward to tackle the challenges related to the traditional certification. This method enables the division of the system into different application components and platform services, also called modules, and enables reducing the development and certification cost. A module can be implemented independently

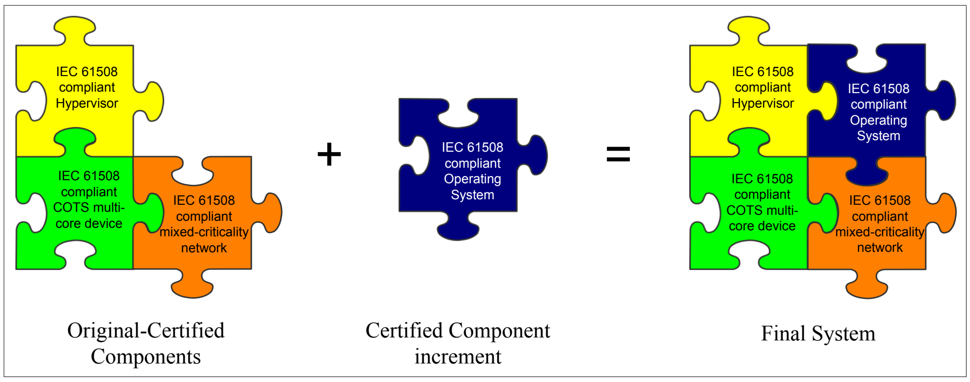
IEC 61508 compliant Hypervisor
IEC 61508 compliant COTS multi-core device
IEC 61508 compliant mixed-criticality network
+
IEC 61508 compliant Operating System
=
IEC 61508 compliant Hypervisor
IEC 61508 compliant Operating System
IEC 61508 compliant COTS multi-core device
IEC 61508 compliant mixed-criticality network
Original-Certified Components
Certified Component increment
Final System

FIGURE 5.1: Modular Certification – Overview

and re-used for developing different mixed-criticality system architectures. For example, in a perfect world, such as shown in Figure 5.1, a set of certified modules (compliant items) (e.g., a hypervisor, a COTS multi-core device and a mixed-criticality network) may be used in conjunction (such as jigsaw pieces) with a certified operating system to compose a certified mixed-criticality system.

The certification process of safety-related systems is usually carried out through safety cases. A safety case is a documented body of evidences that provides convincing and valid arguments that a system is adequately safe for a given application in a given environment (e.g., automotive and railway) [BB98]. In the modular domain, the safety-related arguments are commonly represented by means of MSCs. An MSC is a safety case that enables the reusability of predefined modules, reducing the overall complexity (simplification strategy) and limiting the impacts of changes to specific modules of mixed-criticality systems. Nowadays, the predominant safety case notation languages for representing safety cases are the GSN [KW04, Kel07] and the CAE [BB10, BB98, EC02]. These notation languages are introduced in Section 3.2.

This chapter defines the generic MSCs for fundamental platform services for mixed-criticality systems such as a hypervisor, a safety partition, a COTS multi-core device and a mixed-criticality network. CAE and IEC 61508 are taken as the reference safety-case notation language and safety standard for defining the MSCs. Section 5.1 defines the MSC for an IEC 61508 compliant generic hypervisor, whereas Section 5.2 defined the MSC for an IEC 61508 compliant partition. Section 5.3 defines the MSC for an IEC 61508 compliant COTS multi-core device and Section 5.4 defines the MSC for an IEC 61508 compliant mixed-criticality network.

5.1 A Modular Safety Case for an IEC 61508 compliant Generic Hypervisor

This section presents an MSC for an IEC 61508 compliant generic hypervisor positively assessed by a certification body within the European project DREAMS [LPA+15]. Figure 5.2 presents the MSC and the safety-related arguments that a safety hypervisor must meet to be compliant with the IEC 61508 safety standard. These safety arguments are enriched with several custom tags presented in Subsection 3.2.2 and they may be fulfilled by different safety hypervisors using a variety of strategies and mechanisms.

In this section, the term *hypervisor* is used as the abbreviation of the expression *safety hypervisor*, which is intended to be used in an IEC 61508 compliant safety-related mixed-criticality system. This MSC considers several significant constraints and hypothesis such as that the hypervisor manufacturer provides the safety-related hypervisor 'as it is'. This term means that the hypervisor can be provided by the manufacturer in compliance with a safety standard or not for fail-safe systems. A system is considered fail-safe when in the event of a failure, it can reach a safe state (e.g., stop the execution). For example, a train is a fail-safe system because, in the case of a failure, it can be switched into a safe state. This term is further defined in Subsection 2.2.1. Furthermore, it is assumed that a safety hypervisor follows safe installation procedures, which are contained in the hypervisor's documents. These documents include information regarding the qualified tools used by the hypervisor, the minimum knowledge of the operators, the measures and diagnostic techniques implemented, the failure modes of the hypervisor and the Safety Related Application Conditions (SRAC). The safety installation procedures enable the safe upload and non-volatile storage of the executable files (binary files) of the hypervisor into an ECU. If the *failure-modes* of the safety hypervisor are not documented, it is assumed that the hypervisor fails in an arbitrary way when it is affected by a fault. The SRAC list defines the imported and exported requirements of the safety hypervisor such as the timing requirements, the performance, the resources required and their associated restrictions.

The first two arguments defined in this MSC address the development of a safety hypervisor based on an IEC 61508 compliant *development process* (see Section 4.1) and *qualified tools*. These tools shall be used to reduce the likelihood of introducing or not detecting faults in the safety development process. According to the IEC 61508 safety standard, qualified tools can be classified as off-line or on-line tools. On-line tools can influence in the safety-related systems during their run-time, whereas off-line tools cannot affect the run-time behaviour. Off-line tools can be re-categorized as T1, T2 or T3 depending on their purpose of usage [IEC10d]. T1 tools do not generate outputs which can contribute to the executable code of the safety-related systems (e.g., text editor with no automatic code generation capability such as a configuration control tool). T2 tools support the test or verification of the design or the code, where errors in the tool can fail to reveal defects of the safety-related system without creating errors in the executable software (e.g., a test harness generator and a test coverage measurement tool) and T3 tools generate outputs that can contribute to the executable code of the safety-related system (e.g., a compiler).

In relation to the *safe startup and initialization* of a safety hypervisor, they shall be provided in a known initial state, in a repeatable way and at a specified time window. The startup and initialization time of the hypervisor and the time required for configuring it shall also be considered for achieving determinism. It is assumed that before the startup sequence, a HW based diagnosis and the diagnosis of the hypervisor's binaries are executed. Concerning the *shutdown* sequence of an IEC 61508 compliant hypervisor,

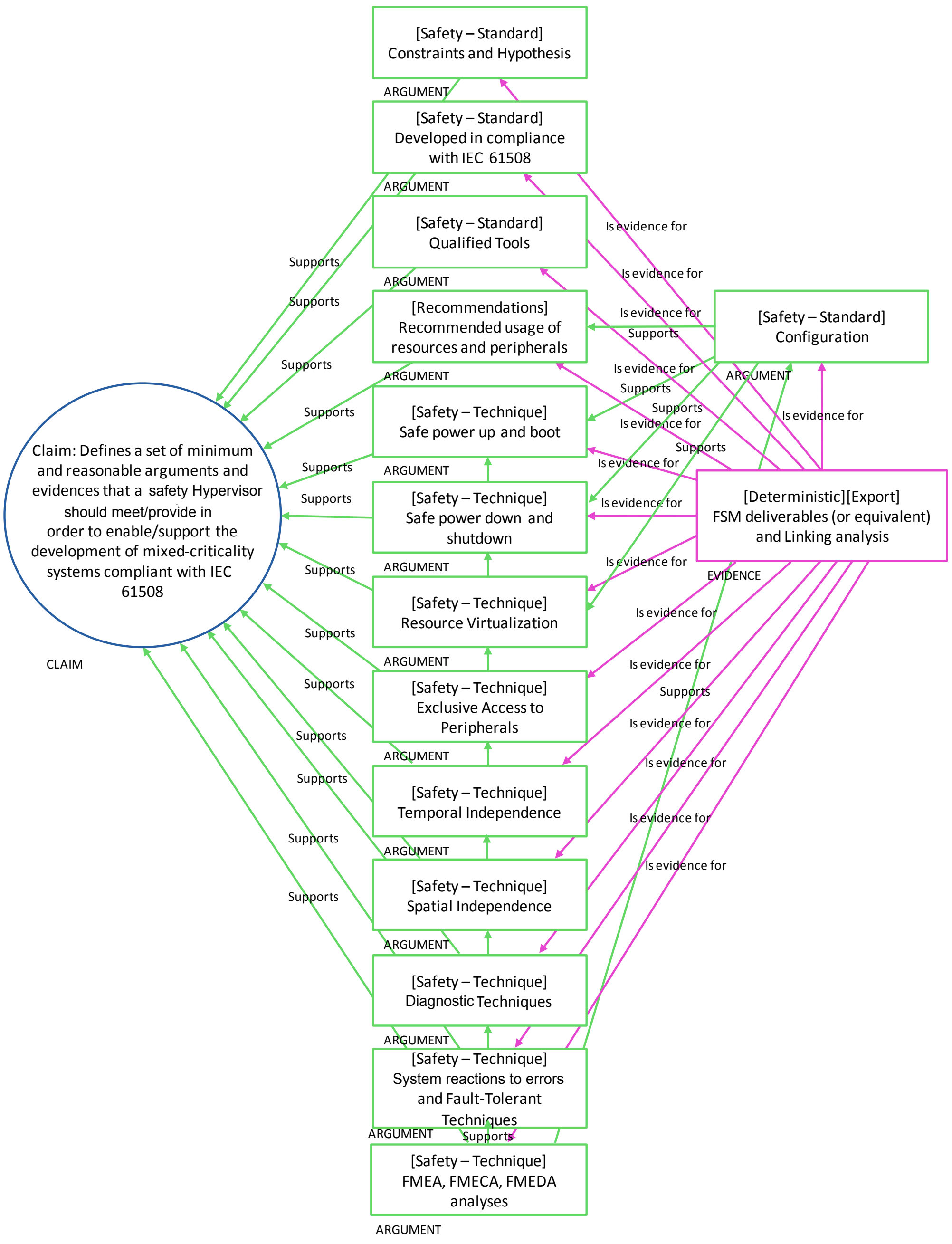
[Safety – Standard] Constraints and Hypothesis
ARGUMENT
[Safety – Standard] Developed in compliance with IEC 61508
ARGUMENT
[Safety – Standard] Qualified Tools
ARGUMENT
[Recommendations] Recommended usage of resources and peripherals
ARGUMENT
[Safety – Technique] Safe power up and boot
ARGUMENT
[Safety – Technique] Safe power down and shutdown
ARGUMENT
[Safety – Technique] Resource Virtualization
ARGUMENT
[Safety – Technique] Exclusive Access to Peripherals
ARGUMENT
[Safety – Technique] Temporal Independence
ARGUMENT
[Safety – Technique] Spatial Independence
ARGUMENT
[Safety – Technique] Diagnostic Techniques
ARGUMENT
[Safety – Technique] System reactions to errors and Fault-Tolerant Techniques
ARGUMENT
[Safety – Technique] FMEA, FMECA, FMEDA analyses
ARGUMENT
[Safety – Standard] Configuration
ARGUMENT
[Deterministic][Export] FSM deliverables (or equivalent) and Linking analysis
EVIDENCE
Claim: Defines a set of minimum and reasonable arguments and evidences that a safety Hypervisor should meet/provide in order to enable/support the development of mixed-criticality systems compliant with IEC 61508
CLAIM
Supports
Is evidence for

FIGURE 5.2: An MSC for an IEC 61508 compliant generic hypervisor – Top.

two possible shutdown scenarios based on the dependency level shall be considered. In the first scenario, the safety hypervisor has absolute control of its shutdown sequence (autonomous shutdown), while, in the second scenario, the hypervisor has no means to know and control its shutdown. It depends partially on external components. Nonetheless, regardless of the scenario, the shutdown process shall lead to a known and repeatable state within a limited time and shall consider the time for shutting down the hypervisor and the partitions.

As a general rule, the sharing of resources (such as memory and peripherals) shall be avoided in mixed-criticality systems for preventing interferences in the temporal and spatial domains. However, resource sharing is a common tactic used in today's partitioned mixed-criticality systems. For example, the memory assigned to a hypervisor can be shared with a communication network, thus possibly leading to interferences (e.g., the bottleneck effect). The *resource virtualization* technique provides a way to reduce problems related to the resource assignment avoiding as much as possible the sharing of resources.

Virtualization enables creating a layer of abstraction above the resources of the physical HW. The virtualization can be done entirely (full-virtualization) or partially (para-virtualization). A safety hypervisor shall support the virtualization of the processor's cores, the resources and the peripherals. Furthermore, the safety hypervisor must ensure that the partitions do not leave their configured environment and that the assigned virtualized resources do not directly access to the hypervisor's resources (such as the memory). For instance, a safety hypervisor shall guarantee that any partition does not influence a safety-related partition. These restrictions shall be configured in the hypervisor's configuration file.

In a similar way, the peripherals shall also be protected from undesirable accesses from partitions. To that end, the safety hypervisor must implement the *exclusive access to peripheral* functions. Those mechanisms guarantees that a peripheral is not assigned to more than one partition at a time and protects the peripherals from unauthorised accesses. The allocation of the resources and peripherals of a device to partitions may lead to failures that can jeopardise the assessment of the *temporal and the spatial independences*. These independences must be guaranteed by the safety hypervisor as defined in Annex F of IEC 61508-3 [IEC10c]. The assurance of temporal freedom means that no safety or non safety partition can influence a safety partition. For example, if a partition takes the processor's available execution time or if it blocks the execution by locking a shared resource, the temporal independence is not be guaranteed. Therefore, a safety hypervisor must ensure that partitions have enough time to complete their execution. The execution time per each partition is configured in the hypervisor configuration file. On the other hand, the spatial independence refers to the ability to guarantee that the data of a partition is not modified by other partitions (see Annex F of IEC 61508-3 [IEC10c]).

The execution of a safety hypervisor shall be isolated and protected against SW faults, thus guaranteeing the integrity of the hypervisor's code, its execution and the virtualization of the CPUs. The hypervisor's code shall be diagnosed periodically to ensure that it has not undergone any modification due to a failure. These periodic checks shall include IEC 61508 compliant measures and diagnostic techniques such as the RAM test and checksum mechanisms (see Tables A.5 and A.6 of IEC 61508-2). Further mechanisms such as a

Memory Management Unit (MMU) shall also be supported by a safety hypervisor to manage independent memory access rights. A MMU virtualizes the physical memory of the system, thus preventing that any code executed within a partition can access to the hypervisor's memory or the memory of another partition. Among others, this unit is responsible for ensuring that:

- no CPU instruction executed at a low level of privilege can access and modify the memory belonging to another partition (also called privileged separation).
- each partition has exclusive access to the CPU's cache
- a partition can only allocate and free its assigned memory
- a call to the MMU does not alter the partitioning of the memory
- only the hypervisor's functions that operate on the MMU can access the data structure of the MMU

It is usual that partitions generated by a safety hypervisor require communicating. This data exchange between partitions must be established by the safety hypervisor in a safe manner, complying with the temporal requirement of the communication protocol that is used for that purpose and avoiding the temporal interferences as much as possible. For example, a safety hypervisor may provide the communication between partitions through a shared memory (e.g., L2 cache) or using a mixed-criticality network (e.g., NoC). If the hypervisor does not establish a communication line for partitions, it should be explicitly justified in the hypervisor's documentation.

The *configuration* of a safety hypervisor is vital for the correctness and adequacy of the partitions' execution and guarantees the safety of the mixed-criticality system. For that reason, the configuration of a safety hypervisor shall be established using qualified tools with a static configuration file as an input (e.g., an .XML file). The configuration file must be defined during the design stage of the safety hypervisor's compliant development process, installed following a safe installation procedure and diagnosed independently at boot time and at run-time to detect misconfigurations.

It is a fact that the safety hypervisor can be subject to interferences derived from the HW platform, the operating system or the communication network implemented by the final system. Those interferences shall challenge the assessment of the temporal and spatial independences, the configuration process, the safe boot, power on, power off and shutdown sequences and the inclusion of the resource virtualization and the exclusive access to peripherals functions. To that end, an IEC 61508 compliant hypervisor must support and implement a set of IEC 61508 compliant *measures and diagnostic techniques* to detect and control random and systematic failures. For instance, a safety hypervisor shall implement platform-independent diagnostic techniques such as the *Defensive programming* and the *Input comparison voting* techniques (see Table A4 of IEC 61508-3 and Table A.7 of IEC 61508-2). In addition, platform related measures and diagnostic techniques (see Table A.5 of IEC 61508-2)), *system reactions to errors* (e.g., safe state) and *fault-tolerance techniques* (e.g., independent clock sources and Watch Dog Timer (WDT)) shall also be supported by the safety hypervisor. If it does not support these measures and diagnostic techniques, they should be implemented by the application SW or additional HW.

The evidences that support the safety-related arguments presented by this MSC shall be compiled in a set of *evidence documents*. These documents shall include, among others, the way in which the safety arguments defined in this section are met by the IEC 61508 compliant hypervisor, the impact analyses for each nonconformity and the identification of exported requirements that shall be completed by a third party. These documents shall also include the safe installation procedure followed by the hypervisor, and the relevant details of the usage of qualified tools in the life cycle of the safety hypervisor.

5.2 A Modular Safety Case for an IEC 61508 compliant Generic Safety Partition

This section presents the MSC for an IEC 61508 compliant safety partition positively assessed by a certification body within the context of the European research project DREAMS [LPA$^+$15]. The arguments set out throughout this section are established as the result of the analysis of the IEC 61508 safety standard and the safety concept for a wind-power mixed-criticality embedded system defined in [PGTT15]. Figure 5.3 presents the graphical representation of the MSC with the claim that a compliant SW partition must fulfil a set of safety arguments and evidences to achieve the compliance with the IEC 61508 safety standard and support reusability. This MSC considers that the intended usage of a safety partition is an IEC 61508 compliant fail-safe system with an SIL up to SIL3 where it is executed on top of an IEC 61508 compliant hypervisor (see Section 5.1). A system is called fail-safe when in the event of a failure, it reaches a safe state (e.g., stop the execution). For example, a train is a fail-safe system because it can be stopped when a failure occurs. The term fail-safe is more detailed in Subsection 2.2.1.

It is assumed that the safety partition can be developed for being compliant with a safety standard (e.g., safety partition or non safety partition). The safety partition shall include an SRAC list where the imported and exported requirements are specified. Examples are timing requirements and performance, required resources and associated restrictions.

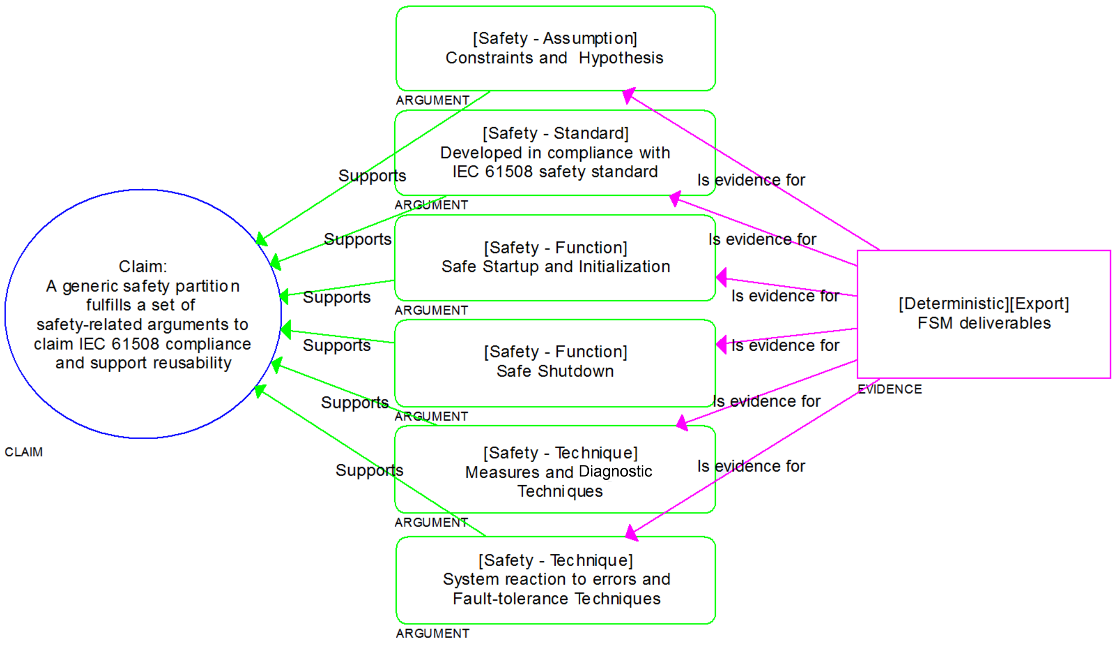

Figure 5.3: An MSC for an IEC 61508 compliant generic safety partition – Top.

This MSC considers that the safety partition shall be abstracted from the HW platform (e.g., abstracted from the COTS multi-core device) and that it must follow an IEC 61508 compliant *development process*. If another safety standard is considered for developing a safety partition, this MSC would need to be reviewed and updated accordingly. Furthermore, it is defined that the development process of a safety partition must support T3 *qualified tools*. These tools generate outputs that can contribute to the executable code of the safety-related partitions [IEC10d] (e.g., a compiler).

The *startup, initialization and shutdown* of a safety partition shall result in a known and repeatable state within a bounded time. The time needed by the components of the safety chain (e.g., the processors, the hypervisor) for completing the startup, initialization and shutdown processes shall also be considered by the safety partition for calculating the startup, boot and shutdown times. These processes may be executed directly by the safety partitions, or they can depend on external requests from a safety-related hypervisor or the HW platform. For each type of shutdown, three scenarios that differ depending on the external components (e.g., a WDT) for executing the shutdown can be identified (*autonomous*, *partial dependency* and *total dependency*). The autonomous shutdown means that the safety partition controls its safe shutdown. Instead, when the safety partition has no mean to know and control its safe shutdown and partially or totally depends on external HW or SW components, it is said that the partition executes a partial or total dependency.

A safety partition shall ensure that its failures do not affect the behaviour of other partitions. To that end, the safety partition shall implement a set of *measures and* *diagnostic techniques* recommended by the IEC 61508 safety standard for detecting random and systematic failures. The safety partition shall also support further methods such as the life-cycle related techniques and platform independent diagnostic techniques (see Table A.4 of IEC 61508-3 [IEC10c] and Table A.7 of IEC 61508-2 [IEC10b]). The abstraction of the measures and diagnostic techniques from the HW platform and the hypervisor enables the reusability of the safety partitions for developing different mixed-criticality systems. Different design techniques such as the standard language subsets for safety (e.g., restricted ANSI C) may be adopted to achieve that goal.

Furthermore, a safety partition shall implement a set of *system reactions to errors and fault-tolerance techniques* to manage the faults. For example, a safety partition may trigger the transition to a sleep state if a failure is detected or an other safety-related partition could supply the functionality of the faulty safety partition. The reactions to errors and fault-tolerance mechanisms supported by the safety partition shall be configured in the hypervisor's configuration file and shall be configured through qualified tools (see Section 5.1).

The safety arguments defined throughout this section and their evidences shall be collected in a set of *evidence deliverables*. These deliverables shall include the way in which a safety partition fulfils the safety-related arguments, the impact analyses for each nonconformity and the identification of exported requirements that have to be met by a third party. The evidence documents shall also define the uploading and the storage of the partitions' configuration and the relevant details of the qualified tools used for designing, implementing, integrating and configuring the safety partition.

5.3 A Modular Safety Case for an IEC 61508 compliant Generic COTS multi-core device

This section presents the MSC for an IEC 61508 compliant fail-safe COTS multi-core device [LPO15]. Figure 5.4 presents the MSC and the safety-related arguments that a COTS multi-core device shall fulfil in order to be compliant with the IEC 61508 safety standard. This MSC uses the term *device* as the abbreviation of a programmable electronic device and a COTS multi-core device whose end user is a safety embedded system engineer. A device is part of the HW design and executes SW. Hence, different dependencies, constraints and hypotheses can be exported to the device and the HW and the SW that is developed by the end user. On the other hand, it is assumed that a safety COTS multi-core device supports virtualization solutions such as hypervisors and on-chip and off-chip mixed-criticality networks. In these cases, the IEC 61508 compliant

COTS multi-core device may be used in conjunction with other IEC 61508 compliant components such as the IEC 61508 compliant hypervisor defined in Section 5.1 and the IEC 61508 compliant mixed-criticality network presented in Section 5.4.

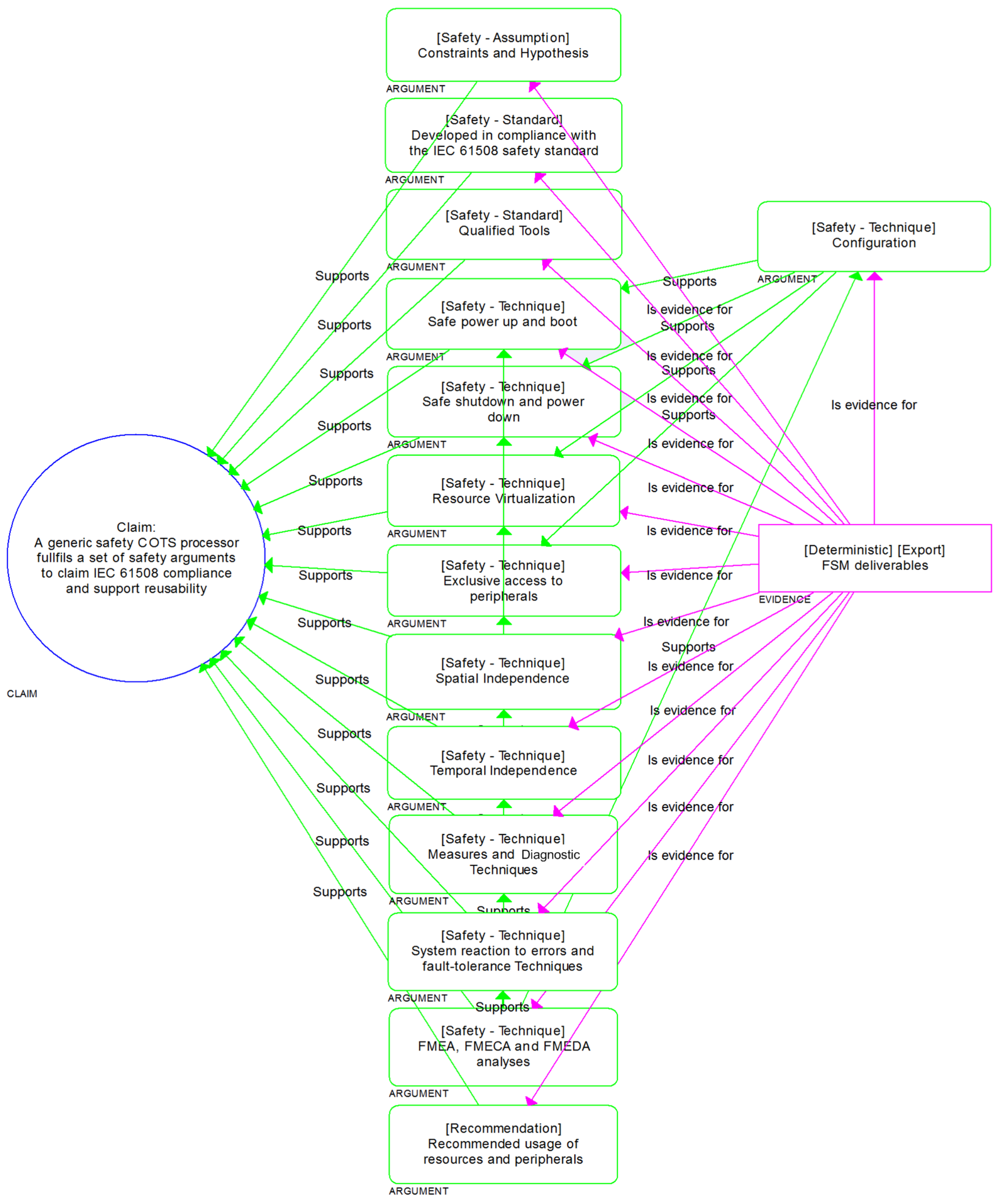
[Safety - Assumption] Constraints and Hypothesis
ARGUMENT
[Safety - Standard] Developed in compliance with the IEC 61508 safety standard
ARGUMENT
[Safety - Standard] Qualified Tools
ARGUMENT
[Safety - Technique] Safe power up and boot
ARGUMENT
[Safety - Technique] Safe shutdown and power down
ARGUMENT
[Safety - Technique] Resource Virtualization
ARGUMENT
[Safety - Technique] Exclusive access to peripherals
ARGUMENT
[Safety - Technique] Spatial Independence
ARGUMENT
[Safety - Technique] Temporal Independence
ARGUMENT
[Safety - Technique] Measures and Diagnostic Techniques
ARGUMENT
[Safety - Technique] System reaction to errors and fault-tolerance Techniques
ARGUMENT
[Safety - Technique] FMEA, FMECA and FMEDA analyses
ARGUMENT
[Recommendation] Recommended usage of resources and peripherals
ARGUMENT
[Safety - Technique] Configuration
ARGUMENT
[Deterministic] [Export] FSM deliverables
EVIDENCE
Claim: A generic safety COTS processor fullfils a set of safety arguments to claim IEC 61508 compliance and support reusability
CLAIM
Supports
Is evidence for

FIGURE 5.4: An MSC for an IEC 61508 compliant generic COTS multi-core device – Top.

The MSC presented in this section defines an IEC 61508 compliant COTS multi-core device that follows an IEC 61508 compliant *development process*. It claims that a safety device shall use *qualified on-line and/or off-line tools* for its design, configuration, verification and validation. The use of qualified tools enables mitigating the undesirable influences in the phases of the safety development process followed by a safety device. These influences derive from random and systematic failures which occur in the design, development, configuration, verification and validation phases of a safety development process. On-line tools can directly impact the safety-related system during the run-time, whereas off-line SW tools support a phase of the SW development life-cycle [IEC10a] and cannot directly affect the safety-related system during the run-time. Further, qualified off-line tools can be classified as T1, T2 and T3 tools in analogy to the MSC of the hypervisor (see Section 5.1).

An IEC 61508 compliant COTS multi-core device must guarantee a *safe power up, power down, boot and shutdown* sequences. From the end user point of view, these sequences represent the transitions between the operating states of the safety device, which shall be performed in a predefined and repeatable way within a limited time window. The power up and boot sequences of a safety device can be given in a cold state or a warm state. A cold sequence is the one where the device is off, and it is unexpectedly powered, thus providing the transition from power off state to active device state. This scenario requires additional HW mechanisms to ensure the device's safe power up and boot. For example, an external HW WDT and an external power supply supervisor are required. On the other hand, in a warm sequence, the device is suddenly restarted due to a reset signal (e.g., Power On Reset (POR)) that provides the transition from the device's current state (e.g., power off to power on). Similarly, the power down and shut-down sequences represent the transition of the safety device's state to the no power state. This transaction can be autonomously commanded by SW or by external events (e.g., power blackout) and managed in collaboration with external HW mechanisms (e.g., external HW WDT).

Multi-core devices provide a wide variety of resources (e.g., peripherals, memory, interrupts) which can be accessed and assigned to the safety and non safety-related components which are executed on top of them (e.g., hypervisor, partitions, operating system, communication network). As a general rule, in multi-core architectures, a resource should not be handled by more than one component at the same time. The main reason for that is that resource sharing may lead to harmful interferences which can hinder the achievement of the temporal independence. The temporal interference issues in COTS multi-core devices are treated in the next paragraph. However, in multi-core COTS devices, there are some exceptions where a resource (e.g., a memory area or a peripheral) can be shared among more than one component at the same time (e.g., two cores) [CAS14].

Resource virtualization techniques generate virtual solutions which help to reduce the complexity of the HW and the interferences caused by the resource sharing. For example, virtual cores, shared buses and communication resources can be directly managed by the end user or by a safety hypervisor, partition or mixed-criticality network. The resource virtualization technique is supported by the *exclusive access to peripherals* in this MSC. This technique aims to guarantee the freedom from interferences between safety and non safety-related components by managing the peripherals of the safety devices. To that end, a safety device shall support segregation (see Annex F of IEC 61508-3) where measures and techniques for achieving the non-interference between the SW components on a single computing system are recommended. The segregation must work in close cooperation with the other safety techniques (such as the resource virtualization, temporal independence, spatial independence and configuration) and shall also include a set of measures and diagnostic techniques to detect and control the unintended access to the peripherals.

COTS multi-core devices are designed to achieve the maximum average performance instead of supporting temporal guarantees. An example of that is the resource sharing between the components that compose the system (e.g., partitions, hypervisor, mixed-criticality networks) and the resulting occurrence of temporal interferences. Annex F of IEC 61508-3 recommends a set of safety techniques for achieving the *temporal independence* between the SW components of a single computing platform [IEC10c]. These measures and diagnostic techniques shall also be supported and implemented by safety COTS multi-core devices to be compliant with the IEC 61508 safety standard and achieve the temporal independence. For instance, a partial or complete temporal independence may be accomplished by implementing a TT network (e.g., TTNoC [ART10], AEthereal [GDR05]). The sources of the temporal interferences which can occur in a safety device shall be analysed and documented in a set of evidence documents. On the other hand, a safety device shall support the *spatial independence* as described in Annex F of the IEC 61508-3 [IEC10c]. For example, an MMU or a HW or/and a SW technique (e.g., a virtualization mechanism) may be used to that end.

The *configuration* of a safety COTS multi-core device shall be established using qualified tools of classes T2 (test tools) and T3 (constructive tools) [IEC10c, IEC10d]. The use of these tools enables achieving a safe configuration of the safety device. Hence, a set of detection and monitoring techniques shall be implemented by the safety multi-core device for guaranteeing a safe configuration. In the case of undocumented components of the device, the system integrator should contact the device manufacturer to identify configuration settings of the behaviour of those components.

An IEC 61508 compliant COTS device must implement further *measures and diagnostic techniques* in addition to the ones mentioned in the paragraphs before for avoiding and controlling HW random and systematic faults (see Annex A and B of IEC 61508-2). For example, the majority comparator technique or the parity bit for RAM may be implemented for detecting and controlling random failures of the processing units and the variable memory. A variable memory is a computer memory that required periodic refresh to maintain its content. Also, the *diverse HW* and the *antivalent signal transmission* techniques may be implemented to avoid systematic failures. For the management of the detected failures, a safety COTS multi-core device must implement hard coded or configurable *system reactions to errors* (e.g., safe state) and *fault-tolerance techniques* (e.g., HW redundancy). If the safety device does not support those techniques as a default, they shall be implemented by means of additional SW or HW mechanisms (e.g., external HW WDT).

Multi-core devices are not designed for safety applications by default. Therefore, they are not usually completely tested, thus leading to possible residual design errors. To address these errors, the device manufacturers rely on errata documents to notify customers of errors uncovered in field use. These errata documents imply that the system integrator must define and implement additional techniques (such as extra measures and diagnostic techniques, system reactions to errors and fault-tolerance techniques) to detect, mitigate and manage the residual errors defined in those documents. *FMEA, FMECA and Failure Mode Effects and Diagnostic Analysis (FMEDA)* analyses support the safety measures, techniques and components implemented by a safety COTS multi-core device and assess random and systematic failures. These analyses must be included in the documentation of the developer, including the description, intended usage, reconfigurability, compliance evidence, time restrictions and dependencies. Furthermore, those documents must contain the information related to the device's fault-hypotheses (including the Failure In Time (FIT), HW Fault-Tolerance (HFT) and SIL values) the functional specification of performed functions (e.g., power up states) and the list of exported/imported requirements (at HW, SW and system levels). The recommended and optimised usage of the device's features with respect to the safety standards must also be provided by the device manufacturer to ease the development and certification of the safety device.

An IEC 61508 compliant COTS multi-core device shall be accompanied, at least, by an *evidence* deliverable. This document must include, among others, the way in which the safety device fulfils the arguments defined in this MSC, the impact analyses per each nonconformity, the identification of exported requirements to be met by a third party, the FMEA / FMECA / FMEDA analyses and the specification of the configuration process. If additional safety techniques are required for a given safety device, they shall be explicitly stated in other evidence documents. In the same way, if further measures and diagnostic techniques, system reactions to errors or fault-tolerance techniques are required, they shall also be explicitly described in additional documents.

5.4 A Modular Safety Case for an IEC 61508 compliant Generic Mixed-Criticality Network

This section presents the MSC for an IEC 61508 compliant generic mixed-critical network based on the CAE notation language (see Figure 5.5). The scope of this MSC includes networks provided by the manufacturer as they are (black channel networks) and networks developed in accordance with a safety standard (e.g., IEC 61508, IEC 61784-3) (white channel networks). White channel networks are designed, implemented and validated according to IEC 61508 [IEC10a, IEC10b, IEC10c] and IEC 61784-3 [IEC10g] or IEC 62280 [IEC02a]. In the case of black channel networks, it is assumed that parts of the communication channel cannot be designed, implemented and validated according to a safety standard. Therefore, additional measures and diagnostic techniques are required to ensure that the failure performance of the communication process is compliant with the IEC 61508 and the IEC 61784-3 safety standards. Those extra measures and diagnostic techniques may be provided, for example, by an IEC 61508 compliant SCL.

In this section, whenever it refers to a safety network, we mean a network composed of HW and SW in the case of white channel networks and a network consisting of HW, SW and an SCL in the case of black channel networks. The safety arguments in green of Figure 5.5 are related to white channel networks, while the remaining arguments apply to both approaches (white channel and black channel networks). Furthermore, it is assumed that the safety concept defined in this section is intended for fail-safe mixed-criticality systems with an SIL up to SIL3.

According to IEC 61508 and as defined in Section 2.10, communication systems can be divided into *white channel* and *black channel* networks. White channel networks and the safety-related parts (the SCL) of black channel networks shall be developed in compliance with the IEC 61508 and IEC 61784-3 safety standards, against the required SIL and residual error rate values. The residual bit error rate is the number of bit errors per time unit based on the total number of bits received during a time interval. For instance, as stated in Table 1 of IEC 61784-3, for an SIL of 3, the probability of dangerous failures per hour, and the maximum permissible residual error rate of the functional safety communication system shall be lower than $10^{-9}/h$ [IEC10g]. A safety network shall guarantee that the reception time interval of two consecutive messages is below a pre-established time value. Otherwise, a communication error shall be assumed. This requirement is usually referred to as the *idle current*, closed-circuit or de-energized to trip principle.

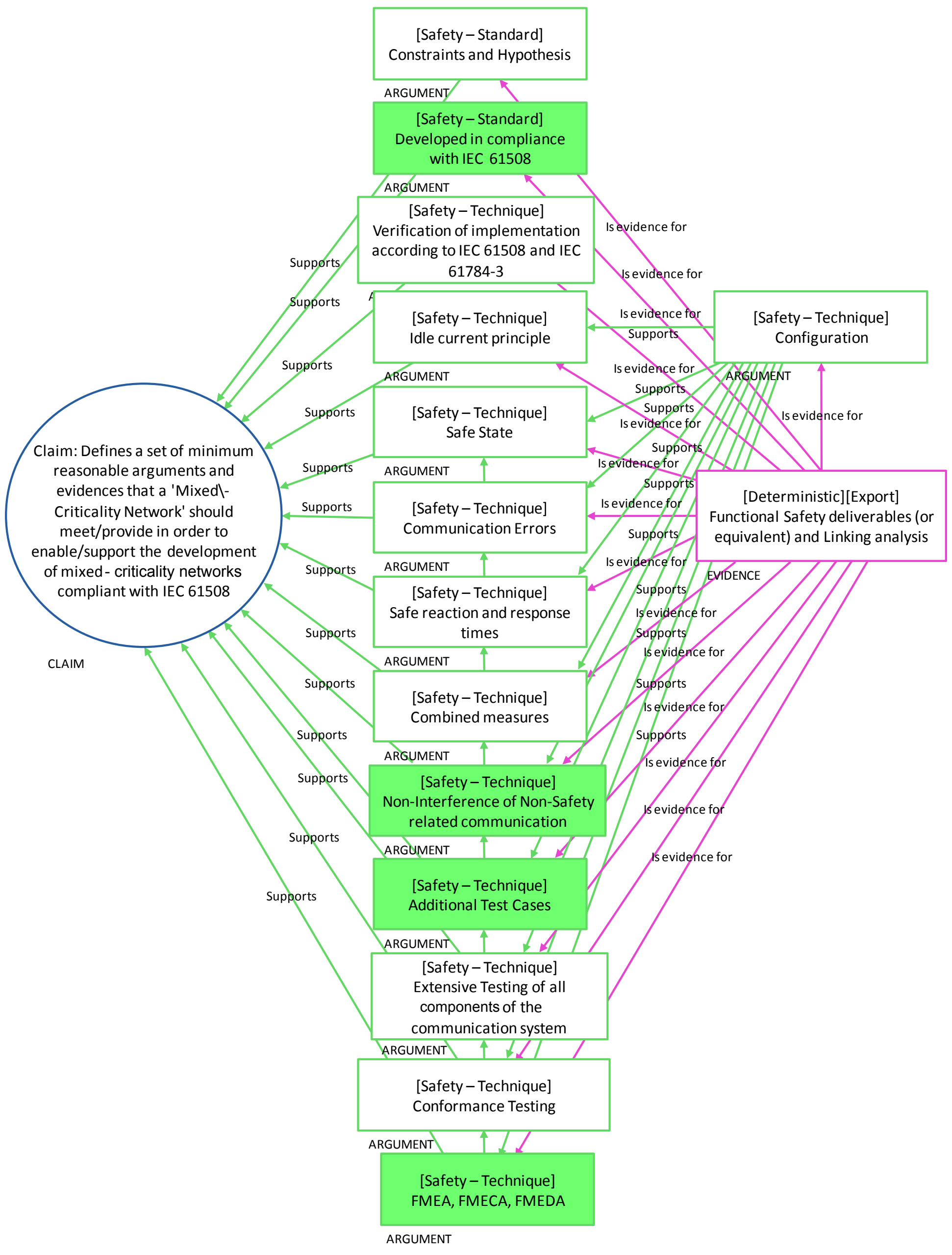
[Safety – Standard] Constraints and Hypothesis
ARGUMENT
[Safety – Standard] Developed in compliance with IEC 61508
[Safety – Technique] Verification of implementation according to IEC 61508 and IEC 61784-3
[Safety – Technique] Idle current principle
[Safety – Technique] Safe State
[Safety – Technique] Communication Errors
[Safety – Technique] Safe reaction and response times
[Safety – Technique] Combined measures
[Safety – Technique] Non-Interference of Non-Safety related communication
[Safety – Technique] Additional Test Cases
[Safety – Technique] Extensive Testing of all components of the communication system
[Safety – Technique] Conformance Testing
[Safety – Technique] FMEA, FMECA, FMEDA
[Safety – Technique] Configuration
[Deterministic][Export] Functional Safety deliverables (or equivalent) and Linking analysis
EVIDENCE
Claim: Defines a set of minimum reasonable arguments and evidences that a 'Mixed\-Criticality Network' should meet/provide in order to enable/support the development of mixed- criticality networks compliant with IEC 61508
CLAIM
Supports
Is evidence for

FIGURE 5.5: An MSC for an IEC 61508 compliant generic mixed-criticality network – Top.

According to the IEC 61784-3 [IEC10g] standard, the common *communication errors* that can occur in a mixed-criticality network include the corruption, unintended repetitions, incorrect sequence, loss, unacceptable delay, insertion, masquerade and addressing. These errors shall be detected and mitigated by implementing a set of *deterministic remedial measures* recommended in IEC 61784-3. For instance, the sequence numbers, time stamps, expectation time, connection authentication, feedback messages, data integrity assurance, redundancy with cross checking and different data integrity assurance techniques may be implemented by a white channel or black channel safety network for detecting and controlling communication errors.

In addition to the defined methods for the estimation of residual errors, *further fault cases* defined in the IEC 62280 safety standard [IEC02a] shall be considered and controlled by white channel networks. IEC 62280 recommends the measures and safety services to reduce the risk associated with the threats of communication systems. For instance, it suggests the message authentication, integrity, timeliness and sequence checking, source and destination identify, feedback message and cryptography techniques to reduce associated risks with the communication threats (e.g., repetition, deletion, insertion).

Moreover, a safety network must provide *error reaction mechanisms* to achieve a safe state in the case that a communication error is detected. For example, it can stop the communication between specific components of the system. In the case of black channel networks, the reaction to error mechanisms shall be defined in the documentation provided by the network developer.

An IEC 61508 compliant communication network shall guarantee that its *safety reaction and response times* do not exceed the time values specified by the network manufacturer, even in the presence of a failure. To that end, the safety communication systems shall provide a predictable and a deterministic communication between the components with different criticality (e.g., safety, non safety).

In addition, a *combination of the measures* quoted before (e.g., sequence numbers, time stamps, expectation time, connection authentication) shall be implemented by the safety communication network to detect communication errors (see an example in Table 5.2). The quantity and timing errors to be considered is provided by the network manufacturer or the Functional Safety Communication Profile (FSCP). The utilisation of a common function within the fieldbus communication by the specific groups of participants is called a profile. For example, communication profile family 12, commonly known as EtherCAT is based on the IEC 61158 standard [IEC03] and the safety communication

Communication errors	**Measure and Diagnostic Techniques**				
	Sequence Number	Echo	CRC	Connection authentication	Time stamp
Unintended repetition	x				x
Loss	x	x			
Insertion	x			x	
Incorrect sequence	x		x		
Corruption		x	x		
Unacceptable delay					x
Masquerade				x	
Wrong addressing		x	x	x	

Table 5.2: Communication errors and combination of measures and detection techniques – Example.

layer specification is defined in IEC 61784-3-12 [IEC10f]. FSCP 12 describes a protocol for transferring safety data up to SIL3 between FSCP 12 devices.

The transmission/reception among safety and non safety-related messages through white channel networks may be affected by interferences, leading to the failure of the communication between the components connected to the network. For that reason, white channel networks shall guarantee the *non-interference of non safety-related communication* to ensure the compliance with the temporal and spatial independence requirements. For instance, TT communication systems may be used to ensure the temporal independence. These networks use a priori knowledge about the permitted component behaviour to block faulty messages.

Safety communication networks can be tested in two ways. In the first instance, the components of the network may be tested together, thus obtaining an *exhaustive diagnosis* of the network. In the second scenario, the components of the network are tested independently by test beds and/or simulators. Nevertheless, both scenarios must follow the parameters set in the FSCP. An FSCP defines suitable conformance testing to assess services and measures of IEC 61784-3. A safety network shall support an adequate diagnostic strategy to avoid configuration related issues that can jeopardise the safety of the communication system and associated subsystems. For that reason, qualified tools of classes T2 (e.g., verification tools) and T3 (e.g., compilers) shall be used for designing, developing and configuring the safety network. The safety-related techniques presented in this section shall be evidenced by *FMEA / FMECA / FMEDA* analyses, only in the case of white channel networks, to assess random and systematic faults. These analyses shall include information related to the safety-relevant failure modes, causes and effects, as well as the measures, diagnostic techniques and system reactions to errors implemented by the safety network.

A safe mixed-criticality network shall be backed up by a set of *evidence documents* which shall be available for the end user to support the integration of the safety network. These documents shall include the evidences that support the fulfilments of the safety techniques, the impact analyses for each nonconformity and the identification of exported requirements that must be met by a third party.

Chapter 6

Cross-Domain Mixed-Criticality Patterns

The current upward trend in the development of mixed-criticality systems where functionalities with different criticality are integrated onto a single embedded computing platform has its basis on virtualization, HW segregation and partitioning solutions. This architecture style is considered for defining the MSCs in the chapter before, where a solution to the associated challenges that lead to a significant and potentially unacceptable increase of engineering and certification costs is presented. As a result of the definition of the MSCs and the analysis of the IEC 61508 safety standard, several components of today's mixed-criticality systems out of the scope of this safety standard have been identified. Consequently, new techniques are required to give solutions to the occurring problems in the development of today's mixed-critical systems.

Cross-domain patterns are widely used universal approaches for describing and documenting recurring solutions for design problems which are used to guide and support engineers towards solutions that solve commonly occurring problems in the development of mixed-criticality products from design to verification and validation (V & V).

This chapter defines several remarkable generic cross-domain patterns, including patterns for hypervisors (see Section 6.1), COTS multi-core devices (see Section 6.2) and mixed-criticality networks (see Section 6.3). These cross-domain patterns are defined in plain text format for better understanding, although they are commonly represented using the structure presented in Section 2.12.

6.1 Hypervisor

A hypervisor is a layer of SW or a combination of SW and HW that allows running several independent execution environments, also called partitions, in a single embedded computing platform. Partitions are logical divisions of memory with static or dynamic cycle and execution time. They can have assigned one or more peripherals and can be developed for different levels of criticality (e.g., SIL1 to 4 according to the IEC 61508 safety standard). Among products in this category, XtratuM [Sol14], PikeOS [SYS15], Wind River [Riv15] and QNX [QNX15] hypervisors can be commercially distinguished.

Partitioning a system using a hypervisor can give rise to spatial independence, temporal independence and real-time constraints. The broad trend of the division of a system into partitions with different criticality requires inter-partition communication mechanisms. In multi-core architectures, partitions can communicate through shared memories or by using NoC communication systems. NoCs are commonly used communication systems for avoiding the problems associated with the use of shared memories [KP15]. These memories imply temporal and spatial interferences due to memory inconsistencies and cache coherency problems. However, the use of NoCs increases the complexity of the system and involves challenges to certification such as guaranteeing that the critical memory assigned to the mixed-criticality network is not accessed by the hypervisor or by the partitions and that the critical memory itself does not cause faults which can jeopardise the system. These and further failure scenarios are handled in Subsections 6.1.1, 6.2.2, 6.1.3 and 6.1.4 where solutions to tackle spatial and temporal interferences and manage the resources in multi-core mixed-criticality systems are presented.

6.1.1 NoC-Accessible Memory Area Diagnosis Pattern

NoCs are widely implemented communication systems to avoid Point-to-Point (P2P) individual communication paths between the components of mixed-criticality systems. They enable the creation of logic paths to interchange data. On-chip networks may access critical memory areas in use by other components, which may imply errors that could jeopardise the safety of the system. The most significant impact caused by a memory access of a NoC communication system is the breaking of the temporal isolation. The temporal isolation can also be endangered due to delays caused by a high amount of traffic in the NoC.

This cross-domain pattern defines the following design solutions to detect, manage and avoid failures in the critical memory areas accessible by the NoC.

▶ $\boxed{\textbf{Solution 1:}}$ **Provide complete HW isolation of NoC**

This solution proposes the assignment of a dedicated memory to the NoC for incoming and outgoing message buffers and its internal operations. In addition, this solution suggests avoiding the attachment of the NoC to the same bus as the processing cores, thus achieving a complete isolation from the processing cores. This solution scheme can be implemented using a dual-port RAM where one port is accessible by the NoC, and the other port is connected to the bus where the processing cores are connected (See Figure 6.1).

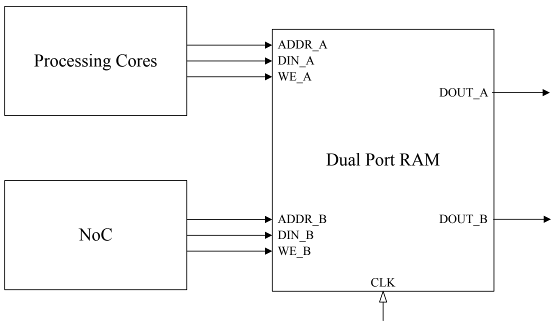

FIGURE 6.1: Dual-port RAM – Overall representation.

► $\boxed{\textbf{Solution 2:}}$ **I/O MMU**

This solution proposes the implementation of an MMU for controlling the access of Direct Memory Access (DMA) transfers programmed by the bus-master capable I/O devices. Consequently, the DMA transfers cannot overwrite and read from the restricted memory addresses. In the case of safety-critical systems, the memory addresses may contain the code and data of safety-critical tasks. An I/O MMU enforces the spatial isolation and avoids the overwriting of the safety-sensitive memory regions by the NoC.

► $\boxed{\textbf{Solution 3:}}$ **Additional monitoring mechanisms**

See design pattern *Critical partition diagnosis pattern* (Subsection 6.1.2).

6.1.2 Critical Partition Diagnosis Pattern

When dealing with partitioned mixed-criticality systems, failures caused by the exchange of information are quite probable. The lower criticality functionalities can lead to interferences on the higher criticality functionalities. This pattern analyses two possible sources of interferences. The first source refers to the occurrence of temporal interferences generated by multiple accesses in parallel to the shared memory (e.g., interferences caused by the processors of a multi-core device). The concurrent accesses will compete for accessing the shared memory cache, possibly leading to interferences in the temporal domain. On the other hand, failures of the hypervisor in the spatial isolation must be considered. These failures can also be found in mono-core architectures. The following generic and scalable design solutions presented by this pattern measure and detect interferences caused by non-critical partitions on critical partitions and guarantee the system's temporal and spatial independences.

▶ **Solution 1: Limit the concurrency**

This solution proposes to execute the critical tasks without concurrency. When a critical task is running on a certain CPU of a multi-core processor, the other CPUs do *idle* only for the duration of the task. This consideration enables avoiding contention. The limitation of the concurrency can be achieved by appropriately configuring at design time the partition execution windows for all the CPUs. The loss in performance can be leveraged by tuning the amount of time that a CPU (running a critical task) executes without concurrency. The maximum amount of interferences suffered by one CPU due to accesses to shared memory and the bus bandwidth used by the other CPUs can be calculated using off-line analyses. The concurrent execution can be guaranteed up to a certain safe time limit based on the temporal constraints of the safety-critical tasks and the maximum amount of interferences.

▶ **Solution 2: Assess the spatial isolation**

This solution presents a diagnostic partition that periodically checks the data of the critical memory areas, including the hypervisor's code and the partitions' code and data. Checksum and similar mechanism recommended by the IEC 61508 safety standard are perfect candidates to ensure that no accidental modification of code or data takes place. Partitions can individually implement measures and diagnostic techniques for checking their code and data. However, the code of the hypervisor shall be verified by at least one partition to ensure that is not unexpectedly modified.

▶ **Solution 3: Assess the temporal isolation**

The measures and techniques recommended by the IEC 61508 safety standard can also be implemented to assess the temporal isolation. For instance, the *Program temporal sequence monitoring* technique recommended in Section A.9 of IEC61508-7 [IEC10e] may be implemented to monitor the execution of safety-critical tasks regarding their temporal response and to ensure that the temporal isolation is not compromised due to a failure of the hypervisor. This proposed solution aims to detect failures in the hypervisor and in the configuration of the partition execution window that may jeopardise the temporal isolation. The task scheduling inside the partitions is out of the scope of this solution and therefore it is not analysed.

6.1.3 Communication Input/Output Server Pattern

The functionalities (e.g., safety-related, non safety-related) executed on multi-core mixed-criticality systems usually require communication. To that end, today's multi-core devices usually implement shared memories, on-chip buses (e.g., TTNoC), off-chip buses (e.g., EtherCAT) and local buses (e.g., PCIe, digital I/Os and RS485). These communication systems have their pros and cons. For instance, shared memories are common sources of interferences in architectures with more than one core [MSM$^+$11]. To overcome issues related to the shared memories, on-chip and off-chip communication networks are one of the alternatives for providing internal and external communications. For example, the communication between partitions with different criticality levels can be carried out through an STmicroelectronics' NoC (STNoC) on-chip network and a TTE network with or without real-time capabilities. However, the use of NoCs implies additional interferences in general with challenges for certification. The challenges regarding NoCs are discussed in Subsection 6.3.1.

On the other hand, the number of functions integrated in a mixed-criticality system which require communication tends to increase, mixing the communication requirements of different criticality (e.g., safety, real-time, security) and hampering the development and certification of mixed-criticality networks. As a result, the underlying mixed-criticality communication systems can require an adaptation procedure or shall be modified to cover new requirements. These processes may incur a higher engineering and certification cost. This pattern defines a communication server that simplifies the system design and development and reduces the cost of certification. This communication server is a centralised communication manager of the communication between partitions and external components that compose a mixed-criticality system. It is logically abstracted from the processor control of the communication network (e.g., using partition ports) and manages the assignment of the peripherals to the partitions implementing the exclusive access to peripherals (see Section 5.1).

6.1.4 Digital Input/Output Server Pattern

Different mixed-criticality system architectures widely use DIOs for communicating with external components such as sensors and actuators. In single core architectures, the DIOs can be managed without significant difficulty. However, in multi-core architectures where hundreds of thousands of functionalities with different criticality levels can be implemented (e.g., safety, non safety, real-time) and where, a DIO can be requested at the same time by all the functionalities, the management of these I/O interfaces is hindered [Ona17]. The simultaneous access to a DIO by more than one function may

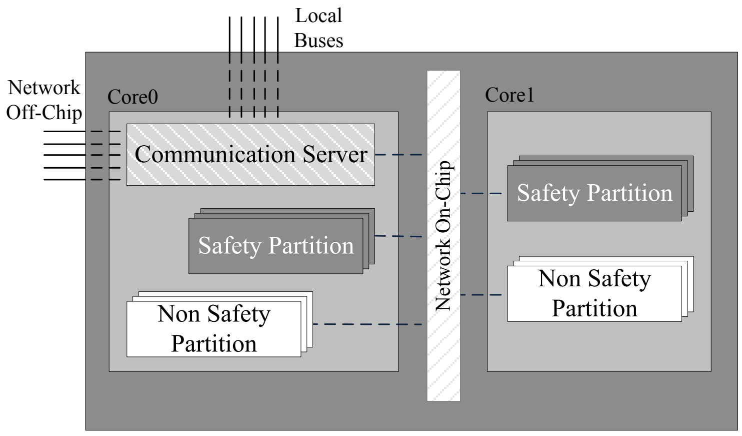

Figure 6.2: Communication I/O Server – Overview.

cause interferences in the temporal domain which could jeopardise the system. Besides, from a product line perspective, the number of DIOs requested by a product might change, which may lead to scalability problems and an adaptation process to fit with the changes.

This cross-domain pattern proposes the implementation of a DIO Server (DIOS) partition for centralizing the management of the DIOs of mixed-criticality systems (see Figure 6.3). The DIOS is a consistent concurrent manager of DIOs which is abstracted from platform and hypervisor details to assure reusability. Furthermore, it periodically updates the values of the inputs and refreshes the information of the partitions where the inputs are required. This server can be reused on different system architectures without significant changes, thus simplifying the system design.

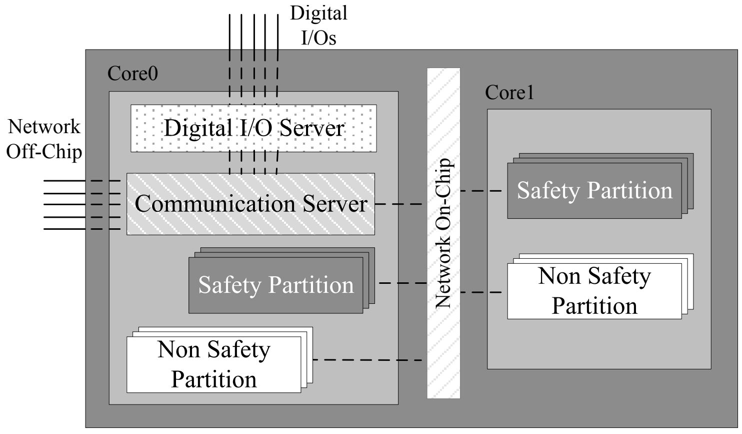

Figure 6.3: Digital Input/Output Server – An overview.

Afterwards, the diagnosis techniques explained in the following paragraphs are executed, so that, if a failure is detected, the outputs will be refreshed with the safe value instead of with the value provided by the partitions. In the case that different partitions try to update the same output with different values, the partitions will be moved to a safe state, and the outputs will be updated to the default value. The conditions, measures and diagnostic techniques implemented by the DIOS partition to assess that a DIO controlled by a safety-related partition does not affect other partitions are the following:

- The safety-related outputs have associated inputs with the same or opposite values. These values vary depending on the configuration.
- The CRCs of the register's values associated to the digital I/Os are periodically compared against the values which are already stored by the digital I/O server. The comparison period is determined by the minimum refreshing period of the digital outputs.
- It is checked that the partitions in charge for updating the digital outputs refresh the values of the DIOS partition. For that purpose, a token is implemented that is updated every time that the communication is performed, always agreeing with the expected values in the DIOS. This solution can also be applied to the remaining partitions of the system, but with the inputs, for assuring the communication between the partitions and the DIOS.
- The register values of the digital outputs are checked against the values supported by the DIOS every time that the values of the digital outputs change.
- Each digital input is checked under a pre-configured timeout to detect whether their values can be modified. If the time-out value is not specified, the default value will be used (a month) and the developer shall integrate an output to change the values of the inputs in a controlled non safety way for testing purposes.

In safety-related applications, the configuration of the digital I/Os of the server like the configuration of partitions shall be established using qualified off-line tools. This pattern relies on the safety-related arguments defined in the MSC for an IEC 61508 compliant generic hypervisor [LPA+15] and the safe communication between partitions provided by an IEC 61508 compliant hypervisor.

6.2 COTS Multi-Core Device

COTS multi-core devices are commonly used for real-time systems due to their low-cost and short time to market. However, multi-core devices provide sophisticated components which increment the complexity and may cause drawbacks, thus jeopardising the safety of the system. For example, the simultaneous running of tasks and the resource sharing between more than one component can cause interferences in temporal and spatial domains.

Different research studies propose techniques to improve the performance of the multi-core devices by reducing the memory interferences of the applications [GBEL10] [PQnCV09] [KHMHB10] and to control the mapping of application's data to memory channels [SHK14]. Some of these techniques focus on scheduling policies which provide request prioritisation and reduce the inter-partition interferences.

On the other hand, the coherency of the cores of the processor and the memories can be a source of issues in general. In single-core systems, the coherency of the cores and the memory is not a problem such as there is only one processing unit that can read or write from/to the memory. Conversely, in multi-core systems, there can be two or more processing units executing at the same time. So, it is possible that they access the same memory location at the same time. If no processor changes the data of the accessed memory location, they can share data indefinitely. However, the coherency protocol may fail, leading to inconsistencies and jeopardising the safety of the system.

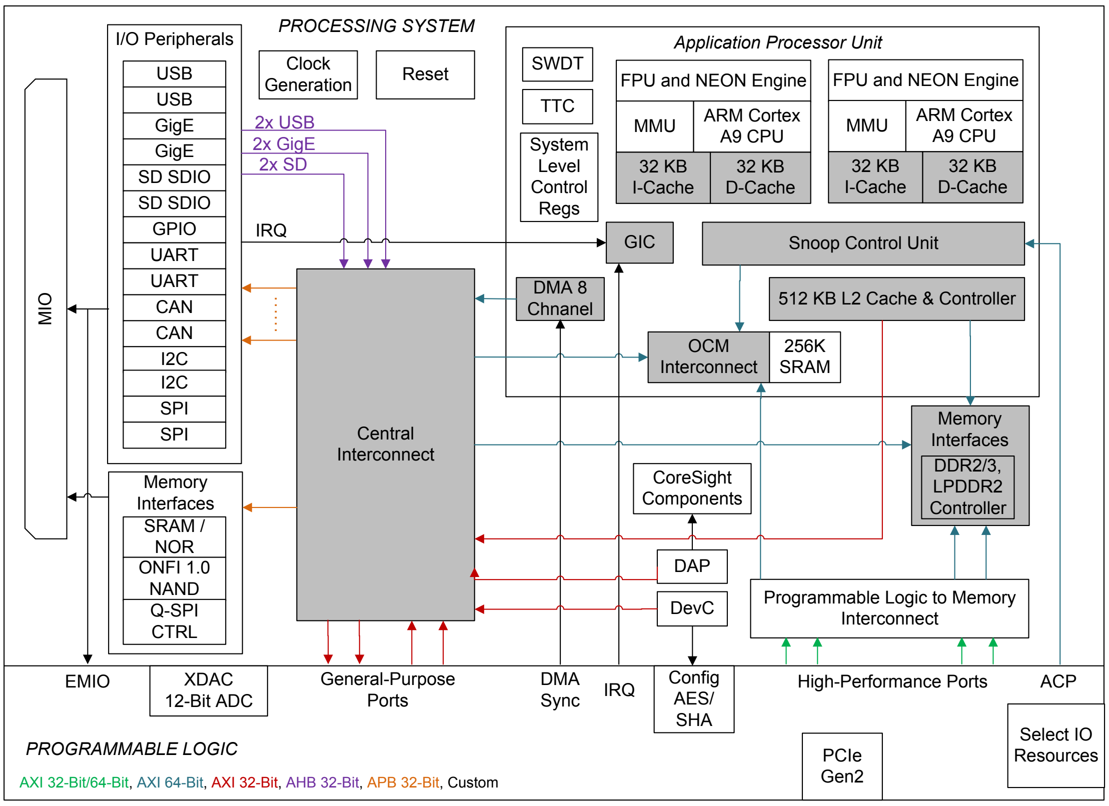

FIGURE 6.4: Zynq-7000 zc706 – Complex and challenging components. (Source [XIL14c])

The following subsections deeply analyse the shared memory and the coherency management units of today's COTS multi-core devices and propose solutions that aim to solve the commonly occurring problems of those components.

6.2.1 Shared Memory Diagnosis Pattern

The sharing of resources is a habitual implementation in today's multi-core mixed-criticality devices for improving the performance. Modern multi-core devices integrate cache memories which can be applied for private use (e.g., L1 cache) or memories which can be shared between more than one component at the same time (e.g., L2 cache). Secondary cache memory is commonly used to improve the performance of the system when the processor generates significant data traffic. In single-core architectures, the IEC 61508 safety standard covers the failures caused by memory sharing such as the causal factors of the execution interference between components of a single computer platform (see Annex F of IEC 61508-3 [IEC10c] *Techniques for achieving non-interference between SW components on a single computer*).

Measures and diagnostic techniques recommended by the IEC 61508 safety standard should be reviewed and extended for covering the issues caused by the use of shared memories in multi-core architectures. Different research studies propose techniques to solve and reduce as much as possible the interferences posed by the shared memories. For example, techniques to improve the performance of the system by reducing the memory interferences of the applications [GBEL10], [PQnCV09] and [KHMHB10] are proposed. These techniques focus on scheduling policies which provide request prioritisation and reduce the inter-partition interferences. Other solutions aim to control the mapping of application's data to memory channels [SHK14].

This cross-domain pattern proposes the following reusable generic solutions to detect, evict and manage the failures related to the shared memories in HW architecture with more than one processing core.

► **Solution 1: Limit the use of shared memories**

This solution proposes to limit as much as possible the use of the shared memory and in the case that it is implemented to control its access for avoiding parallel accesses. For instance, the shared memory of the Zynq zc706 multi-core device can be disabled for avoiding interferences [XIL14c]. Another possibility is to replace the shared memory by an on-chip network, thus avoiding interferences caused by the use of shared memory. NoC systems provide benefits regarding spatial and temporal segregation. However, as analysed in Subsection 6.3.1, the management of NoCs for communicating components with different criticality implies fundamental challenges to certification [DAM$^+$13].

► **Solution 2: Cyclic redundancy check with comparison**

This solution proposes a CRC based diagnosis with comparison technique to detect failures in the shared memory. It considers two implementation scenarios where the first scenario builds on a COTS multi-core device provided *as it is* by the device manufacturer and the second scenario presents a partitioned COTS multi-core device where functionalities with different criticality are executed on top of partitions (e.g., safety, security, real-time). The term *as it is* refers to a device provided without modifications (e.g., without partitioning) that would alter its properties (e.g., safety-related properties). The graphical representation of this solution is shown in Figure 6.5, where it implements two independent memories (memoryA and memoryB) for storing and reading the application data and the CRCs.

For reasons of simplification and understanding, a scenario where both CPUs execute the same functionalities is taken as a basis for describing the steps for implementing the CRC and the comparison-based diagnosis pattern. The first step to undertake this diagnosis pattern is to send the application data of both CPUs to separate memory areas of memoryB. The application data is stored in memoryB through the shared memory. Then, the CRCs of the application data are calculated and stored in separate memory locations of memoryA. These CRC values are usually referred to as *golden CRC*. A golden CRC is the result of the first calculation of the CRC. This CRC is used to perform the comparison against the CRC values of the data that is sent to the shared memory and to determine if a failure occurs in the shared memory. The CRCs can be calculated at the beginning or the end of the execution of the tasks, although, in the case that the CRCs are calculated at the beginning, a synchronisation mechanism may be required to synchronise the calculation and the comparison of the CRCs.

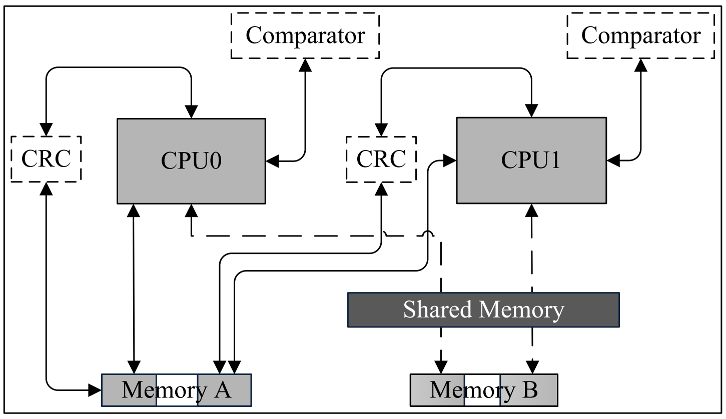

FIGURE 6.5: Shared memory diagnosis pattern – Solution 2: CRC with comparison.

The successful implementation of this pattern depends on additional components of the mixed-criticality system, including among other things the processing cores, the timers, the interrupt controllers, the coherency management units, the interconnection management units and the memories. Therefore, this cross-domain pattern assumes that all those components are checked in advance and that they do not interfere with the shared memory, thus achieving the optimum implementation of this solution.

- **Scenario 1:** Non-partitioned COTS multi-core system

– ***Sub-scenario 1.1:*** *Non partitioned system where the processors execute the same functionality and generate the same outputs*

This sub-scenario considers a non-partitioned processing system that supports the CRC with comparison solution and allows the execution of applications with the same or different criticality levels on top of the processor's cores (see Figure 6.7). The steps for implementing the proposed settlement in this scenario are defined below.

Step A) CPU0 and CPU1 write the application data in separate memory areas of memoryB. This memory is accessed through a shared memory. Then, each CPU calculates the CRC of the application data. These CRCs usually referred to as golden CRC, are stored in separate memory regions of memoryA.

Step B) The application data stored in memoryB is periodically read (e.g., each 50ms) by each CPU, and the CRC of the data is calculated (one CRC per CPU). The calculation of the CRCs can be performed at the beginning or at the end of the task which is executed by each CPU.

- If the execution of the CPUs' tasks is synchronous, steps **c**, **d** and **f** shall be followed.
- Otherwise, if the execution of the tasks is asynchronous, a synchronisation mechanism should be implemented to synchronise the execution of the tasks and the associated CRC calculations. In that case, once a synchronization mechanism is implemented, steps **e** and **f** shall be followed.

Step C) The CRCs of the application data read from memoryB are calculated. Next, the CRCs are compared on the fly against the golden CRCs stored in memoryA (see Figure 6.6).

Step D) The CRCs of the data read from memoryB are calculated. During the calculating process, the CRCs are compared on the fly against the golden CRC read from memoryA. This is a particular case where it is assumed that the golden CRCs of each CPU do not differ (same functionality and data), thus resulting in the same golden CRC value.

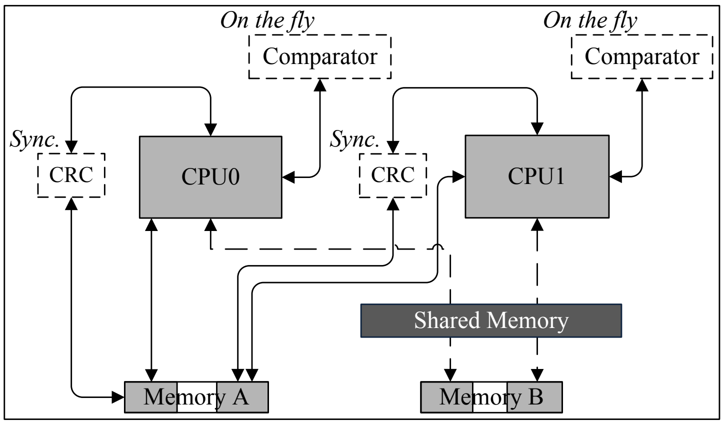

FIGURE 6.6: Shared memory diagnosis pattern – Solution 2 – Scenario 1: Non partitioned system with the same functionality execution and on-fly comparison.

Step E) The comparison of the CRCs takes place. CPU0 reads the golden CRC that is stored in memoryA and compares it against the current CRC. If the CRCs match, the execution will continue. Otherwise, a fault-tolerance technique or a safe state must be executed. CPU1 shall executes the same process.

Step F) Once the CRCs are compared and it is checked that they are not corrupted, the current CRCs of the application data are stored in memoryA such as the new golden CRCs.

– ***Sub-scenario 1.2:*** *Non partitioned multi-core system where the processors execute different functionality and generate different outputs*

This sub-scenario assumes that the processor's cores execute applications with different criticality levels. In that case, a variant of the solution presented at the beginning of this section can be implemented. This solution is represented in Figure 6.7 and it contains the following steps:

Step A) CPU0 and CPU1 execute different functionalities. Before sending data to memoryB, the golden CRC of the data is calculated and stored in memoryA. This process is realised per each CPU and the resultant golden CRCs are stored in different memory locations in memoryA.

Step B) CPU0 and CPU1 write the application data in memoryB (e.g., DDR). Each CPU has its own memory region.

Step C) The data stored in memoryB is periodically read (e.g., 50ms) and independent CRCs are calculated (one CRC per CPU). The CRCs can be calculated at the beginning or the end of the tasks executed by each CPU.

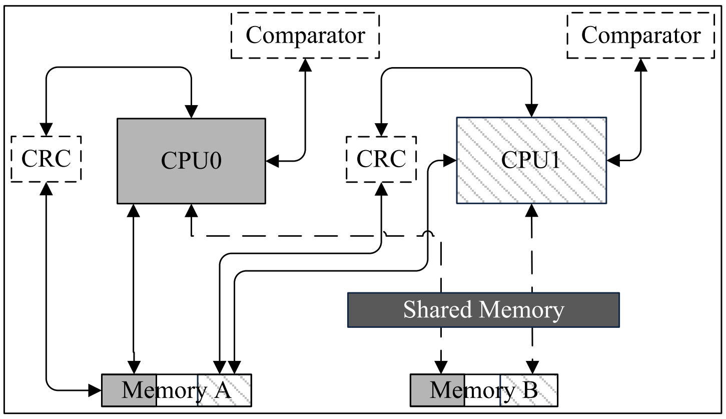

FIGURE 6.7: Shared memory diagnosis pattern – Solution 2 – Scenario 1: Non partitioned system with diverse functionality execution.

Step D) The golden CRCs are compared against the CRCs calculated . The comparator unit of CPU0 reads the CPU0's golden CRC from memoryA and compares it against the current CRC. If the CRCs match, the execution continues. Otherwise, a fault-tolerance technique or a safe state is executed. CPU1 shall execute the same process.

Step E) Once the CRCs are compared and it is checked that the shared memory does not corrupt the data, the current CRCs (one CRC per CPU) are stored in memoryA such as the new golden CRCs.

– ***Sub-scenario 1.3:*** *Non partitioned COTS multi-core system with a PS and a PL*

The solution defined at the beginning of this section can also be implemented for a COTS multi-core device with a PS and a PL. The PL enables implementing more than one soft-core processors in the same device. From a safety perspective, the diagnosis coverage provided by this architecture would be better than the ones provided by the other diagnosis scenarios. The main reason for that is that according to the IEC 61508 safety standard, the PL can be considered as an additional independent HW, where its failure modes, causes and effects differ from the ones of the PS. This extra scenario considers that the functionalities executed by the soft-core processors may vary from the executed ones in the PS. Independently of the scenario, the steps defined in the sub-scenarios 1.1 and 1.2 should be followed to diagnose the shared memory. The major differences between these sub-scenarios lie in the execution environments of the tasks and the access methods to the memories (e.g., OCM and DDR) such as shown in Figure 6.8 and Figure 6.9.

- **Scenario 2:** Partitioned COTS multi-core system

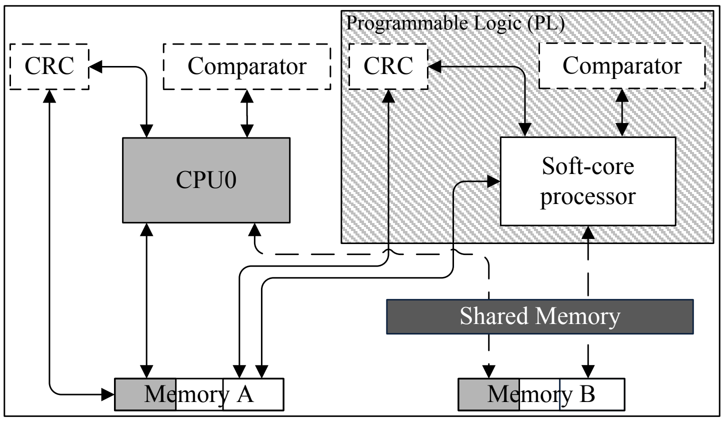

FIGURE 6.8: Shared memory diagnosis pattern – Solution 2 – Scenario 1.3.A.

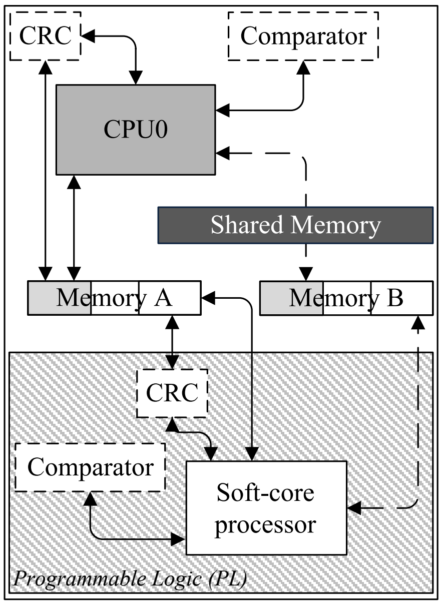

FIGURE 6.9: Shared memory diagnosis pattern – Solution 2 – Scenario 1.3.B.

Mixed-criticality systems implement virtualization mechanisms such as hypervisors for dividing the system into partitions where functionalities with different criticality can be implemented. This second scenario considers the diagnosis of the shared memory in a partitioned system where functionalities with different criticality levels are executed. From a safety perspective, the proposed diagnosis scenario can be improved if the comparison is given between two partitions which are located in different cores or if the HW redundancy is implemented. Nevertheless, this scenario assumes that the partitions are scheduled by the hypervisor, thus providing bounded execution time of the tasks.

– ***Sub-scenario 2.1:*** *Partitioned system where safety-related partitions execute functionalities with different criticality levels*

In sub-scenario 1.1 the overall solution for a device where each CPU executes a different functionality is presented. That solution can also be applied to partitioned multi-core devices where functionalities with different criticality levels are executed on the same CPU (see Figure 6.10). This sub-scenario shall follow the steps stated in the sub-scenario 1.1, taking into account that instead of having two CPUs, only a single CPU is available.

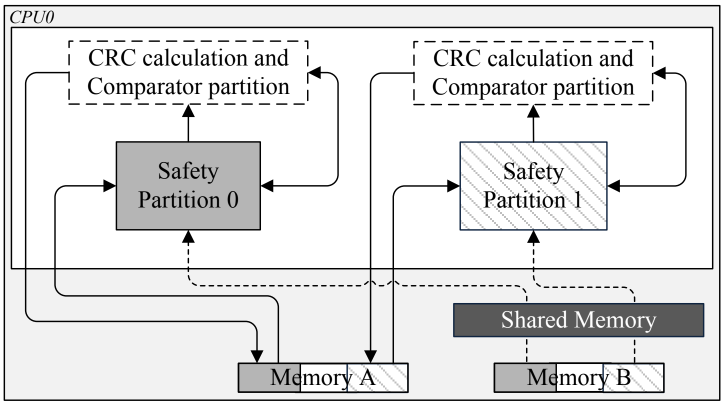
CPU0
CRC calculation and Comparator partition
CRC calculation and Comparator partition
Safety Partition 0
Safety Partition 1
Shared Memory
Memory A
Memory B

FIGURE 6.10: Shared memory diagnosis pattern – Solution 2 – Scenario 2.1: Partitioned system with functionalities with different criticality.

– ***Sub-scenario 2.2:*** *Partitioned system where some safety-related partitions execute the same functionality*

In sub-scenario 1.2 the overall solution for a device where two CPUs execute the same functionally is presented. That solution can also be applied to the partitioned multi-core device shown in 6.11. This scenario shall follow the steps stated in sub-scenario 1.2, taking into account that instead of having two CPUs, we have a single partitioned CPU.

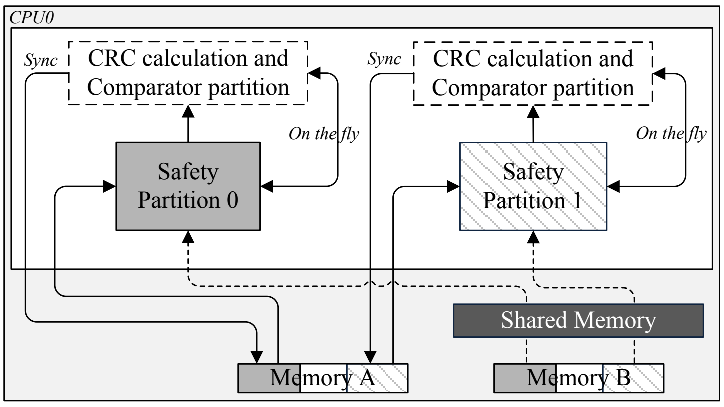

FIGURE 6.11: Shared memory diagnosis pattern – Solution 2 – Scenario 2.2: Partitioned system where safety-related partitions execute the same functionalities.

– ***Sub-scenario 2.3:*** *Partitioned-system where the partitions are executed on a CPU and a soft-core processor*

In sub-scenario 1.3 the overall solution for a COTS multi-core device where the same or different functionalities are executed on a CPU and a soft-core processor is presented. That solution can also be extended to partitioned multi-core devices such as shown in Figure 6.12. This scenario shall follow the steps stated in the sub-scenario 1.2, taking into account that instead of having two CPUs, we have a partitioned CPU and a soft-core processor.

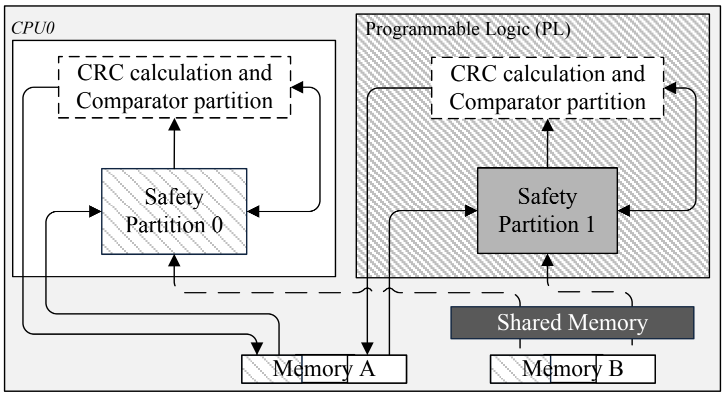

FIGURE 6.12: Shared memory diagnosis pattern – Solution 2 – Scenario 2.3: Partitioned system with a PS and a PL.

– ***Additional considerations***

In the above scenarios, several solutions to solve the issues regarding shared memories are identified. These solutions may be extended for improving their diagnosis coverage. For instance, partition redundancy can be implemented, thus enabling the comparison of the CRCs in different CPUs (see Figure 6.13).

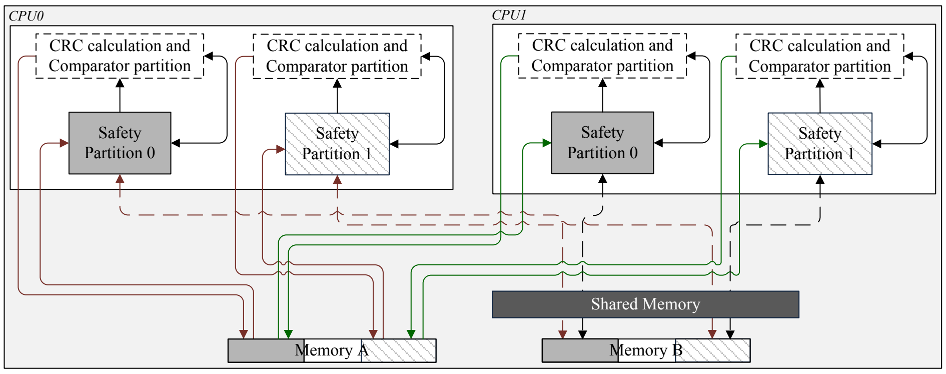
CPU0
CPU1
CRC calculation and Comparator partition
CRC calculation and Comparator partition
CRC calculation and Comparator partition
CRC calculation and Comparator partition
Safety Partition 0
Safety Partition 1
Safety Partition 0
Safety Partition 1
Shared Memory
Memory A
Memory B

FIGURE 6.13: Shared memory diagnosis pattern - Solution 2 - Scenario 2.4a: Additional considerations and solutions.

In addition, a diagnosis scenario with redundant HW can also be implemented (see Figure 6.14). This scenario executes the comparison of the CRCs at partition level and system level. This comparison can be realised internally by a SW comparator or externally by a HW comparator. In the case that a SW based comparator is implemented, a communication network wold be required to spread of the resulting CRCs through the entire system. This approach enables to detect whether the shared memory fails due to a failure of some component of the device or the device itself.

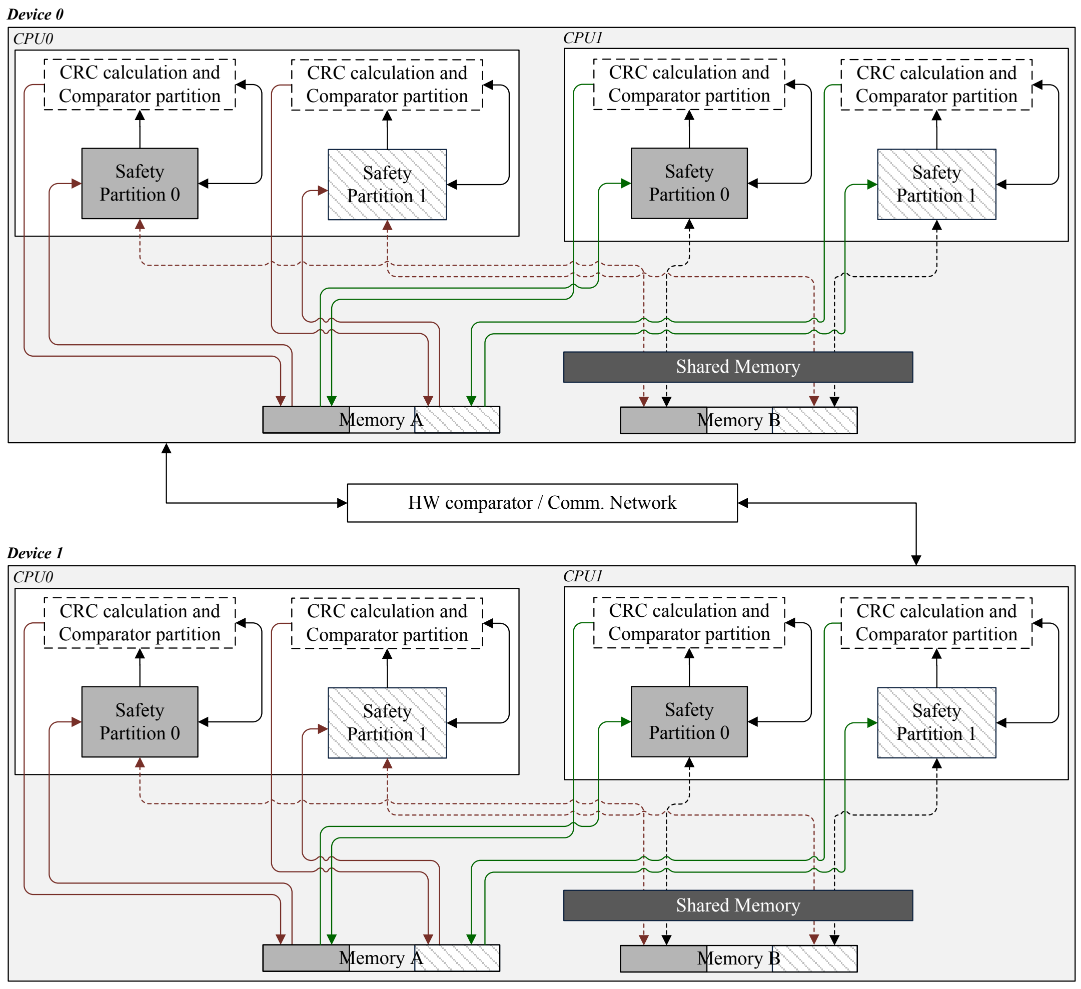

FIGURE 6.14: Shared memory diagnosis pattern - Solution 2 - Scenario 2.4b: Additional considerations and solution.

Chapter 7 presents the implementation of the solutions proposed in this cross-domain pattern, providing results that evidence the applicability of these solutions in a realistic wind turbine system.

6.2.2 Cache Coherency Management Diagnosis Pattern

Cache coherency is the consistency of shared resource data that ends up stored in multiple local caches (such as the L1 cache and L2 cache). For example, a coherency mechanism can store the copies of the data saved in several caches. When one copy of the data is modified, the other copy shall also be changed. Otherwise, an inconsistency arises. The *directory-based* approach, *snooping* and *snarfing* are the usually used coherency mechanisms. In a *directory-based* system, the shared data is stored in a common directory that is used for guaranteeing the coherency between the caches. This directory acts as a filter that grants permission to the processor to load an entry to the cache. When an entry is updated, the directory-based mechanism updates the other caches with the new entry. *Snooping* is a coherency protocol where the individual caches monitor address lines for accessing memory locations that they have cached, whereas *snarfing* is a mechanism where a cache controller watches both address and data in an attempt to update its own copy of a memory location when a second master modifies a location in the main memory. If a write operation is observed to a memory area where the cache has a copy of data, the cache controller will update its own copy located in the snarfed memory with the new data.

Today's mixed-criticality systems based on multi-core devices implement a coherency management unit for managing, among others, the coherency of the processors, the memory and the PL (if applicable). For instance, the Zynq-7000 zc706 multi-core device implements a SCU for managing the coherency of the memories (e.g., L1 cache, L2 cache, OCM) using the snooping coherency technique [XIL14c]. However, the snooping coherency technique does not fully guarantee the coherency of the memories. For example, the CPUA of the Figure 6.15 has a copy of a memory block from a previous read and CPUB changes the memory block. Consequently, in the case that the coherency management unit fails, the data of CPUA is not updated, leading to an inconsistency of data that can jeopardise the behaviour of the system. Here is where this cross domain pattern is focused, ensuring that changes of the data are propagated through the device and if not, detecting whether a coherency failure occurs.

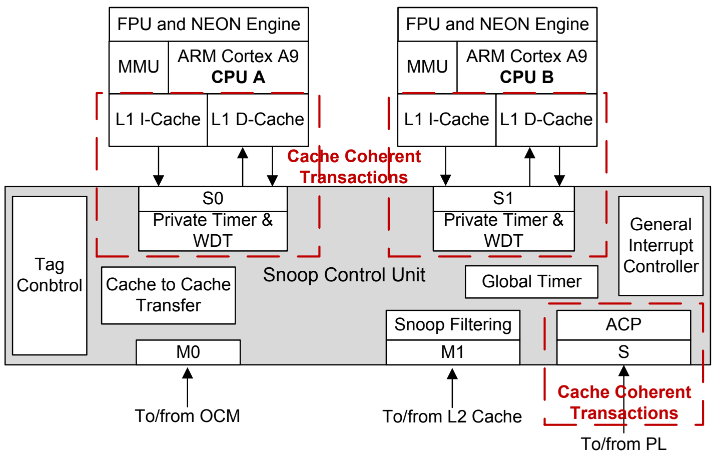

FIGURE 6.15: Coherency management diagnosis pattern – Zynq-7000 multi-core device – Overview.

This pattern defines the following three design solutions for checking the memory coherency unit from a safety perspective:

▶ **Solution 1:** **Check the configuration of the coherency management unit**

The coherency management unit shall be configured in a reasonable manner to minimise the interferences. The wrong or incorrect configuration of the coherency management unit may lead to the loss of coherency and the resultant failure of the system. Therefore, this first solution proposes to periodically check the configuration of the coherency management unit, comparing it with the expected configuration or the last valid configuration set. In addition, this solution assumes that the chosen configuration is free of systematic faults and that, therefore, a set of measures and diagnostic techniques for ensuring that it is protected against unexpected configuration changes should be implemented. For instance, the periodic read-back (see Table A.10 of IEC 61508-2), modification protection (see Table A.17 of IEC 61508-2) and the failure detection by on-line monitoring (see Table A.15 of IEC 61508-2) techniques may be implemented.

▶ **Solution 2:** **Diagnose random failures**

The software can manage the memory regions shared among certain sets of coherent masters. In addition, it can ensure that the shareability mappings between the masters are consistent to avoid unexpected behaviours and inconsistencies. For instance, a protection mechanism such as a MMU can be used to control the memory, manage permissions to blocks of the memory and translate the virtual addresses to physical addresses. This solution assumes that the coherency management unit implements a set of measures and diagnosis techniques to detect and control random faults such as the wrong addressing, partial update or single bit errors faults. For instance, WDTs can be used for detecting the temporal deviations, a CRC with comparison technique may be implemented for detecting unexpected data modifications, (see pattern ”Shared memory diagnosis” in Subsection 6.2.1) and an ECC and/or a parity bit technique can be implemented to detect data consistency violations, including partial update or single bit error failures.

▶ **Solution 3:** **Diagnose systematic failures**

Systematic faults can also affect the coherency management unit. These faults can be sourced from the HW design, the environmental stress, external influences and operational failures. This third solution considers the implementation of the measures and diagnostic techniques recommended in Tables A.15 to A.17 of IEC 61508-2 [IEC10b] for managing the systematic faults in the coherency unit. The selection of the measures and techniques depends on the HW available and the SW supported by the system architecture, giving rise to different combinations of measures and diagnostic techniques.

In addition to the solutions defined, this pattern considers that fault avoidance and fault control measures should be implemented in systems with cache coherency. For instance, the use of the shared memory may be limited to an absolute minimum required for operation, the use of multiple threads and tasks for one safety function can be restricted to a minimum and write accesses to the memory can be assigned statically to the tasks. The automatic invalidation of cache lines after a defined period is required to ensure that caches are flushed periodically and that they keep coherent. The faults can be controlled implementing communication protocols with additional messaging between sender and receiver of the information. For example, flags to indicate whether the information is updated and received can be applied. This procedure enables detecting the order violation fault.

Extra coding information may be integrated with the data to detect data consistency violations. To that end techniques such as CRC, ECC or parity information may be used. It is safe to assume that a HW that implements the ECC technique or the parity technique may have bugs (e.g., ARM: 751475—Parity error may not be reported on full cache line access (eviction / coherent data transfer / cp15 clean operations [Sem13]). Further techniques implement data structures that match the cache architecture (e.g., the maximum size of one cache line for optimal performance) and implement additional measures and diagnosis techniques. Examples are, ECC techniques, scrubbing, the implementation of timing expectations and error detection for the shared memory communication or common communication error related measures such as sequence numbers.

Chapter 7 presents the implementation of the solutions proposed in this cross-domain pattern in a wind turbine system, providing results that evidence the applicability of these solutions.

6.2.3 Inter-Connection Management Unit Diagnosis Pattern

COTS multi-core devices implement interconnection buses for the communication of the components with different criticality levels which are integrated on them. These interconnection buses can provide different protocols to switch the traffic through the components such as AMBA AXI and AXI Coherency Extension (ACE). For instance, the Zynq-7000 zc706 device implements an interconnection manager that includes a set of interconnect blocks or switches for managing, among others, the communication between the CPUs of the processor, the memories (e.g., OCM, DDR, L2 cache), the peripherals (e.g., I/O Peripheral (IOP)), the PL (if applicable). Figure 6.16 presents the block-diagram of the interconnections inside the Zynq device which are based on *interconnect* *master* blocks (e.g., Accelerator Coherency Port (ACP), AXI_HP (High Performance), AXI_GP (Generic Purpose), DMA, IOP), the *SCU*, the *central interconnect*, the *master interconnect*, the *slave interconnect*, the *memory interconnect* and the *OCM interconnect*.

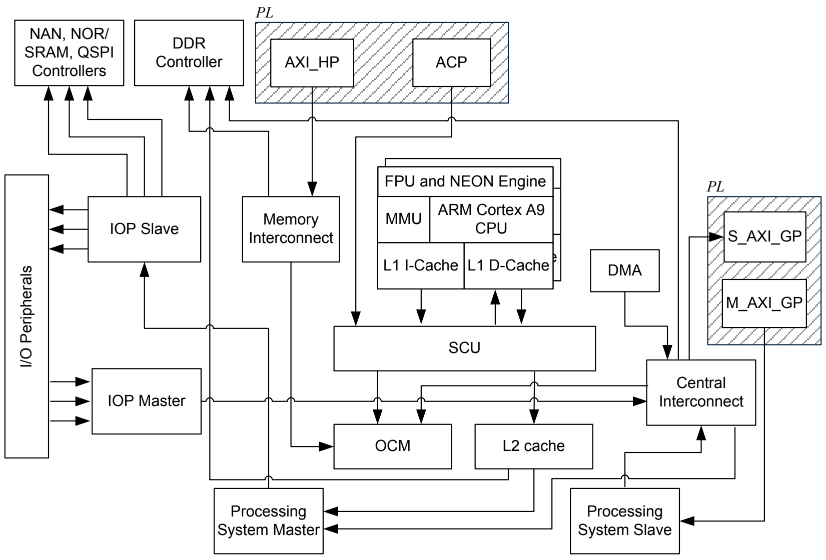

FIGURE 6.16: Zynq-7000 zc706 – Interconnection scheme – Overview.

The interconnection units, in general, are prone uncertainties related to their behaviours (e.g., lack of information). The interconnection scheme of each COTS multi-core device is unique. For instance, the Zynq zc706 multi-core device provides the interconnection scheme shown in Figure 6.16. This interconnection scheme includes several ports and interconnection blocks such as the AXI_HP and AXI_GP ports, going through several interconnect blocks such as the *central interconnect*, the *OCM*, the *SCU*, the *memory interconnect* or the *IOP*.

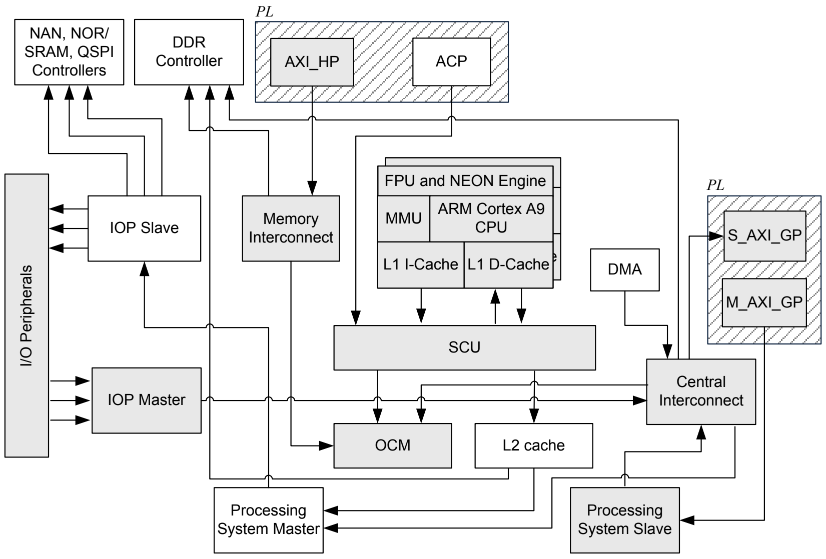

FIGURE 6.17: Interconnection – Certification challenges.

This cross-domain pattern aims to provide generic solutions for measuring and detecting faults in the interconnection management units. To that end, it assumes that the processing units of the multi-core processor, the cache memories (e.g., L1 and L2 memories), the OCM memory, the PL (if applicable) and its associated components (e.g., memory), the timers and the interrupt controller are checked in advance. This pattern considers the following three solutions for testing the interconnection management units.

▶ **Solution 1: Check the configuration of the interconnect management unit**

The interconnect management unit shall be configured in a reasonable manner to provide minimum possible interferences. The components or blocks that compose the interconnect manager are set up by means of registers. The configuration of their registers will be used to manage their behaviour, thus leading to an incorrect or partial behaviour. Therefore, to detect whether the configuration of the interconnection management unit changes, this solution proposes the implementation of the periodic read-back check with comparison of the interconnect manager's configuration registers.

▶ **Solution 2: Diagnose random failures**

In the case that the implemented system requires a maximum latency, the Quality-of-Service (QoS) modules can be used to ensure expected throughput and latency in the system design. The modules regulate the masters that do not guarantee maximum latency (e.g., CPU, DMA and IOP). Furthermore, they can be used to resolve issues related to contention by means of a two-level arbitration abstraction scheme. The first scheme is based on the priority indicated by the QoS register. The highest QoS value has top priority. The second scheme is based on a least recently granted scheme and is used when multiple requests are pending with the same QoS signal value. An interconnect manager shall consider, among others, measures and diagnostic techniques for typical faults such as wrong addressing or wrong data forwarding, including partial transmissions or single bit errors. For instance, a WDT can be implemented to detect temporal deviations and the CRC with comparison (see *Shared memory diagnosis pattern*), the ECC and the parity bit diagnostic techniques may be implemented to detect data consistency violations, considering partial updating or single bit error failures.

▶ **Solution 3: Diagnose systematic failures**

The interconnect manager can be affected by systematic faults which can be caused by the HW design, environmental stress or operational failures. This solution considers the implementation of the measures and diagnostic techniques recommended in Tables A.15 to A.17 of IEC 61508-2 for detecting and controlling the systematic faults of the interconnection management unit. Furthermore, it is assumed that the selection of measures and diagnostic techniques depends on the HW platform or the SW that is supported by the system architecture. Therefore, the selection of measures and diagnostic techniques for this purpose may vary. On the other hand, the possibility of systematic errors in the configuration of the interconnection management unit shall be addressed by these techniques.

6.2.4 Interrupt Controller Diagnosis Pattern

The interrupt controllers are components of today's COTS multi-core devices that may imply certification challenges. They are key components for managing the prioritisation of the tasks in multi-core devices. For instance, the Zynq-7000 device implements an interrupt controller named GIC to that end. The GIC is internally composed of one or more distributed blocks depending on the number of CPUs, and one or more CPU interface blocks (see Figure 6.18). The interrupt distributor centralises the sources of interrupts before dispatching the one with the highest priority level to an individual CPU. This controller ensures that one CPU can only take an interrupt targeted to several CPUs at a time. The sources of interrupts are identified by a unique interrupt ID number, a configurable priority and a list of the cores which are targeted. The interrupts that are handled by the interrupt controller can originate in cores (Private Peripheral Interrupts (PPIs), the PL and the PS (Shared Peripheral Interrupts (SPIs) and SW Generated Interrupts (SGIs)).

On the other hand, the core interfaces perform the interrupt priority masking and pre-emption handling for the cores of the device. Each core interface block provides an interface for each processor that operates within the GIC. In the case that the core interface implements the security extension, the Interrupt Request (IRQ) (non-secure) and Fast Interrupt reQuest (FIQ) (secure) signals may be used. The GIC also provides the write protection lock mechanism for preventing unauthorised accesses to the critical configuration registers.

However, at the event that the interrupt controller fails or that the request for an interrupt or the assignment of an interrupts fails, the execution of the processors will be affected. In multi-core mixed-criticality systems, the interrupt controller is used for guaranteeing and managing the execution of the functionalities with different criticality level. Table A.1 of IEC 61508-2 [IEC10b] defines the requirements for faults that shall be detected and measured to guarantee the safety of the interrupt handling. However, this standard is intended for single computing systems where a resource is not shared between more than one component, and therefore, the measures and diagnosis techniques recommended by this standard (see Annex F of IEC 61508-2) are not at all applicable to the interrupt controllers used for managing the execution of functionalities with different criticality on

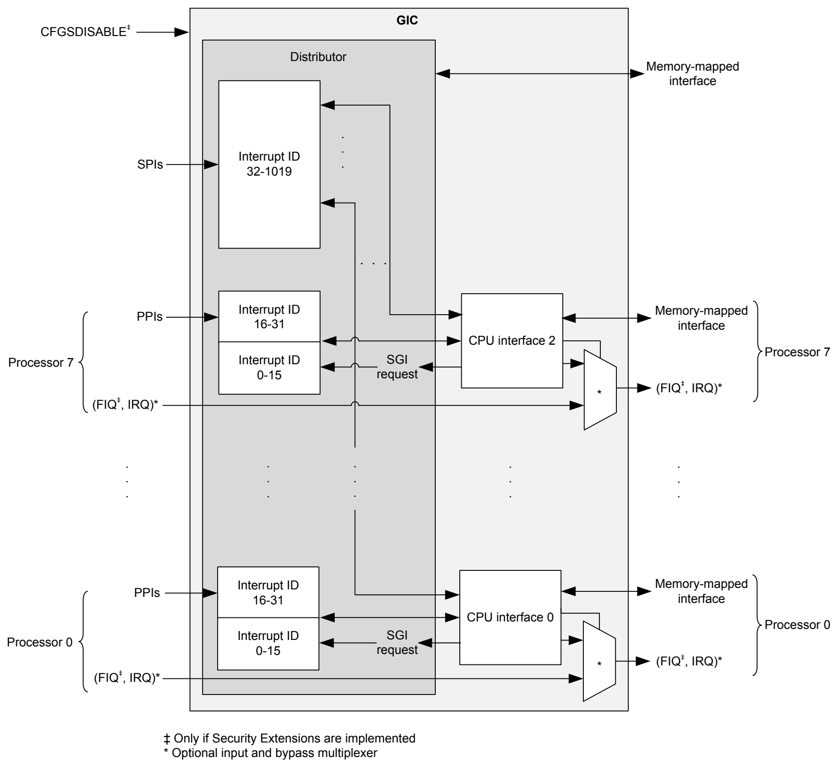

FIGURE 6.18: Generic Interrupt Controller of Zynq-7000 device – Block diagram (Source [XIL14c]).

today's mixed-criticality systems. This cross-domain pattern defines the following design solutions for diagnosing the interrupt controllers. Furthermore, it is assumed that the CPUs of the device, the L1 and L2 cache memories and the OCM memory, the PL and associated components (such as memories), the timers, the interconnection management units and the coherency management units are checked in advance.

▶ **Solution 1: Check the configuration of the interrupt controller unit**

Figure 6.18 shows the block diagram architecture of an interrupt controller composed of a distributor and one or more CPU interfaces. These components can be configured independently by means of registers. This first solution proposes to periodically check the configuration registers of the interrupt controller to detect whether the configuration is modified.

▶ **Solution 2: Diagnose random failures**

The interrupt controller component can be the subject to unexpected internal failures which can be caused by direct-current (DC) faults, drift and oscillations and reset-related faults. In Table A.1 of IEC 61508-2 [IEC10b] techniques and measures for diagnostics and recommended maximum levels of diagnostic coverage for an interrupt controller are defined.

► **Solution 3:** **Diagnose systematic failures**

Tables A.15 to A.17 of IEC 61508-2 [IEC10b] recommend techniques and measures for controlling systematic failures, including techniques and measures to control systematic failures caused by the HW design, environmental stress or operational failures. These techniques shall be implemented to detect systematic faults that can occur in the GIC. For example, the possibility of systematic errors in the configuration of the interconnection management unit shall be addressed using these techniques.

6.3 Mixed-Criticality Network

May mixed-criticality systems implement shared memories for the communication of their components including the transactions between functionalities with different criticality. However, the use shared memories may imply interferences on the communication such as the bottleneck effects. NoCs embed the solutions associated with traditional off-chip networks into the chip. They can be implemented for safety-critical applications, providing support for TT and Event-Triggered (ET) (RC and BE) traffic. TT NoCs are predictable communication systems with inherent non-interferences among their components (such as processing cores, peripherals and memories). Instead, ET NoCs are non-predictable networks.

The shift towards NoCs leads to challenges such as supporting multiple types of communication as well as supporting applications with different criticality levels. For instance, the TT Network-on-Chip (TTNoC) communication system [OESHK08] does not support the transmission of ET messages and AEthereal NoC [GDR05] does not support the transmission of RC messages. However, the integration of functionalities with different criticality on a single computing platform communicating through a NoC can lead to communication errors.

This cross-domain pattern aims to provide a generic fault-tolerant on-chip communication network with support for different computation and communication models as well as mixed-criticality systems.

6.3.1 Network-on-Chip Diagnosis Pattern

This pattern defines a generic solution that manages the prioritisation of communication for different criticalities with scheduling, routing, traffic shaping and error detection. In accordance with the IEC 61508 safety standard, this pattern can be considered as a SCL network implemented on top of a *black channel* network (see Figure 6.19). Therefore, it is assumed that parts of the communication channel,i.e., the NoC cannot be designed, implemented and validated according to a safety standard. Instead, this SCL shall be compliant with a safety standard (such as IEC 61508 and IEC 61784-3). To that end, the SCL must fulfil the safety-related requirements stated in the safety standard, including a safety life-cycle development process. A linking analysis of this NoC pattern is presented in [LPN^{+}16] where the way in which this SCL fulfils the safety-related requirements defined in the MSC for an IEC 61508 compliant mixed-criticality network (see Section 5.4) is analysed.

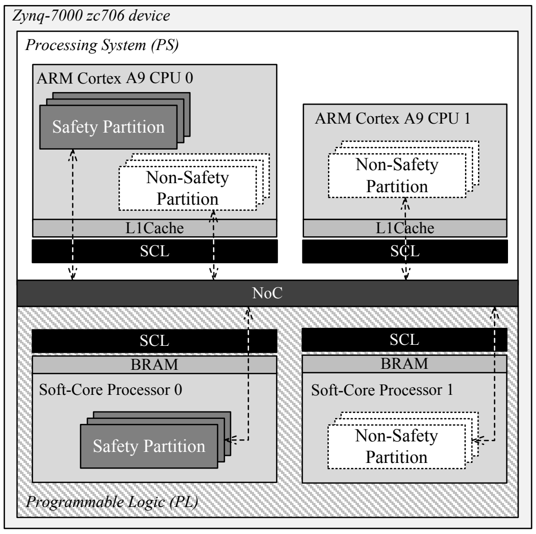

FIGURE 6.19: NoC cross-domain pattern – Overview.

This pattern supports the following requirements to schedule, route, shape traffic and detect the communication errors:

– *Support for TT and ET (BE and RC) timing models:* TT messages are periodically transmitted for achieving predictable timing with minimal latency and no jitter. Instead, BE messages do not have timing restrictions and do not fulfil requirements of non safety applications. RC messages offer a reasonable trade-off between resource reservation and latency.

– *Provide fault-isolation:* This NoC pattern provides fault-isolation at SW level using a hypervisor virtualization mechanism and establishes partitioning at HW level using encapsulated communication channels.

– *Compatibility to a wide range of NoCs:* This pattern shall be integrable on a broad variety of NoCs, enabling the system to support both TT and ET communications, despite only ET transmission are backed by the underlying network.

– *Support of hard real-time applications:* This pattern shall ensure that messages of the system meet the pre-specified deadlines in all situations defined in [45]. For this purpose, this pattern shall provide a scheduler that enables to achieve deterministic communication.

– *Support of mixed-criticality systems:* The communication of applications with different criticality levels that interact and coexist on a shared computing platform requires protection mechanisms that establish chip-wide segregation. Virtualization mechanisms such as hypervisors are not considered enough to establish the segregation because non safety partitions can influence the safety-related ones. Therefore, this approach shall provide rigid temporal and spatial partitioning by setting up a chip-wide partitioning.

– *Establish temporal and spatial segregation:* It establishes temporal segregation among TT messages assigning different time slots to the TT messages of each tile and guaranteeing that no other tile can inject messages within the slots of other tiles. In addition, this NoC pattern ensures the temporal segregation between TT and ET messages. ET messages can be injected into the network only if there is no ongoing or upcoming TT communication.

  On the other hand, this SCL ensures spatial segregation. To that end, it assigns separate memory areas to the ports of the network for storing the messages and establishes that the messages with different criticality levels are routed through separate paths (source based routing).

This NoC shall further consider IEC 61508-2 and IEC 61784-3 compliant measures and diagnostic techniques to assure that all safety-related failures are detected and controlled. An overview of the measures and diagnostic techniques recommended by those standards is presented in Section 5.4.

Chapter 7

Case Study – Wind Turbine

This chapter presents the integration and evaluation of the solutions defined throughout this dissertation in a wind power case study. The solutions envisaged include a modular development process based on the DREAMS architecture style (see Chapter 4), the MSCs for an IEC 61508 compliant hypervisor, partition, COTS multi-core processor and mixed-criticality network presented in Chapter 5 and the reusable generic cross-domain patterns defined in Chapter 6. A real wind turbine installation, which is commonly referred to as Wind Park, is largely composed of a set of interconnected wind turbines and a centralised control centre (see Figure 7.1). The control unit is made up of one or more supervision subsystems, Human Machine Interface (HMI), communication and safety-related subsystems. Supervision subsystems perform real-time control and supervision of the wind turbine, while the safety-related subsystems assure that the design limits of the wind turbine are not exceeded. These subsystems can be accessed directly or via

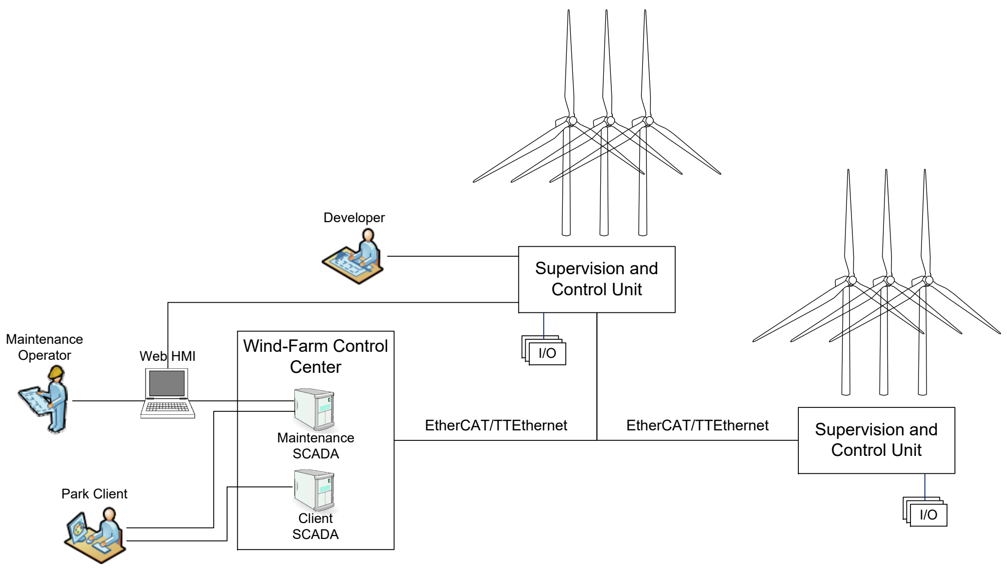

FIGURE 7.1: A wind-farm – Overview.

the web for fixing bugs, upgrading the system, performing corrective and/or preventive maintenance activities and collecting data for performance analysis.

This case study focuses on the integration of the safety-related solutions defined in this thesis onto a safety wind-turbine system. It identifies and lists the safety-related standards applicable for developing wind-turbine systems (see Section 7.1), presents the HW architectural style of a simplified wind turbine system (see Section 7.2), introduces the safety-related requirements of a safety wind turbine system from a product line perspective (see Section 7.3) and exposes the results from executing selected mixed-criticality cross-domain patterns (see Section 7.4).

7.1 Safety Standards

This section lists the different directives and standards that are applied for the developing of a wind power system. However, the IEC 61508 safety standard is taken as the reference standard in this thesis for defining the technical integration of the solutions identified in the previous chapters into a wind turbine system.

- Directives:
    - 2001/108/EC Electromagnetic Compatibility (EMC)
    - 2011/65/EU Restriction of the use of certain hazardous substances (RoHS)
- Standards:
    - IEC 61508 Functional safety of E/E/PE safety-related systems [IEC10a] [IEC10b] [IEC10c]
    - IEC 61400 Wind turbine generator systems [AS14a] [AS14b]
    - ISO 13849 Safety machinery – Safety-related parts of control systems [ISO06][ISO12]
    - IEC 60204-1 Safety of machinery – Electrical equipments of machines [IEC06]

7.2 System Architecture

This case study for the sake of simplicity is based on the wind turbine system architecture that can be shown in Figure 7.2, where the supervision, control and protection units are only considered. These units run on top of a real-time platform named Galileo and a Zynq-7000 zc706 harmonised platform [XIL14c], which are interconnected through a PCIe bus. Nevertheless, according to the IEC 61508 safety standard, these HW platforms can be considered as independent HW systems. The Galileo platform supervises and controls the wind turbine system. It is based on an industrial PC APC 910 (commercial HW) [Hea15] and it is customised at the operating system level and SW level. On the other hand, the harmonised platform implements the safety-related functionalities and integrates the results from this dissertation. The Zynq-7000 zc706 multi-core device is a programmable System on a Chip (SoC) that supports a PS with a dual ARM Cortex A9 MPCore and a PL into a single silicon chip.

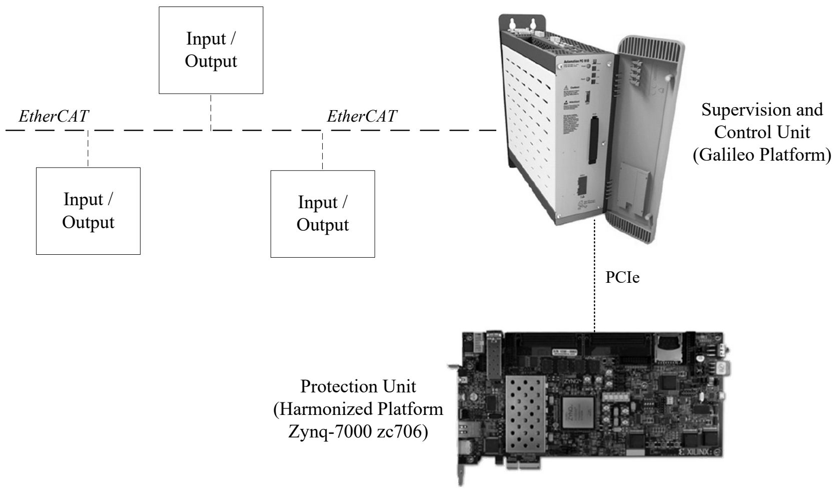
Input / Output
EtherCAT
EtherCAT
Input / Output
Input / Output
Supervision and Control Unit (Galileo Platform)
PCIe
Protection Unit (Harmonized Platform Zynq-7000 zc706)

FIGURE 7.2: Wind turbine use case – Overall architecture.

This system architecture supports the execution of functionalities with different criticality levels (such as SIL1 to 3 according to IEC 61508). To that end, XtratuM hypervisor [Sol14] is used, splitting the CPUs of the PS and the soft-core(s) of the PL into partitions where the functionalities with different criticality are executed. Partitions carried out by the wind turbine system can be classified into safety-related partitions (e.g., safety-protection partition), non safety-related partitions (e.g., supervision partitions), diagnostic partitions and further partitions (e.g., server patterns). Safety-related partitions execute the safety-related functionalities such as reacting to the failures detected by the diagnostic partition(s). Instead, non safety partitions supervise the behaviour of the system's execution, and diagnostic partitions execute the safety-related tests for detecting failures that can jeopardise the safety of the system. Further partitions may be implemented in this system architecture to centralise certain functionalities such as the communication peripherals of the HW platform (e.g., digital I/Os).

The protection unit shown in Figure 7.2 communicates with external sensors (e.g., wind speed sensor) and actuators (e.g., safety relay) through a safe fieldbus protocol composed of a non-safe fieldbus EtherCAT and a SCL integrated on top of a NoC. The combination of the NoC and the SCL enables temporal and spatial independences, which depend if a shared memory is used or not to communicate the partitions. The NoC implemented in this case study is the STNoC, which is complemented with the NoC SCL cross-domain pattern (see Subsection 6.3.1). The SCL guarantees a safe communication between the partitions. Figure 7.3 presents the overall architecture style of the protection system (combination of the HW and the SW) implemented onto the partitioned and networked Zynq-7000 zc706 multi-core platform.

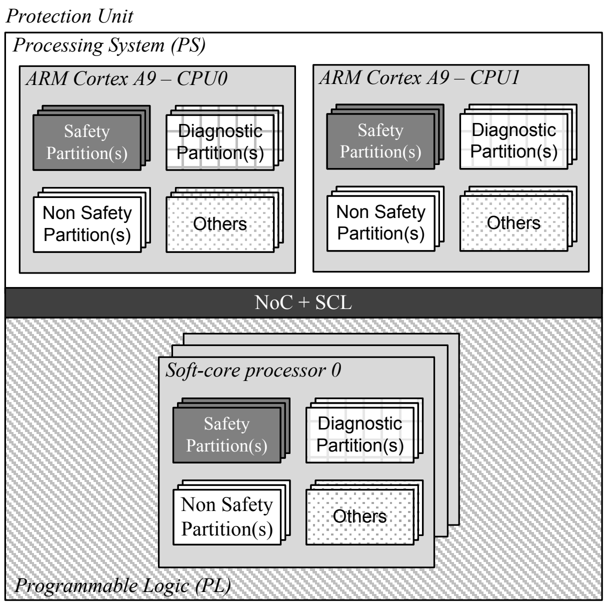

FIGURE 7.3: Wind turbine case study – Block diagram of the protection unit based on the Zynq-7000 zc706 device.

The components presented in the preceding paragraphs have been analysed from a safety perspective to identify the safety-related arguments that they fulfil and detect inconsistencies. To that end, the following subsections present the linking analyses where the way in which those components fulfil the safety-related arguments presented in the MSCs is analysed.

7.2.1 XtratuM Hypervisor – Linking Analysis

This subsection defines a brief representative example of a *linking analysis* based on the commercially available hypervisor XtratuM used in dependable aerospace applications. XtratuM is not certified as a compliant item. Instead, it implements multiple IEC 61508 compliant safety techniques, safety functions and requirements. Within the scope of this analysis, XtratuM is considered to be an IEC 61508 compliant hypervisor by means of *Route 3S "assessment of non compliant development"* [IEC10c]. This route implies that the hypervisor shall fulfil all the requirements stated in subsection 7.4.2.13 of IEC 61508-3 [IEC10c]. Among others, the XtratuM hypervisor shall meet the specification of the functional safety requirements and the safety behaviour for hypervisors in its application.

XtratuM uses up to five qualified tools to automate its configuration (*xmeformat*, *xm-pack*, *rswbuid* and *xmcparse*). Figure 7.4 shows the inputs, outputs, interactions and intermediate results of the tools used by this hypervisor. From a safety perspective and according to the IEC 61508 safety standard, these tools can be considered off-line tools of category T3 with support for a *safe installation procedure*. T3 tools generate outputs which can directly or indirectly contribute to the executable code of the safety-related system (e.g., compilers).

– ***xmcparser*** tool handles the hypervisor's configuration. To that end, it reads the XtratuM Configuration File (XMCF) and generates a binary file containing the hypervisor's and partitions' configuration data. This tool also checks the consistency of the configuration parameters of the hypervisor.

– ***xmeformat*** and ***xmpack*** tools generate a binary package (*container.bin*) containing the hypervisor's and partitions' configuration.

– ***rswbuild*** tool appends the package (*container.bin*) right after the resident SW code to generate the final image which is written to an invariable memory. The resident SW code is the SW that is executed before the hypervisor's execution.

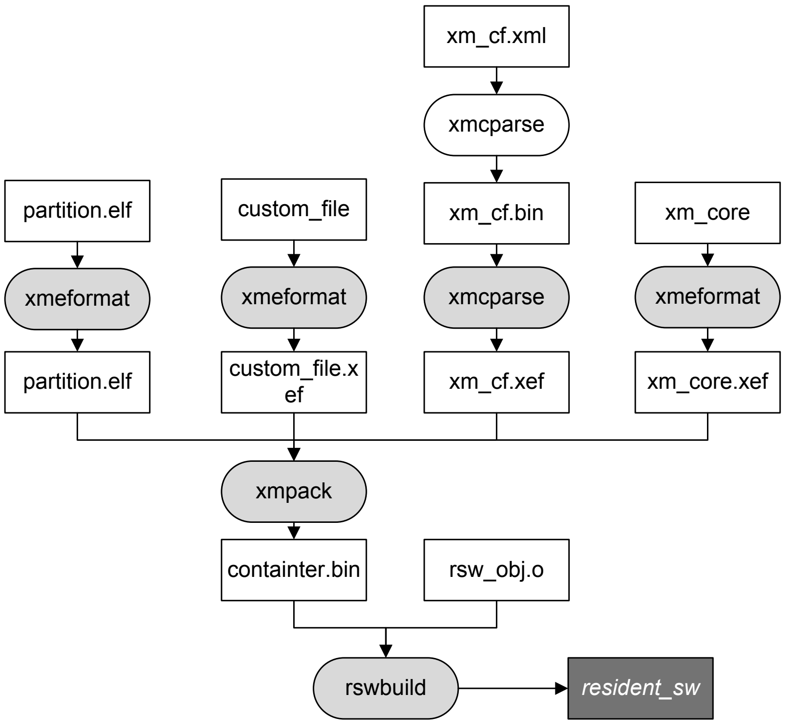

FIGURE 7.4: XtratuM Tool set. (Source DREAMS D5.1.1 Annex A [DRE15])

This hypervisor is considered to be active (*startup*) when the resident SW has finished copying the code and the data of the hypervisor and partitions to the memory. The *boot* sequence continues with the configuration and checking of the hypervisor, partitions and their associated resources. Once finished, XtratuM initializes the partitions' internal data and waits until the initialization reaches the state of all the cores. Finally, the hypervisor executes the schedule for each core.

The *shutdown* sequence of the XtratuM hypervisor is given in a deterministic way where partitions finish their executions first and the hypervisor is shut down after wards. The temporal boundaries of this process are configured in the XMCF and they are also included as a part of the hypervisor's documentation.

On the other hand, XtratuM supports a basic virtualization interface for *virtualizing resources*. This interface avoids the failures related to the assignment of one resource to more than one partition. The resource virtualization is usually based on hypercalls which provide access to the hypervisor's services such as the real-time clocks, interrupts and communications. The only restriction is that the virtualization is limited to resources that may endanger the hypervisor's integrity, temporal independence or spatial independence.

- ***Timer virtualization*** is supported to get the status of the virtualized timers and simulate the arrival of the partition's interruption when the timer expires.
- Interrupt masks, exceptions and IRQs are managed by the hypervisor by means of ***IRQ controller virtualization*** and vector based exception/IRQ handling.
- The HW dependency which is related to the presence of any component of the processor such as a MMU or a Memory Protection Unit (MPU) is an issue to be considered. The ***MMU virtualization*** enables modifying the page tables of the partitions that must be compliant with the physical memory areas. These physical memory areas are specified in the XMCF.

The XtratuM hypervisor also avoids interferences caused by the assignment of the peripherals (e.g., I/Os). To that end, it limits the assignments of peripherals to certain specific partitions (*exclusive access to peripherals*). The assignment is configured off-line, during the system design phase, by means of the XMCF. In that regards, XtratuM considers to set the access to certain bits of a specific I/O address using a bit mask or to configure the access to contiguous I/O regions.

The execution of safety-related and non safety-related partitions can lead to interferences precluding the achievement of the *temporal independence*. To address issues of this nature, XtratuM supports a cyclic execution scheduling that follows the ARINC 653 specification [ARI06]. The hypervisor's cyclic scheduling is supported by one-shot timer interrupts which are statically allocated to the partitions in the XMCF. Figure 7.5 shows an example of the deterministic execution scheduling implemented by the XtratuM hypervisor. Nevertheless, the scheduler of the hypervisor and partitions can be changed depending on the needs of the system to be partitioned (e.g., maximum and minimum execution time, maximum and minimum response time).

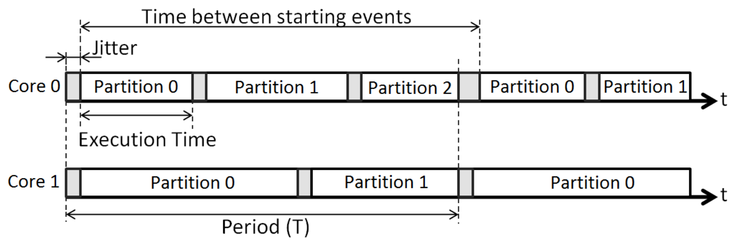

FIGURE 7.5: Cyclic execution scheduling for the XtratuM hypervisor – Example.

The basic concept of spatial independence aims to detect and avoid unauthorised accesses to the memory areas and the peripherals assigned to partitions. Partitions can be allocated by XtratuM in different physical memory addresses to prevent that one partition can access to memory areas of other partitions. Safety partitions check that the contents of their memory areas are not accidentally or intentionally modified. A safety partition periodically calculates the CRC of the hypervisor's code and compares it with the off-line calculated CRC value. This check enables detecting when a modification on the memory occurs. In the same vein, the direct I/O memory accesses are prohibited for partitions and they can only be granted using the hypervisor's hypercalls. All the access requests for peripherals are managed by the MMU and MPU.

The communication between partitions and between the partitions and their assigned components is supported by specific services provided by the XtratuM hypervisor (e.g., port based communication). These services operate in a similar manner to the inter-partition services described in the ARINC 653 standard [ARI06]. They are based on channels and endpoints, which are configured off-line in the XMCF. The memory used for storing the messages of the communication channels is only accessible by XtratuM. If a partition requires communication, the communication messages shall be copied to the memory areas controlled by XtratuM and the respective partition. Therefore, there is no way that partitions can accidentally or intentionally modify the contents of the messages. In addition, the limitation of the number of messages and the message size preserves the system integrity by statically allocating the required memory areas during the initialisation of the XtratuM hypervisor and bounding interferences in the communication.

This hypervisor uses a global notion of time when it is executed on Symmetric Multi-Processing (SMP) architectures. The fault-tolerant clock synchronisation is academic, with no claim of compliance with the IEC 61508 standard. The fault-tolerant global notion of time relies on a fault-tolerant HW clock and fault-tolerant timers which are configured in the XMCF. Furthermore, as defined in Section 5.1, an IEC 61508 compliant hypervisor must implement a set of measures and diagnostic techniques to detect and control random and systematic faults. In the case of the XtratuM hypervisor, it provides the following set of IEC 61508 compliant measures and diagnostic techniques to that end.

▶ **Random Failures**

- ***Program sequence monitoring*** to diagnose failures related to the critical execution patterns of the hypervisor, e.g., initialisation, partition context-switch, exception handling. The last one is especially relevant since it contributes to the fault-tolerance mechanisms.
- ***Temporal monitoring mechanisms*** such as WDTs with a separate time-base complement the program sequence monitoring technique.
- XtratuM is deployed with a variable monitor by the resident SW tool. Therefore, it does not require any invariable memory monitoring mechanism. The resident SW tool is responsible of checking, by means of a checksum, that the hypervisor's image is not modified. If any partition is deployed into the invariable memory, a safety partition shall be responsible for checking the appropriate memory areas during the startup and initialization sequences.
- The run-time data of XtratuM shall be protected by *HW ECC* mechanisms. The correctness of the ECC shall be tested by a third-party entity.
- XtratuM does not use any I/O unit. It is assumed that the partitions which use I/Os for safety purposes are responsible for their diagnoses.
- The implementation of extra measures and diagnostic techniques for the detection of communication and mass-storage failures is the responsibility of the partitions.

▶ **Random Failures**

- A ***program sequence monitoring*** technique is supported by XtratuM for random failure diagnosis.
- The XtratuM hypervisor implements the ***defensive programming*** technique to diagnose the interfaces between the partitions and the HW.
- ***Failure assertion programming*** is employed on the upper and lower limits of the stacks that are used by XtratuM to manage the partitions.

Once a random or systematic failure is detected, the hypervisor should react to the unusual event or state by executing a system reaction to an error or a fault-tolerance mechanism. XtratuM implements a Health Monitoring (HM) function for discovering the faults at an early stage and masking or isolating the failure to avoid its consequences [Sol13, MRC11]. The HM defines four different execution scopes, depending on which part of the system is affected (partition process, operating system, XtratuM code or resident SW) and defines their reactions in the XMCF (e.g., ignore the event, notify to the partition, warm restart or cold restart).

It is worth stressing that when a safety hypervisor is ported to a new ECU, a recertification process must be carried out. This process includes the redefinition of its associated safe states, intended uses and documentation.

7.2.2 Safety Protection Partition – Linking Analysis

This subsection defines a brief representative *linking analysis* of a safety protection partition defined in the Multi-cores Partitioning for Trusted Embedded Systems (MultiPARTES) project [PGN$^+$14]. Figure 7.6 shows an example of a partitioned heterogeneous quad-core processor that implements partitions with different criticality levels (i.e., safety partitions, non-safety partitions) and where resources are shared between the partitions (e.g., L2 cache, extended shared memory, I/Os).

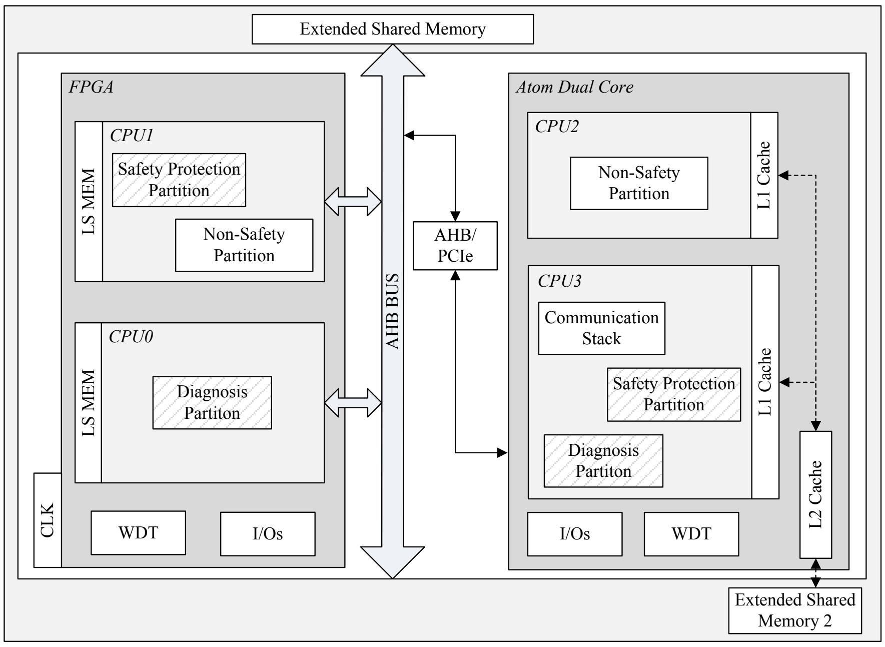

FIGURE 7.6: Partitioned mixed-criticality system – Example.

The safety protection partition does not require additional startup and initialization functions. The safety hypervisor performs these functions. The safety startup and initialization implemented by the safety hypervisor manages and updates the outputs of the safety partitions at execution time. Likewise, no new *shutdown* function is required by the safety partitions. In the specific case of the diagnosis safety partitions shown in Figure 7.6, they support platform independent *measures and diagnostic techniques* to detect faults during execution. For instance, the Error Detection Correction (EDC), *temporal and logical monitoring*, *defensive programming* and *diverse monitoring* techniques may be provided by a safety-related diagnosis partition. In addition, platform dependent diagnosis techniques such as the reciprocal comparison of memory and I/Os may be provided by those partitions.

The overall *system response to errors* executed by the safety partitions considers different scenarios where reaching a safe state is required. For instance, if a fatal error is detected, the safety protection partition will stop the execution. Furthermore, in the case that a diagnostic partition does not receive the periodic confirmation from the safety protection partition, the watchdog of the system will not be refreshed, thus leading to a safe state.

7.2.3 Zynq-7000 Multi-Core Device – Linking Analysis

This subsection provides a brief representative example of a *linking analysis* between the MSC that is defined in Section 5.3 and the commercially available Zynq-7000 zc706 device. This device is a programmable SoC property of Xilinx, Inc., which integrates a PS with dual ARM Cortex A9 MPCore and a PL in a single device. Figure 7.7 shows the major functional blocks of a Zynq-7000 device. This multi-core device includes, among others, two processor cores, a programmable logic, instruction and data L1 caches for each core, an L2 shared cache, an interconnect and coherency management unit (SCU), an OCM, DDR memories and multiplexed I/Os. Regarding safety standards and certification, this device is not an IEC 61508 compliant device. Therefore, it is used as provided by the *device manufacturer* for fail-safe systems.

This multi-core device offers different processing units, including the CPUs of the PS and the soft-core processors of the PL, in lockstep and normal modes [HCM15]. These implementation modes (shown in Figure 7.9 and Figure 7.8) impact on the selection of the device's measures, diagnostic techniques, system reactions to errors and fault-tolerance techniques. For instance, if the scenario composed of two soft-core processors in lockstep mode is implemented, the *Comparator* technique will be one of the recommended mechanisms to diagnose the processing units (see Annex A of IEC 61508-2). Conversely, if lockstep mode is not implemented, the *Self-Test by SW: Walking bit* [IEC10b] diagnostic technique will be recommended.

For the sake of simplification, the scenario composed of a soft-core processor and a single CPU is used as the reference scenario in this linking analysis (see Figure 7.8). This analysis identifies the safety-related arguments which are met by the Zynq-7000

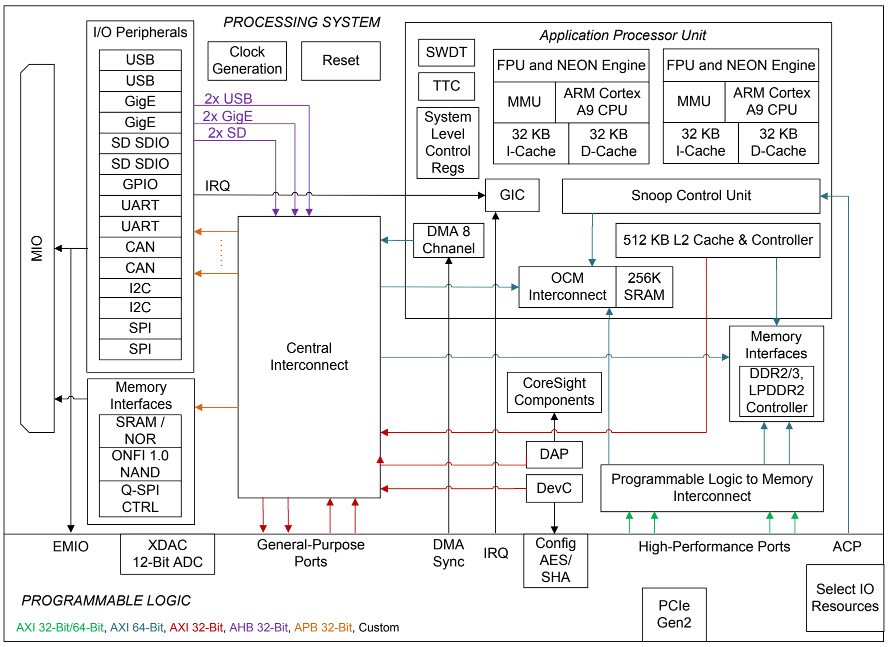

FIGURE 7.7: Zynq-7000 – Block diagram. (Source UG585 [XIL14c])

device, the measures and diagnostic techniques which are implemented and supported by this device and the parts of the safety techniques stated in the MSC for an IEC 61508 compliant COTS multi-core device which are delegated to other components. This MSC is defined in Section 5.3.

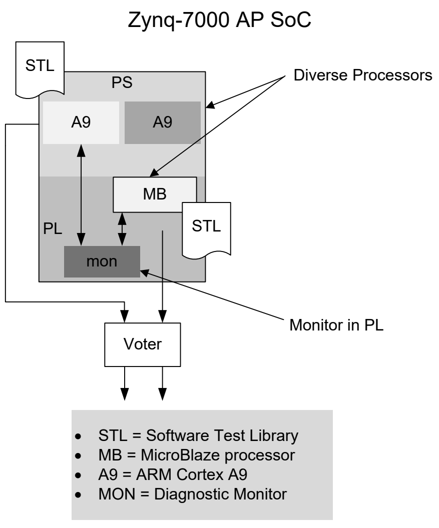

FIGURE 7.8: Zynq-7000 – Normal mode. (Source WP461 [HCM15])

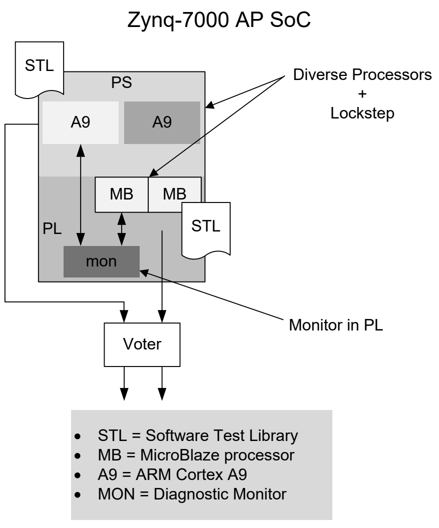

FIGURE 7.9: Zynq-7000 – Lockstep mode. (Source WP461 [HCM15])

In relation to the safe power up and power down techniques, the Zynq device provides separated and isolated power domains for the PS and the PL. These power domains are powered up and powered down independently and in a deterministic way (see Figure 7.10). The PS can be powered up and powered down regardless of the PL, although for security reasons, the PL cannot be powered before the PS [XIL15]. The feed sequence and time bounds for the power up and power down of the Zynq device are specified in [XIL15], where the optimised sequences for achieving the minimum current are indicated. In the case of the Zynq-7000 device, it executes an optimised power up sequence where the PS_VCCINT is first powered up, followed by the activation of the PS_VCCAUX, PS_VCCPLL, PS_VCCO, PL_VCCINT, PL_VCCBRAM, PL_VCCAUX and PL_VCCO power signals. In the case of the power down sequence, it follows a reverse sequence of the power up sequence.

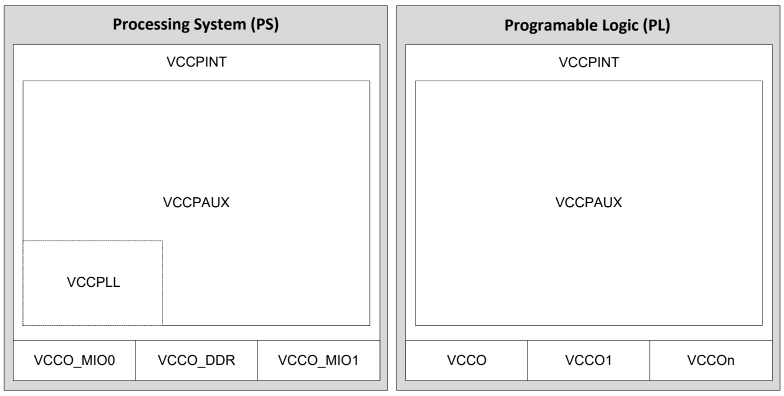

FIGURE 7.10: Zynq-7000 – Power Up and Power Down sequences. (Source UG585 [XIL15])

The *boot* sequence is an inclusive safety technique in the Zynq device [XIL14c]. This device supports two boot modes, a normal mode and a secure mode (see Figure 7.11). The main difference between them is that in the secure boot mode, the First Stage Boot Loader (FSBL)/User code is authenticated (Rivest, Shamir and Adleman (RSA)) and decrypted (Advanced Encryption Standard (AES)/Hash Message Authentication Code (HMAC)), whereas in the normal boot the FSBL/User code is executed directly from the boot device (e.g., Secure Digital (SD) card or RAM). If any of the checks fails during the boot sequence, the contents of the memory and registers of the PS and the PL shall be deleted; the PL shall be shutdown and a secure lock-down state shall be executed.

The Zynq device is always powered up in secure boot mode, only switching to non-secure boot mode when it detects that the FSBL is not encrypted. The boot of the Zynq device is a two-stage process, where the code of the internal BootROM and the FSBL are executed. In Stage 0, CPU0 starts executing the code from the internal BootROM, whereas, in stage 1, CPU0 executes the FSBL from the OCM. Once the PS is fully operational, the configuration of the PL is carried out by providing a bitstream.

Regarding the *shutdown* of the Zynq device, this process takes place before the powering-down of the device to keep the system stable, thereby avoiding the corruption of the file

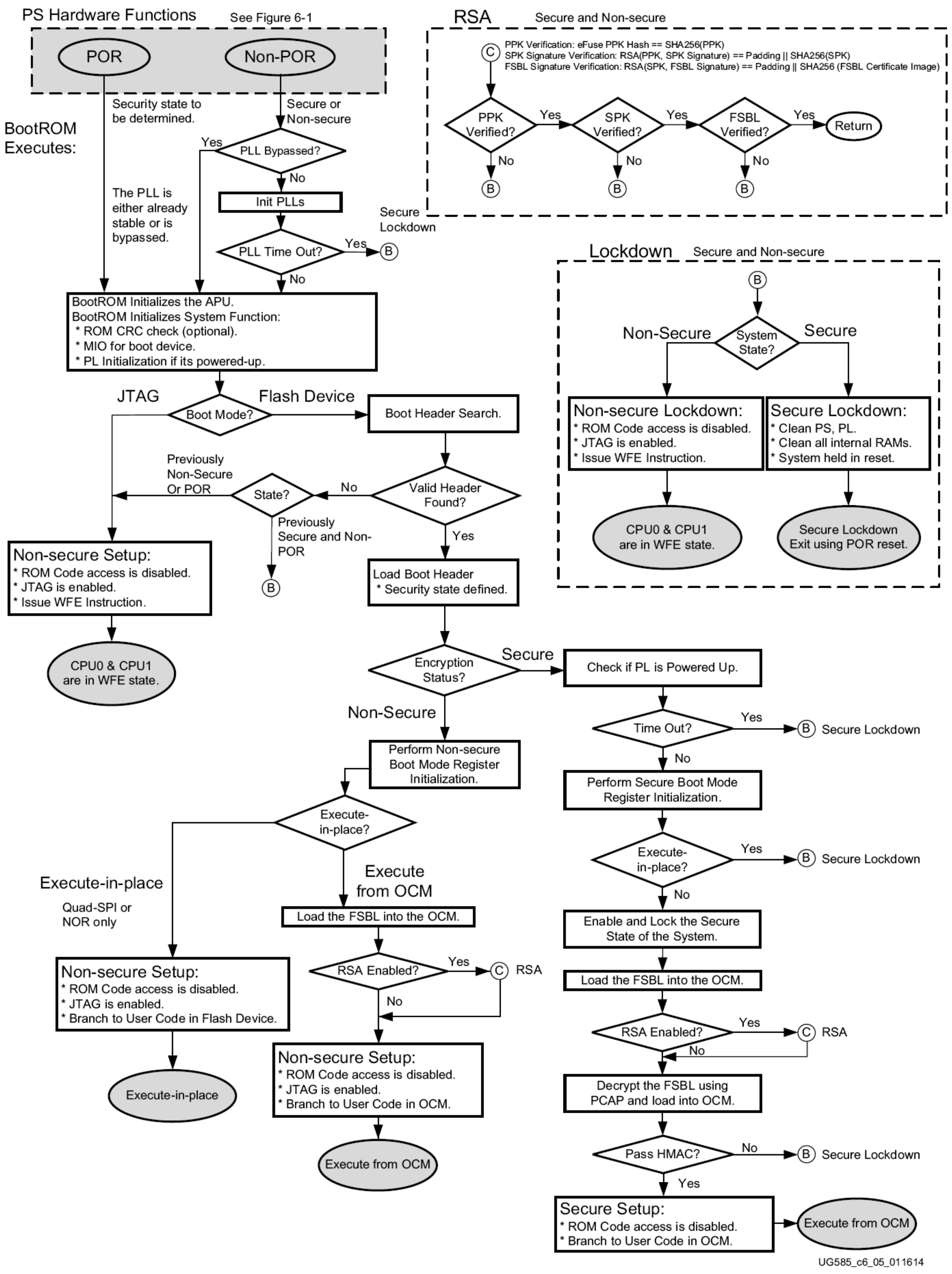
PS Hardware Functions
See Figure 6-1
POR
Non-POR
Security state to be determined.
Secure or Non-secure
BootROM Executes:
Yes
PLL Bypassed?
No
Init PLLs
The PLL is either already stable or is bypassed.
PLL Time Out?
Yes
B
No
BootROM Initializes the APU.
BootROM Initializes System Function:
* ROM CRC check (optional).
* MIO for boot device.
* PL Initialization if its powered-up.
JTAG
Boot Mode?
Flash Device
Boot Header Search.
Previously Non-Secure Or POR
State?
No
Valid Header Found?
Previously Secure and Non-POR
Yes
Non-secure Setup:
* ROM Code access is disabled.
* JTAG is enabled.
* Issue WFE Instruction.
B
Load Boot Header
* Security state defined.
CPU0 & CPU1 are in WFE state.
Encryption Status?
Secure
Check if PL is Powered Up.
Non-Secure
Time Out?
Yes
B Secure Lockdown
No
Perform Non-secure Boot Mode Register Initialization.
Perform Secure Boot Mode Register Initialization.
Execute-in-place?
Execute-in-place?
Yes
B Secure Lockdown
No
Execute-in-place
Quad-SPI or NOR only
Execute from OCM
Load the FSBL into the OCM.
Enable and Lock the Secure State of the System.
Non-secure Setup:
* ROM Code access is disabled.
* JTAG is enabled.
* Branch to User Code in Flash Device.
RSA Enabled?
Yes
C RSA
No
Load the FSBL into the OCM.
RSA Enabled?
Yes
C RSA
No
Execute-in-place
Non-secure Setup:
* ROM Code access is disabled.
* JTAG is enabled.
* Branch to User Code in OCM.
Decrypt the FSBL using PCAP and load into OCM.
Pass HMAC?
No
B Secure Lockdown
Yes
Execute from OCM
Secure Setup:
* ROM Code access is disabled.
* Branch to User Code in OCM.
Execute from OCM
UG585_c6_05_011614
RSA
Secure and Non-secure
C
PPK Verification: eFuse PPK Hash == SHA256(PPK)
SPK Signature Verification: RSA(PPK, SPK Signature) == Padding || SHA256(SPK)
FSBL Signature Verification: RSA(SPK, FSBL Signature) == Padding || SHA256 (FSBL Certificate Image)
PPK Verified?
Yes
SPK Verified?
Yes
FSBL Verified?
Yes
Return
No
No
No
B
B
B
Secure Lockdown
Lockdown
Secure and Non-secure
B
Non-Secure
System State?
Secure
Non-secure Lockdown:
* ROM Code access is disabled.
* JTAG is enabled.
* Issue WFE Instruction.
Secure Lockdown:
* Clean PS, PL.
* Clean all internal RAMs.
* System held in reset.
CPU0 & CPU1 are in WFE state.
Secure Lockdown Exit using POR reset.

FIGURE 7.11: Zynq-7000 – Boot. (Source UG585 [XIL14c])

systems (e.g., operating system, functionality) and the invariable memory [XIL14d].

The resource sharing is a common source of interferences in nowaday's multi-core devices. These interferences lie in the fact that as a general rule, a resource should not be allocated to more than one component of the device at the same time. This assertion is however not fulfilled in the case of multi-core devices where specific resources are shared among their components (e.g., memory, peripherals). The *resource virtualization* technique overcomes the interferences in multi-core devices. This technique manages the non-exclusive resources for guaranteeing the interference freeness among safety and non safety-related functionalities executed on top of the multi-core device. The Zynq device supports the virtualization of the CPUs, the memory and the interrupt controller, thereby enabling the isolation of functionalities executed on the system. In addition, the resource virtualization technique allows the migration of applications from one platform to another, reducing the power consumption and improving the availability of critical applications.

– *CPU virtualization:* The ARM Cortex A9 processors inside the Zynq device are specifically designed to execute complex applications, providing high performance, efficiency, scalability and low-power architectures. These processors support ARM V7-A architectures with full virtualization of memory [ARM11b]. The Cortex-A9 CPU employs speculative execution of instructions which are enabled by the dynamic renaming of physical registers into an available pool of virtual registers. The CPU employs the renaming of the virtual register to eliminate dependencies across registers without jeopardising the correct execution of programs.

  This feature allows code acceleration through HW based program execution optimisation and increases the pipeline utilisation removing data dependencies between adjacent instructions (e.g., pointer arithmetic), simultaneously reducing the interrupt latency of the Zynq device.

– *Memory:* Each Cortex A9 processor in the Zynq device includes a separate L1 cache that supports 4KB, 64KB, 1MB and 16MB virtual memory pages [XIL14c]. The MMU works in close cooperation with the L1 and L2 memories in the process of translating their virtual addresses to physical addresses and controlling access to the external memories. It is compatible with the requirements of the Virtual Memory System Architecture v7 (VMSAv7) with 4 KB, 64 KB, 1 MB, and 16 MB page table entries and 16 access domains. It uses an in-memory table of items called *page table* that contains one *page table entry* per page. The tables are used to map the virtual page numbers to physical page numbers in the main memory. For that end, the MMU contains a single unified Translation Lookaside Buffer (TLB) that is used to fetch instructions and data accesses.

– *Interrupt controller:* The GICv1.0 [ARM08] has been enhanced by a newer version, the GICv2.0 [ARM13]. The new version enables the virtualization extension that allows handling both virtual and physical interrupts.

Another safety technique supported by the Zynq device is the *exclusive access to peripherals.* This technique is implemented by means of the AXI interconnect IP [ARM11a] and the TrustZone security extension [GP14], thus hiding the peripherals and memories from the non secure accesses (see Figure 7.12). Table 7.1 summarizes the exclusive accesses which can be executed by the Cortex A9 cores of the PS, the L2 cache and the PL masters of the ACP "S_ACP", Generic Product (GP) "S_GP" and High Performance (HP) "S_HP" ports. The ACP port of the PL does not support exclusive accesses to the coherent memory and the L2 cache does not support exclusive monitors.

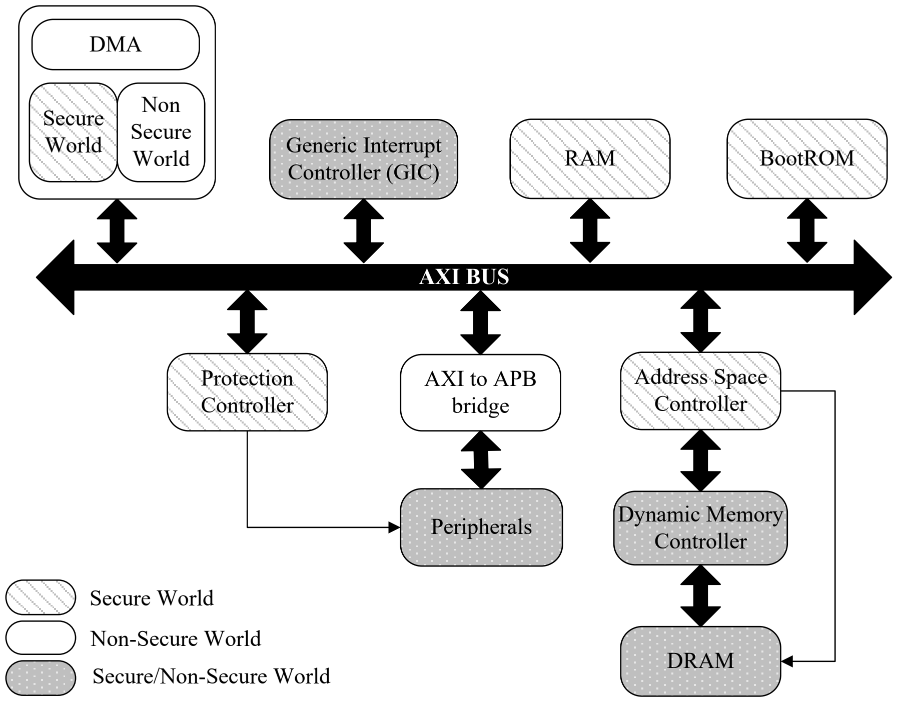

FIGURE 7.12: Zynq-7000 – Exclusive access to peripherals.

Exclusive Operation	**Exclusive Accesses Supported**
ACP to L1 cache	Yes
Two A9 CPUs to L2 cache	No
Two A9 CPUs and ACP to DDR memory	No
ACP and one CPU to DDR	Yes
Two A9 CPUs to DDR	Yes
Masters on GP and HP to DDR: - GP0 and GP1 can do exclusive access with each other only. - HP0 and HP1 can do exclusive access with each other only. - HP2 and HP3 can do exclusive access with each other only.	Yes

TABLE 7.1: Zynq-7000 – Exclusive AXI accesses. (Source [XIL15])

The *resource virtualization* and *exclusive access to peripherals* safety techniques and the techniques described in Annex F of IEC 61508 [IEC10c] for achieving the non-interference between SW components focuses on reaching the *temporal and spatial independences.* In the case of the Zynq device, it is assumed that unacceptable temporal delays may occur (e.g., concurrent cache access, contention by cores). Nevertheless, the boundaries of the *temporal interference* can be estimated and measured by the *system developer* (e.g., estimate and measure the WCET and Best Case Execution Time (BCET)). The delays can be measured and detected, for example, by means of an external HW WDT.

On the other hand, the *spatial independence* can be achieved by means of HW memory protection mechanisms, partitioning mechanisms, operating systems and *bare metal* user applications (see Annex F of IEC 61508-3 [IEC10c]). In the case of the Zynq device, it implements an MMU, a partitioning mechanism (TrustZone security extension) and a DMA to ensure it.

– An MMU effectively performs the management of the virtual memory, preventing SW running in a CPU from accessing other CPUs' address spaces. At the same time, it handles memory protection, cache control, bus arbitration and translates the addresses for mapping the logic memory spaces to physical memory regions. As stated in Table E.2 item 4 of IEC 61508-2 [IEC10b], the isolation of the Zynq device's memory regions can be achieved by means of qualified tools. For instance, the Isolation Design Flow (IDF) qualified tool can be implemented to guarantee the isolation through the use of *fences*.

– *ARM TrustZone security extension* [GP14] helps to create a secure environment to run applications and protect their contents by splitting the processor's cores into two virtual cores, where secure and non-secure functionalities are implemented. This extension, in conjunction with certain peripherals (see Table 7.2) enables a secure system to handle private data and encrypted information without leaking to the non-secure core. The transition of data between the two cores of the Zynq device is switched by a secure monitor that runs in a secure core.

Entity	**TrustZone Security**
ARM A9 Core	Secure and Non Secure
L1 cache controller	Secure
L1 cache	Secure and Non Secure
MMU	Secure
SCU	Secure
L2 cache controller	Secure
SLCR	Secure
Triple Timer-Counter0	Secure
Triple Timer-Counter1	Configurable
Watchdog	Secure
SoC CoreSight Debug	Secure
OCM	Secure and Non Secure
DDR memory	Secure and Non Secure
IOU devices	Configurable (I^2C, GPIO, CAN, Ethernet and UART)

Table 7.2: Zynq-7000 device – TrustZone Security Summary. (Source DS190 [XIL15])

– The DMA provides the spatial separation of the SW tasks of the processor memory by means of DMA data transfers between the memories and peripherals of the Zynq device. If a violation of a memory policy is given, a memory protection exception will be generated.

The *configuration* of resources, peripherals and user applications required for implementing the safety techniques specified above (resource virtualization, exclusive peripheral access, temporal independence and spatial independence) is mostly carried out at the devices' startup using qualified tools. However, it can also be performed at run-time. The *qualified tools* must follow proper methodologies and procedures for the implementation and configuration of the device and they shall be used as defined in the documentation provided by the *tool manufacturer*. In the case of the Zynq device, Xilinx provides the Integrated Synthesis Environment (ISE) and IDF qualified tools. These tools are certified tools for IEC 61508 and ISO 26262 and follow proper methodologies and procedures for configuring the device.

– ISE design tool [HCM15, XIL14c] is used to develop safety-related applications for FPGAs. This qualified tool includes the description of the design methodology and the flow for the FPGA design. It also guides the user toward the application of the design flow and guides verification and testing of the implementation (e.g., timing, temperature, power consumption, violation of design rules and isolation violations).

– IDF [XIL14b, XIL13, Hal14] is a SW methodology and tool that enables the logical and physical isolation among functionalities by using *fences*. The Isolation Verification Tool (IVT) verifies the isolated regions and assesses that a partitioned FPGA design fulfils the stringent security and safety standards. The blocks obtained using the IDF tool can be considered as a structure that separates and decouples the physical blocks (see Table E.2 of IEC 61508-2 [IEC10b]). On the other hand, this tool helps to accelerate the development of secure and safety-related systems and provides techniques such as the modular redundancy, watchdog alarms and segregation by safety levels, enhancing the development of independent functional modules on a single device.

The safety techniques defined in the MSC of chapter 5.3 and the components of the Zynq device require verification and validation activities for detecting failure conditions which may hinder the system. The measures and diagnostic techniques implemented by a multi-core device shall be provided by the *device manufacturer* or defined in the *FMEA / FMECA / FMEDA* analyses. As stated at the beginning of this subsection, a device scenario composed of two CPUs and a soft-core processor in the PL is used as the default device scenario to be analysed. The purpose of this subsection is not to specify at length the diagnostic strategy, system reactions to errors and fault-tolerance techniques which are or must be implemented by the Zynq multi-core device, but the fundamentals of the techniques for both random and systematic faults that are supported by the Zynq device are described in the following paragraphs.

Tables 7.4 and 7.5 summarise the IEC 61508 compliant measures and diagnostic techniques for detecting and controlling random and systematic faults, highlighting if they are supported by the Zynq multi-core device and if not, defining the way in which they can be supported.

▶ **Random Failures**

The diagnosis techniques for random failures summarised below support the requirements of IEC 61508-2 Table A.1 [IEC10b].

– ***Electrical Components***

  Electrical components (e.g., cables, wires, relays) should follow the recommended measures and diagnostic techniques in Table A.2 of IEC 61508-2 [IEC10b]. For instance, the *Failure detection by online monitoring*, *Monitoring of relay contacts*, *Comparator* or *Majority Voter* techniques can be implemented in order to diagnose the electrical components of the device. The Zynq-7000 device does not support those measures and diagnostic techniques *by default.*

– ***Electronic Components***

  Electronic components (e.g., diodes, transistors, integrated circuits, displays, power supply) can be diagnosed by means of recommended technologies in Table A.3 of IEC 61508-2 [IEC10b]. The Zynq device contains a wide variety of electronic components such as the flip-flops and multiplexers, which should be checked following the recommended techniques stated in the same table. For instance, the *Failure detection by on-line monitoring*, *Comparator*, *Majority Voter*, *Monitored redundancy* or *HW with automatic check* techniques can be applicable to the Zynq device. The *Standard test access port and boundary-scan architecture* technique can also be implemented, although this technique depends on whether the device has been designed following the design for test approach.

  On the other hand, the *Test by redundant HW*, *Dynamic principles* and *Analogue signal monitoring* techniques are not supported by the Zynq device. The *Test by redundant HW* technique is not applicable in the first instance because it requires HW redundancy. The same occurs for the *Dynamic principle* technique.

  The *Analogue signal monitoring* technique shall not be implemented due to its low Diagnosis Coverage (DC) level (60%). The main reason for that is that, as stated in the MSC for an IEC 61508 compliant COTS processor, a SIL up to SIL3 is expected and consequently, a minimum DC of 90% is required.

Electronic devices occasionally exhibit erroneous behaviour for no apparent reasons. Soft errors (such as Single Event Upsets (SEUs)) are the common source of these failures. SEUs are non-permanent errors that can lead to transient current pulses, can change the memory data values or cause latch-ups. A latch-up is a type of short circuit that can occur in an integrated circuit, which can disrupt the proper functioning of the device. To prevent latch-ups, the PL side of the Zynq device follows proprietary design rules that specially mitigate that kind of effects [JED01]. Also, to detect and correct soft errors of the PL, the Soft Error Mitigation (SEM) IP is provided by Xilinx. SEM IP cores [XIL14a, HS12, GdL00, CFBC99] perform SEU detection, correction, classification for the configuration memory and perform the emulation of SEUs by injecting errors into the configuration memory.

– ***Processing Units***

The Zynq product family relies on a dual-core ARM Cortex A9 CPU with NEON co-processor. Each processor is a high-performance and low-power core [XIL15] that implements the ARMv7 instruction set architecture. In our particular case, as the processing units (CPUs and soft-core processors) of the Zynq device do not execute the same SW and the lockstep architecture is not supported, the *Comparator* technique recommended by the IEC 61508-2 [IEC10b] is precluded. In the same way, the *majority voter* and *reciprocal comparison by SW* techniques are also excluded. Similarly, the *self-test by SW (limited number of patterns)* technique is not applicable due to its low DC value (60%). In addition, although the *self-test supported by HW* technique is suitable for use, it requires extra HW facilities and therefore, it is not supported by the Zynq device.

The processing units of the Zynq device may be diagnosed using well-known IEC 61508 compliant measures and diagnostic techniques such as the *Self-test by SW: walking bit (one-channel)*, *Code Processing* and *Reciprocal comparison by SW* techniques (see Table A.4 of IEC 61508-2 [IEC10b]). For instance, the *Self-test by SW: walking bit (one channel)* technique (DC of 99%) may be used for diagnosing the processing units by means of additional SW functions responsible for the physical storage (data and address registers of the CPUs) and the instruction decoder. Instead, the *code processing* and *reciprocal comparison by SW* techniques may be implemented as additional diagnosis techniques with a maximum achievable DC of 99%.

– ***Invariable Memory***

Table A.5 of IEC 61508-2 [IEC10b] recommends a set of measures and diagnostic techniques for invariable memories. For instance, the *Word-protection multi-bit redundancy* technique is not supported by the Zynq device because it requires additional HW to monitor the processing units and additional memory to store the parity of each location, thus increasing the cost of the device. In the same way, the *Modified Checksum* technique is inadequate as it has a maximum achievable DC of 60%. On the other hand, although the *signature of one word (8-bit)* technique is adequate to be used, the *Signature of doubleword (16-bit)* technique is preferable because it provides the detection of all one-bit and multi-bit failures with a DC value of 99%. In addition, the *Block replication* technique can be optionally applied. This technique aims to replicate the content of the memory and compares the replicated information to ensure that the memory is error-free.

– ***Variable Memory***

The Zynq device is composed of several variable memories such as the DDR, the L1 and L2 caches and the OCM. The IEC 61508 safety standard recommends a set of measures and diagnosis techniques for variable memory diagnosis (see Table A.6 of IEC 61508-2 [IEC10b]). However, those techniques are not completely supported by the Zynq device because, among others, the communication between the CPUs, the L2 cache, the OCM and the DDR is mainly carried out through the SCU. The SCU manages the coherency between the CPUs, the PL and the cache memories, and controls the communication of the components of the device (such as the CPU and the PL, the peripherals and the memory). Therefore, if the SCU fails, it is unknown if the write/read processes in the memory are concluded properly.

Additionally, in multi-core devices the L2 cache is shared among a wide variety of subsystems and components, which leads to interferences in general. For these reasons, in Chapter 6 the subsystems and components of nowaday's multi-core devices (e.g., shared memory) are analysed, defining at the same time new solutions and diagnosis techniques to overcome issues related to them.

However, in the case of the Zynq device, some variable memories such as the OCM and DDR can be accessed from the PL side, without going through the SCU and L2 cache (see Figure 7.13). Those variable memories can be diagnosed following the diagnostic techniques stated in Table A.6 of IEC 61508-2. For instance, techniques such as *March C-* or *March U* or equivalent can be implemented for RAM diagnosis at boot time. In addition, a cyclic execution of the *RAM test checkerboard* technique would be recommendable to ensure adequate coverage for permanent faults. In the same vein, the *parity bit for RAM* technique would be recommendable to be implemented to improve the detection of the random failures (permanent and transient) that can affect the variable memories. Moreover, the parity bit diagnosis technique can be extended with the EDC code, increasing the detection capability and including the detection of all odd-bit failures, all two bit-failures, some three-bit failures and some multi-bit failures.

As previously mentioned, the SEM core IP is provided by Xilinx for the detection, correction and classification of SEUs that can occur in the PL (e.g., in the Block RAM (BRAM)). SEM IP implements device primitives such as the Internal Configuration Access Port (ICAP) and the ECC to clock and check the read-back CRC as part of the detection function of SEUs. This IP also provides the ECC mechanism for error correction and performs the emulation of the SEUs by injecting errors into the configuration memory. The fault injector provides a means to evaluate and test the mitigation capabilities of variable memories.

– ***I/O Units and Interfaces***

The Zynq device uses a handful of I/O peripherals for on-chip and off-chip communications. The I/Os are organised into four banks of registers (Bank 0, 1, 2 and 3), providing individual (separate) power domains and configuration (e.g., sensitivity level (edge or level sensitivity), interrupts). The PS of the Zynq device accesses the peripherals through the SCU. The SCU manages the coherency of the CPUs, PL and memory and controls, among others, the communication of the PS's CPUs, PL, memory and peripherals. Consequently, if the SCU fails, it is unknown what happens with the peripherals.

Chapter 6 defines additional techniques to diagnose the coherency management unit. However, not all the accesses to peripherals are made through the SCU. In those cases, the recommended measures and diagnostic techniques by the IEC 61508 standard are applicable (see Table A.6 of IEC 61508-2 [IEC10b]). For instance, the *Test pattern* technique is a static (stuck-at) and cross-talk failure detection technique that measures the values of the I/Os and compares the obtained values with the ones expected. The *Multi-channel parallel output* technique makes use of the device's independent outputs for random HW failure detection (stuck-at). The detection is done via external comparators. To ensure that predefined tolerance ranges (e.g., time, addressing, value) are not modified due to external influences (drift failures or transient faults), the use of the *Monitored outputs* diagnosis technique is highly recommendable.

Furthermore, to address permanent and transient failures of the input lines, the *Input comparison/voting (1oo2, 1oo3 or better redundancy)* technique is recommended. A 1oo2 scheme connects the external safety-relevant signal to two independent I/O lines. To reduce the potential impact of common causes of failures, I/O lines belonging to different I/O ports are recommended. Optionally, the *antivalent signal transmission* diagnosis might be implemented by SW (if applicable).

– ***Data Paths***

The interconnect block located within the PS comprises the resources to connect the device's internal components via the ARM Advanced Microcontroller Bus Architecture (AMBA) 3.0 on-chip interconnect bus. The communication bus implements complete interconnect communication capabilities by means of interconnect masters and slaves, the SCU, the Central interconnect, the Master interconnect, the Slave interconnect, the memory interconnect, the OCM interconnect and the L2 cache controller (see Figure 7.13).

The transactions among the masters and slaves must be diagnosed to detect and control transmission errors. To that end, the IEC 61508 safety standard recommends a set of measures and diagnosis techniques to detect and control the failures of the data paths (see Table A.7 of IEC 61508-2 [IEC10b]). For the sake of simplification, techniques such as the *One-bit HW redundancy*, *Multiple-bit HW redundancy* or *Complete HW redundancy* are excluded, among others, due to their implementation complexities and maximum achievable DC values (60%), which are not enough for SIL3.

On the other hand, diagnosis techniques such as the *Inspection using test pattern* and/or the *transmission and information redundancies* might be implemented by SW and supported by the Zynq device. The *Inspection using test pattern* is based on periodical connectivity tests and it must be verified through fault injection mechanisms to ensure that the diagnosis technique itself is free of failures. In addition, the *transmission and information redundancy* techniques are recommended to detect and control both permanent and transient faults.

– ***Power Supply***

The Zynq device provides separate and independent power supplies for the PS and PL. These layers reside on different power areas, and they can be connected to independent power rails. Therefore, in the case that the PL is not used, it can be completely shut down to save power. When the PL is powered-off, signals between the PS and the PL would not be accessible. PS includes an independent power supply for the DDR memory (e.g., DDR2 1V8, DDR3 1v5, Low-Power DDR (LPDDR)2 1V2) and two separate voltage banks for the Multiplexed I/Os (MIOs) (e.g., High-Speed Transceiver Logic (HSTL) 1V8, Logic Voltage Complementary Metal-Oxide Semiconductor (LVCMOS) 1V8, 2V5, 3V3) [XIL15].

In our case, as the Zynq device has a single main power supply that is divided for supplying both the PL and the PS, the diagnosis of the device's power supply can be

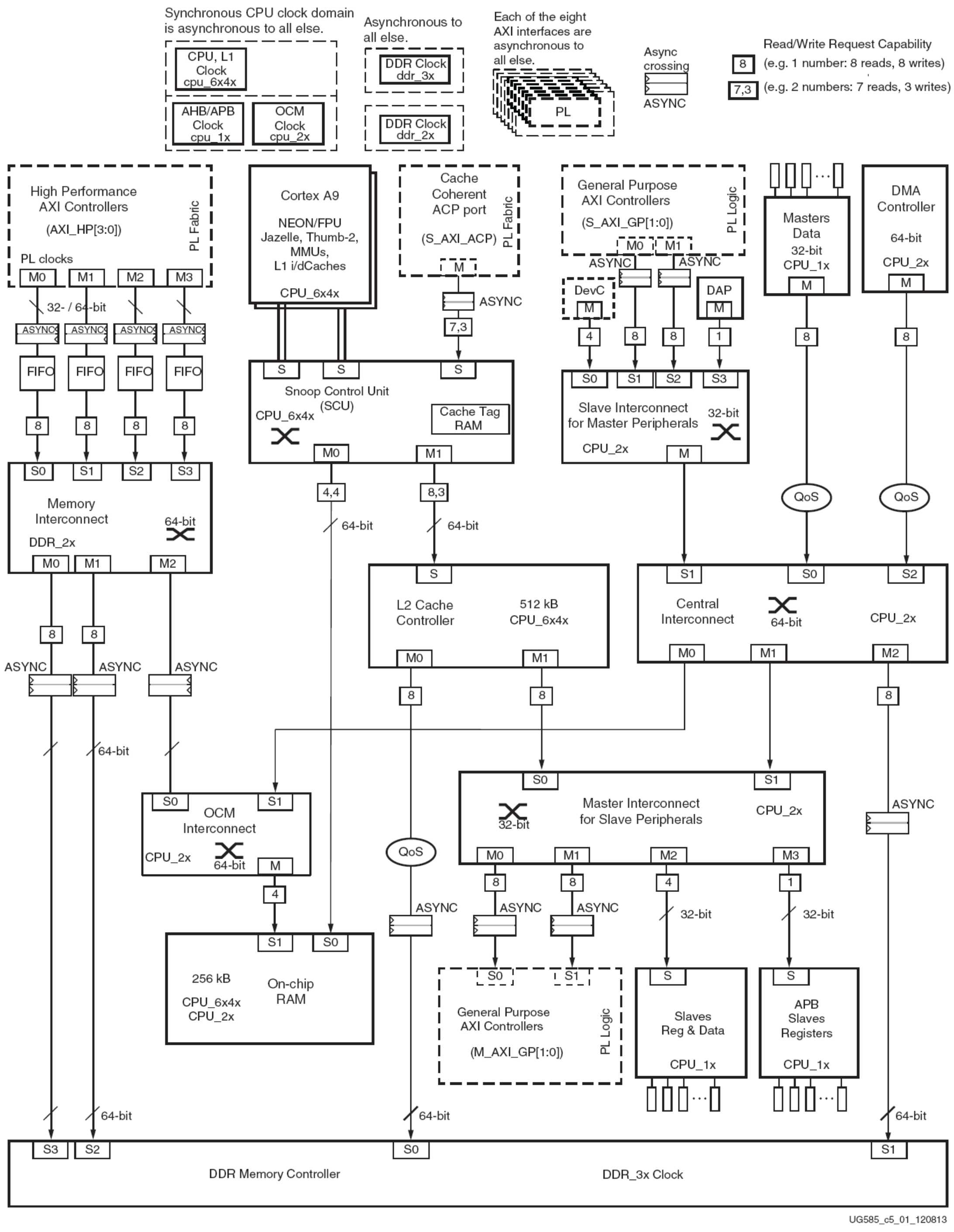
Synchronous CPU clock domain is asynchronous to all else.
CPU, L1 Clock cpu_6x4x
AHB/APB Clock cpu_1x
OCM Clock cpu_2x
Asynchronous to all else.
DDR Clock ddr_3x
DDR Clock ddr_2x
Each of the eight AXI interfaces are asynchronous to all else.
PL
Async crossing
ASYNC
Read/Write Request Capability
(e.g. 1 number: 8 reads, 8 writes)
(e.g. 2 numbers: 7 reads, 3 writes)
High Performance AXI Controllers (AXI_HP[3:0])
PL Fabric
PL clocks
Cortex A9 NEON/FPU Jazelle, Thumb-2, MMUs, L1 i/dCaches CPU_6x4x
Cache Coherent ACP port (S_AXI_ACP)
General Purpose AXI Controllers (S_AXI_GP[1:0])
PL Logic
DevC
DAP
Masters Data 32-bit CPU_1x
DMA Controller 64-bit CPU_2x
32- / 64-bit
FIFO
Snoop Control Unit (SCU) CPU_6x4x
Cache Tag RAM
Slave Interconnect for Master Peripherals 32-bit CPU_2x
Memory Interconnect 64-bit DDR_2x
QoS
L2 Cache Controller 512 kB CPU_6x4x
Central Interconnect 64-bit CPU_2x
OCM Interconnect CPU_2x 64-bit
Master Interconnect for Slave Peripherals 32-bit CPU_2x
256 kB CPU_6x4x CPU_2x On-chip RAM
General Purpose AXI Controllers (M_AXI_GP[1:0])
Slaves Reg & Data CPU_1x
APB Slaves Registers CPU_1x
DDR Memory Controller
DDR_3x Clock
UG585_c5_01_120813

FIGURE 7.13: Zynq-7000 – Interconnect. (Source UG585 [XIL14c])

performed using an external power supply supervisor. In addition, the *over-voltage protection with safety shut-off*, *Voltage control secondary voltages* or/and *Power down with safety shutdown* diagnosis techniques can be implemented (see Table A.9 of IEC 61508-2 [IEC10b]). Ideally, the Zynq device should support separate power supplies, one for the PS and another one for the PL, which shall be monitored by two external power supply supervisors, one per each power supply.

– ***Program Sequence***

A defective program sequence is a sequence where the individual components of a program are processed incorrectly or in a wrong period. IEC 61508 recommends a set of measures and diagnostic techniques for diagnosing defective program failures (see Table A.10 of IEC 61508-2 [IEC10b]). For instance, the *Watchdog with separate time-base without time-window* technique is not applicable in our case due to its maximum achievable DC value of 60%, which is not high enough for our purpose (SIL3). In exchange, the *Watchdog with separate time-base and time-window* technique which has a maximum achievable DC value of 99% might be applicable. Two watchdog timers are required, one for the PL and another one for the PS.

The *temporal and logical monitoring* technique monitors the correct behaviour and sequence of the individual program sections. If the program sequence is executed as expected, the watchdog timer monitor will be re-triggered. Otherwise, it will not be re-triggered. Hence, the *logical monitoring of program sequence* diagnosis technique is recommended for our purpose. Furthermore, the *temporal monitoring with on-line check* diagnosis technique may be required during the processor's boot-time to detect faults in the temporal monitoring.

– ***Clock***

The Zynq device has two separated crystals which can be used to generate the PS's and PL's clocks. The PS consists of three clock domains which are generated from the *PS_CLK* clock. The ARM processing units, the DDR memory and the I/Os (e.g., CAN, Ethernet, UART) make use of those clock domains. On the other hand, the four clock signals that are received by the PL from the PS are derived from the Phase Locked Loops (PLLs) (see Figure 7.14). However, the PL cannot provide any clock to the PS. In the case that any clock of the device is faulty, the techniques recommended by the IEC 61508 standard should be implemented to detect and control clock-related failures (see Table A.11 of IEC 61508-2 [IEC10b]). For instance, the *external watchdog timer* is highly recommended due to its reliability and coverage. Additional diagnosis techniques can be implemented to improve the diagnosis coverage of the device up to SIL3.

► **Systematic Failures**

The diagnosis techniques for systematic failures are summarised and enumerated in Table A.18 of IEC 61508-2 [IEC10b], including the guidance on effectiveness levels (e.g., low, high). Table A.15 of IEC 61508-2 [IEC10b] defines the techniques and measures to control systematic failures caused by the HW design. The *Program sequence monitoring*,

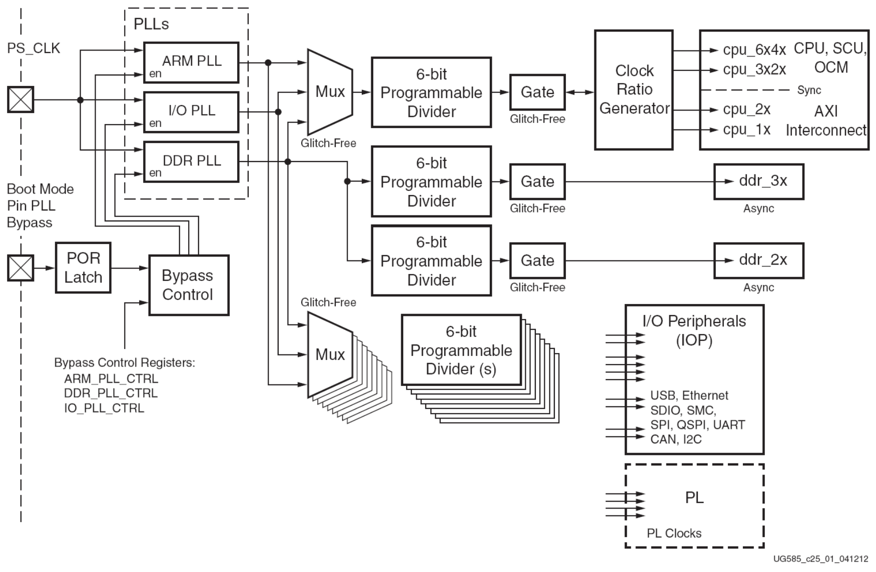

FIGURE 7.14: Zynq-7000 – Clock. (Source UG585 [XIL14c])

Failure detection by on-line monitoring and/or *diverse HW* techniques and measures can be highlighted. At least one of the techniques highlighted in grey (see Table 7.5) must be implemented to guarantee the detection and control of the HW design failures. For instance, the *Standard test access port and boundary-scan architecture* technique can be applied, although it is dependent on whether the HW has been designed in accordance with the design for test approach. The *test by redundant HW* technique might be applicable, although it requires a redundant HW, and the *code protection* technique cannot be implemented because it is based on information and/or time redundancy, which is not provided by the Zynq device.

In order to detect the systematic failures caused by environmental stress or influences (see Table A.16 of IEC 61508-2 [IEC10b]), the Zynq device should implement the *measures against power failures*, *separation of electrical energy lines from information lines*, *increase of electromagnetic immunity* and *measures against physical environment* (e.g., temperature and humidity) techniques. At least one of the light-grey and black-grey techniques related to the environmental failure detection techniques stated in Table 7.5 shall be provided by the Zynq device. For instance, the *Measure to detect breaks and shorts in signal line* and the *diverse HW* techniques are already provided by the Zynq device, although they are dependent on the requirements of IEC 61508-2 Annex E [IEC10b]. On the other hand, the *Modification protection* technique can be used to detect and control systematic operational failures (see Table A.17 of IEC 61508-2 [IEC10b]). At least one of the techniques highlighted in light-grey of Table 7.5 shall be implemented to detect and control systematic operational failures.

An IEC 61508 compliant COTS device must include *system reactions to errors and fault-tolerance techniques* for managing detected failures. As shown in Table 7.3, whenever a processing unit, peripheral or the device itself is reset, a reaction to the error is executed by the device to manage and minimise the impact of the failure. For instance, in the case that the internal temperature of the Zynq device overcomes the maximum allowed temperature value, an interrupt status is sent to the PS, providing the automatic shutdown of the PL. The PL will remain shut down until the temperature alarm goes inactive. Further system reactions to errors can be provided via SW-controlled reactions (e.g., safe state).

Reset name	**Portion of system that is reset**
Power On Reset (POR)	
External System Reset	Entire chip reset
System SW	
CPU 0 Watchdog Timers	CPU(s) reset
CPU 1 Watchdog Timers	
Peripherals	Selected peripheral or CPU reset

Table 7.3: Reset effects (Source [XIL15]).

The safety techniques stated in this subsection, and the components of the Zynq-7000 device must provide functional safety failure analyses to identify the causes and effects of the failures. Hence, according to Table B.6 of IEC 61508-2 [IEC10b], the techniques and components of a safety device shall be commonly backed up by *FMEA / FMECA / FMEDA* analyses. Those analyses aim to evaluate the reliability, safety and integrity of presented techniques and components.

Annex A of this dissertation includes the FMEA / FMECA / FMEDA analyses to assess the random and systematic failures of the safety-related requirements and remarkable components of the Zynq-7000 device. These analyses include the failure modes, causes, effects, related measures and diagnosis techniques and associated DC values (random failures), failure rate values (systematic failures) and mitigation measures (systematic failures) per each failure mode.

	Measures and Diagnostic Techniques	See IEC 61508-7	Maximum achievable DC	Supported by the Zynq-7000 device	Arguments
Electrical Components	Failure detection by on-line monitoring	A1.1	Low (Low demand mode). Medium (High demand or continuous mode).	No	It can be supported.
	Monitoring of relay contacts	A1.2	High (99%).	No	–
	Comparator	A1.3	High (99%).	No	It can be supported.
	Majority voter	A1.4	High (99%).	No	It can be supported.
Electronic Components	Failure detection by on-line monitoring	A1.1	Low (Low demand mode) Medium (High demand or continuous mode)	No	It can be supported.
	Comparator	A1.3	High (99%).	No	It can be supported.
	Majority voter	A1.4	High (99%).	No	It can be supported.
	Test by redundant HW	A2.1	Medium (90%).	No	–
	Dynamic principles	A2.2	Medium (90%).	No	It can be supported.
	Standard test access port and boundary-scan architecture	A2.3	High (99%).	No	It can be supported if the device's design follows the design for test.
	Monitored redundancy	A2.5	High (99%).	No	It can be supported.
	HW with automatic check	A2.6	High (99%).	No	It can be supported.
	Analogue signal monitoring	A2.7	Low (60%).	No	–
Processing Units	Comparator	A1.3	High (99%)	No	–
	Majority voter	A1.4	High (99%)	No	–
	Self-test by SW (limited number of patterns)	A3.1	Low (60%)	No	–
	Self-test by SW: Walking bit	A3.2	Medium (90%)	No	It can be supported by SW implementation.
	Self-test supported by HW	A3.3	Medium (90%)	No	–
	Coded processing	A3.4	High (99%)	No	It can be supported by SW implementation.
	Reciprocal comparison by SW	A3.5	High (99%).	No	It can be supported by SW implementation.
Invariable Memory	Word-protection multi bit redundancy	A4.1	Medium (90%)	No	–
	Modified checksum	A4.2	Low (60%)	Yes	It is implemented at boot time to verify the boot header.
	Signature of one word (8-bit)	A4.3	Medium (90%)	No	–
	Signature of a double word (16-bit)	A4.4	High (99%)	No	It is provided by SW implementation.

	Block replication	A4.5	High (99%)	No	It is provided by SW implementation.
Variable Memory	RAM Test "Checkerboard" or "March"	A5.1	Low (60%)	No	It is provided by SW implementation.
	RAM Test "Walkpath"	A5.2	Medium (90%)	No	–
	RAM test "Galpat" or "Transparent Galpat"	A5.3	High (99%)	No	–
	RAM test "Abraham"	A5.4	High (99%)	No	–
	Parity-bit for RAM	A5.5	Low (60%)	Yes	It is supported by both L1 and L2 caches and OCM.
	RAM monitoring with a modified Hamming code or detection for data failure with EDC	A5.6	Medium (90%)	Yes	**ECC**: It is supported by the DDR memory controller, QSPI and NAND flash controller. **CRC**: It is supported by the SD/SDIO host controller, the PL and the OCM.
	Double RAM with HW or SW comparison and read-/write test	A5.7	High (99%)	No	–
I/O units	Failure detection on-line monitoring	A1.1	Low (Low demand mode) Medium (High demand or continuous mode)	No	It is supported by SW implementation.
	Test pattern	A6.1	High (99%)	No	It is supported by SW implementation.
	Code protection	A6.2	High (99%)	No	–
	Multi-channel parallel output	A6.3	High (99%)	No	It is supported by SW implementation.
	Monitored outputs	A6.4	High (99%)	No	It is supported by SW implementation.
	Input comparison/voting (1oo2, 2oo3 or better redundancy)	A6.5	High (99%)	No	It is supported by SW implementation.
	Antivalent signal transmission	A11.4	High (99%)	No	It is supported by SW implementation.
Data Paths	One-bit HW redundancy	A7.1	Low (60%)	No	–
	Multi-bit HW redundancy	A7.2	Medium (90%)	No	–
	Complete HW redundancy	A7.3	High (99%)	No	–
	Inspection using test patterns	A7.4	High (99%)	No	It is supported by SW implementation.
	Transmission redundancy	A7.5	High (99%)	No	It is supported by SW implementation.
	Information redundancy	A7.6	High (99%)	Yes	It is supported when the multi-core device is reset.
Power Supply	Over-voltage protection with safety shut-off	A8.1	Low (60%)	No	An external power supply supervisor is required.
	Voltage control secondary voltages	A8.2	High (99%)	No	–
	Power down with safety shutdown	A8.3	High (99%)	No	–
	Watchdog with separate time-base without time-window	A9.1	Low (60%)	No	–

Program Sequence Monitoring	Watchdog with separate time-base and time-window	A9.2	Medium (90%)	No	The device provides three integrated SW watchdog timers (two CPU watchdog timers and a system watchdog timer). These watchdogs reset the CPUx or the entire system when they are enabled and the timer expires. One or more external watchdogs are required.
	Logical monitoring of program sequence	A9.3	Medium (90%)	No	–
	Temporal and logical monitoring	A9.4	High (99%)	No	It is supported by an external watchdog.
	Temporal monitoring with on-line check	A9.5	Medium (90%)	No	It is supported by an external watchdog.
Clock	Watchdog with separate time-base without time-window	A9.1	Low (60%)	No	–
	Watchdog with separate time-base and time-window	A9.2	Medium (90%)	No	The device provides three integrated SW watchdog timers (two CPU watchdog timers and a system watchdog timer). These watchdogs reset the CPUx or the entire system when they are enabled and the timer expires. One or more external watchdogs are required.
	Logical monitoring of program sequence	A9.3	High (99%)	No	–
	Temporal and logical monitoring	A9.4	High (99%)	No	It is supported by an external watchdog.
	Temporal monitoring with on-line check	A9.5	Medium (90%)	No	It is supported by an external watchdog.

Table 7.4: IEC 61508 compliant measures and diagnostic techniques for random failures.[1] (Source Tables A.2 to A.14 of IEC 61508-2 [IEC10b])

[1] ***Light-Gray:*** Optional measures and diagnostic techniques. ***Black-Gray:*** Recommended measures and diagnostic techniques.

	Measures and Diagnostic Techniques	See IEC 61508-7	SIL1	SIL2	SIL3	SIL4	Supported by the Zynq-7000 device	Argumentation
Failures caused by HW design	Program sequence monitoring	A9	HR Low	HR Low	HR Medium	HR High	No	–
	Failure detection by on-line monitoring	A1.1	R Low	R Low	R Medium	R High	No	–
	Test by redundant HW	A2.1	R Low	R Low	R Medium	R High	No	–
	Standard test access port and boundary-scan architecture	A2.3	R Low	R Low	R Medium	R High	No	Might be applied to the Zynq-7000 device if it has been designed following the design for test approach.
	Code protection	A6.2	R Low	R Low	R Medium	R High	No	–
	Diverse HW	B1.4	– Low	– Low	R Medium	R High	No	–
Failures caused by environmental stress or influences	Measure against voltage breakdown, variations and over voltage	A8	M Low	M Medium	M Medium	M High	No	The Zynq-7000 device supports an internal voltage parameter supervisor. There are required two external power supply supervisors.
	Separation of electrical energy lines from information lines	A11.1	M	M	M	M	Yes	–
	Increase of interference immunity	A11.3	M Low	M Low	M Medium	M High	No	SEM core IP can be used for protecting the device against SEUs.
	Measures against physical environment (temperature, humidity, vibration and dust)	A14	M Low	M High	M High	M High	No	The Zynq-7000 device supports an internal temperature monitor with alarm.
	Program sequence monitoring	A9	HR Low	HR Low	HR Medium	M High	No	External watchdog timers are required, one for the PS and another for the PL.
	Measures against temperature increase	A10	HR Low	HR Low	HR Medium	M High	Yes	The Zynq-7000 device provides an internal temperature alarm that reaches a safe state until the temperature reaches a normal temperature value range.
	Spatial separation of multiple lines	A11.2	HR Low	HR Low	HR Medium	M High	No	–
	Idle current principle	A1.5	R	R	R	R	No	It could be supported if an external relay is used.
	Measure to detect breaks and shorts in signal lines	–	R	R	R	R	No	It could be implemented by additional HW and/or SW.
	Failure detection by on-line monitoring	A1.1	R Low	R Low	R Medium	R High	No	It could be implemented by additional HW.
	Test by redundant HW	A2.1	R Low	R Low	R Medium	R High	No	It could be implemented.
	Code protection	A6.2	R Low	R Low	R Medium	R High	No	See Tables A.2 and C.2 of IEC 61508-3.
	Antivalent signal transmission	A11.4	R Low	R Low	R Medium	R High	No	–
	Diverse HW	B1.4	– Low	– Low	– Medium	– High	No	It could be supported if Annex E of IEC 61508-2 is accomplished.

	SW architecture	IEC 61508-3 Subsection 7.4.3	See tables A.2 and C.2 of IEC 61508-3				No	–
Failures caused by operational failures	Modification protection	B4.8	M Low	M Medium	M High	M High	No	It should be implemented by the Zynq-7000 device.
	Failure detection by online monitoring	A1.1	R Low	R Low	R Medium	R High	No	It can be implemented in the Zynq-7000 device.
	Input acknowledgement	B4.9	R Low	R Low	R Medium	R High	No	–
	Failure assertion programming	C3.3	See tables A.2 and C.2 of IEC 61508-3				No	–

TABLE 7.5: IEC 61508 compliant measures and diagnostic techniques for systematic failures.[2] (Source Tables A.15 to A.18 of IEC 61508-2 [IEC10b])

[2]Per each set of measures and diagnostic techniques, at least one of the techniques in the gray shaded group and at least of the techniques of the black gray shaded group is required.

7.2.4 Mixed-Criticality Networks – Linking Analysis

This subsection provides a brief representative example of two *linking analyses* between the MSC presented in Section 5.4, a *black channel* network and a *white channel* network. The *black channel* approach is based on an EtherCAT network, where on top of it, a SCL is implemented. In relation to the *white channel* approach, it is based on a TTE network which has been developed following the development process defined in the IEC 61508 safety standard.

7.2.4.1 EtherCAT – Linking analysis

EtherCAT is a high-performance, low-cost, easy to use industrial Ethernet technology with a flexible topology. It was introduced in 2003 and has been an international standard since 2007. Its key functional principle lies in how its nodes process Ethernet frames. Each node reads its corresponding data and writes the data back to the frame, while the frame is moving downstream. This leads to improved bandwidth utilisation (one frame per cycle is often sufficient for communication) while also eliminating the need for switches or hubs. The host microprocessors in the slave devices are not involved in the processing of the EtherCAT data transfer, thus achieving short cycle times and high performance.

Custom or commercial SCLs can be implemented on top of an EtherCAT network to provide safety-related functionality to the *black channel* network. The SCLs shall be compliant with a safety standard (e.g., IEC 61508, IEC 61784). For instance, Functional Safety over EtherCAT (FSoE) [Eth11] is a commercial certified protocol that fulfils the requirements for SIL3 systems, and it is suitable for both centralised and decentralised control systems. FSoE defines a safety single-channel communication layer (SCL) for transferring messages of different criticality. The transport medium (EtherCAT) is referred to as a *black channel* network, which is not included in safety considerations. It is assumed that the SCL (custom or commercial) on top of EtherCAT network is applicable to fail-safe systems and that it follows an IEC 61508 compliant development process with a residual error rate probability. For example, the FSoE is developed with a residual error rate lower than $10^{-9}/h$.

According to the IEC 615784-3, a safety network shall operate according to the *idle current principle*, thus guaranteeing that the time intervals of two consecutive message receptions do not exceed a predefined time window. To that end, the SCL diagnoses the EtherCAT *black channel* network by separate watchdog timers to detect delays on the communication paths, detecting whether an EtherCAT frame arrives within the specific time and when not. The parametrization and documentation of diagnosis mechanisms must be provided by the system integrator.

The SCL includes the measures defined in Section 5.4 for controlling the common errors that can jeopardise the communication among safety and non safety-related components of the network. The communication errors to consider by a safety network are presented below. For more information see Section 5.3 of IEC 61784-3 [IEC10g]).

- ***Corruption*** of the messages due to errors on the transmission lines or interferences between the messages
- ***Unintended repetitions*** of the messages due to an error, fault or interference.
- ***Incorrect sequence*** of messages due to an error, fault or interference.
- ***Loss*** of the reception of a message or missing acknowledgement due to an error, fault or interference.
- ***Unacceptable delay*** of the messages beyond their permitted arrival time window.
- ***Insertion*** of messages in the communication medium with an unexpected or unknown source.
- ***Masquerade*** errors where a message is inserted from an apparently valid safety source into a safety-related source, but the source is not a safety source; it masks its source.
- ***Addressing*** errors where a safety-related message is sent to a wrong safety receiver.

Table 7.6 shows the measures and diagnostic techniques implemented by the SCL to cover the *communication errors* presented in the previous paragraph. In the case that a communication error is detected, the SCL on top of an EtherCAT network provides safe state mechanisms to react to the error. For instance, the erroneous communication system is reset.

The *reaction time* of the SCL depends on the arriving time of the order from the master node (calculated) plus the order's propagation time required to reach the upper communication layer. The processing time of the application on top of the slave depends on the application itself. Therefore, it is out of the scope of this paper. In the case of the SCL on top of a *black channel* network, its deterministic *response time* is determined, among other things, by the network topology.

The custom or commercial SCL provides *non interference of non safety-related communications*, ensuring temporal and spatial independences. These independences are provided

		Measures			
		Sequence Number	Watchdog Timer	Connection ID	CRC calculation
Errors	Unintended repetition	X			X
	Loss	X	X		X
	Insertion	X			X
	Incorrect sequence	X			X
	Corruption				X
	Unacceptable delay		X		
	Masquerade		X		X
	Repeating memory errors in switches	X			X
	Incorrect forwarding between segments			X	

Table 7.6: SCL – Communication errors vs Safety measures. (Source [Eth11])

using detection and correction techniques explained before. In the case of temporal independence, Time Division Multiple Access (TDMA) based protocols are frequently used in industrial networks. Although the average performance is not as good as for randomly accessed networks, it is one way to assure the temporal independence. Data networks such as Avionics Full-Duplex Switched Ethernet (AFDX) with rate constraints can also be implemented for the same purpose. The SCL supports the time multiplexed concept, where the access to the medium is given via a token. The token is represented by the communication frame where each network slave writes and reads the transmitted information. On the other hand, spatial independence is supported by the SCL through isolation of the application from the HW by means of a HW network controller. The HW only passes the information to the application layer when the received information is marked as correct. The spatial independence between *black channel* slaves is inherent to the network definition and the communication generation. The HW network controller provides random and systematic failure diagnosis in order to detect failures during design time and execution time (see IEC 61508-2 and IEC 61508-3 [IEC10b, IEC10c]).

As defined in Section 5.4, the components of a safety network shall be tested together to obtain an exhaustive diagnosis scenario. If diagnosis in conjunction is not applicable, the components of the network shall be diagnosed separately. In that case, the SCL on top of a *black channel* network shall be tested using the *extensive testing* approach where the worst case scenarios shall be considered. The worst case scenarios shall be defined by the network manufacturer or the FSCP.

Conformance testing is mandatory to verify that the device complies with the communication requirements of IEC 61508 [IEC10b] and IEC 61784-3 [IEC10g]. The tests are required to diagnose the communication devices that shall work together with other communicating devices provided by other manufacturers, where an integration test is not possible. The conformance testing is also required for the *black channel* approach and the SCL. For instance, the EtherCAT conformance test tools [Eth12].

It is assumed that the SCL is developed in accordance with a safety standard. Therefore, the *configuration* of the SCL on top of the EtherCAT network is performed by means of qualified tools to avoid configuration errors that could lead to errors of the communication network.

The EtherCAT network is used as a *black channel* network, therefore, no *FMEA / FMECA / FMEDA* analyses must be provided. Instead, the SCL should provide FMEA / FMECA / FMEDA analyses to assess its random and systematic failures and evince that it is safe enough for its intended use.

7.2.4.2 TTEthernet – Linking analysis

TTE is an extension of the traditional Ethernet standard, with additional services that guarantee reliable and deterministic delivery of time-critical messages. TTE offers the TT, RC and BE classes of traffic [SD11]. The TT traffic class serves for deterministic communication, RC traffic is subject to traffic policing by TTE switches and BE traffic does not support timing guarantees and priorities. A TTE network is a white channel network composed of HW and SW. It has been developed according to the IEC 61508 safety standard, and it applies to fail-safe and fail-operational systems. This network can be seen as an implementation of the TT Architecture (TTA). The prime concept of TTA is the common perception of the time in the devices that form the distributed system.

According to the IEC 61784-3, a network in safety-related applications shall operate according to the *idle current principle.* This principle claims that a safety network must guarantee that the time intervals of two consecutive message receptions do not exceed a predefined time value. To that end, redundancy mechanisms (such as the information or transmission redundancy) or the majority voting mechanism may be implemented. For instance, an end-system can be configured such that it delivers all redundant messages to the host CPU. The host CPU will execute a SW layer, and it will perform different voting mechanisms.

In relation to *communication errors*, a TTE network considers the communication errors stated in Section 5.4 of IEC 61784-3 [IEC10g], Table A.1 of IEC 61508-2 Annex A [IEC10b] and a set of well-known failure modes, including the fail-silence, fail-omission, fail-inconsistent and fail-arbitrary failure modes [Int11]. The communication errors to consider are defined in the previous linking analysis for EtherCAT black channel network.

The global time-stamping service simplifies the process of reconstructing a distributed event chain. TTE supports periodic diagnostic information as well as inner self-checking mechanisms. The period of the status and diagnosis data can be scheduled with the other TT messages or the ET traffic (RC or BE traffic). TTE can be queried for state and diagnostic information. For instance, it provides the diagnostic parameters associated with the TTE network, the diagnostic values of the protocol state (e.g., the state of the protocol state machine of the TTE network) and the diagnostic information about the clock synchronisation mechanism (e.g., the largest correction value ever applied by the clock synchronisation), the membership (e.g., number of integration frames received in the most recent acceptance window) and the current sync priority of the Synchronization Master (SM).

On the other hand, a TTE communication network provides *error detection* and *reaction* mechanisms. It allows the development of critical systems parts according to fail-safe or fail-operational application requirements. In fail-safe systems, it must detect and react (e.g., reach a safe state) to errors of the network. For instance, the TTE networks support fault hypotheses with a single-failure, dual-failure and arbitrary disturbances (transient disturbances). An even higher level of fault-tolerance can be achieved by a SoS approach that supports system-level fault-tolerance. The overall failure or power down of a TTE based system can be mitigated by another redundant TTE based system.

For safety-critical applications that require *deterministic* communication, the TT frames are used. TTE components can send TT frames, which are received by any Ethernet component. If necessary, those frames can implement higher priorities than the RC and BE traffic classes. Triggered frames are sent and routed at defined points in time, which gives to TTE its highly deterministic timing properties and minimal jitter. The path and timing of TT frames are determined by the schedules configured in the senders and the switches.

Due to the statically configured schedule, the sending instants of messages are triggered by a globally synchronised clock and each TT frame is received with a defined latency and a minimal (bounded) jitter. There is no external (application) control over the protocol progression. The global time base and common knowledge about the action times reside inside the protocol controllers. The application CPU can not modify them. This concept guarantees predictable *reaction and response times*. In relation to the *non interference of/from safety-related communication*, TTE provides temporal independence by means of several traffic classes in parallel: TT, RC and BE. TT traffic dispatches messages according to a predefined communication schedule. Since the transmission delay of any frame can be bounded, TT operations constitute a TDMA communication network that guarantees temporal independence. Additionally, TTE networking components provide spatial independence of messages in switches and end systems at all times. In end-systems, the network controller is implemented in the HW. Any fault in the communication system is thus spatially guarded against the application system.

When a TTE device supports multiple traffic classes with different time criticality, the designer of the respective TTE device should ensure that appropriate memory protection mechanisms are in place. In particular, it should ensure that the TTE devices independently treats the frames belonging to different traffic classes. For instance, TTE devices can use physically separated memories or statically reserved memory space in a single physical memory. Parts of TTE networks are tested using the *extensive testing* approach, including the worst case scenarios (e.g., delays, queue length and shuffle delays). This method defines that the components of a safety network shall be tested together to obtain an exhaustive diagnosis scenario. If it is not applicable, it determines that the parts of the safety network could be diagnosed separately. In relation to the *configuration* of a TTE network, its overall configuration work-flow relies on a set of tools, which are shown in Figure 7.15 and described in the next paragraphs [TTT15, Cha13].

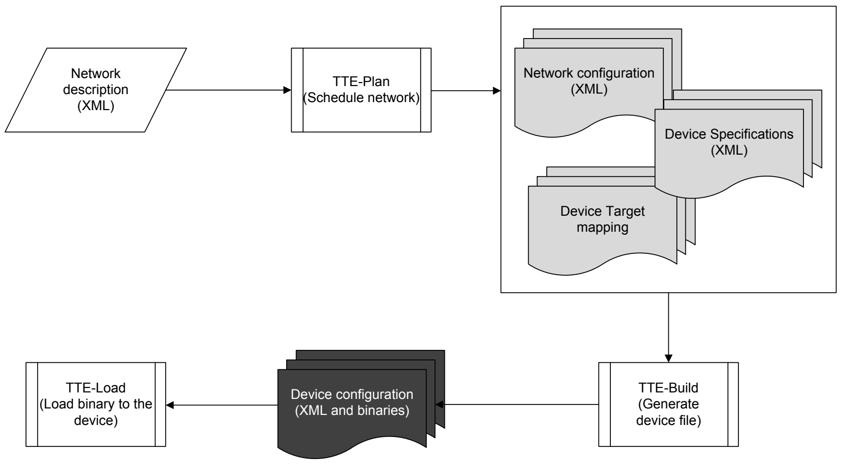
Network description (XML)
TTE-Plan (Schedule network)
Network configuration (XML)
Device Specifications (XML)
Device Target mapping
TTE-Build (Generate device file)
Device configuration (XML and binaries)
TTE-Load (Load binary to the device)

FIGURE 7.15: TTE configuration toolset. (Source [TTT15])

– ***TTE-Plan*** [TTT15] is a scheduling and network analysis tool that performs checking and validation of the Network Description (ND). It analyses the schedulability of Critical Traffic (CT) and outputs a schedule, provided that the ND is valid, in the form of corresponding network configuration files. Additionally, a checking report is output by TTE-Plan.

  The resulting TTE Network Configuration database (NCDB) consist of a network configuration file (e.g., devices and ports mapping, synchronisation parameters), device specification file (e.g., device type information, traffic routeing information) and a device target mapping that maps the devices to HW.

– ***TTE-Build*** [TTT15] is used to handle the TTE NCDB with a Graphical User Interface (GUI). It allows navigating through the NCDB in an Eclipse GUI, displaying and editing individual Virtual Links (VLs) going through the network, manually changing the timing, creating new TT/RC - VLs and adjusting timing parameters of TT/RC VLs.

– The ARINC 615A ***Data Loader*** [TTT15] comes with the following key features: 615A-v2&3 Data Loading, command line operation, A615 FIND [TTT15], TFTP C API [TTT15] with support for client and server services, ARINC-615A C API that supports high-level data load operations and support for multiple, simultaneous load operations.

– The diagnosis tool ***TTE-View*** is [TTT15] based on a standard PC and the open source *WireShark* network protocol analyser. It allows capturing traffic live for offline analysis, to monitor multiple TTE switches (channels or hops), to dissect frame content according to the TTE protocol specification and to inspect TTE traffic and protocols other than TTE.

FMEA / FMECA / FMEDA analyses are required by *white channel* networks in order to evidence that the relevant failure modes, causes and effects, diagnosis techniques and reaction techniques are taken into consideration. The TTE network manufacturer shall provide these analyses.

7.3 System of System, Product Line and Modularity

A SoS is the integration of independently useful systems into a larger system that delivers unique capabilities [PHZ+14]. On the other hand, a product line is set of systems that share and manage a common set of features satisfying the needs of particular market areas. Those systems may be developed from a common set of reusable core. The concepts from SoS engineering can be helpful in Product Line Engineering. As an example, we can consider the scenario shown in Figure 7.16 where a wind park system that kept at different levels of detail (granularity) is presented. The granularity levels state the abstraction levels of a wind turbine system. I.e., SoS, system, subsystem and component. Those levels of abstraction should follow a proper development process to be considered compliant items and be reused. For instance, the development process presented in Section 4.1 for developing an IEC 61508 compliant items may be applied to that end.

Vertically (see Figure 7.16), we can distinguish a SoS (i.e., a wind park) that is composed of different systems (e.g., wind turbines, wind park control centre systems). These systems

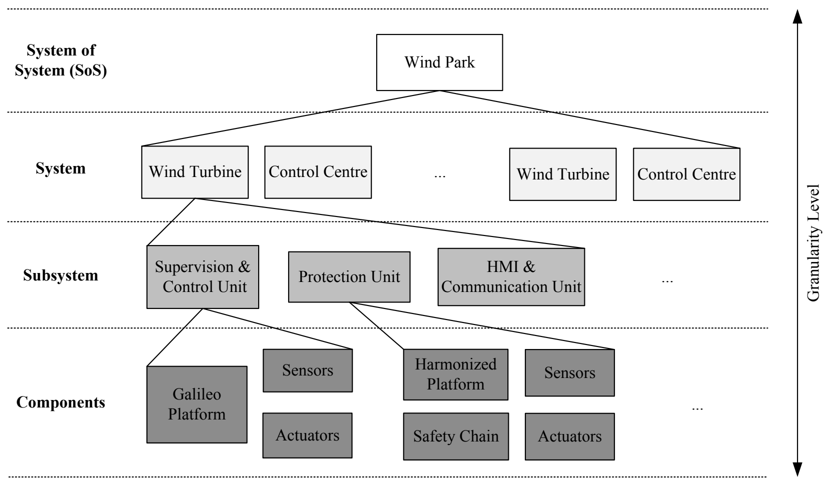

FIGURE 7.16: Dimensions of abstraction: Development Viewpoint and Granularity Level.

consist of subsystems which can be in turn systems on their right. This hierarchical structure can occur recursively at many levels of abstraction. For instance, a wind turbine system can be composed of HMI and communication, supervision and control, and protection subsystems. In this case study, the supervision and the control of the wind turbines are provided by a commercial real-time platform which monitors the external sensors and acts on the external actuators (see Figure 7.1). Instead, the protection subsystem consists of a heterogeneous platform component that executes the safety-related functionalities, assuring that the design limits of the wind turbine system are not exceeded and controlling the safety chain. These components may be connected to each other, within the subsystems and systems. For instance, the protection subsystem may be composed of a COTS multi-core device, a NoC communication network, a hypervisor and a wide variety of SW with different safety levels (e.g., SIL1 to 4 according to the IEC 61508 safety standard).

When dealing with SoS engineering, we can consider each system to be potentially a product of a product line. The motivation to that (in a SoS context) can come from the following causes. First, in many cases, a supplier of systems (e.g., a manufacturer of the protection system of a wind turbine) may have families of similar systems. Product line techniques promise considerable benefits in handling such families of products in a systematic fashion. Therefore, product lines can be seen as a mechanism to develop components, subsystems and systems in a SoS approach. Second, from the perspective of an end user of systems, which manages a SoS, it can be beneficial to handle groups of systems together rather than addressing them independently. For instance, an end user can use modelling to represent the variability of the protection systems which are executed in a wind park.

Variability is the quality, state or degree of a system to be changeable. For example, the product samples of a product line can vary depending on the safety standard (i.e., IEC 61508, ISO 26262, IEC 50126) and the level of criticality (i.e., SIL1 to 4 according to the IEC 61508, ASIL A to D according to ISO 26262). Variability and safety-related arguments of those systems can be represented using the GSN, CAE and SACM safety case notation languages (see Section 3.2). For instance, the MSCs for IEC 61508 compliant components, including a hypervisor, safety partition, COTS multi-core device and mixed-criticality network are defined in Chapter 5 using the CAE notation. These MSCs define the basic safety requirements that shall be met by those components to be IEC 61508 compliant items.

Furthermore, this dissertation presents the linking analyses of a commercial/custom hypervisor, safety-partition, COTS multi-core device and mixed-criticality network which are generated taken the generic MSCs as a basis. These MSCs and the linking analyses defined in this dissertation have been ported to the GSN language due to the limitations of the CAE notation language for representing complex safety cases and the facilities and extensions provided by GSN (see Subsection 3.2.1). For instance, the modularity extension supports the development of compositional safety cases, reducing the effort to the reassessment of a safety case after a modification of the system.

The MSCs and linking analyses presented in this thesis provide the safety-related argumentation for the design branch of an IEC 61508 compliant product sample development process. These safety arguments can be taken as the basis to extend the argumentation scheme up to a realistic wind turbine product line including further components and considering different levels of criticality and safety-related standards.

Based on the representation scheme exposed in the previous paragraphs and the use case presented at the beginning of this chapter, the following four abstraction layers are identified (see Figure 7.17).

a) The *generic safety-related arguments for a safety standard* (e.g.,IEC 61508) compliant components. E.g., a hypervisor, a safety partition, a COTS multi-core device.

b) The *safety-related arguments for application independent safety standard compliant commercial or custom components.* E.g., a Zynq-7000 multi-core device, the XtratuM hypervisor.

c) The *safety arguments for a specific product sample.* I.e., a wind turbine product sample based on the DREAMS architecture style.

d) The *safety-related arguments for a product line* which should be common to all possible product samples of a product line. I.e., a wind turbine product line.

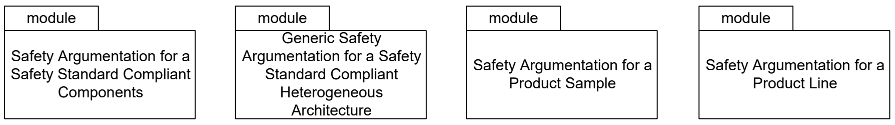
module
Safety Argumentation for a Safety Standard Compliant Components
module
Generic Safety Argumentation for a Safety Standard Compliant Heterogeneous Architecture
module
Safety Argumentation for a Product Sample
module
Safety Argumentation for a Product Line

FIGURE 7.17: Product line development abstraction layers.

GSN also provides the contract element for specifying the interrelationships between the safety arguments of each abstraction layer, linking the goals to be supported with the supporting goals and minimising the impact of changes between interrelated layers. In this particular case, the contract element details how a commercial/custom component fulfils the safety-related requirements defined in the generic safety argumentation for heterogeneous safety architectures (see Figure 7.18).

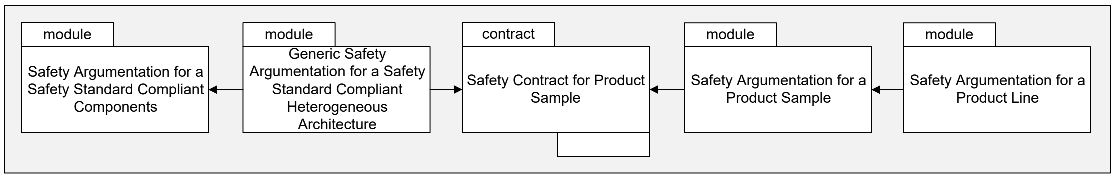
module
Safety Argumentation for a Safety Standard Compliant Components
module
Generic Safety Argumentation for a Safety Standard Compliant Heterogeneous Architecture
contract
Safety Contract for Product Sample
module
Safety Argumentation for a Product Sample
module
Safety Argumentation for a Product Line

FIGURE 7.18: Linkage between the abstraction layers and the contract.

A product line representation can be automated for achieving an optimum product configuration, depending on the safety requirements for a product sample. Automation can be accomplished using tools, which also automate the generation of the safety reports. In addition, this kind of tools can complete the information of the product sample safety contract, linking the requirements of the product sample to the safety-related requirements of the optimum application independent components (see Figure 7.19). For instance, the Design Space Exploration (DSE) toolset [For16] resolves variability models, selecting alternative candidate deployments of the logical components on the HW target and automatically assembles the safety argumentation models per each product sample. The outputs of DSE toolset include a partial argumentation model that is mapped to a set of evidence documents which are semi-automatically generated.

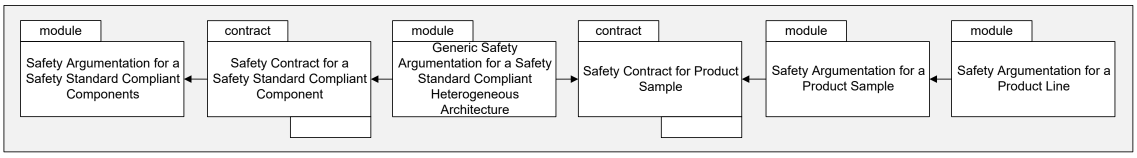
module
Safety Argumentation for a Safety Standard Compliant Components
contract
Safety Contract for a Safety Standard Compliant Component
module
Generic Safety Argumentation for a Safety Standard Compliant Heterogeneous Architecture
contract
Safety Contract for Product Sample
module
Safety Argumentation for a Product Sample
module
Safety Argumentation for a Product Line

FIGURE 7.19: Automatizing the deployment of a product line.

Continuing with representation style presented in the previous paragraphs, an extended version based on a wind turbine product line with a basis on the DREAMS architecture style is generated. This product line architecture exposes the safety arguments related to the wind turbine product line, identifying the way in which a wind-turbine product sample meets those requirements and presenting the linkage between the product sample and the commercial or custom components that make it up. Figure 7.20 represents the

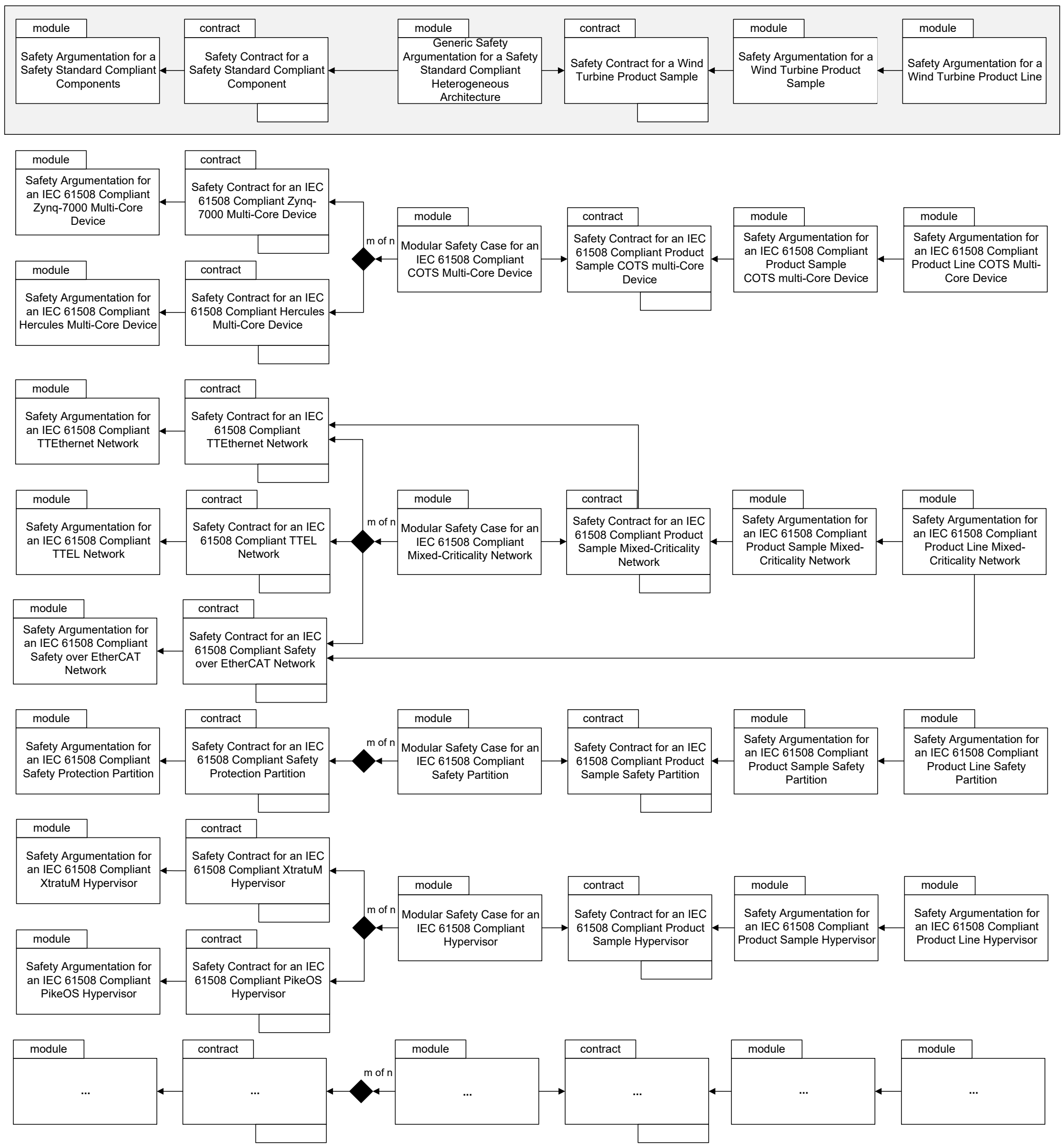
module
Safety Argumentation for a Safety Standard Compliant Components
contract
Safety Contract for a Safety Standard Compliant Component
module
Generic Safety Argumentation for a Safety Standard Compliant Heterogeneous Architecture
contract
Safety Contract for a Wind Turbine Product Sample
module
Safety Argumentation for a Wind Turbine Product Sample
module
Safety Argumentation for a Wind Turbine Product Line
Safety Argumentation for an IEC 61508 Compliant Zynq-7000 Multi-Core Device
Safety Contract for an IEC 61508 Compliant Zynq-7000 Multi-Core Device
Safety Argumentation for an IEC 61508 Compliant Hercules Multi-Core Device
Safety Contract for an IEC 61508 Compliant Hercules Multi-Core Device
m of n
Modular Safety Case for an IEC 61508 Compliant COTS Multi-Core Device
Safety Contract for an IEC 61508 Compliant Product Sample COTS multi-Core Device
Safety Argumentation for an IEC 61508 Compliant Product Sample COTS multi-Core Device
Safety Argumentation for an IEC 61508 Compliant Product Line COTS Multi-Core Device
Safety Argumentation for an IEC 61508 Compliant TTEthernet Network
Safety Contract for an IEC 61508 Compliant TTEthernet Network
Safety Argumentation for an IEC 61508 Compliant TTEL Network
Safety Contract for an IEC 61508 Compliant TTEL Network
Safety Argumentation for an IEC 61508 Compliant Safety over EtherCAT Network
Safety Contract for an IEC 61508 Compliant Safety over EtherCAT Network
Modular Safety Case for an IEC 61508 Compliant Mixed-Criticality Network
Safety Contract for an IEC 61508 Compliant Product Sample Mixed-Criticality Network
Safety Argumentation for an IEC 61508 Compliant Product Sample Mixed-Criticality Network
Safety Argumentation for an IEC 61508 Compliant Product Line Mixed-Criticality Network
Safety Argumentation for an IEC 61508 Compliant Safety Protection Partition
Safety Contract for an IEC 61508 Compliant Safety Protection Partition
Modular Safety Case for an IEC 61508 Compliant Safety Partition
Safety Contract for an IEC 61508 Compliant Product Sample Safety Partition
Safety Argumentation for an IEC 61508 Compliant Product Sample Safety Partition
Safety Argumentation for an IEC 61508 Compliant Product Line Safety Partition
Safety Argumentation for an IEC 61508 Compliant XtratuM Hypervisor
Safety Contract for an IEC 61508 Compliant XtratuM Hypervisor
Safety Argumentation for an IEC 61508 Compliant PikeOS Hypervisor
Safety Contract for an IEC 61508 Compliant PikeOS Hypervisor
Modular Safety Case for an IEC 61508 Compliant Hypervisor
Safety Contract for an IEC 61508 Compliant Product Sample Hypervisor
Safety Argumentation for an IEC 61508 Compliant Product Sample Hypervisor
Safety Argumentation for an IEC 61508 Compliant Product Line Hypervisor
...

FIGURE 7.20: Wind Turbine Product Line Safety Argumentation – Example.

wind turbine product line architecture where a safety contract regarding the qualified tools selects the best combination of components for a particular wind turbine product sample. For instance, as shown in the same figure, the wind turbine product sample can choose from amongst different multi-core processors (Zynq-7000, Hercules), hypervisors (XtratuM, PikeOS) and mixed-criticality networks (TTEthernet, Safety over EtherCAT). In this case, this product sample selects the combination formed by a Zynq-7000 multi-core device, a XtratuM hypervisor, a safety protection partition and a TTEthernet mixed-criticality network as the optimum combination. It can be highlighted that the safety over EtherCAT network shall be used in all the product sample developments since it is defined as a critical requirement. Therefore, this network is directly selected from the product line abstraction layer (see Figure 7.20).

As defined in the previous paragraphs, this representation hierarchy can be extended for developing product lines of different application domains (e.g., railway, automotive) as it enables reusing the safety argumentation blocks of the commercial and custom components. Each domain specific safety standard defines additional requirements and measures and diagnostic techniques that shall be met to accomplish safety certification. In addition, this representation hierarchy can be extended to develop product samples with different levels of criticality (e.g., SIL1 to 4 according to the IEC 61508 safety standard). Figure 7.21 presents a partial representation of the safety argumentation for COTS multi-core devices that support the variations from safety standards and safety requirements.

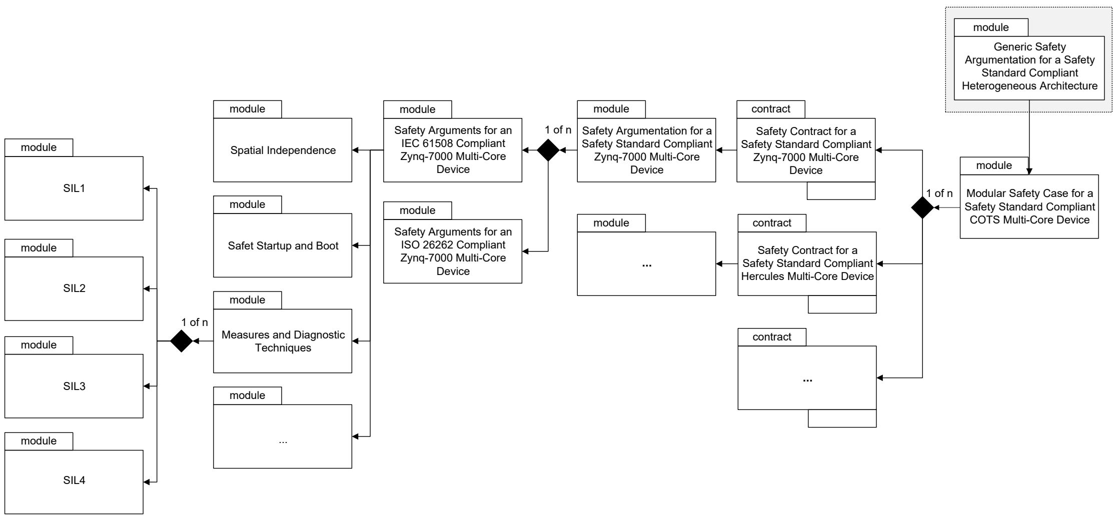
module
Generic Safety Argumentation for a Safety Standard Compliant Heterogeneous Architecture
module
Modular Safety Case for a Safety Standard Compliant COTS Multi-Core Device
1 of n
contract
Safety Contract for a Safety Standard Compliant Zynq-7000 Multi-Core Device
contract
Safety Contract for a Safety Standard Compliant Hercules Multi-Core Device
contract
...
module
Safety Argumentation for a Safety Standard Compliant Zynq-7000 Multi-Core Device
module
...
1 of n
module
Safety Arguments for an IEC 61508 Compliant Zynq-7000 Multi-Core Device
module
Safety Arguments for an ISO 26262 Compliant Zynq-7000 Multi-Core Device
module
Spatial Independence
module
Safet Startup and Boot
module
Measures and Diagnostic Techniques
module
...
1 of n
module
SIL1
module
SIL2
module
SIL3
module
SIL4

Figure 7.21: Variation points in a Wind Turbine product line – Example.

7.4 Cross-Domain Patterns

This section presents the implementation and evaluation results of the cross-domain patterns for multi-core processors, namely *Shared memory diagnosis pattern (SMDP)* and *Coherency management unit diagnosis pattern (CMUDP)*. All other patterns have been implemented and tested by third party entities and therefore, their implementation is out of the scope of this dissertation.

7.4.1 Shared Memory Diagnosis Pattern (SMDP)

The subsection presents the integration of the solutions provided by the shared memory diagnostic pattern into a Zynq-7000 zc706 multi-core device. More specifically, this subsection presents the application of the sub-scenarios 1.1 and 1.2 introduced by this cross-domain pattern within the PS, whereas the sub-scenarios 1.3 and 2.x are not implemented due to time limitations. Consequently, the PL of the Zynq-7000 device is not used in the implementation process.

Figure 7.22 shows the interconnection scheme provided by the Zynq-7000 multi-core device. This interconnection scheme enables accessing the memories through different routes. For instance, the DDR memory can be accessed by the CPUs through the SCU and the shared memory (red arrows). Instead, the OCM memory can be accessed through the SCU without going through the shared memory (green arrow). These two memories are used by this cross-domain pattern for implementing the shared memory diagnosis. The OCM memory is used for storing the values of the golden CRCs calculated by the CPUs (CPU0 and CPU1). Instead, the DDR memory is used for storing the data of the CPUs and reading them for comparing against the golden CRC.

On the other hand, for test purposes only, this implementation scenario supports a SW fault injector that causes periodical failures in the execution of this diagnostic pattern. The injector inserts modified data to the DDR memory region of CPU1 before calculating the CRCs. Consequently, the comparison of the CRCs results in an inconsistency, which stops the execution of the CPU1. This fault injection scheme can also be commanded through a soft-core processor as the memory B can be accessed through the PL.

The implementation of scenarios 1.1 and 1.2 requires SW tools such as the *Vivado Design Suite* [XIL16a] and the *SW Development Kit (SDK)* [XIL16b]. The main reason for that is that the Zynq-7000 zc706 device is a property of Xilinx Inc., and therefore all application SW generated for this device shall be compiled and programmed through tools belonging to Xilinx. Vivado SW is used for defining the Zynq device's system architecture, whereas the SDK tool is used for implementing the application code in C for the CPUs. The main difference between the scenarios 1.1 and 1.2 lies in the functionality executed by the CPUs. Sub-scenario 1.1 executes the same functionality in both CPUs, while the CPUs run different functionalities in sub-scenario 1.2.

Tables 7.7 and 7.8 present the execution states of the scenarios 1.1 and 1.2 defined in the SMDP. The right column defines the execution sequence followed by CPU0 while the left column defines the sequence of CPU1.

CPU0 execution	*CPU1 execution*
CPU0 - Cycle 0	CPU1 - Cycle 0
Disable cache on the OCM	Disable cache on the OCM
Disable cache on the FSBL	Disable cache on the FSBL
Initialize the SCU Interrupt Distributed (ICD)	Initialize the SCU Interrupt Distributed (ICD)
CPU0 - writing start address for CPU0	CPU1 - writing start address for CPU1
Golden CRC calculated and stored in the memory	Golden CRC calculated and stored in the memory
Write data to the DDR	Write data to the DDR
Read from the DDR	Read from the DDR

Comparing...	Comparing...
Successful comparison!	Successful comparison!
Writing new golden CRC in the memory	Writing new golden CRC in the memory
CPU0 - Cycle 1	CPU1 - Cycle 1
Disable cache on OCM	Disable cache on OCM
Disable cache on FSBL	Disable cache on FSBL
Initialize the SCU Interrupt Distributed (ICD)	Initialize the SCU Interrupt Distributed (ICD)
CPU0 - writing start address for CPU0	CPU1 - writing start address for CPU1
Golden CRC calculated and stored in the memory	Golden CRC calculated and stored in the memory
Write data to the DDR	Write data in the DDR
CPU0 modifying the data of CPU1 and writing it	...
in the DDR...	
Read from the DDR	Read from the DDR
Comparing...	Comparing...
Successful comparison!!!	Unsuccessful comparison!
...	Activating a safe state. Stopping CPU1...
CPU0 - Cycle 2	
Disable cache on OCM	
Disable cache on FSBL	
Continue...	

Table 7.7: SMDP – Implementation results of Scenario 1.1.

CPU0 execution	*CPU1 execution*
CPU0 - Cycle 0	CPU1 - Cycle 0
Disable cache on the OCM	Disable cache on the OCM
Disable cache on the FSBL	Disable cache on the FSBL
Initialize the SCU Interrupt Distributed (ICD)	Initialize the SCU Interrupt Distributed (ICD)
CPU0 - writing start address for CPU0	CPU1 - writing start address for CPU1
Golden CRC calculated and stored in the memory	Golden CRC calculated and stored in the memory
Write data to the DDR	Write data to the DDR
CPU0 modifying the data of CPU1 and writing it	...
in the DDR...	
Read from the DDR	Read from the DDR
Comparing...	Comparing...
Successful comparison!	Unsuccessful comparison!!!
...	Activating safe state. Stopping CPU1...
CPU0 - Cycle 1	
Disable cache on OCM	
Disable cache on FSBL	
Continue...	

Table 7.8: SMDP – Implementation results of Scenario 1.2.

These implementations are supported by the dispersion diagrams shown in Figures 7.23 and 7.24. A dispersion diagram represents the measures of data variability using Cartesian coordinates. These diagrams use two axes that define the relationship between two variable values. The horizontal axis represents the execution and failure detection times of the SMDP while the vertical axis represents the probability that the execution and failure detection happen.

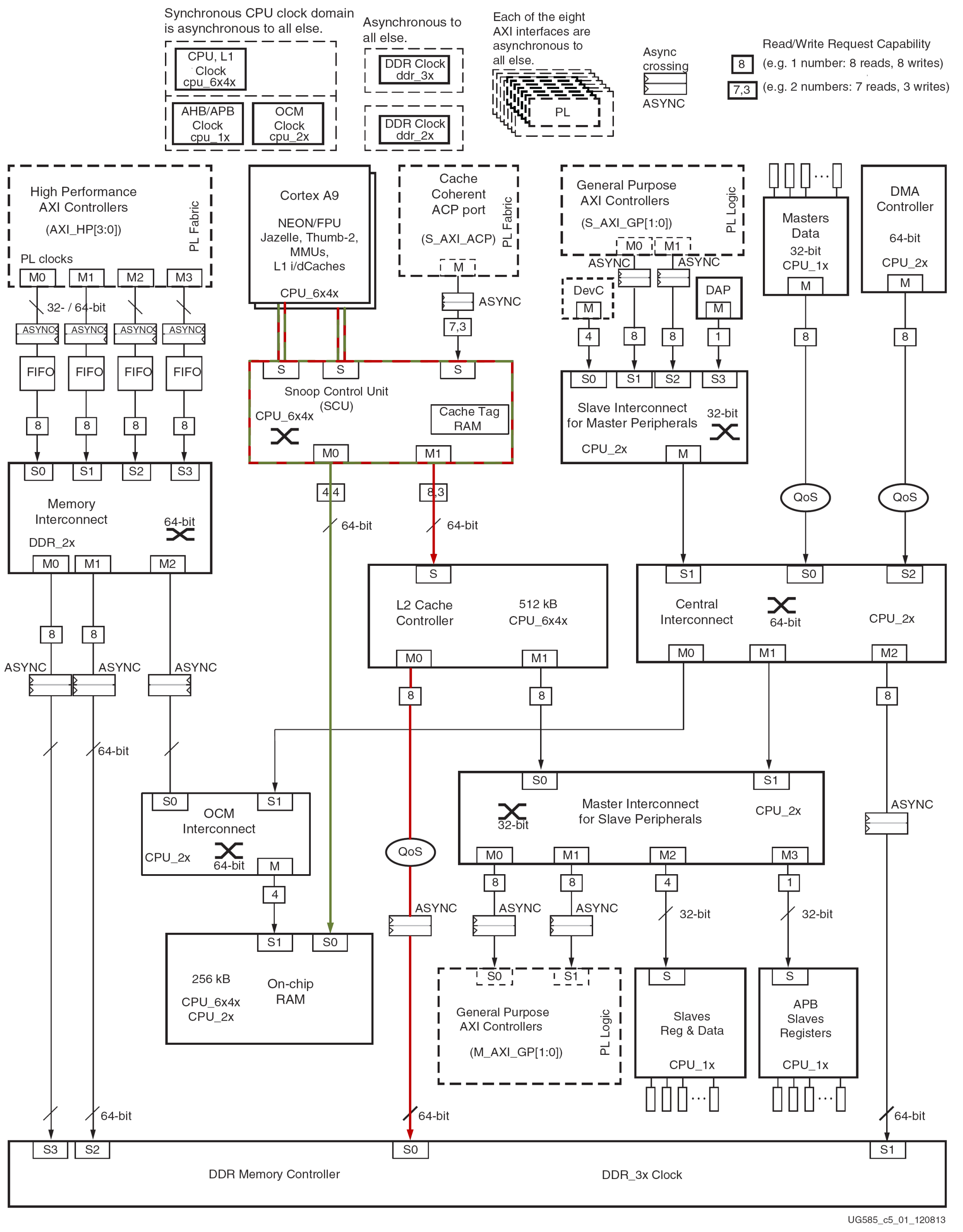
Synchronous CPU clock domain is asynchronous to all else.
CPU, L1 Clock cpu_6x4x
AHB/APB Clock cpu_1x
OCM Clock cpu_2x
Asynchronous to all else.
DDR Clock ddr_3x
DDR Clock ddr_2x
Each of the eight AXI interfaces are asynchronous to all else.
PL
Async crossing
ASYNC
Read/Write Request Capability
(e.g. 1 number: 8 reads, 8 writes)
(e.g. 2 numbers: 7 reads, 3 writes)
High Performance AXI Controllers (AXI_HP[3:0])
PL Fabric
PL clocks
Cortex A9 NEON/FPU Jazelle, Thumb-2, MMUs, L1 i/dCaches CPU_6x4x
Cache Coherent ACP port (S_AXI_ACP)
General Purpose AXI Controllers (S_AXI_GP[1:0])
PL Logic
Masters Data 32-bit CPU_1x
DMA Controller 64-bit CPU_2x
DevC
DAP
32- / 64-bit
FIFO
Snoop Control Unit (SCU) CPU_6x4x
Cache Tag RAM
Slave Interconnect for Master Peripherals CPU_2x 32-bit
Memory Interconnect DDR_2x 64-bit
QoS
L2 Cache Controller 512 kB CPU_6x4x
Central Interconnect 64-bit CPU_2x
OCM Interconnect CPU_2x 64-bit
Master Interconnect for Slave Peripherals 32-bit CPU_2x
256 kB CPU_6x4x CPU_2x On-chip RAM
General Purpose AXI Controllers (M_AXI_GP[1:0])
Slaves Reg & Data CPU_1x
APB Slaves Registers CPU_1x
DDR Memory Controller
DDR_3x Clock
UG585_c5_01_120813

FIGURE 7.22: Zynq-7000 ZC706 – OCM and DDR interconnect. (Source [XIL14c])

The following dispersion diagrams are generated as a result of one million executions of the SMDP where the same amount of failures is injected in each execution. The failures are injected only once per cycle to the data of the CPU1 (i.e., 50ms). It is considered that random failures that may occur in an industrial environment can also be detected by this diagnostic pattern. However, the assessment of this assumption is out of the scope of this thesis and therefore, it should be evaluated in future trials.

The execution time diagram shown in Figure 7.23 presents a bimodal (double-peak) distribution with two local maximums at 0,0892ms and 0,09015ms. A bimodal distribution is a continuous distribution with two or more peaks. This dispersion diagrams also presents a bounded execution of the shared memory diagnostic pattern with a minimum execution time value of 0,0887ms and a maximum value of 0,0911ms.

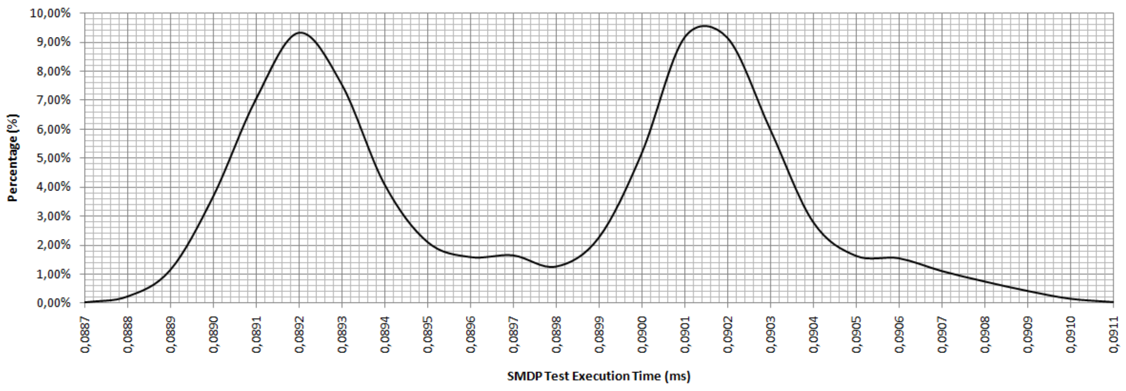

FIGURE 7.23: SMDP – Execution Time Dispersion.

Instead, the failure detection diagram shown in Figure 7.24 follows a normal distribution, also known as *Gaussian bell* distribution. This distribution is the most commonly observed probability distribution. The bell curve shown in this figure represents the failure detection time distribution of the shared memory diagnostic pattern that is bounded between 0,003477ms and 0,003877ms with the highest peak at 0,03711ms.

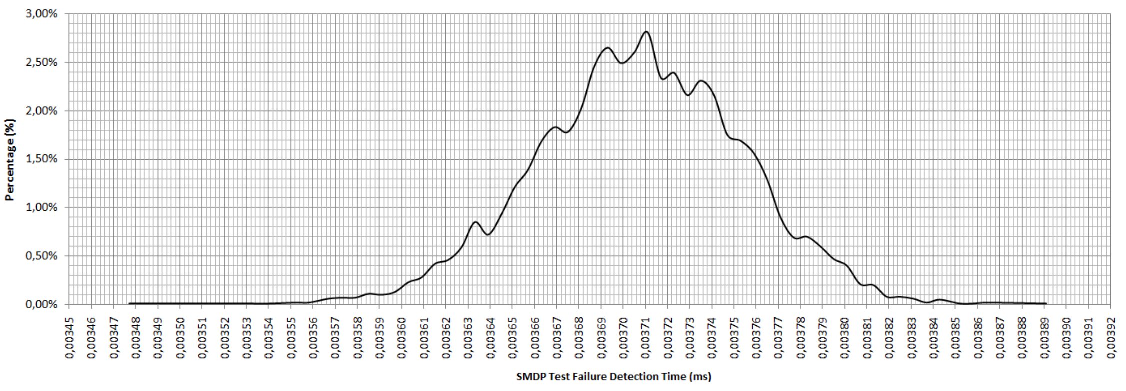

FIGURE 7.24: SMDP – Failure Detection Time Dispersion.

7.4.2 Coherency Management Unit Diagnosis Pattern (CMUDP)

This subsection presents the implementation and the results of the solutions proposed in Subsection 6.2.2 for detecting and controlling the failures in the coherency management units. This implementation is based on the Zynq-7000 zc706 multi-core device that implements a SCU for managing the coherency between the processing cores and the memories.

The first solution defined in this pattern proposes to actively diagnose the configuration of the coherency management unit to detect configuration errors that can disrupt the behaviour of the coherency unit. Table 7.9 shows the registers associated with the coherency management unit of the Zynq-7000 zc706 device. These registers are periodically checked (e.g., each 50ms) and compared against the values expected (pre-defined values at design time) to detect whether the configuration of the coherency management unit changes. If the configuration registers coincide, the device will continue working. Otherwise, a fault is asserted and the device should reach a safe state.

Control register bit assignment		
Bit	Name	Description
[2]	SCU RAMs parity	1 = Parity on. 0 = Parity off.
Configuration register bit assignment		
Bit	Name	Description
[7:4]	CPUs SMP	Defines the processor mode: 0: Asymmetric MultiProcessing (AMP) mode not taking part in the coherency or not present. 1: SMP mode taking part in the coherency. [7] CPU3 [6] CPU2 [5] CPU1 [4] CP0
SCU CPU power status register bit assignment		
Bit	Name	Description
[25:24]	CPU status	Power status of the processor b00: Normal mode. b01: Reserved. b10: the processor is about to enter (or is in) dormant mode. No coherency request is sent to the processor. b11: the processor is about to enter (or is in) powered-off mode, or is non-present. No coherency request is sent to the processor.
AXI user attributes encodings		
Bit	Name	Description
[0]	ARUSERMx	Shared bit 1: Coherent request. 0: Non-coherent request.
[1]	AWUSERMx	Shared bit 1: Coherent request. 0: Non-coherent request.

TABLE 7.9: Zynq-7000 zc706 SCU's coherency configuration registers. (Source [XIL14c])

The results of executing this solution are presented in the next lines where all registers related to the coherency management unit are read and compared against the expected values. For demonstration purposes only, the configuration register of the *SCU CPU power status* is altered to check the effectiveness of this diagnostic technique. Consequently, whenever a modified register value is detected, the system should be restarted and the CMUDP should be re-executed.

```
#### Checking configuration errors

#### Reading registers of SCU Controller....
10001000001101111000000001011010
#### Checking if read values match with the expected configuration values
#### Successful Comparison!
#### Summary of SCU controller's registers:
SMP mode activated for CPU0
AMP mode activated for CPU1
AMP mode activated for CPU2
AMP mode activated for CPU3

#### Reading registers of the SCU CPU power status....

00000000000000001101001000011101

#### Checking if read values match with the expected configuration values

#### Unsatisfactory Comparison!

#### Restarting...
```

This diagnostic pattern implements further measures and diagnostic techniques to diagnose the coherency control unit of the Zynq-7000 device. These measures and diagnostic techniques include CRC with comparison, ECC and parity bit check techniques which aim to detect random failures in the coherency control unit. The CRC with comparison technique is defined in Subsection 7.4.1. On the other hand, the DDR, L1 and L2 caches and the OCM memory and the SCU of the Zynq-7000 device support the parity bit check technique [XIL14c]. In the case of the DDR memory, the parity bit can be enabled or disabled in the configuration registers (*register DRAM Controller (DDRC) ECC_scrub [4:0] with relative resolution address 0x000000F4 and absolute address 0xF80060F4, bit [2:0] in b010*). In the same vein, the parity bit of the L2 cache can be enabled in the configuration registers. The parity bit of the L2 cache is by default disabled.

The steps followed by diagnose the parity bit errors in the L2 cache are the following:

i) Disable L2 cache

ii) Disable the parity

iii) Enable L2 cache

iv) Write data

v) Disable L2 cache

vi) Enable parity

vii) Enable L2 cache

viii) Read data

The OCM memory supports both single and multiple bit parity bit errors. In the event that a parity error is detected, an interrupt is asserted and the parity bit errors of the OCM memory is returned. The steps followed to diagnose random faults using the parity bit technique are the following:

i) Disable D cache of L1 and L2 cache memories.

ii) Disable I cache of L1 and L2 cache memories.

iii) Configure the *OCM_PARITY_CTRL* register to enable the AXI read and the use of interrupts for reporting the parity error to the CPU.

iv) Write data to the OCM to generate a parity error.

v) Read data from the OCM.

The execution results of this pattern to check parity bit error in the OCM and L2 cache memories are shown below, where an error is inserted for checking the application.

```
#### OCM parity error test
#### An exception processed
IRQ No.35 OCM interrupt processed
#### OCM parity error test is run successfully run

#### L2 cache parity error test

#### An exception processed

IRQ No.34 L2 cache interrupt processed

#### L2 cache parity error test is run successfully run
```

The Zynq-7000 device supports the ECC technique in half-bus (16bit) data width configuration. The ECC provides single error correction and dual error detection. When this technique is enabled, a write operation computes and stores the ECC code along with the data. Then, the ECC code that is stored in the memory is compared against the data that is read. To that end, all memory locations are written before being read, thus avoiding reading ECC errors. The errors detected by ECC diagnosis can be classified into correctable and uncorrectable errors. For correctable errors, there is no error actively signalled via an interrupt or AXI response. Instead, for uncorrectable errors, the controller returns a signal response back to the requesting AXI bus master. In both cases, information regarding the errors (such as a column, row and bank error address and error byte lane) is logged in the controller register space. In the case that the controller detects a correctable ECC error, it automatically corrects the error and sends the right data to the bus master. Instead, when the driver (e.g., DDRC) detects an uncorrectable ECC error, it returns a response signal to the bus master with the uncorrectable data. In that case, if the shared memory is disabled, the signal response is directly received by the CPU, causing a data abort. Otherwise, the ECC error is reported to the CPU using an interrupt sourced in the shared memory.

This solution considers the four scenarios that depend on the ECC error detection requirements (detection of correctable and uncorrectable errors) and the availability of the shared cache memory (shared memory enable and disable). The steps followed for executing this technique in the Zynq-7000 zc706 device include:

i) Disable the cache

ii) Read ECC registers

iii) Initialize data in the DDR memory

iv) Disable the ECC

v) Depending on the errors which are required to be detected, this pattern injects uncorrectable or correctable errors in the DDR.

vi) Enable the ECC

vii) Read ECC resisters

viii) If the cache memory shall be implemented, enable the cache, set up an interrupt for reporting the ECC errors to the CPU and read data from the DDR memory.

ix) If uncorrectable ECC errors are detected, an interrupt is generated by the cache memory to report the ECC errors to the CPU.

x) Otherwise, the ECC error is directly transmitted to the CPU.

The results of the execution of the ECC diagnosis sequence presented before are the following:

```
#### Disable L1 and L2 Cache
#### Read ECC registers
DDRC.CHE_CORR_ECC_LOG_REG_OFFSET:00000000 (No Correctable Error)
DDRC.CHE_UNCORR_ECC_LOG_REG_OFFSET:00000000 (No Uncorrectable Error)
DDRC.CHE_ECC_STATS_REG_OFFSET:00000000 (0 Correctable Error(s), 0 Uncorrectable Error(s))
#### Initialize Data on DDR3
00100000:  00000000
00100004:  00000000
00100008:  00000000
0010000C: 00000000
```

```
00100010:  00000000
00100014:  00000000
00100018:  00000000
0010001C: 00000000

#### Disable ECC
ADDR: 0x000000F4 W: 0x04
#### Insert Correctable Errors (1bit error) on DDR3
00100000:  00000001
00100004:  00000000
00100008:  00000000
0010000C: 00000000
00100010:  00000000
00100014:  00000000
00100018:  00000000
0010001C: 00000000

#### Enable ECC
ADDR: 0x000000F4 W:0x04, ADDR:0x000000C4 W:0x03, W:0x00
#### Read ECC registers
DDRC.CHE_CORR_ECC_LOG_REG_OFFSET:00000000 (No Correctable Error)
DDRC.CHE_UNCORR_ECC_LOG_REG_OFFSET:00000000 (No Uncorrectable Error)
DDRC.CHE_ECC_STATS_REG_OFFSET:00000000 (0 Correctable Error(s), 0 Uncorrectable Error(s))
#### Enable Cache
#### Enable D-Cache (L1 and L2)
#### Enable I-Cache (L1 and L2)
#### Set Up Interrupt
#### Read Data from DDR3 again
00100000:  00000000

#### Read ECC registers

DDRC.CHE_CORR_ECC_LOG_REG_OFFSET:00000007 (Correctable Error Detected)

DDRC.CHE_CORR_ECC_ADDR_REG_OFFSET:00040000 (Correctable Error:  Bank=0x0, Row=0x40, Column=0x0)

DDRC.CHE_UNCORR_ECC_LOG_REG_OFFSET:00000000 (No Uncorrectable Error)

DDRC.CHE_ECC_STATS_REG_OFFSET:00000100 (1 Correctable Error(s), 0 Uncorrectable Error(s))
```

Further considerations when using the coherency management unit include that it is assumed that the components of the multi-core device are diagnosed in advance to detect and control systematic failures and that the use of the shared memory is minimised to an absolute minimum required for operating (fault avoidance). To that end, the SCU of the Zynq-7000 zc706 device should be disabled (see configuration registers in [ARM11b]).

The following figures present the dispersion diagrams of the tests executed by the CMUDP. These diagrams are generated as a result of one million executions of the pattern where the same amount of failures are injected to the SCU configuration, L2 cache data, OCM data and ECC (one failure per cycle). The horizontal axes of these dispersion diagrams represent the time required for executing the tests of CMUDP, including the failures detection times. The vertical axes define the probability that the failure detection happens.

It is assumed that this cross-domain pattern can be applied in an industrial environment where failures may occur at a random time. The main reason for this assertion is that, in a real scenario: *(a)* we do not need to inject faults and *(b)* the diagnostic pattern will be periodically executed to detect the occurrence of faults. The assessment of this cross-domain pattern in a realistic scenario is out of the scope of this dissertation, and therefore, it should be evidenced in future trials.

Figure 7.25 shows the dispersion diagram of the SCU configuration test execution where one failure (a modified configuration register) is injected in the configuration each cycle. This diagram follows a normal dispersion bounded between 0,00466ms and 0,00526ms with a maximum peak at 0,00484ms and a maximum execution time percentage of 13,32%.

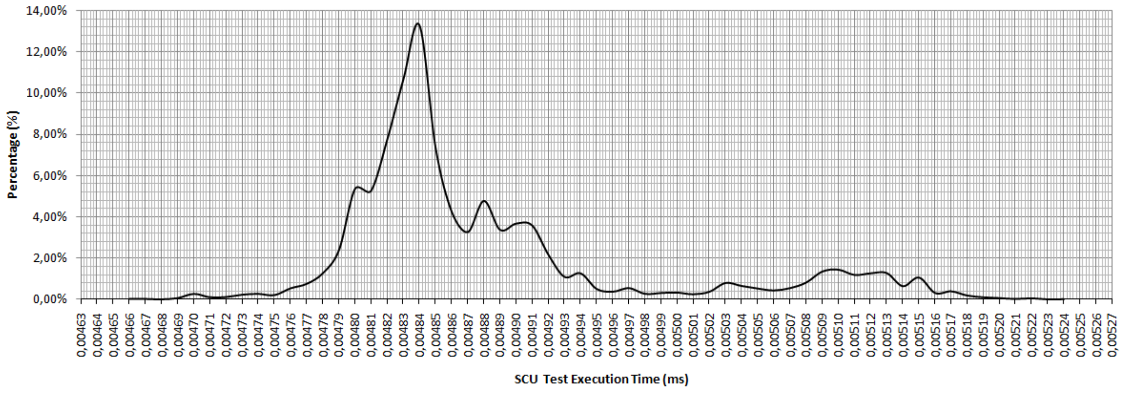

FIGURE 7.25: CMUDP – SCU Configuration Test Dispersion.

On the other hand, Figure 7.26 shows the dispersion diagram of the L2 cache test execution with fault injection (one failure per cycle). This diagram presents an approximation of the normal distribution with a maximum peak at 97,9ms and media at 98,14ms. The execution of the L2 cache test is bounded between 97,7ms and 98,56ms and reaches a maximum execution time percentage of 36,07% at its highest peak.

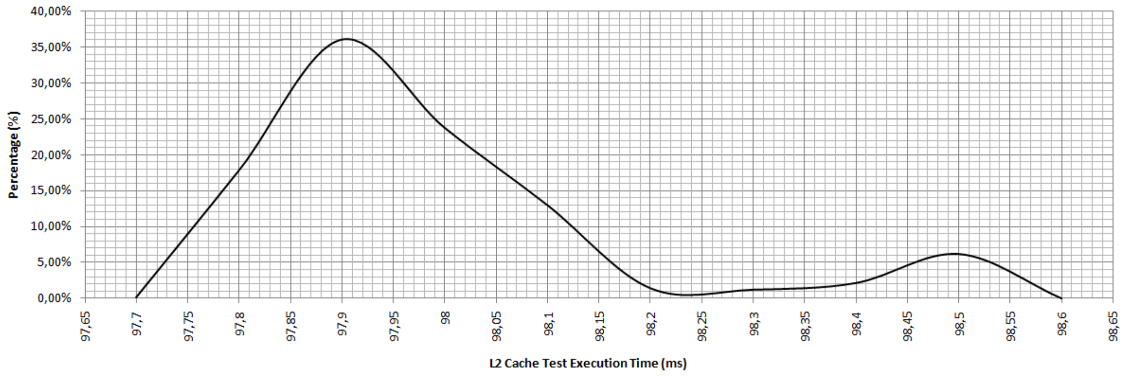

FIGURE 7.26: CMUDP – L2 cache Test Dispersion.

Figure 7.27 exhibit also a normal distribution dispersion where one failure is injected per cycle. The failure injects modified data to the OCM memory. This dispersion shows the highest peak at 0,2670ms with an execution percentage of 16,79%. The execution of the OCM test is bounded between 0,261ms and 0,281ms with a media of 0,2707ms, which comparing with values of the previous figure shows a more linear dispersion with lower execution times.

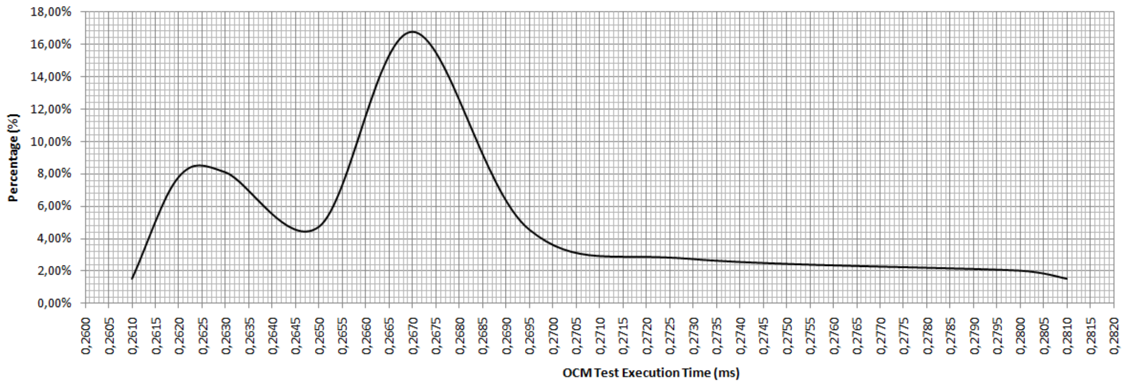

Figure 7.27: Coherency Management Diagnostic Pattern – OCM Test Dispersion.

The ECC test execution dispersion diagram also shows a normal distribution with a maximum peak at 0,0575ms and an execution percentage of 17.97% (see Figure 7.28). This diagram presents a bounded dispersion between 0,0571ms and 0,0584ms and a media of 0,0577ms.

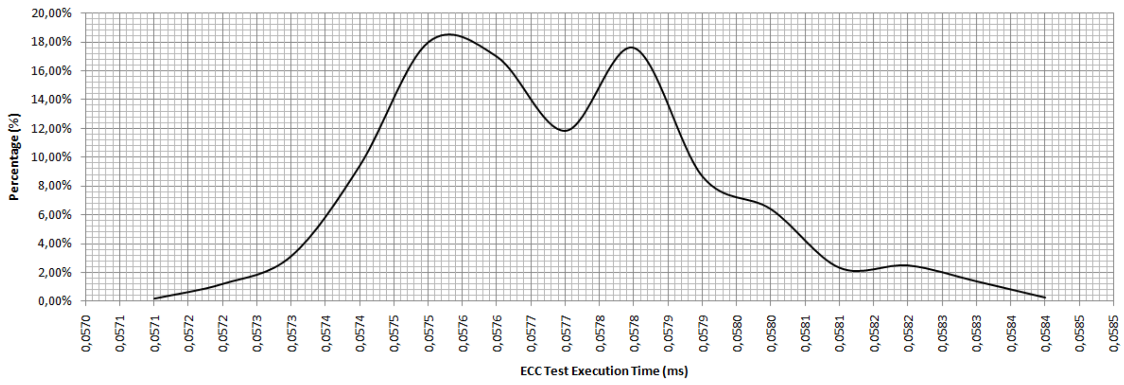

Figure 7.28: Coherency Management Diagnostic Pattern – ECC Test Dispersion.

The dispersion diagrams presented in the previous figures follow a normal distribution where each distribution has a maximum peak and bounded execution times. The sole exception focuses on the execution time required by the L2 cache test which requires higher execution time comparing with the rest of the tests and results in an increase of the cycle execution time. Nevertheless, the execution cycle time of this diagnostic pattern can be modified under demand depending on the requirements of the system under test. For instance, this pattern is executed by the wind turbine use case each

100ms, although it may be configured for being executed each 150ms, always considering the minimum time that is required to run the diagnostic pattern (maximum execution time).

Chapter 8

Conclusions

This chapter reviews the work defined throughout this dissertation and identifies possible future opportunities for research.

8.1 Review

This thesis provides reusable generic solutions to commonly occurring challenges in today's mixed-criticality systems. To that end, this dissertation defines a modular safety development process based on the IEC 61508 compliant development process and the modularity methodology. Modularity enables limiting the impact of changes to reduced areas of the safety case, allowing the reusability of those areas. Based on the modularity method, this thesis introduces the MSCs for an IEC 61508 compliant generic hypervisor, partition, COTS multi-core device and mixed-criticality network. These modular safety concepts can be used as the reference documents for developing and certifying mixed-criticality systems based on the IEC 61508 safety standard. In addition, this dissertation provides the analyses of a selected set of commercial components (including the XtratuM hypervisor, the Zynq-7000 multi-core device, the Safety over EtherCAT and Time-Triggered Ethernet networks) and exposes the way in which those components fulfil the safety-related arguments defined in the MSCs.

As a result of the definition of the MSCs and the analysis of the IEC 61508 safety standard, reusable generic cross-domain patterns to solve the common certification challenges in today's mixed-criticality systems are defined. These patterns enable reducing the engineering and certification cost and provide benefits regarding scalability. The patterns exposed in this thesis can be taken as the basis for developing other cross-domain patterns that will solve future challenges in mixed-criticality systems and update today's safety related standards such as the IEC 61508 safety standard for E/E/PE functional safety systems.

Finally, this thesis presents the theoretical integration of the DREAMS development process, the MSCs and the cross-domain patterns on a simplified wind-turbine case study. The development of the wind turbine system has been analysed from a product line perspective, identifying the linkage between the safety-related requirements of the subsystems and components that compose a wind turbine. This integration provides the evidence to sustain the applicability of the contributions of this dissertation, including the benefits gained regarding scalability, reusability and lower engineering and certification cost.

8.2 Future work

The contributions of this thesis can be extended leading to future research lines:

- *Extra MSCs*: In this dissertation, the MSCs for some common components of mixed-criticality systems are defined (e.g., safety hypervisor, partition, COTS multi-core device, mixed-criticality network). However, the mixed-criticality systems are also composed of other subsystems (e.g., operating system) which shall be compliant with a safety standard to allow a successful certification. Therefore, further MSCs could be defined to support the MSCs already defined in this thesis.
- *Realistic implementation:* This dissertation provides a theoretical definition of the MSCs for an IEC 61508 hypervisor, partition, COTS device and mixed-criticality network. However, the industrial application of those MSC in the development and certification of a commercial IEC 61508 compliant hypervisor, partition, COTS multi-core device and mixed-criticality network is not provided (outside the scope of this research). Therefore, the industrial certification of those components and the subsequent development and certification of a mixed-criticality system that uses the certified components could be the following course of action.
- *Automatization:* Manual linking analyses have been performed in this thesis for the safety-related arguments defined in the MSCs for the commercial hypervisor, a safety protection partition, a COTS multi-core device and a mixed-criticality network. Therefore, a future research line would be the automatization of the generation of the linking analyses, thus reducing the time spent for their definition. For instance, a qualified tool could be used for managing the safety-related requirements.

- *Extra cross-domain patterns:* This dissertation defines several reusable generic cross-domain patterns to solve the commonly occurring problems in mixed-criticality systems. In addition, some of those patterns have been implemented and integrated into a system architecture based on a simplified wind turbine system. However, the implementation of the remaining cross-domain patterns is still pending. Therefore, another future research line could be to implement the remaining cross-domain patterns.

- *Extension to other safety standards:* This dissertation is focused on the IEC 61508 safety standard. This standard is the basis for other domain-specific safety standards such as ISO 26262 (automotive), EN 5012X (railway) and ISO 13849 (machinery). However, the applicability of IEC 61508 does not cover all engineering domains. For instance, the domain of avionics uses of its own safety standards where the requirements differ from the ones defined by the IEC 61508. Therefore, further MSCs with safety-related requirements in compliance with other safety standards could be defined. For example, future work can consider the DO-178C as the basic avionic safety standard.

- *Extension to security standards:* The MSCs, the cross-domain patterns and the case study provided in this thesis are defined from a safety perspective. However, other domains such as the security or real-time domain are still not analysed. Therefore, the work presented in this dissertation could be extended to cover other approaches such as security.

Bibliography

[Ada95] Sam S. Adams. *Functionality ala carte*, pages 7–8. ACM Press/Addison-Wesley Publishing Co., 1995.

[Ade98] Adelard. Adelard safety case development manual (ASCAD), 1998.

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Computers*, 1(1):11–33, January 2004.

[AM95] Vangalour S. Alagar and R Missaoui. *Object-Oriented Technology for Database and Software Systems.* World Scientific Publishing Co. Pte. Ltd., 1995.

[ARA11] ARAMIS. ARAMIS - Automotive, Railway and Avionics Multicore Systems, 2011.

[ARI06] ARINC. ARINC specification 653. October 16 2006.

[ARM08] ARM. ARM Generic Interrupt Controller (GIC): Architecture specification v1.0. Report, ARM, 9 2008.

[ARM11a] ARM. AMBA AXI and ACE protocol specification. Report, ARM, 10 2011.

[ARM11b] ARM. Cortex A9 - Technical reference manual. Report, ARM, 7 2011.

[ARM13] ARM. ARM Generic Interrupt Controller: Architecture specification v2.0. Report, ARM, 7 2013.

[ART10] ARTEMIS ACROSS. ARTEMIS ACROSS, 2010.

[AS14a] DNV GL AS. *Project certification of wind farms according to IEC 61400-22*, 12 2014.

[AS14b] DNV GL AS. *Type and component certification of wind turbines according to IEC 61400-22*, 12 2014.

[ASC12] ASCOS. Aviation safety and certification of new operations and systems, 7 2012.

[Bau11] T. Blasum H. Tverdyshev S. Baumann, C. Bormer. Proving memory separation in a microkernel by code level verification. In *14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pages 25–32, 3 2011.

[BB98] Peter Bishop and Robin Bloomfield. A methodology for safety case development. In *Proceedings of the Sixth Safety-critical Systems Symposium*, pages 194–203. Springer London, 1998.

[BB10] Robin Bloomfield and Peter Bishop. *Safety and Assurance Cases: Past, Present and Possible Future - an Adelard Perspective*, pages 51–67. Springer London, 2010.

[CAS14] CAST-32. Certification Authorities Software Team (CAST), 2014.

[CFBC99] Carl Carmichael, Earl Fuller, Phil Blain, and Michael Caffrey. SEU Mitigation Techniques for Virtex FPGAs in Space Applications, 1999.

[Cha13] Jean-Baptiste Chaudon. TTEthernet theory and concepts. In *Proceedings of the 2009 Eighth IEEE International Symposium on Network Computing and Applications*, NCA '09, pages 319–322, 2013.

[Com12] European Commission. Mixed-criticality systems. In *Workshop on Mixed-Criticality Systems*, volume 41, page 31, Brussels, Belgium, 2012.

[CON14] FP7 CONTREX. CONTREX - Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties, 2014.

[DAM$^+$13] R. Das, R. Ausavarungnirun, O. Mutlu, A. Kumar, and M. Azimi. Application-to-core mapping policies to reduce memory system interference in multi-core systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 107–118, 2 2013.

[DAN$^+$13] Dajshina Dasari, Benny Akesson, Vicent Nelis, Mihammad Ali Awan, and Stefan M. Petters. Identifying the sources of unpredictability in COTS based multi-core systems. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 39–48, 2013.

[Dou99] Bruce Powel Douglass. *Doing hard time: Developing real-time systems with UML, objects, frameworks, and patterns.* Addison-Wesley Longman Publishing Co., Inc., 1999.

[Dou02] Bruce Powell Douglass. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems.* Addison-Wesley Longman Publishing Co., Inc., 2002.

[DP15] Ewen Denney and Ganesh Pai. Towards a formal basis for modular safety cases. In *Proceedings of the 34th International Conference on Computer Safety, Reliability, and Security - Volume 9337*, SAFECOMP 2015, pages 328–343, Cham, 2015. Springer International Publishing.

[DRE13] FP7 DREAMS. DREAMS - Distributed REal-time Architecture for Mixed Criticality Systems, 2013.

[DRE15] FP7 DREAMS. D5.1.1 distributed real-time architecture for mixed-criticality systems - A modular safety case for hypervisor, 1 2015.

[EC02] Luke Emmet and George Cleland. Graphical notations, narratives and persuasion: a pliant systems approach to hypertext tool design. In *ACM Hypertext (HT) Proceeding of*, HYPERTEXT '02, pages 55–64, 2002.

[EMC14] ARTEMIS EMC2. EMC2 - Embedded Multi-Core systems for Mixed-Criticality applications in dynamic and changeable real-time environments, 2014.

[EN99] EN. *EN 50126 Railway applications - The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS).* EN, 9 1999.

[EN03] EN. *EN 50129 Railway applications - Communications, signalling and processing systems - Safety related electronic systems for signalling.* EN, 2 2003.

[ESEH$^+$12] Christian El Salloum, Martin Elshuber, Oliver Hollberg, L.ftberger, Haris Isakovic, and Armin Wasicek. The ACROSS MPSoC: A new generation of multi-core processors designed for safety-critical embedded systems. In *15th Euromicro Conference on Digital System Design (DSD 2012)*, pages 105–113, 9 2012.

[Eth11] EtherCAT. Safety over EtherCAT, 2011.

[Eth12] EtherCAT. EtherCAT slave implementation guide. techreport, EtherCAT, 1 2012.

[FJKM11] Yong-Yi Fan Jiang, Jong.Yih Kuo, and Shang-Pin Ma. An embedded software modeling and process by using aspect-oritented approach. *Journal of Software Engineering and Applications*, 4(2):106–122, 4 2011.

[For16] Fortiss. Design space exploration tool, 2016.

[Fre13] Freescale. Safety manual for Qorivva MPC5643L. Report, Freescale, 4 2013.

[GBEL10] Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. The Mälardalen WCET benchmarks - past, present and future. In Björn Lisper, editor, *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, volume 15, pages 136–146. Schloss Dagstuhl, 2010.

[GdL00] Fernanda Gusmao de Lima. *Single Event Upset Mitigation Techniques for Programmable Devices*. Graduado, 2000.

[GDR05] Kees Goossens, John Dielissen, and Adrei Radulescu. AEthereal network on chip: concepts, architectures and implementations. *IEEE Design Test of Computers*, 22:414–421, 9 2005.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Wesley, Addison, 1994.

[GP14] Yashu Gosain and Prusholthaman Palanichamy. TrustZone technology support in Zynq-7000 All Programmable SoCs, 5 2014.

[Hal14] Ed Hallett. Developing secure and reliable single device design with Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs using the Isolation Design Flow. Report, XILINX, 4 2014.

[Ham03] Rob Hammett. Flight-critical distributed systems - Design considerations [avionics]. *IEEE Aerospace and Electronic Systems Magazine*, 18(6):30–36, 2003.

[HCM15] Ed Hallett, Giulio Corradi, and Steven McNeil. Xilinx reduces risk and increases efficiency for IEC 61508 and ISO 26262 certified safety applications, 4 2015.

[Hea15] B & R Corporate Headquarters. Automation PC 910, 2015.

[HGZ$^+$13] Brahim Hamid, Jacob Geisel, Adel Ziani, Jean-Michel Bruel, and Jon Perez. *Model-Driven Engineering for Trusted Embedded Systems Based on Security and Dependability Patterns*, pages 72–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[HS12] Jameel Hussein and Gary Swift. Mitigating single-event upsets, 4 2012.

[IEC02a] IEC. *IEC 62280 - Railway applications - Communication, signalling and processing systems*. IEC, 10 2002.

[IEC02b] IEC. *IEC 62280-1 Railway applications - Communication, signalling and processing systems - Part 1: Safety-related communication in closed transmission systems*. IEC, 10 2002.

[IEC03] IEC. *IEC 61158 Digital data communications for measurement and control - Fieldbus for use in industrial control systems.* IEC, 6 2003.

[IEC06] IEC. *IEC 60204-1 Safety of Machinery - Electrical equipment of machines - Part 1: General Requirements*, 6 2006.

[IEC10a] IEC. *IEC 61508-1 Functional safety of Electrical/Electronic/Programmable Electronic safety-related systems - Part 1: General Requirements.* IEC, 4 2010.

[IEC10b] IEC. *IEC 61508-2 Functional safety of Electrical/Electronic/Programmable Electronic safety-related systems - Part 2: Requirements for Electrical/Electronic/Programmable Electronic safety-related systems.* IEC, 4 2010.

[IEC10c] IEC. *IEC 61508-3 Functional safety of Electrical/Electronic/Programmable Electronic safety-related systems - Part 3: Software requirements.* IEC, 4 2010.

[IEC10d] IEC. *IEC 61508-4 Functional safety of Electrical/Electronic/Programmable Electronic safety-related systems - Part 4: Definitions and abbreviations.* IEC, 4 2010.

[IEC10e] IEC. *IEC 61508-7 Functional safety of Electrical/Electronic/Programmable Electronic safety-related systems - Part 7: Overview of techniques and measures.* IEC, 4 2010.

[IEC10f] IEC. *IEC 61784-3-12 Industrial communication networks - Profiles - Part 3-12: Functional safety fieldbuses - Additional specifications for CPF 12.* IEC, 6 2010.

[IEC10g] IEC. *IEC 61784-3 Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions.* IEC, 6 2010.

[Ins13] Texas Instruments. Safety manual for TMS570LS31x and TMS570LS21x Hercules ARM safety-critical micro-controllers - User guide. Report, Texas Intruments, 4 2013.

[Int11] SAE International. SAE Aerospace Standard: As-2d2 deterministic ethernet and unified networking, 2011.

[ISO04] ISO/IEC. *ISO/IEC 17000 Conformity assessment – Vocabulary and general principles.* ISO/IEC, 6 2004.

[ISO06] ISO. *ISO 13849-1: Safety of machinery – Safety-related parts of control systems – Part 1: General principles for design*, 11 2006.

[ISO09] ISO. *ISO 26262-10 Road Vehicles – Functional Safety: Guideline*. ISO, 11 2009.

[ISO11] ISO. *ISO 26262-1 Road Vehicled – Functional Safety – Part 1: Vocabulary*. ISO, 11 2011.

[ISO12] ISO. *ISO 13849-2: Safety of machinery – Safety-related parts of control systems – Part2: Validation*, 10 2012.

[ISO15a] ISO. *ISO 26262-4 Road vehicles – Functional safety – Part 4: Product development at the system level*. ISO, 1 2015.

[ISO15b] ISO. *ISO 26262-5 Road vehicles – Functional safety – Part 5: Product development at the hardware level*. ISO, 1 2015.

[ISO15c] ISO/IEC. *ISO/IEC 15408 Information Technology – Security Techniques – Evaluation Criteria for IT Security – Part 1: Introduction and general model*. ISO/IEC, 8 2015.

[JED01] JEDEC Solid State Technology Association. Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices, 8 2001.

[Kel01] Tim Kelly. Concepts and principles of compositional safety case construction, 5 2001.

[Kel07] Tim Kelly. Modular certification: Acknowledgements to the industrial avionic working group (IAWG), 2007.

[KHMHB10] Yoongu Kim, Dongsu Han, Onur Mutlu, and Mor Harchol-Balter. Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *2010 International Conference on High Performance Computing Systems (HPCS)*, page 12, 2010.

[Kop08] Hermann Kopetz. The complexity challenge in embedded system design. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 3–12. IEEE, 5 2008.

[Kop11] Hermann Kopetz. *Real-Time Systems: Design principles for distributed embedded applications*. Springer, second edition, 2011.

[KP15] Woo-Cheol Kwon and Li-Shiuan Peh. A universal ordered NoC design platform for shared-memory MPSoC. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '15, pages 697–704, 2015.

[KW04] Tim Kelly and Rob Weaver. The Goal Structuring Notation – A safety argument notation. In *Proceedings of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[LAN$^+$15] Asier Larrucea, Irune Agirre, Carlos Fernando Nicolas, Jon Perez, Mikel Azkarate-Askasua, and Ton Trapman. Temporal independence validation of an IEC 61508 compliant mixed-criticality system based on multi-core partitioning. In *2015 Forum on Specification and Design Languages (FDL)*, pages 1–8, September 14-16 2015.

[LPA$^+$15] Asier Larrucea, Jon Perez, Irune Agirre, Vicent Brocal, and Roman Obermaisser. A modular safety case for an IEC 61508 compliant generic hypervisor. In *18th Euromicro Conference on Digital Systems Design (DSD 2015)*, pages 571–574, 8 2015.

[LPN$^+$16] Asier Larrucea, Jon Perez, Carlos F. Nicolas, Hamidreza Ahmadian, and Roman Obermaisser. A realistic approach to a network-on-chip cross-domain pattern. In *19th Euromicro Conference on Digital System Design (DSD 2016)*, pages 396–403, 10 2016.

[LPO15] Asier Larrucea, Jon Perez, and Roman Obermaisser. A modular safety case for an IEC 61508 compliant COTS multi-core device. page 8, October 2015.

[MRC11] Miguel Masmano, Ismael Ripoll, and Alfons Crespo. XtratuM hypervisor for LEON3. Report, XtratuM, 2 2011.

[MSM$^+$11] Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 374–385, New York, NY, USA, 2011. ACM.

[NAS15] NASA. Airspace operations and safety program, 2015.

[OBM$^+$14] André L. de Oliveira, Rosana T. V. Braga, Paulo C. Masiero, Yiannis Papadopoulos, Ibrahim Habli, and Tim Kelly. A model-based approach to support the automatic safety analysis of multiple product line products.

In *Proceedings of the 2014 Brazilian Symposium on Computing Systems Engineering*, SBESC '14, pages 7–12, Washington, DC, USA, 2014. IEEE Computer Society.

[OCM16] OCM. Structured assurance case metamodels (SACM), 2016.

[OESHK08] Roman Obermaisser, Christian El Salloum, Bernhard Huber, and Hermann Kopetz. The time-triggered system-on-a-chip architecture. In *2008 IEEE International Symposium on Industrial Electronics*, pages 1941–1947, 6 2008.

[OKG04] Roman Obermaisser, Hermann Kopetz, and Kastner Gruppe. *An Integrated Architecture for Event-Triggered and Time-Triggered Control Paradigms.* PhD thesis, 2004.

[Ona17] Peio Onaindia. *Design patterns for mix-criticality applications.* Master, 2017.

[PGN+14] Jon Perez, David Gonzalez, Carlos Fernando Nicolas, Ton Trapman, and Jose Miguel Garate. A safety certification strategy for IEC 61508 compliant industrial mixed-criticality systems based on multi-core partitioning. In *2014 17th Euromicro Conference on Digital System Design*, pages 394–400, 8 2014.

[PGT+13] Jon Perez, David Gonzalez, Salvador Trujillo, Anton Trapman, and Alfonso Garate. A safety concept for a wind power mixed-criticality embedded system based on multi-core partitioning. In *Proceedings of 11th International Symposium on Functional Safety in Industrial Applications*, page 9, 2013.

[PGTT15] Jon Perez, David Gonzalez, Salvador Trujillo, and Anton Trapman. *A safety concept for an IEC 61508 compliant fail-safe wind power mixed-criticality embedded system based on multi-core partitioning*, volume 9111, pages 3–17. Springer International Publishing, 2015.

[PHZ+14] Dave Prochnow, Laura Hilton, Anita Zabek, Mike Willoughby, and Cindy Harrison. Systems of systems and product line best practices from the DoD modeling and simulation industry, 9 2014.

[PQnCV09] Marco Paolieri, Eduardo Quiñones, Francisco J. Cazorla, and Mateo Valero. An analyzable memory controller for hard real-time cmps. *IEEE Embedded Systems Letters*, 1(4):86–90, 2009.

[PRO14] FP7 PROXIMA. PROXIMA - Probabilistic real-time control of mixed-criticality multi-core and manycore systems, 2014.

[PvdH06] Michael P. Papazoglou and Willem-Jan van den Heuvel. Service-ortiented design and development methodology. *Web Engineering and Technology (IJWET), International Journal of*, 2(4):412–442, 2006.

[QNX15] QNX. QNX hypervisor, 2015.

[RGG$^+$12] Petar Radojkovic, Sylbain Girbal, Arnaud Grasset, Eduardo Quiñones, Sami Yehia, and Francisco J. Cazorla. On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments. *CM Transactions on Architecture and Code Optimization (TACO) - HIPEAC Papers*, 8(4), 2012.

[Riv15] Wind River. Wind River hypervisor, 2015.

[RTC92] RTCA. *DO-178B Software Considerations in Airbone Systems and Equipment Certification*. RTCA, 12 1992.

[RTC11] RTCA. *DO-178C, Software Considerations in Airborne Systems and Equipment Certification*, 2011.

[Rub95] Barry Rubel. Pattern languages of program design, 1995.

[Rus99] John Rushby. Partitioning in avionics architectures: Requirements, mechanisms and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, 3 1999. Also to be issued by the FAA.

[SAS$^+$08] Erwin Schoitsch, Egbert Althammer, Gerald Sonneck, Henrik Eriksson, and Jonny Vinter. Support for modular certification of safety-critical embedded systems in DECOS – The generic safety case. In *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pages 258–263, 7 2008.

[SD11] Wilfried Steiner and Bruno Dutertre. Automated formal verification of the TTEthernet synchronization quality. In *Proceedings of the Third International Conference on NASA Formal Methods*, pages 375–390, 4 2011.

[Sem10] Freescale Semiconductor. P4080 development system user's guide. Report, Freescale Semiconductor, 8 2010.

[Sem13] Freescale Semiconductor. Chip errata for the i.MX 6Dual/6Quad, 2013.

[SHK14] Hardik Shah, Kai Huang, and Alois Knoll. Timing anomalies in multi-core architectures due to the interference on the shared resources. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 708–713, 2014.

[Sol13] Fent Innovative Software Solutions. XtratuM hypervisor for INTEL x86. Report, XtratuM, 10 2013.

[Sol14] Fent Innovative Software Solutions. XtratuM, 2014.

[SVM01] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of Computers*, 18(6):23–33, 2001.

[SYS15] SYSGO. PikeOS hypervisor, 2015.

[TOG+14] Salvador Trujillo, Roman Obermaisser, Kim Gruttner, Francisco J. Cazorla, and Jon Perez. European project cluster on mixed-criticality systems. In *DATE 2014 Workshop, Perfomance, Power and Predictable of Many-Core Embedded Systems (3PMCES)*, page 6, 3 2014.

[Tou58] Stephen Edelston Toulmin. *The Use of Argument*. Number 241. Cambridge University Press, 1958.

[Tou14] Stephen Edelston Toulmin. The Toulmin method. Technical report, University of Cambridge, 2014.

[TTT15] TTTech. TTTE Tools - The software tools for developing a TTEthernet network, 2015.

[USE12] TIMMO-2 USE. Timmo-2 Use – Methodology description V2. Web page, ITEA2, 7 2012.

[WESK10] Armin Wasicek, Christian El Salloum, and Hermann Kopetz. A system on a chip platform for mixed-criticality applications. In *2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 210–216, Carmona, Seville, 2010.

[XIL13] XILINX. 7 series isolation design flow lab using ISE design suite 14.4. Report, XILINX, 7 2013.

[XIL14a] XILINX. Soft error mitigation controller v4.1: LogicCORE IP product guide, 11 2014.

[XIL14b] XILINX. Xilinx all programmable functional safety design flow solution. Report, XILINX, 7 2014.

[XIL14c] XILINX. Zynq-7000 All Programmable SoC: Technical Reference Manual. Report, XILINX, 9 2014.

[XIL14d] XILLYBUS. Getting started with Xillinux for Zynq-7000 EPP, 3 2014.

[XIL15] XILINX. Zynq-7000 All Programmable SoC Overview, 5 2015.

[XIL16a] XILINX. Vivado design tool. Technical report, XILINX, 2016.

[XIL16b] XILINX. Xilinx software development kit (SDK). Technical report, XILINX, 2016.

Appendices

Appendix A

Zynq-7000 COTS multi-core device – FMEA/FMECA/FMEDAs

The functional safety/failure analyses are usually applied to identify the causes of the failures and their effects (see Table B.6 of IEC 61508-2 [IEC10b]). A typical method to assess systematic failures is the use of FMECAs, while FMEAs are used to assess random failures. Those analyses aim to evaluate the reliability, safety and integrity of the system. FMECAs extend FMEAs by including the criticality analysis to chart the probability of the failure modes against the severity of their consequences. Tables A.1 and A.2 show the patterns for FMEAs and FMECAs which include the following elements:

- **FMEA:**
  - **Name:** A meaningful name to describe the system, subsystem or component to be analysed.
  - **Failure mode:** The manner or way by which a failure of the system, subsystem or component occurs.
  - **Failure cause:** Cause or sequence of causes that initiates a failure of the system, subsystem or component.
  - **Failure effect on:** Consequences of a failure of the system, subsystem or component.
  - **Detection:** The means of detection of the failure.
  - **DC:** Diagnostic coverage value.
    - * High (99%).

  - ∗ Medium (90%).
  - ∗ Low (60%).

- – **Failure rate (λ):** The frequency with which a system, subsystem or component fails, expressed in failures per unit of time.
  - ∗ λ_S: Failure rate of safe failures.
  - ∗ λ_D: Failure rate of dangerous failures.
  - ∗ λ_{DD}: Failure rate of dangerous detected failures.
  - ∗ λ_{DU}: Failure rate of dangerous undetected failures.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}

Table A.1: FMEA template.

– **FMECA:**

- – **Name:** A meaningful name to describe the system, subsystem or component that is being analysed.
- – **Failure mode:** The manner or way by which a failure of the system, subsystem or component occurs.
- – **Failure cause:** Cause or sequence of causes that initiates a failure of the system, subsystem or component.
- – **Failure effects without failure control measures:** Consequences of a failure of the system, subsystem or component.
- – **Without mitigation measures:** Information about the risk level scenario without mitigation measures, including:
  - ∗ *Probability (P):* Probability of the occurrence of the failure.
    - · 5: High (e.g., once per day).
    - · 4: Rather High (e.g., once per week).
    - · 3: Low (e.g., once per year).
    - · 2: Very Low (e.g., once in 10 years).
    - · 1: Improbable (e.g., once in the millennium).
  - ∗ *Severity (S):* The worst potential consequence of the failure.
    - · 5: Loss of safety function.
    - · 4: Safety function is still available, although a failure occurs.

- 3: Safety function is available, although the system, subsystem or component is unusable.
- 2: Safety function is available, although safety system, subsystem or component is affected.
- 1: No effect on safety function.

* *Detection (D):* The means of detection of the failure.
  - 5: None (D $<$ 20%).
  - 4: Low (20% $\leq$ D $<$ 60%). DC = Low (60%).
  - 3: Low (60% $\leq$ D $<$ 90%). DC = Medium (90%).
  - 2: Medium (90% $\leq$ D $\geq$ 99%). DC = High (99%).
  - 1: High (D $>$ 99%).
* *Risk Priority Number (RPN):* Probability (P) x Severity (S) x Detection (D).

- **Safety Measure:** Mitigation measures to improve or justify risk.
- **With mitigation measures:** Information about the risk level scenario with mitigation measures, including:
  * *Probability (P):* Probability of the occurrence of the failure.
    - 5: High (e.g., once per day).
    - 4: Rather High (e.g., once per week).
    - 3: Low (e.g., once per year).
    - 2: Very Low (e.g., once in 10 years).
    - 1: Improbable (e.g., once in the millennium).
  * *Severity (S):* The worst potential consequence of the failure.
    - 5: Loss of safety function.
    - 4: Safety function is still available, although a failure occurs.
    - 3: Safety function is available, although the system, subsystem or component is unusable.
    - 2: Safety function is available, although safety system, subsystem or component is affected.
    - 1: No effect on safety function.
  * *Detection (D):* The means of detection of the failure.
    - 5: None (D $<$ 20%).
    - 4: Low (20% $\leq$ D $<$ 60%). DC = Low (60%).
    - 3: Low (60% $\leq$ D $<$ 90%). DC = Medium (90%).
    - 2: Medium (90% $\leq$ D $\geq$ 99%). DC = High (99%).
    - 1: High (D $>$ 99%).

∗ *Risk Priority Number (RPN):* Probability (P) x Severity (S) x Detection (D).

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	With mitigation measures			
				P	S	D	RPN		P	S	D	RPN

Table A.2: FMECA template.

This annex defines the FMEAs and FMECAs to assess the random and systematic faults of safety techniques defined in the MSC for an IEC 61508 compliant generic COTS device (see Section 5.3) and remarkable components of the Zynq-7000 device. It is assumed that the estimations of the *Failure Rates* of the FMEAs and the *Risk Priority Numbers* of the FMECAs are provided as part of the *device manufacturer documentation.*

A.1 Safety Requirements

A.1.1 Power Up

The *power Up* requirement is defined in Section 5.3. The FMEA of this safety-related requirement is defined in Table A.4 and its FMECA in Table A.27.

A.1.2 Boot

The *boot* requirement is defined in Section 5.3. The FMEA of of this safety-related requirement is defined in Table A.6 and its FMECA in Table A.28.

A.1.3 Reset

The *Reset* requirement is defined in Section 5.3. The FMEA of of this safety-related requirement is defined in Table A.8 and its FMECA in Table A.29.

A.1.4 Power Down

The *power down* requirement is defined in Section 5.3. The FMEA of this safety-related requirement is defined in Table A.10 and its FMECA in Table A.30.

A.1.5 Shutdown

The *shutdown* requirement is defined in Section 5.3. The FMEA of this safety-related requirement is defined in Table A.12 and its FMECA in Table A.31.

A.1.6 Resource Virtualization

The *resource virtualization* requirement is defined in Section 5.3. The FMEA of this safety-related requirement is defined in Table A.14 and its FMECA in Table A.32.

A.1.7 Exclusive Access to Peripherals

The *exclusive access to peripherals* requirement is defined in Section 5.3. The FMEA of this safety-related requirement is defined in Table A.16 and its FMECA in Table A.33.

A.1.8 Temporal Independence

The *temporal independence* requirement is defined in Section 5.3. The FMEA of this safety-related requirement is defined in Table A.18 and its FMECA in Table A.34.

A.1.9 Spatial Independence

The *spatial independence* requirement is defined in Section 5.3. The FMEA of this safety-related requirement is defined in Table A.20 and its FMECA in Table A.35.

A.1.10 Configuration

The *configuration* requirement is defined in Section 5.3. The FMEA of this safety-related requirement is defined in Table A.22 and its FMECA in Table A.36.

A.1.11 Measures and Diagnostic Techniques

The *measures and diagnostic techniques* requirement is defined in Section 5.3. The FMEA of this safety-related requirement is defined in Table A.24 and its FMECA in Table A.37.

A.1.12 System Reactions to Errors

The *system reactions to errors* requirement is defined in Section 5.3. The FMEA of this safety-related requirement is defined in Table A.26 and its FMECA in Table A.38.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Power up ramp time and/or sequence	Incorrect power up ramp time and/or sequence	Line regulator damaged	Unpredictable function of the device. I/Os do not function as designed.	External power supply supervisor and WDT (see Tables A.9 and A.10 of IEC 61508-2).	99%	–	–	–	–	–
Over or under voltage during the power up sequence	Over/under voltage	Line regulator damaged	Unpredictable function of the device. Incorrect functioning of I/Os.	External power supply supervisor and WDT (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–
Short circuit	Short circuit	Line regulator damaged	Permanent damage to the device might be caused.	Over current protection (see Table A.9 of IEC 61508-2).	60%	–	–	–	–	–
Supply voltage over the specified operational range	Over voltage of power supply	Supply voltage damaged	Permanent damage to the device.	External power supply supervisor (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–
Supply voltage below the specified operational range	Under voltage of power supply	Supply voltage damaged	Configuration errors or sampling of wrong data.	External power supply supervisor (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–
Loss of power supply	Loss of power supply	Power supply damaged	Applications running on the device are no longer in execution.	External power supply supervisor (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–
Excessive power consumption	Excessive power consumption	Line regulator damaged	Unpredictable function of the device. Incorrect functioning of I/Os.	External power supply supervisor (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–
Power supply below to the brown-out voltage level	Power supply below to the brown-out voltage level	Power supply drops below the nominal value	Unknown and unpredictable behaviour of the device.	External power supply supervisor (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–
Time-out	Time-out of the power up sequence	Line regulator damaged	The device is not powered up in the optimal and expected manner.	External WDT (see Table A.10 of IEC 61508-2).	99%	–	–	–	–	–

TABLE A.4: Power Up – FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Boot mode	Incorrect boot mode (Secure or Non-Secure)	Incorrect sampling of boot mode	The device is locked-down.	Read-back of boot mode register at boot time.	90%	–	–	–	–	–
Power up time-out	Power up time-out	See Table A.4.	Incorrect boot of the device	External WDT (see Table A.10 of IEC 61508-2).	60%	–	–	–	–	–
Boot files	Faulty boot files	Faulty BootROM Header, FSBL or User code.	The device is locked-down.	CRC mechanism (see Table A.5 of IEC 61508-2).	60%	–	–	–	–	–
Boot time-out	Time-out	Power up time-out, Faulty boot file, Faulty boot loader.	The device is locked-down.	External WDT (see Table A.10 of IEC 61508-2).	99%	–	–	–	–	–

Table A.6: Boot – FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Reset signal assertion	Unexpected reset assertion	Violation of secure lock-down window.	The device is reset.	Periodic read-back of reset registers.	90%	–	–	–	–	–

Table A.8: Reset – FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Power up ramp time or sequence	Incorrect power up ramp time or sequence	Line regulator damaged	Unpredictable function of the device. Incorrect functioning of I/Os.	External power supply supervisor (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–
Over or under voltage during the power up sequence	Over/under voltage	Line regulator damaged.	Unpredictable function of the device. Incorrect functioning of I/Os.	External power supply supervisor (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–
Short circuit	Short circuit	Line regulator damaged	Permanent device damage.	Over current protection (see Table A.9 of IEC 61508-2).	60%	–	–	–	–	–
Excessive power consumption	Excessive power consumption	Line regulator damaged	Increase of the internal temperature.	External power supply supervisor (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–
Power supply below to the brown-out voltage level	Power supply below to the brown-out voltage level	Power supply drops below the nominal value.	Unknown and unpredictable behaviour of the device.	External power supply supervisor (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–
Time-out	Time-out of the power up sequence	Line regulator damaged	The device is not powered up in the optimal/expected manner.	External WDT (see Table A.10 of IEC 61508-2).	99%	–	–	–	–	–

Table A.10: Power Down – FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Incomplete shutdown	Incomplete shutdown	Random failure	Corruption of the file system and issues with invariable memory.	External power supply supervisor (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–
No shutdown	No shutdown	Random failure	Corruption of the file system and issues with invariable memory.	External power supply supervisor (see Table A.9 of IEC 61508-2).	99%	–	–	–	–	–

Table A.12: Shutdown – FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Virtualization failure	Virtualization failure	Virtual to physical address translation failure on memory virtualization, Hypervisor failure or GIC failure.	Memory virtualization, CPU virtualization or Interrupt virtualization.	Read-back of virtual memory control registers, Diagnosis technique of Hypervisor and Read-back configuration registers of GIC v2.0.	90%	–	–	–	–	–

Table A.14: Resource Virtualization – FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Failure of exclusive access assignment	Exclusive access assignment failure	Register value modification	System failure	Read-back configuration registers	90%	–	–	–	–	–

Table A.16: Exclusive access to peripherals – FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Temporal interferences	Temporal interferences	Failure of some component of the device (e.g., GIC or shared memory). Simultaneous access to memory. Erroneous time scheduling.	System failure	External WDT (see Table A.10 of IEC 61508-2).	99%	–	–	–	–	–

Table A.18: Temporal Independence – FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Partial spatial independence	Partial spatial independence	No enough mechanisms or tools are provided to ensure the spatial independence. Incorrect or unexpected behaviour of the mechanisms or tools which provide the spatial independence.	System failure	EDC mechanism (see Table A.6 of IEC 61508-2).	90%	–	–	–	–	–
In-existent spatial independence	In-existent spatial independence	A momentary failure of mechanisms that provide spatial independence.	System failure	N/A.	–	–	–	–	–	–

TABLE A.20: Spatial Independence – FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Configuration Failure	Incorrect configuration	Random HW failure.	PS failure, PL failure, System failure or Component failure.	Periodic read-back of configured registers.	90%	–	–	–	–	–
No configuration	No configuration	A momentary failure of mechanisms that provide spatial independence.	PS failure, PL failure or System failure.	Periodic read-back of configured registers.	90%	–	–	–	–	–

TABLE A.22: Configuration – FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Incomplete diagnosis execution	Incomplete diagnosis execution	Component is disabled suddenly. Unexpected data value change. System failure.	Component is not diagnosed completely.	Periodic read-back of diagnosis execution results. External WDT (see Table A.10 of IEC 61508-2).	60%	–	–	–	–	–
No Diagnosis	No Diagnosis technique cannot be executed	Component is disabled or System is powered down.	Component is not diagnosed on the system.	Periodic read-back of diagnosis execution results. External WDT (see Table A.10 of IEC 61508-2).	60%	–	–	–	–	–

TABLE A.24: Measures and Diagnostic Techniques – FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Incomplete system reaction to error	Incomplete system reaction to error	Component is disabled suddenly, Unexpected data value change or System failure.	System failure.	See Diagnostic techniques in Table A.24.	N/A.	–	–	–	–	–
No reaction to error	The system reaction to error cannot be executed	System is powered down.	System failure.	See Diagnostic techniques in Table A.24.	N/A.	–	–	–	–	–

TABLE A.26: System reactions to errors – FMEA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Over temperature	Temperature range exceeded	Break or ageing or incorrect use	System failure	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–
		Excessive power consumption		–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and measures against the physical environment and measures to detect breaks and shorts in signal lines and failure detection by on-line monitoring (see Table A.16 of IEC 61508-2).	–	–	–	–
Power up	Incorrect power up sequence	Break or ageing or incorrect use	Power up failure. System failure.	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–

TABLE A.27: Power Up – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Boot loader	Faulty boot loader (e.g., SD, RAM)	HW break caused by ageing or incorrect use	The device is not booted.	–	–	–	–	HW diversity (see Table A.15, IEC 61508-2).	–	–	–	–

Table A.28: Boot – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Reset activation	Unpredictable HW reset activation	Reset switch break due to ageing or incorrect use.	System reset	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–
		Break of reset line due to ageing	System reset	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–
		Environmental or external influences	System reset	–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and measures against the physical environment and measures to detect breaks and shorts in signal lines and failure detection by on-line monitoring (see Table A.16 of IEC 61508-2).	–	–	–	–
No reset	HW reset is not activated	Reset switch break due to ageing or incorrect use.	System failure	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–
		Break of reset line due to ageing or incorrect use.	System failure	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–

Table A.29: Reset – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Power Down	Power down sequence failure	Break or ageing or incorrect use	System power down failure	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–

TABLE A.30: Power Down – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Shut Down	Shut down failure	Break or ageing or incorrect use	Incorrect or non shutdown of the system.	–	–	–	–	Program sequence monitoring (see Table A.15 of IEC 61508-2). Failure detection by online monitoring (see Table A.15 of IEC 61508-2).	–	–	–	–

TABLE A.31: Shutdown – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
No Virtualization	Incorrect or in-existent resource virtualization	HW break due to ageing or incorrect use	Virtualization failure	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture or Diverse HW (see Table A.15 of IEC 61508-2).	–	–	–	–

		Environmental causes or external influences	Virtualization failure	–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–
		Operational failures	Virtualization failure	–	–	–	–	Modification protection and Failure detection by on-line monitoring or failure assertion programming (see Table A.17 of IEC 61508-2).	–	–	–	–

Table A.32: Resource Virtualization – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
No Exclusive Access	Incorrect or in-existent access	HW break due to ageing or incorrect use	Exclusive access failure	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture or Diverse HW (see Table A.15 of IEC 61508-2).	–	–	–	–
		Environmental causes or external influences	Exclusive access failure	–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–
		Operational failures	Exclusive access failure	–	–	–	–	Modification protection and Failure detection by on-line monitoring or failure assertion programming (see Table A.17 of IEC 61508-2).	–	–	–	–

Table A.33: Exclusive access to peripherals – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Temporal interferences	Temporal independence	HW break due to ageing or incorrect use	Temporal interferences (e.g., delays)	–	–	–	–	Program sequence monitoring (see Table A.15 of IEC 61508-2).	–	–	–	–
				–	–	–	–	Failure detection by on-line monitoring (see Table A.15 of IEC 61508-2).	–	–	–	–
		Environmental causes or external influences	Exclusive access failure	–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–

Table A.34: Temporal Independence – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Design failure	Partial or in-existent spatial independence	The HW is not designed to provide spatial independence.	System failure	–	–	–	–	Diverse HW (see Table A.15 of IEC 61508-2).	–	–	–	–
Mechanism(s) or Tool(s) failure	Partial or independent spatial independence.	Break or ageing of mechanisms/tools of spatial independence.	System failure	–	–	–	–	Diverse HW (see Table A.15 of IEC 61508-2).	–	–	–	–

Table A.35: Spatial Independence – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Configuration device	Fault configuration device	SEUs	System is not configured	–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and measures against the physical environment and measures to detect breaks and shorts in signal lines and failure detection by on-line monitoring (see Table A.16 of IEC 61508-2).	–	–	–	–
		Break or ageing of the device.	System is not configured.	–	–	–	–	Failure detection by on-line monitoring or Diverse HW (see Table A.15 of IEC 61508-2).	–	–	–	–
Configuration file	Fault configuration file	SEUs	System failure	–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and measures against the physical environment and measures to detect breaks and shorts in signal lines and failure detection by on-line monitoring (see Table A.16 of IEC 61508-2).	–	–	–	–
		Operational failure	System failure.	–	–	–	–	Modification protection (see Table A.17 of IEC 61508-2).	–	–	–	–
				–	–	–	–	Failure detection by on-line monitoring.	–	–	–	–
SEUs	External influences	SEUs	Electronic units and memory failure	–	–	–	–	SEM IP	–	–	–	–
				–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and measures against the physical environment and measures to detect breaks and shorts in signal lines and failure detection by on-line monitoring (see Table A.16 of IEC 61508-2).	–	–	–	–

				–	–	–	–	Failure detection by on-line monitoring or Diverse HW (see Table A.15 of IEC 61508-2).	–	–	–	–

Table A.36: Configuration – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Incomplete diagnosis execution	Incomplete diagnosis	HW failure	The components or system cannot be completely assessed.	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture or Diverse HW (see Table A.15 of IEC 61508-2).	–	–	–	–
		External influence	The components or system cannot be completely assessed.	–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–
		Operational failure	The components or system cannot be completely assessed.	–	–	–	–	Modification protection and Failure detection by on-line monitoring or failure assertion programming. see Table A.17 of IEC 61508-2).	–	–	–	–
No diagnosis	No diagnosis	HW random failure	The components or system cannot be completely assessed.	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture or Diverse HW (see Table A.15 of IEC 61508-2).	–	–	–	–

		External influence	The components or system cannot be completely assessed.	–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–
		Operational failure	The components or system cannot be completely assessed.	–	–	–	–	Modification protection and Failure detection by on-line monitoring or failure assertion programming. see Table A.17 of IEC 61508-2).	–	–	–	–

Table A.37: Measures / Diagnostic Techniques – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Incomplete system reactions to errors	Incomplete system reactions to errors	HW failure	The system is under continuous failure.	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture or Diverse HW (see Table A.15 of IEC 61508-2).	–	–	–	–
		External influence	The system is under continuous failure.	–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–
		Operational failure	The system is under continuous failure.	–	–	–	–	Modification protection and Failure detection by on-line monitoring or failure assertion programming (see Table A.17 of IEC 61508-2).	–	–	–	–
No system reactions to errors	No system reactions to errors	HW random failure	The system is under continuous failure.	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture or Diverse HW (see Table A.15 of IEC 61508-2).	–	–	–	–

		External influence	The system is under continuous failure.	–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–
		Operational failure	The system is under continuous failure.	–	–	–	–	Modification protection and Failure detection by on-line monitoring or failure assertion programming (see Table A.17 of IEC 61508-2).	–	–	–	–

TABLE A.38: System reactions to errors – FMECA.

A.2 Remarkable components of the Zynq-7000 device

A.2.1 Processing Unit

The *processing unit* of the Zynq device is defined in Subsection 7.2.3. The FMEA of this component is defined in Table A.4 and its FMECA in Table A.44.

A.2.2 Coherency Management Unit

Subsection 6.2.2 analyses the *coherency management unit* implemented by the Zynq-7000 multi-core device for managing the coherency between the CPUs and the memories. The FMEA analysis of this component is defined in Table A.39 and its FMECA in Table A.46.

A.2.3 Data Paths

The *data paths* of the Zynq device are defined in Subsection 7.2.3. The FMEA of this component is defined in Table A.40 and its FMECA in Table A.47.

A.2.4 Interrupt Controller

The Zynq-7000 device supports a GIC for managing the prioritisation of the tasks executed in the processor cores. This interrupt controller has been introduced in Subsection 6.2.4. The FMEA analysis of the generic interrupt controller is defined in Tables A.41 and A.42 and its FMECA in Table A.48.

A.2.5 Memory Areas and Registers

The FMEA of the *Memory Areas and Registers* component is defined in Table A.43 and its FMECA in Table A.45.

Name	Registers	Failure Mode	Failure Cause	Failure effect on			Detection	DC	Failure rate				
				SCU	CPU	PL			λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Configuration Registers	SCU Control Register^I, SCU CPU Power Status^{II}, SCU Invalidate all register in Secure state^{III} (WO), Filtering Start Addresses^{IV}, Filtering End Addresses^V, SCU Access Control (SAC)^{VI} and Non Secure SCU Access Control (NSAC)^{VII}	Data or Address modification	Modification of SCU's registers	IC field is set to '1'. SCU is reached to standby mode^I.	CPU cannot perform a read / write request.	PL cannot perform a read / write request.	Periodic readback of configuration registers.	90%	–	–	–	–	–
				IC field is set to '0', the SCU is activated^I.	CPU(s) can perform a write/read request.	PL can perform a write/read request.			–	–	–	–	–
				Bit [5] of ^{I} is set. The SCU is stopped^I. [5]	CPUs cannot access to PL, OCM, L2 cache, DDR.^{I} [5]	PL cannot access to L2 cache and CPU(s).			–	–	–	–	–
				SCU is launched^I. [5]	CPU can communicate with the PL, OCM, L2 cache.	PL can communicate with the CPUx, OCM.			–	–	–	–	–
				Value of field [4] is set to '1'^{I}. [4]	All requests from CPUs are forced to be issued on the master port 0.	All requests from PL are forced to be issued on the master Port 0.			–	–	–	–	–
				Value of field [4] is set to '0'^{I}. [4]	All requests from CPUs are forced to be issued on the master ports M0 and M1.	All requests from PL are forced to be issued on the master ports M0 and M1.			–	–	–	–	–

				Bit [1] of ^{I} is set to '0'^{I}. [1]	All requests from CPUs are forced to be issued on the master port M0.	All requests from PL are forced to be issued on the master port M0.			–	–	–	–	–
				Bit [1] of ^{I} is set to '1'^{I}. [1]	All requests from CPUs are forced to be issued on the master ports M0 and M1.	All requests from PL are forced to be issued on the master ports M0 and M1.			–	–	–	–	–
				SCU is disabled^{I}. [0]	CPU(s) cannot access to PL, cache or memory.	PL cannot access to CPUs and memories.			–	–	–	–	–
				SCU is enabled. Bit [0] is set^{I}. [0]	CPU(s) can access to PL, caches.	PL can access to processors, memory.			–	–	–	–	–
				Value of register ^{II} is set to b00.	The power state of CPUx changes to normal operation mode.	–			–	–	–	–	–
				Value of register ^{II} is set to b10.	The power state of CPUx changes to dormant mode. No coherency request is sent to the CPUx.	–			–	–	–	–	–
				Value of register ^{II} is set to b11.	The power state of CPUx changes to powered-off. No coherency request is sent to the CPUx.	–			–	–	–	–	–

				Invalidates specified ways of CPUx. ^{III} [15:0] Some ways of CPUx are invalidated by the SCU.	–	–			–	–	–	–	–
				Start address for use with master port 1 in a two master port configuration is changed. ^{IV} [31:20]	–	–			–	–	–	–	–
				End address for use with master port 1 in a two master port configuration is changed.^{V} [31:20]	–	–			–	–	–	–	–
				SAC register's value is set to '0'. ^{VI} [1] [3:0]	CPUx cannot access the registers[1].	–			–	–	–	–	–
				SAC register's value is set to '1'. ^{VI}[1] [3:0]	CPUx can access the registers[1].	–			–	–	–	–	–
				Any bit from bit 11 to 8 of NSAC is set to '1'. ^{VII}	CPUx can access to the global timer in secure and non-secure modes.	–			–	–	–	–	–
				Any bit from bit 11 to 8 of NSAC is set to '0'. ^{VII}	CPUx can access only to the global timer in secure mode.	–			–	–	–	–	–

[1]Accessible/Inaccessible registers due to modification of SAC register are the SCU Control Register, SCU CPU Status Register, SCU Invalidate All Register in Secure State, filtering registers and SCU CPU Power Status register.

				Any bit from bit 7 to 4 of NSAC is set to '1'.	CPUx can access to the private timers and watchdogs in secure and non-secure modes.	–			–	–	–	–	–
				Any bit from bit 7 to 4 of NSAC is 0set to '0'.	CPUx can access only to the private timers and watchdogs in secure mode.	–			–	–	–	–	–
				Any bit from bit 3 to 0 of NSAC is set to '1'. ^{VII2}	CPUx can access the registers[2].	–			–	–	–	–	–
				Any bit from bit 3 to 0 of NSAC is set to '0'. ^{VII2}	CPUx cannot write the registers[2].	–			–	–	–	–	–
		No access to the registers of the SCU	Modification of the SCU's registers	SCU cannot be accessed or configured.	CPUx cannot communicate with PL, OCM and other components of the device.	PL cannot communicate with CPUx, memories and other components of the device.	Periodic read-back of configuration registers	90%	–	–	–	–	–
Reading registers SCU control register	SCU Configuration register, SCU CPU Power Status Register, Filtering Start and End Address register, SAC, NSAC	Inconclusive read values of register(s)	Modification of the SCU's registers.	–	–	–	Periodic read-back of configuration registers	90%	–	–	–	–	–

Table A.39: SCU – FMEA.

[2]Accessible/Inaccessible registers due to modification of NSAC register are SAC, SCU control, SCU CPU status, filtering and SCU CPU power status registers.

Name	Failure Mode	Failure Cause	Failure effect on			Detection	DC	Failure rate				
			SCU	CPU	PL			λ	λ_S	λ_D	λ_{DD}	λ_{DU}
CPU – SCU	Wrong data	Information modification	Intelligible or incomplete data is received	–	–	Inspection Using Test pattern (see Table A.8 of IEC 61508-2).	60%	–	–	–	–	–
						Transmission redundancy (see table A.8 of IEC 61508-2).	90%	–	–	–	–	–
	Wrong destination address	Address modification	The SCU does not receive any data.	The CPU cannot communicate with the SCU.	–	Inspection Using Test pattern (see Table A.8 of IEC 61508-2)	60%	–	–	–	–	–
	No transmission	Modification of the SCU's configuration registers.	The SCU is misconfigured	The CPU cannot communicate through the SCU.	The PL cannot communicate through the SCU.	Periodic read-back of configuration registers.	90%	–	–	–	–	–
AXI_ACP - SCU	Wrong data	Information modification	Intelligible or incomplete data is received	–	–	Inspection Using Test pattern (see Table A.8 of IEC 61508-2).	60%	–	–	–	–	–
						Transmission redundancy (see Table A.8 of IEC 61508-2).	90%	–	–	–	–	–
	Wrong destination address	Address modification	The SCU does not receive the expected data.	–	–	Inspection Using Test pattern (see Table A.8 of IEC 61508-2).	60%	–	–	–	–	–
	No transmission	Modification of the SCU's configuration registers.	The SCU is misconfigured	The CPU cannot communicate through the SCU.	The PL cannot communicate through the SCU.	Periodic read-back of configuration registers.	90%	–	–	–	–	–
SCU – L2 cache	Wrong data	Information modification	–	The L2 cache contains an intelligible or incomplete data.	–	Inspection Using Test pattern (see Table A.8 of IEC 61508-2).	60%	–	–	–	–	–
						Transmission redundancy (see Table A.8 of IEC 61508-2).	90%	–	–	–	–	–

	Wrong destination address	Address modification	–	The L2 cache stores data on different address memory.	–	Inspection Using Test pattern (see Table A.8 of IEC 61508-2).	60%	–	–	–	–	–
	No transmission	Modification of the SCU's configuration registers.	The SCU cannot access to the L2 cache.	The CPU cannot access to the L2 cache.	The PL cannot access to the L2 cache.	Periodic read-back of configuration registers (see Table A.8 of IEC 61508-2).	90%	–	–	–	–	–
SCU - OCM	Wrong data	Information modification	–	–	Data read from the PL is erroneous or incomplete.	Inspection Using Test pattern (see Table A.8 of IEC 61508-2).	60%	–	–	–	–	–
						Transmission redundancy (see Table A.8 of IEC 61508-2).	90%	–	–	–	–	–
	Wrong destination address	Address modification	–	–	The PL cannot does not find the expected data in the specified address.	Inspection Using Test pattern (see Table A.8 of IEC 61508-2).	60%	–	–	–	–	–
	No transmission	Modification of the SCU's configuration registers.	The SCU cannot access to the OCM.	The CPU cannot access to the OCM.	The PL cannot access to the OCM through the SCU.	Periodic read-back of configuration registers.	90%	–	–	–	–	–

Table A.40: Data Paths – FMEA.

Name	Registers	Failure Mode	Failure Cause	Failure effect on			Detection	DC	Failure rate				
				SCU	CPU	PL			λ	λ_S	λ_D	λ_{DD}	λ_{DU}

Configuration Registers	ICDDCR, ICDISR, ICDICPR, ICDICPR, ICDIPTR, ICDICFR, ICDSGIR[3],[4] ICDISER, ICDICER	Data or address modification	Modification of GIC's registers.	The bit '1' of register ICD-DRCR is set up to '1'. Hence, enables the distributor to update register locations for non-secure interrupts.	–	–	Periodic read-back of configuration registers.	90%	–	–	–	–	–
				The bit 1 of register ICDDRCR is set up to '0'. Hence, disables all non-secure interrupts control bits in the distributor.	–	–			–	–	–	–	–
				The bit 0 of register ICDDRCR is set up to '1'. Hence, enables the distributor to update register locations for secure interrupts.[5]	–	–			–	–	–	–	–
				The bit 0 of register ICDDRCR is set up to '0'. Hence, disables all secure interrupts control bits in the distributor.[5]	–	–			–	–	–	–	–

[3]Only writeable registers (WO).
[4]If the security extension of GIC is not enables, bit[15] of ICDSGIR is reserved.
[5]Registers accessible only in Secure mode. Otherwise can be accessed in non-secure and secure mode.

				Some bit of ICDISR is set to '0'. This means that corresponding interrupts are secure. [31:0] [5]	–	–		–	–	–	–	–
				Some bit of ICDISR is set to '1'. This means that corresponding interrupts are non-secure. [31:0] [5]	–	–		–	–	–	–	–
				Some bit of ICSIDER is set of to '0'. This means that interrupt X is disabled.	–	–		–	–	–	–	–
				Some bit of ICDISER is set to '1'. This way, enabling the corresponding interrupt.	–	–		–	–	–	–	–
				Incorrect interrupts are disabled. (ICDICER)	Interrupts allocated to CPUx are deactivated.	–		–	–	–	–	–

				The state of interrupt X is changed. If it is previously inactive, it changes to pending and if it is previously active, it changes to active and pending. (ICDICPR)	–	–			–	–	–	–	–
				None or incorrect de-assignment of the interrupts pending. (ICDICPRn)	The status of allocated interrupts to CPUx can be randomly changed.	–			–	–	–	–	–
				Incorrect or no assignment of interrupt priority. (ICDIPRn)	CPU's interrupts have incorrect priority assignment.	–			–	–	–	–	–
				Interrupt X is not assigned to CPU interface X. (ICDIPTRn)	–	–			–	–	–	–	–
				Interrupt X is assigned incorrectly to CPU interface X. (ICDIPTRn)	–	–			–	–	–	–	–
				Incorrect interrupt configuration (edge triggered or level sensitive). (ICDICFR)	CPUx's interrupt is configured with incorrect trigger type.	–		–	–	–	–	–	–

Name	Registers	Failure Mode	Failure Cause	Failure effect on SCU	Failure effect on CPU	Failure effect on PL	Detection	DC	Failure rate λ	λ_S	λ_D	λ_{DD}	λ_{DU}
				Incorrect assignment of interrupt X to CPU interfaces. (ICDSGIR)^{III}	–	–		–	–	–	–	–	–
				CPUx does not have assigned any interrupt. (ICDSGIR)^{III}	–	–		–	–	–	–	–	–
				An interrupt is assigned to an incorrect CPU interface. (ICDSGIR)^{III}	CPUx has assigned an incorrect interrupt.	–		–	–	–	–	–	–
				The SGI interrupt X is not assigned to any CPU. (ICDSGIR)^{III}	CPUx has not any SGI interrupt assigned.	–		–	–	–	–	–	–
				Incorrect interrupts are assigned to CPUx. (ICDSGIR)^{III}	CPUx has assigned one or more incorrect interrupts.	–		–	–	–	–	–	–
		No access to the registers of the SCU	Modification of the SCU's registers.	GIC cannot be configured or is not accessible	CPU(s) does not have any interrupt assigned.	–	Periodic read-back of configuration registers	90%	–	–	–	–	–
Reading registers	ICDICTR, ICABRn, ICPPISRn, ICSPISRn, ICPIDR [0-7], ICCIDR [0-3]	Inconclusive read values of register(s)	Modification of the SCU's registers.	–	–	–	Periodic read-back of configuration registers	90%	–	–	–	–	–

Table A.41: GIC Distributor– FMEA.

Configuration Registers	ICCIR (0NS, 4S, 3S, 2S, 1S, 0S), ICCPMR (S), ICCBPR (S), ICCEOIR*1	Data or address modification	Modification of GIC's registers	None of the interrupts is signalled by the CPU interface to the connected processor. (ICCIR) [0]	CPUx does not have any interrupt assigned.	–	Periodic read-back of configuration registers.	90%	–	–	–	–	–
				All interrupts are signalled by the CPU interface to the connected processor. (ICCIR) [0]	All CPUs have at least an interrupt assigned.	–			–	–	–	–	–
				All interrupts are signalled by the CPU interface to the connected processor. (ICCIR) [0]	All CPUs have at least an interrupt assigned.	–			–	–	–	–	–
				Incorrect assignments of the binary point register to CPU interface X. (ICCIR) [4]	–	–			–	–	–	–	–
				None assignment of the binary point register to CPU interface X. (ICCIR) [4]	–	–			–	–	–	–	–

				None assignment of the secure interrupts to a target processor using signals IRQ or FIQ.(ICCIR) [3] –	–			–	–	–	–	–	
				Incorrect assignment of the secure interrupts to a target processor using signals IRQ or FIQ.[6](ICCIR) [3]	The secure interrupts are signalled using an incorrect interrupt type (FIQ, instead of IRQ and vice versa).	–			–	–	–	–	–
				Interrupt status is not refreshed[6]. (ICCIR) [2]	CPU's tasks are locked.	–			–	–	–	–	–
				All non-secure interrupts are signalled[6]. (ICCIR) [1]	–	–			–	–	–	–	–
				Non-secure interrupts are not signalled[6]. (ICCIR) [1]	–	–			–	–	–	–	–
				All secure interrupts are signalled[6]. [ICCIR) [0]	–	–			–	–	–	–	–
				Secure interrupts are not signalled[6]. (ICCIR) [0]	CPUx does not have assigned any secure interrupt.	–			–	–	–	–	–

[6]Registers accessible only in Secure mode. Otherwise can be accessed in non-secure and secure mode.

				The priority mask of CPU interface X is assigned erroneously[6]. (ICCPMR).	–	–			–	–	–	–	–
				The interrupt X is assigned to an incorrect CPU. (ICCEOIR) [12:10]	CPUx has an incorrect assignment of interrupt X.	–			–	–	–	–	–
		No access to the registers of the SCU	Modification of the SCU's registers.	GIC cannot be configured or is not accessible.	CPU(s) does not have any interrupt assigned.	–	Periodic read-back of configuration registers	90%	–	–	–	–	–
Reading Registers	ICCIAR, ICCRPR, ICCHIPR, ICCIDR	Inconclusive read values of registers.	Modification of the SCU's registers.	–	–	–	Periodic read-back of configuration registers.	90%	–	–	–	–	–

Table A.42: GIC Interrupt Interface– FMEA.

Name	Failure Mode	Failure Cause	Failure effect on	Detection	DC	Failure Rate				
						λ	λ_S	λ_D	λ_{DD}	λ_{DU}
Data modification	Register data modification	Change of information caused by soft-errors	System failure	Periodic read-back of register values. EDC (see Table A.6 of IEC 61508-2). Information redundancy (see Table A.8 of IEC 61508-2).	90%	–	–	–	–	–
No access	No access to the registers	No, wrong or multiple addressing or Wrong address decoding or Wrong access time or Time out.	System failure	Periodical software test. External watchdog timer (see Table A.10 of IEC 61508-2).	60%	–	–	–	–	–
Read	Inconclusive read values of registers	No, wrong or multiple addressing.	–	Periodical software test. CRC (see Table A.7 of IEC 61508-2).	60%	–	–	–	–	–

Table A.43: Memory and Register areas – FMEA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Boot	Unable to boot	Breaking or ageing of HW	PS failure. The PL can work normally.	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–
Malfunction	Malfunction	Breaking or ageing of HW	PS failure. The PL can work normally.	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–
		External influence	PS failure. The PL can work normally.	–	–	–	–	Modification protection and Failure detection by on-line monitoring or failure assertion programming (see Table A.17 of IEC 61508-2).	–	–	–	–
		Operational failure	PS failure. The PL can work normally.	–	–	–	–	Modification protection and Failure detection by on-line monitoring or failure assertion programming (see Table A.17 of IEC 61508-2).	–	–	–	–
No function	No function	Break or ageing of HW	PS failure. The PL can work normally.	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–

Table A.44: Processing Unit – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Memory break	Memory break	Break due to ageing or incorrect use.	Possible system failure	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–

		Environmental or external influences	Possible system failure	–	–	–	–	Measures against voltage breakdown and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–

Table A.45: Memory – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
SCU is broken	SCU stops working correctly	Deterioration of the SCU	System failure	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–
		Failure of the PS.	System failure	–	–	–	–	Measures against voltage break down and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–
		Power Supply failure.	System failure	–	–	–	–		–	–	–	–

Table A.46: SCU – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
Data Paths	Failure of Data Paths	Deterioration of the data paths.	System failure, PL failure PS failure or Component failure	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–

		External influence	System failure, PL failure, PS failure Component failure	–	–	–	–	Measures against voltage break down and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–

Table A.47: Data Paths – FMECA.

Name	Failure Mode	Failure Cause	Consequence without failure control measures	without mitigation measures				Safety integrity measure	with mitigation measures			
				P	S	D	RPN		P	S	D	RPN
GIC is broken	GIC stops working	Deterioration of the GIC.	System failure	–	–	–	–	Failure detection by on-line monitoring or Standard test access port and boundary-scan architecture (see Table A.15 of IEC 61508-2).	–	–	–	–
		Failure of the PS.	System failure	–	–	–	–	Measures against voltage break down and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–
		Power Supply failure	System failure	–	–	–	–	Measures against voltage break down and Separation of electrical energy lines from information lines and Increase of interference immunity and Measures against physical environment and Measure to detect breaks and shorts in signal lines and Failure detection by on-line monitoring or Diverse HW (see Table A.16 of IEC 61508-2).	–	–	–	–

Table A.48: GIC – FMECA.

Publications

- Asier Larrucea, Jon Perez, Irune Agirre, Vicent Brocal, Roman Obermaisser **A Modular Safety Case for an IEC 61508 compliant Generic Hypervisor**. *The 18th Euromicro Conference on Digital System Design, DSD, 2015, Funchal (Portugal).*

- Asier Larrucea, Jon Perez, Roman Obermaisser **A Modular Safety Case for an IEC 61508 compliant Generic COTS device**. *The 13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC, 2015, Liverpool (UK).*

- Asier Larrucea, Irune Agirre, Carlos Fernando Nicolas, Jon Perez, Mikel Azkarate-Askasua, Ton Trapman **Temporal Independence Validation of an IEC 61508 compliant Mixed-Criticality System based on Multi-core Partitioning**. *2015 Forum on Specification and Design Languages, FDL, 2015, Barcelona (Spain).*

- Asier Larrucea, Hamidreza Ahmadian, Roman Obermaisser, Jon Perez **A Realistic Approach to a Network-on-Chip Cross-Domain Pattern**. *The 19th Euromicro Conference on Digital System Design, DSD, 2015, Limassol (Cyprus).*

- Asier Larrucea, Imanol Martinez, Hamidreza Ahmadian, Roman Obermaisser, Vicent Brocal, Salvador Peiró, Jon Perez **DREAMS: Cross-Domain Mixed-Criticality Patterns**. *Workshop on Mixed Criticality Systems, WMC, 2016, Porto (Portugal).*

- Asier Larrucea, Imanol Martinez, Vicent Brocal, Salvador Peiró, Hamidreza Ahmadian, Roman Obermaisser, Jon Perez **Reusable generic design patterns for mixed-criticality systems based on DREAMS**. *Microprocessor and Microsystems Journal (MICPRO)*

- Asier Larrucea, Imanol Martinez, Carlos F. Nicolas, Jon Perez, Roman Obermaisser **Modular development and certification of mixed-criticality systems**. *20th Euromicro Conference on Digital Systems Design, 2017, Vienna, Austria.*

- Carlos Fernando Nicolas, Fernando Eizaguirre, Asier Larrucea, Simon Barner, Franck Chaivel, Goiuria Sagardui, Jon Perez **Support of Mixed-Criticality Systems Certification**. *12th ERCIM/ERCIM/EWICS/ARTEMIS Workshop on "Dependable Smart Embedded Cyber-physical Systems and Systems-of-Systems" at SAFECOMP 2017 (DECSoS '17), Trento, Italy.*

- Irune Agirre, Mikel Azkarate-Askasua, Asier Larrucea, Jon Perez, Tullio Vardanega, Francisco J. Cazorla **A safety concept for a railway mixed-criticality embedded system based on multi-core partitioning**. *The 13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC, Liverpool (UK).*

- Iban Ayestaran, Carlos Fernando Nicolas, Jon Perez, Asier Larrucea, Peter Puschner **A Novel Modeling Framework for Time-Triggered Safety-Critical Embedded Systems**. *Forum on Specification and Design Languages, FDL, 2014, Munich (Germany).*

- Iban Ayestaran, Carlos Fernando Nicolas, Jon Perez, Asier Larrucea, Peter Puschner **A Simulated Fault Injection Framework for Time-Triggered Safety-Critical Embedded Systems**. *The 33rd International Conference on Computer Safety, Reliability and Security, SAFECOMP, 2014, Florence (Italy).*

- Iban Ayestaran, Carlos Fernando Nicolas, Jon Perez, Asier Larrucea, Peter Puschner **Modeling and Simulated Fault Injection for Time-Triggered Safety-Critical Embedded Systems**. *The 17th IEEE International Symposium on Object, Component, Service-Oriented Real-Time Distributed Computing, ISORC, 2014, Reno (NV).*