D4.4 FRACTAL SAFETY SERVICES

Deliverable Id:	**D4.4**
Deliverable name:	**FRACTAL Safety Services**
Status:	**Final**
Dissemination level:	**PUBLIC**
Due date of deliverable:	**2022-10-31(M26)**
Actual submission date:	**2022-20-25**
Work package:	**WP4 "Safety, Security & Low Power Techniques"**
Organization name of lead contractor for this deliverable:	**SIEG**
Authors:	Rakotojaona Nambinina (SIEG), Daniel Onwuchekwa (SIEG), Boris Bulatovic (AVL) Bernhard Peischl (AVL) Tomislav Bukic (AVL) Harisyam Manda (AVL) Bernhard Winkler (Virtual Vehicle Research) Markus Postl (Virtual Vehicle Research) Florian Thaler (Virtual Vehicle Research) Jaume Abella (BSC) Ramon Canal (BSC) Bertaccini Luca (ETHZ) Artur Kaufmann (BEEWEN Automation GmbH) Ilya Tuzov (UPV)
Reviewers:	Jaume Abella (BSC), Leticia Pascual (SML)

Abstract:
This deliverable aims to report the results and implementations of T4.3 SAFETY SERVICES in the Fractal systems. The deliverable reports the development of the safety services in different hierarachy such as the node level, and systems level using different platform. The explored platforms include VERSAL ACAP, PULP, and NOEL-V.

ECSEL Joint Undertaking
Electronic Components and Systems for European Leadership

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 877056.

Co-funded by the Horizon 2020 Programme of the European Union under grant agreement No 877056.

Contents

1 History (SIEG) 5
2 Summary (SIEG) 6
3 Introduction (SIEG) 7
3.1 Structure of the deliverable 10
4 High Level Picture (SIEG) 11
4.1 Time-Triggered On-Chip Communication 12
4.2 Software diverse redundancy library 12
4.3 Performance monitoring services 13
4.4 Safety services for PULP systems 14
4.5 FPGA based fault Injection 15
4.6 Safety analysis of the NOEL-V Platform redundant acceleration scheme ..15
4.7 Safety concept based on ISO26262 16
5 Time-Triggered On-Chip Communication for Multi-core Architecture (SIEG) ...17
5.1 Time-Triggered Extension Layer for Versal NoC 17
5.1.1 Description of Versal NoC 17
5.1.2 Description of Time-Triggered Extension Layer 18
5.1.3 Experiment setup for Time-Triggered VERSAL NoC 20
5.2 Adaptive Time-Triggered NoC 25
5.2.1 Component Description 25
5.2.2 Architectural Building Blocks of ATTNoC 25
5.2.3 Fault tolerance techniques using Multiple Schedule 27
5.2.4 Fault tolerance techniques using Seamless redundancy 29
6 Software diverse redundancy library (BSC) 39
6.1 Component description 39
6.2 Design and implementation 39
6.3 Testing and evaluation 41
7 Performance monitoring services (BSC) 43
7.1 Component description 43
7.2 Design and implementation 43
7.3 Testing and evaluation 45
8 Safety services for PULP systems (ETH) 47

8.1 Component description 47

8.2 Design and implementation 47

8.3 Testing and evaluation 49

9 FPGA-based Fault Injection (UPV) 50

9.1 Component description 50

9.2 Design and implementation 51

9.3 Testing and evaluation 54

10 Safety Analysis of the NOEL-V Platform redundant acceleration scheme (UPV) 58

10.1 Component description 58

10.2 Design and implementation 58

10.2.1 Transient faults of varied multiplicity affecting single replica. 59

10.2.2 Multiple transient faults affecting multiple replicas. 59

10.2.3 Permanent faults affecting one or several replicas 60

10.3 Testing and evaluation 61

11 Safety concept based on ISO26262 (VIF,AVL) 63

11.1 Introduction to vehicle safety and the relevant ISO standard 63

11.2 UC2 65

11.2.1 Description of the work-flow using the standard 65

11.3 Safety concept with UC7 (SPIDER autonomous robot) 68

11.3.1 Description of the work-flow using the standard 68

11.3.2 Item Definition 68

11.3.3 Hazard Analysis and Risk Assessment 69

11.3.4 Description of use-case 7 69

11.3.5 Hazard analysis and risk assessment 71

11.3.6 Result for functional safety concept 75

12 Safety concept based on IEC 62061 (BEEA) 80

12.1 Introduction for industrial applications of the standard IEC 61508 and corresponding additional standards 80

12.2 Safety concept of UC8 (Shuttle system with swarm intelligence) 81

12.2.1 Description of the workflow using the standard 81

12.2.2 Risk assessment 82

12.2.3 Hardware architecture 87

12.2.4 Safety concept UC8 88

13 Conclusions (SIEG) 91

14 List of figures 92

15 List of tables 94

16 List of Abbreviations 95

1 History (SIEG)

Version	Date	Modification reason	Modified by
0.1	2022-05-31	First contributions	All partners
0.2	2022-07-10	Refined contributions and outline tests	All partners
1.0	2022-07-20	Draft version	All partners
2.0	2022-08-01	New organization of the contents of the document	All partners
2.1	2022-10-21	D4.4 After Internal Review	All partners

2 Summary (SIEG)

This deliverable aims to report the outcomes of T4.3 on safety services. The result of the implementations carried out in the task is presented according to the components developed, which reflect the objectives of the task.

Namely, T4.3 aims to develop safety service at different hierarchies such as the node level and systems level. The structure of this deliverable is divided into different sections and each section describes the implementation, and evaluation of components. Time-Triggered Extension Layer for VERSAL Network-on-Chip, Adaptive Time-Triggered Network-on-Chip, with fault tolerance techniques for NoC are described in section 5 (SIEG); as for safety services at the system level, the work was focused on suitable fault tolerance techniques, specifically software diverse redundancy library, described in section 6 (BSC). The task also investigated performance monitoring services in section 7 (BSC). Section 8, 9 (ETHZ, UPV), describes the safety services for PULP systems and FPGA-based fault injection, in section 10 (UPV), Safety analysis of the NOEL-V Platform redundant acceleration scheme has been explored. In section 11,12 safety concepts based on ISO26262 and IEC 62061 with various use cases have been explored (AVL, VIF, BEEW).

3 Introduction (SIEG)

The goal of the WP4 is to develop safety, security, and low-power services for individual FRACTAL nodes. In this document, we will shed light upon safety services for FRACTAL systems extending the preliminary implementation reported in the previous deliverable D4.1. The development, and implementation, will include both the node level and the system level in accordance with the Fractal system architecture depicted in Figure 1. To this aim, we will provide safety services for multi-core chips that use an Adaptive Time-Triggered Network-on-Chip (ATTNoC) to interconnect heterogenous types of computing resources such as general-purpose processor cores. Moreover, we developed a safety and fault-tolerant services for the communication between FRACTAL nodes including a seamless redundancy mechanism. Performance monitoring services are implemented and investigated in this deliverable. Safety services for PULP systems are developed, in addition, FPGA-based fault injection techniques are investigated and developed. Two safety concepts are considered in this deliverable including ISO 26262, and IEC 62061.

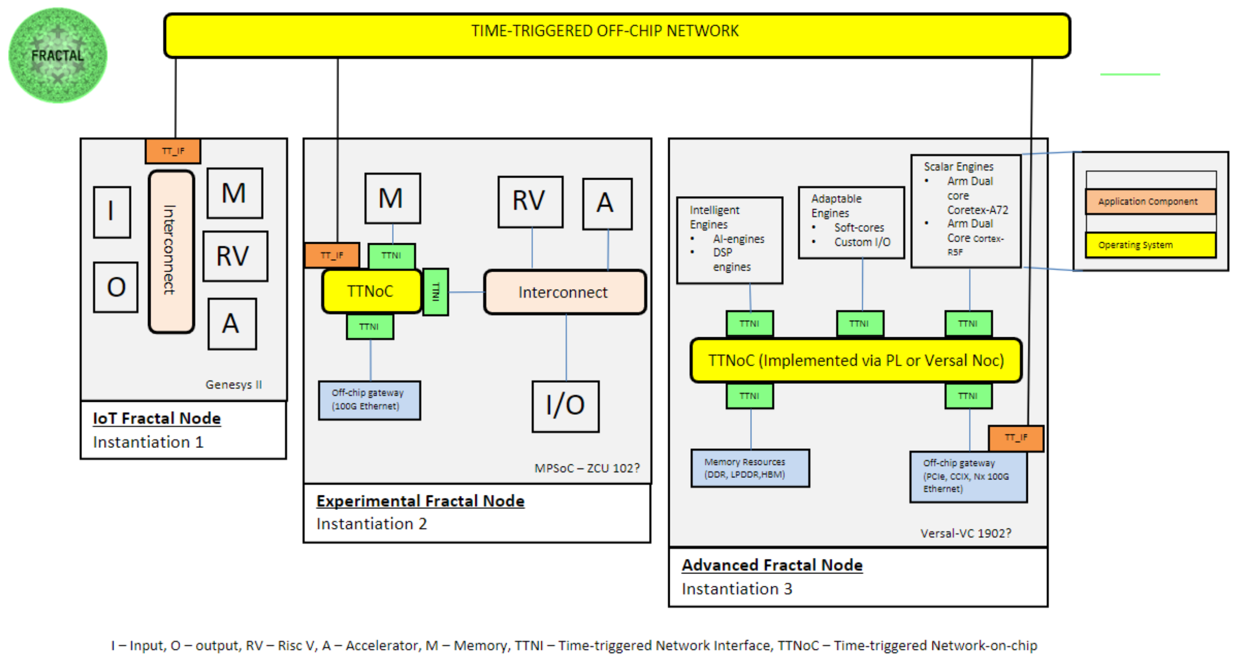

FRACTAL
TIME-TRIGGERED OFF-CHIP NETWORK
TT_IF
I
O
Interconnect
M
RV
A
Genesys II
IoT Fractal Node
Instantiation 1
M
RV
A
TT_IF
TTNI
TTNoC
TTNI
TTNI
Interconnect
Off-chip gateway (100G Ethernet)
I/O
MPSoC – ZCU 102?
Experimental Fractal Node
Instantiation 2
Intelligent Engines
AI-engines
DSP engines
Adaptable Engines
Soft-cores
Custom I/O
Scalar Engines
Arm Dual core Coretex-A72
Arm Dual Core Cortex-R5F
Application Component
Operating System
TTNI
TTNI
TTNI
TTNoC (Implemented via PL or Versal Noc)
TTNI
TTNI
TT_IF
Memory Resources (DDR, LPDDR,HBM)
Off-chip gateway (PCIe, CCIX, Nx 100G Ethernet)
Versal-VC 1902?
Advanced Fractal Node
Instantiation 3
I – Input, O – output, RV – Risc V, A – Accelerator, M – Memory, TTNI – Time-triggered Network Interface, TTNoC – Time-triggered Network-on-chip

Figure 1 Fractal System Architecture

The strategic objective of this task is to provide safety services for the FRACTAL system. To achieve this goal, first, the task was realized by several building blocks/ components which contribute to fulfilling the T4.3 objectives and are reusable and could be used by any use case (UC). A brief description of each of these components and how they contribute to fulfilling the T4.3 Objectives are reported in Table 1.

Table 1 Summary of the components developed in T4.3

Time-Triggered On-Chip Communication	
Description	ATTNoC is an extension of Network-on-Chip, used to interconnect multiple processing elements within SoC, equipped with Adaptive time-triggered features. Time-Triggered communication provides temporal guarantees within NoC and enables the NoC to switch between schedules whenever a context event occurs. This work has also focused on improving fault tolerance of ATTNoC by using a redundancy mechanism at the communication level of NoC that can handle the transient faults that occur during transmission of critical messages within NoC. A single error when transferring critical messages between two cores can be detected and recovered by employing seamless redundancy mechanism features.
Contribution in achieving the T4.3 objectives	This component provides fault-tolerant services for the communication between several processing elements by switching between schedules when a context event occurs within the NoC. It also supports time-triggered communication to achieve deterministic communication within NoC. This component supports message replication/fusion or redundancy mechanisms in FRACTAL systems.
Software diverse redundancy	
Description	With the aim of achieving diverse redundancy, needed to support the highest criticality level across multiple domains, we develop a library that schedules redundant threads of tasks, and preserves enough staggering across redundant threads, so that unprotected hardware resources are used with physical and timing redundancy.
Contribution in achieving the T4.3 objectives	This component provides fault detection, and fault containment based on replication. Random hardware faults are considered, including those leading to common cause failures (i.e., a single fault affecting redundant threads could lead both to identical errors).
Multicore interference control/ monitoring service - Hardware	
Description	The component provides monitoring services built upon a Performance Monitoring Unit (PMU) for the verification of timing concerns (e.g., interference channels and shared resource clogging). This service builds upon configuring WP3T31-01 to measure a specific set of events, typically related to the core where the time-critical task is intended to run. Then, the unit is reset and enabled right before the critical task whose behaviour needs monitoring is about to run. When the task execution finishes, event counters are read and system level decisions on the user application side could be taken based on the amount of shared resource accesses performed and timing interference experienced. Typically, such service is planned to be linked to tasks executing periodically (e.g., every 100ms) with the aim of detecting abnormal timing behaviour during operation, and/or for optimization purposes during system design by, for instance, rescheduling conflicting tasks producing high mutual contention.
Contribution in achieving the T4.3 objectives	The component provides safety features when using shared hardware components. This component provides fault detection, and fault containment. Both random hardware faults and systematic faults will be considered in faulty assumptions. It is planned to support safety critical AI applications. It is a form of safety monitor and provides also support to achieve segregation.
Safety services for PULP	
Description	The aim of this component is to enhance the PULP systems to support safety services. In particular, the work has focused on improving fault-tolerant features through redundancy, ECC protection for memories, and watchdog timers. A single error on a RISC-V core can be detected and recovered by employing redundancy features, eg, having three cores that compute the

	same program in parallel and comparing the output of the three cores. In the case of a single error on one RISC-V core, the redundancy allows to detect the error and recover the corrected status of the faulty core.
Contribution in achieving the T4.3 objectives	The component satisfies the objective of the task through: Increasing safety and fault-tolerant features on PULP platform for IoT applications.
FPGA Based fault injection	
Description	A tool to perform fault-injection in modern FPGA devices. In particular, the tool has been adapted to inject faults in the VCU118, but fault-injections in any device of the Ultrascale+ and Virtex7 families is also possible. The fault-injection process has been adapted to allow fault-injection in the NOEL-V platform allowing performing the robustness characterization of this SoC.
Contribution in achieving the T4.3 objectives	The component helps validating the fault-tolerant capabilities of a built-in safety mechanism such as redundant execution schemes. Thus, the tool helps analyzing the robustness of a particular designs and derive fault metrics at early stages of the product development (i.e before physical implementation).
Redundant Acceleration Scheme Safety Analysis – Analysis/ Safety Concept	
Description	Analysis of the robustness provided by an AI acceleration scheme when implementing different solutions like algorithmic and implementation diversity and for different levels of modularity. The goal if the Identification of protection gaps in the SoC and software layers.
Contribution in achieving the T4.3 objectives	The component helps assess the safety and the robustness of the platform when using non-memory protected accelerators.
Safety Analysis (ISO 26262)	
Description	The ISO 26262 "Road vehicles – Functional safety" represents the automotive specialization of IEC 61508 and provide a framework for achieving Functional Safety for electrical and/or electronic(E/E) systems in road vehicles. A functional safety concept based on the ISO 26262 is applied for the developed functions of UC7. Requirements derived from the safety concept are integrated to the UC architecture.
Contribution in achieving the T4.3 objectives	The component demonstrates how to derive safety requirements for an application in the automotive sector.
Safety Analysis (IEC 61508)	
Description	Safety concept by performing a risk analysis within the scope of the concept phase of IEC 61508 by application of DIN EN ISO 3691-4 (item definition, risk assessment and functional safety concept) on the system, in context of VAL_UC8. The application of fractal components in the shuttle system dissolves many limitations in the hardware and software levels and requires a new risk analysis with the new possibilities.
Contribution in achieving the T4.3 objectives	The safety concept for the shuttle system provides a possible application of the new abilities in respect to the standard DIN EN ISO 3691-4 in an automated environment.

3.1 Structure of the deliverable

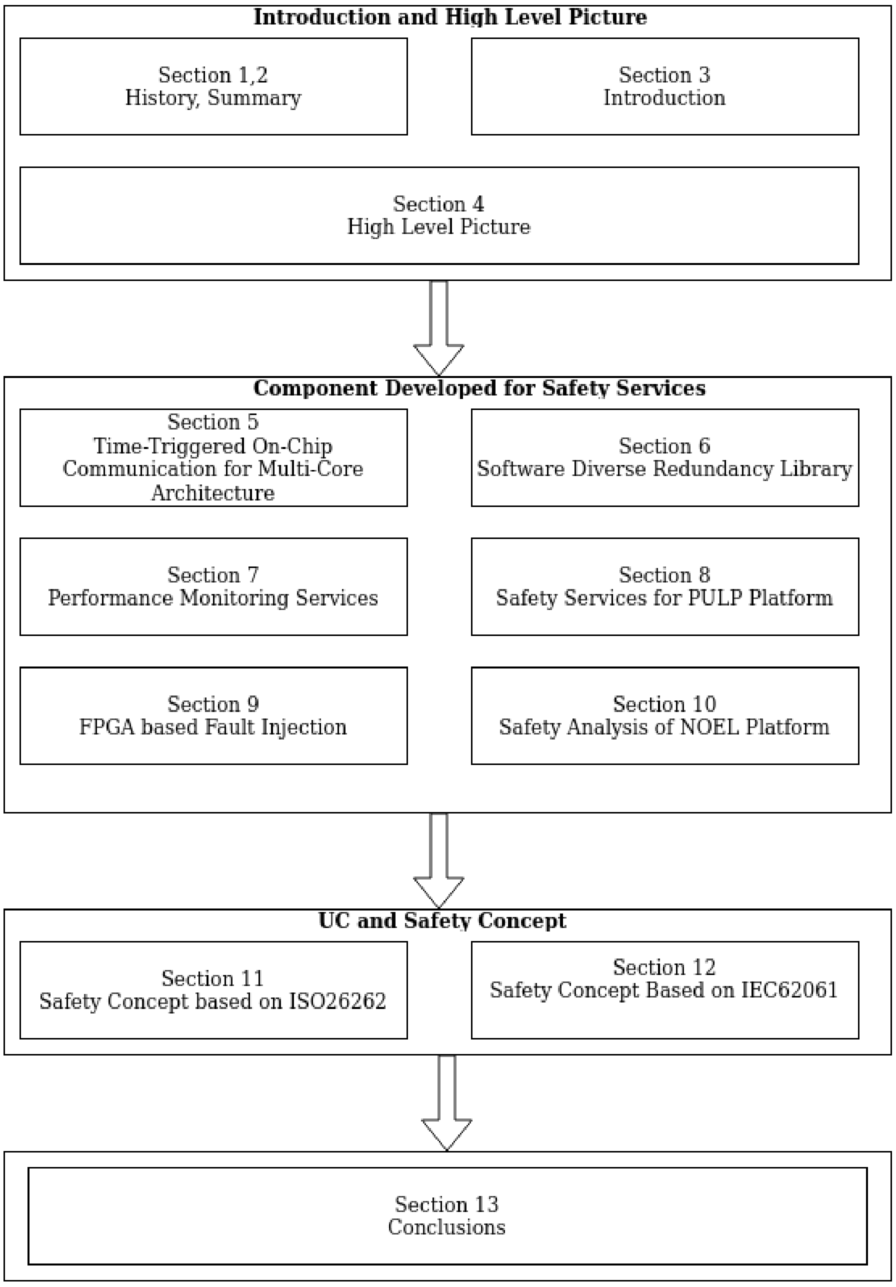

Introduction and High Level Picture
Section 1,2 History, Summary
Section 3 Introduction
Section 4 High Level Picture
Component Developed for Safety Services
Section 5 Time-Triggered On-Chip Communication for Multi-Core Architecture
Section 6 Software Diverse Redundancy Library
Section 7 Performance Monitoring Services
Section 8 Safety Services for PULP Platform
Section 9 FPGA based Fault Injection
Section 10 Safety Analysis of NOEL Platform
UC and Safety Concept
Section 11 Safety Concept based on ISO26262
Section 12 Safety Concept Based on IEC62061
Section 13 Conclusions

Figure 2 Document Organization

4 High Level Picture (SIEG)

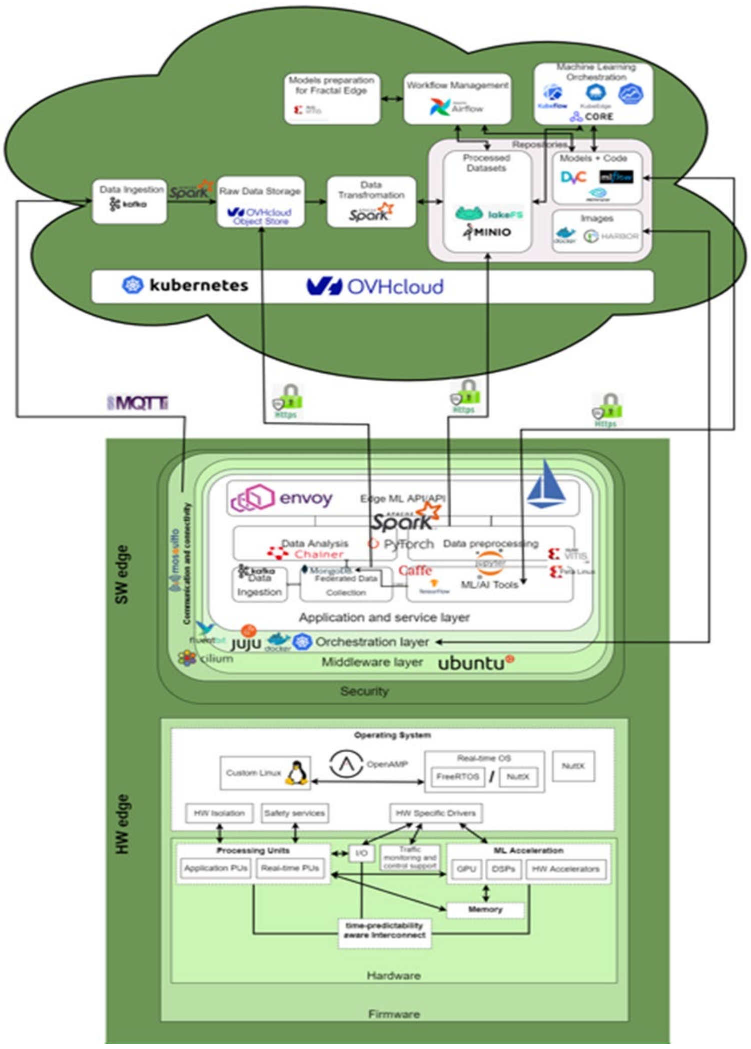


Figure 3 FRACTAL Big Picture

The big picture of the project, illustrated in Figure 3, is a holistic representation of the FRACTAL solution. It provides an answer to the use case requirements, which are the functional and non-functional needs captured by FRACTAL use cases at the beginning of the project. Starting from these requirements, a set of features could be established to give a technical notion to the requirements.

The components, which are developed in the WP4 and could be made of software or hardware, participate in fulfilling some of the FRACTAL features. In the following subsections, we report how each of our components is integrated in the big picture

4.1 Time-Triggered On-Chip Communication

The Hardware edge of the big picture could be represented as it is shown in Figure 4. Thus, the Time-Triggered On-Chip Communication could be regarded as part of the time-predictability aware interconnect and can be embedded into any platform needing a safe-on-chip communication and deterministic communication.

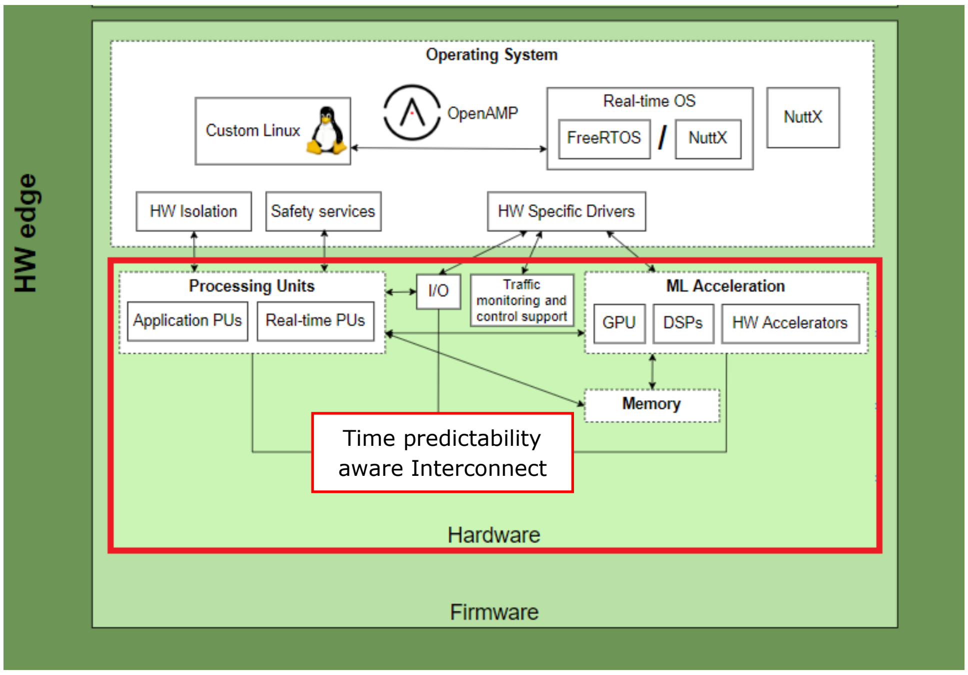


Figure 4 Time-Triggered On-Chip Communication location in the big picture

4.2 Software diverse redundancy library

The software diverse redundancy library is a safety service at the edge level, which builds on some hardware specific drivers to collect some performance monitoring counters and send some signals to cores, but does not use any specific hardware module developed ad-hoc for this service.

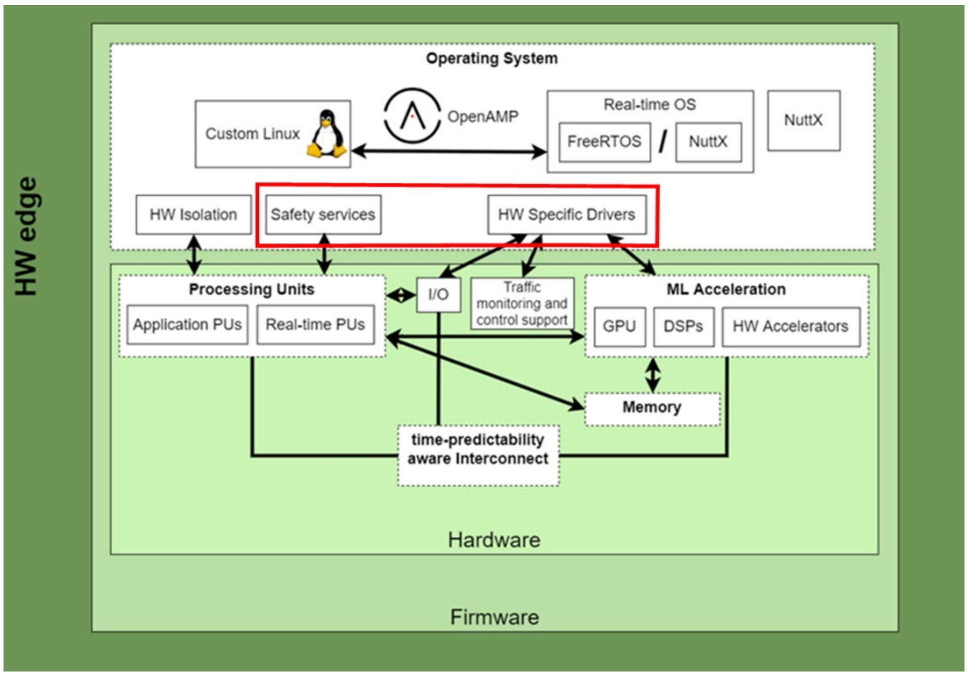


Figure 5 Software diverse redundancy library in the big picture

4.3 Performance monitoring services

The performance monitoring services provide timing monitoring capabilities to end user applications to measure multicore interference and collect other statistics on the on-chip traffic generated by cores. Such safety service interacts with an ad-hoc hardware component monitoring relevant on-chip events, as well as with the interface for such hardware component.

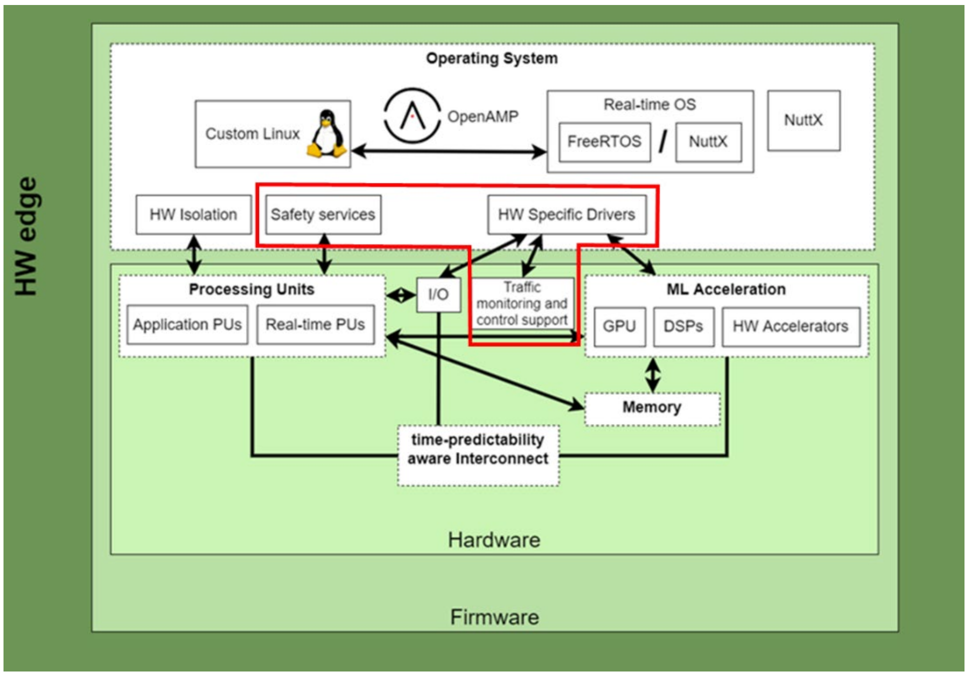


Figure 6 Performance monitoring services in the big picture

4.4 Safety services for PULP systems

FRACTAL targets reliable cognitive end nodes. This component aims at enhancing PULP-based systems with safety features. In particular, fault-tolerant features are added through redundancy, ECC protection for memories, and watchdog timers.

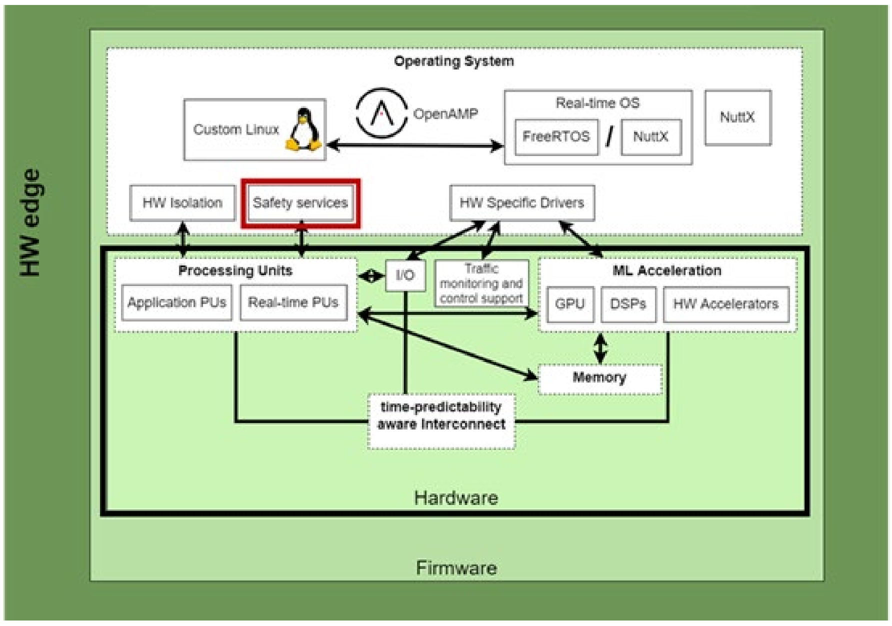


Figure 7 Safety services for PULP systems in the big picture

4.5 FPGA based fault Injection

FPGA-based fault injection tool serves for verification and robustness assessment of FPGA prototypes. It comprises two main components: an on-chip FFI controller and a host-side FFI application. The former is in charge of manipulating the configuration memory of target FPGA to emulate the appearance of selected faults. The latter is in charge of generating the fault load for the FFI controller, as well as supplying the tests (workload) and checking the responses from the Design Under Test (DUT) in order to analyze the effects of injected faults on the DUT.

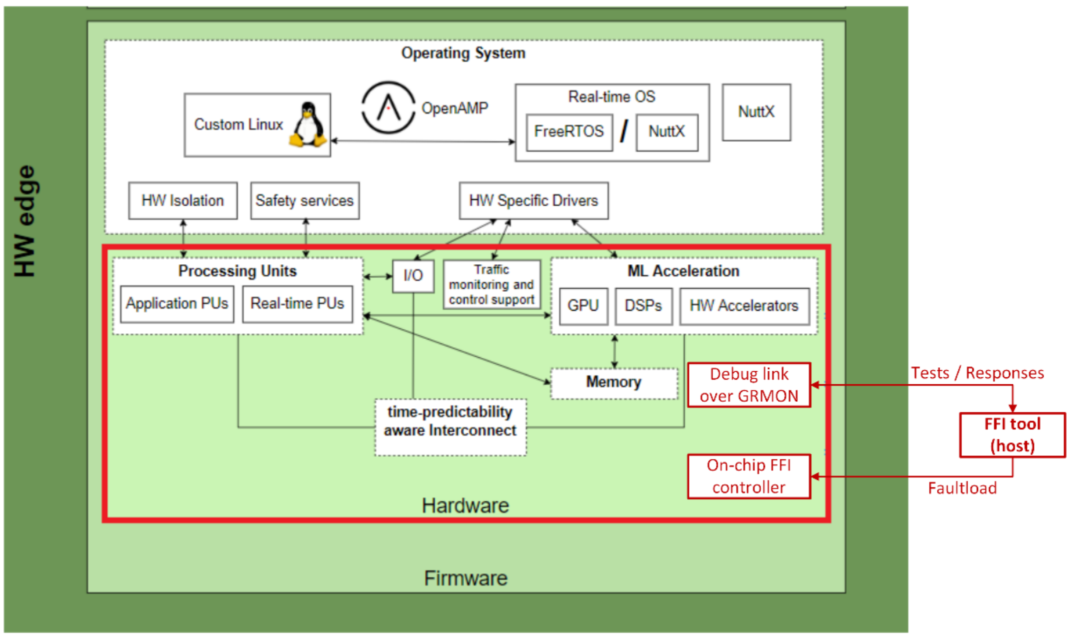


Figure 8 FPGA based fault Injection in the big picture

4.6 Safety analysis of the NOEL-V Platform redundant acceleration scheme

The redundant acceleration scheme comprises an array of replicated CNN (convolutional neural network) accelerators, connected to the SoC via the AXI network, and monitoring component (implemented in software or hardware) responsible for synchronizing the operation of replicas, checking the correctness of results, and taking corrective actions if needed. The reliability/safety analysis of this scheme is partially supported by FPGA-based fault injection experiments.

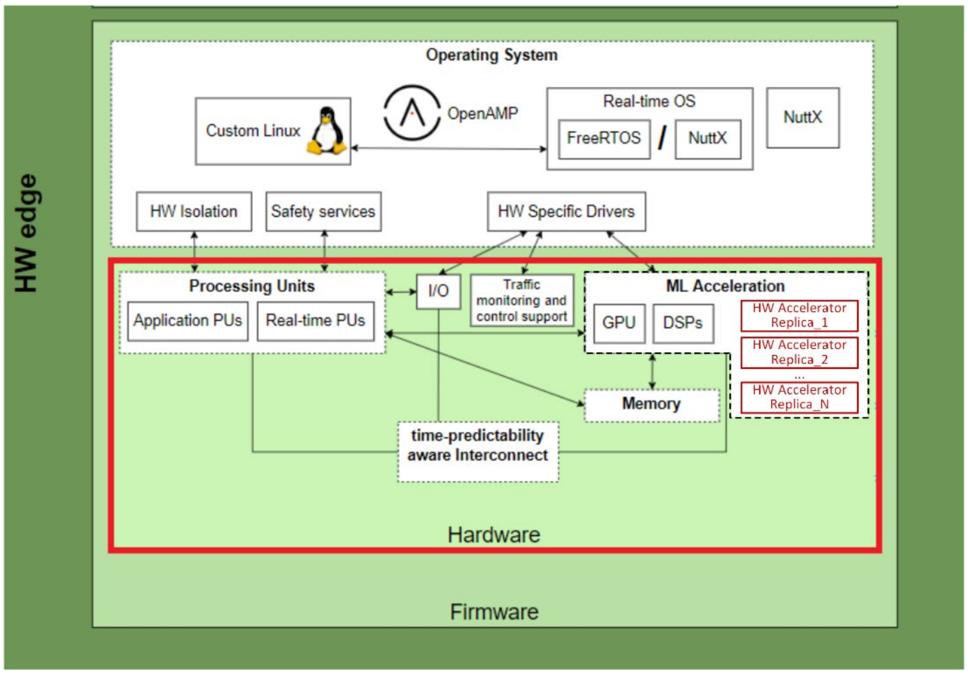


Figure 9 Safety analysis of the NOEL-V Platform redundant acceleration scheme in the big picture

4.7 Safety concept based on ISO26262

The safety concept is implemented by redundant execution of the safety-relevant functions (collision avoidance function and path tracking function) of the SPIDER.

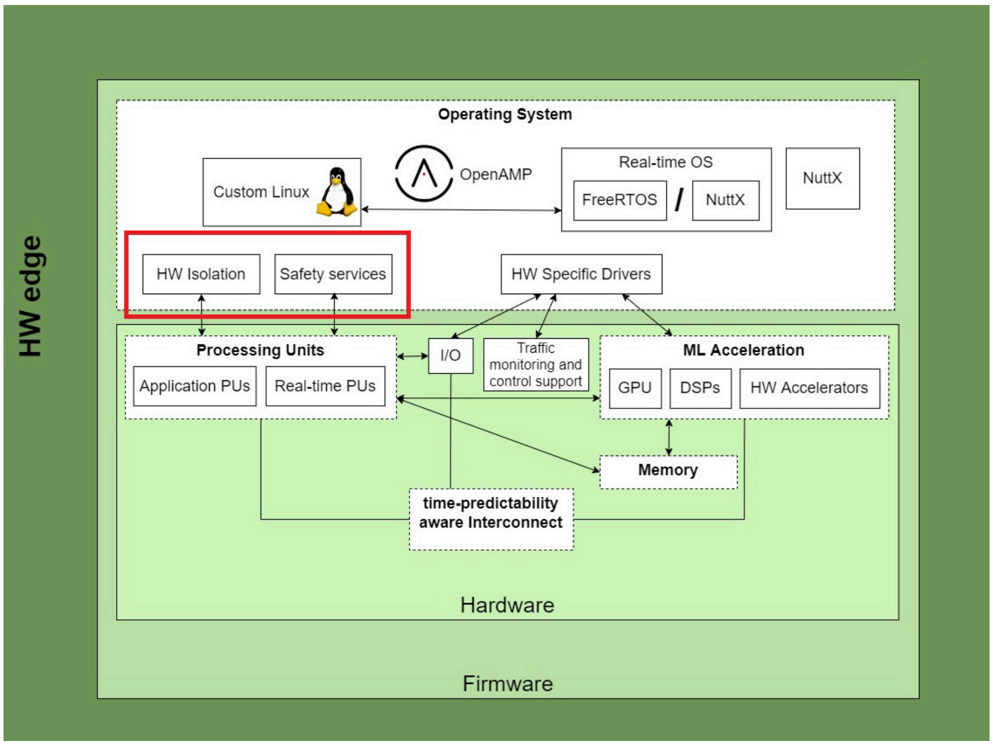


Figure 10 Safety concept based on ISO26262 in the big picture

5 Time-Triggered On-Chip Communication for Multi-core Architecture (SIEG)

5.1 Time-Triggered Extension Layer for Versal NoC

This section describes the time-triggered extension layer used in the VERSAL NoC to establish the temporal partitioning over the Chip. By default, the VERSAL NoC does not support Time-Triggered Traffic. Time-Triggered traffic has been proven to be one of the methods used in safety-critical systems, since it provides deterministic communication, and low jitter, and reduces the risk of message's collision during transmission of messages between multiple processing elements **[SIEG-TTEL1]**. A Time-Triggered Extension Layer (TTEL) is introduced in the VERSAL NoC, which allows the VERSAL NoC to inject messages to the NoC, according to predefined time from the schedule, that is computer offline.

The TTEL components is located in the PL of the VERSAL, between Processing elements (Hard core, soft core, Gateway (GW), etc.) and the NoC Master Unit (NMU) of the VERSAL NoC as depicted in Figure 11.

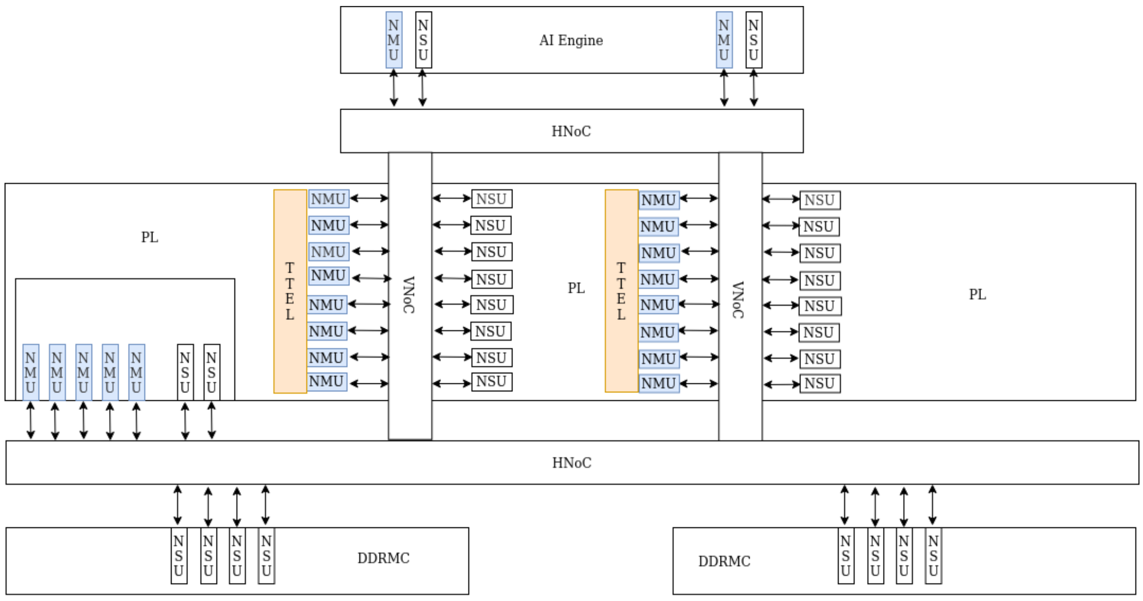


Figure 11 TTEL on VERSAL NoC

5.1.1 Description of Versal NoC

The Xilinx Versal programmable NoC is an AXI based interconnect, used for exchanging data between IP (cores, DDR, AI Engine, Custom IP). This device-wide infrastructure is a high-speed, integrated data path with dedicated switching. The VERSAL NoC has a series of interconnected horizontal (HNoC) and vertical (VNoC) paths as depicted in Figure 12, and supported by a set of customizable, hardware implemented components that can be configured in different ways to meet design timing, speed, and logic utilization requirements. The HNoC and VNoC are dedicated, high-bandwidth paths connecting integrated blocks between the processor system and the programmable logic (PL) without consuming large amounts of programmable logic. The NoC components comprise NoC master units (NMU), NoC slave units (NSU), NoC Packet Switches (NPS), and NoC Inter-Die-Bridge (NIDB). The NMU is the traffic ingress point; the NSU is the traffic egress point. All IPs have some number of these master and slave connections. The NIDB connects two Super Logic Regions (SLRs) together, providing high bandwidth between dies. The NPS is the crossbar switch, used to fully form the network.

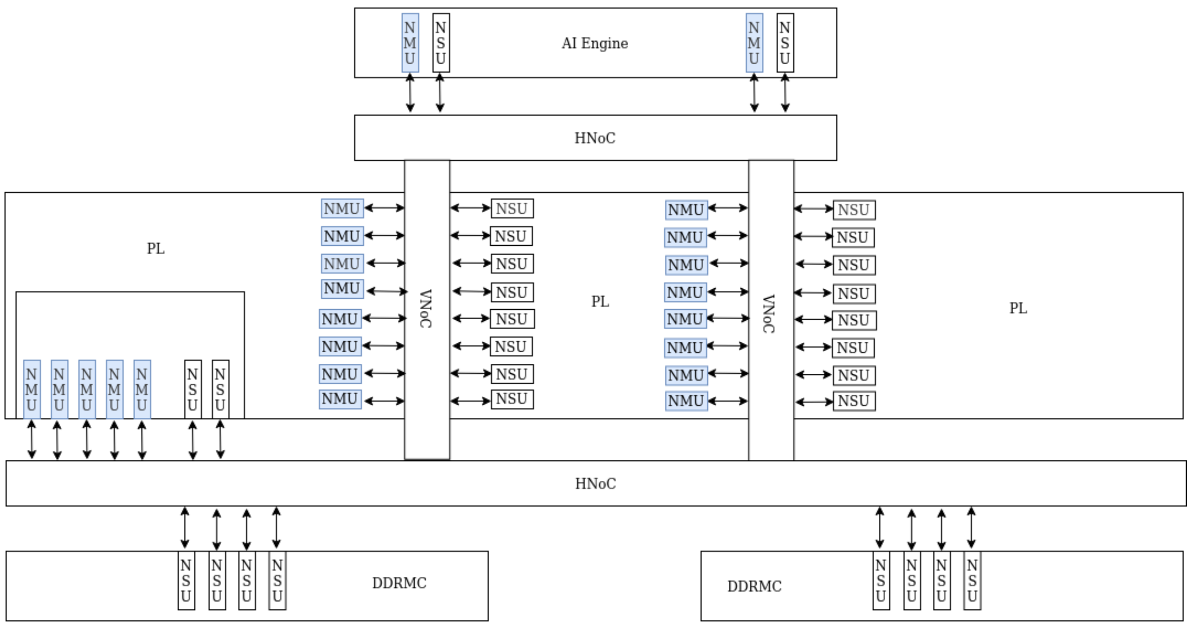


Figure 12 Block Diagram of VERSAL NoC **[SIEG-TTEL2]**

5.1.2 **Description of Time-Triggered Extension Layer**

Time-Triggered Extension Layer (TTEL) is a temporal and spatial partitioning layer that is constructed on top of NoC. The TT extension layer allows the underlying NoC to support several types of communication (e.g., time-triggered, rate-constrained, and best-effort) as well as mixed-criticality applications.

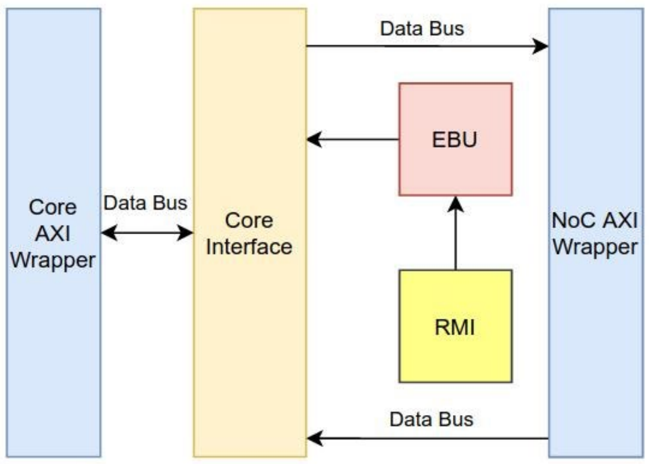


Figure 13 Time-Triggered Extension Layer for VERSAL NoC **[SIEG-TTEL3]**

Figure 13, illustrates the extension layer's building components. In addition, to the extension layer on the core and NoC sides, core and NoC wrappers offer the hardware interface, i.e., AXI Full interface. Other building blocks of the TTEL are the core interface, RMI (Reconfiguration and Monitoring Interface) and EBU (Egress Bridging Unit). The functionalities of each block will be described below:

Core and NoC AXI Wrapper: The purpose of the core AXI wrapper is to provide the required communication service between the core and the TTEL. Similarly, the NoC AXI wrapper supports communication between the TTEL and the Versal NoC.

Core Interface: Core Interface is one of the main components of the TTEL. Versal NoC and Core's communication is handled through Core Interface. The core is composed of Ports; the primary element of the Core Interface which can deliver and receive data. Each port contains a memory that can be used to store data temporarily. The ports on the core interface are separated into input and output ports. Input ports and output ports share the same structure. Both input and output ports are in charge of reading and writing messages.
The concurrent implementation of time-triggered and event-triggered messages is possible with the fully functional Core Interface. However, this approach focuses only on time-triggered message exchange and ignores the transmission of event-triggered messages and other functions for which the Core Interface was designed.

Egress Bridging Unit (EBU): Scheduler and schedule-memory are the two main parts of EBU. To realize time-triggered communication, the Scheduler extracts the scheduling message from memory and compares it with the current time from the GTB (Global Time Base). The message dequeue is caused by the trigger signal, which is generated and forwarded to the core interface.

Reconfiguration and Monitoring Interface (RMI): The Resource Management Interface (RMI) is another multipurpose component. RMI in TTEL can be utilised as a channel between resource management and the internal parts. Based on instructions from resource management, the RMI can reconfigure the TTEL, and can assist by gathering and sending monitoring data. Through the communication ports, the RMI and resource manager communicate with one another on top of the communication services. Additionally, it communicates with the TT schedule memory Unit. Its only purpose is to configure time-triggered schedule information in memory.

5.1.3 Experiment setup for Time-Triggered VERSAL NoC

In order to evaluate the performance of the TT-VERSAL NoC. Various experimental setups were created. The performance of VERSAL NoC with TTEL and without TTEL is considered in this experiment. The following tables illustrate the different experimental setups.

Table 2 Experiment setup without TTEL, ISOCHRONOUS (100MB/s)

Without TTEL		
Exp No	Message Size	NoC QoS
1	2	Isochronous 100MB/s
	4	Isochronous 100MB/s
	8	Isochronous 100MB/s
	16	Isochronous 100MB/s

Table 3 Experiment setup with TTEL, ISOCHRONOUS (100 MB/s)

With TTEL		
Exp No	Message Size	NoC QoS
2	2	Isochronous 100MB/s
	4	Isochronous 100MB/s
	8	Isochronous 100MB/s
	16	Isochronous 100MB/s

Table 4 Experiment setup without TTEL, BEST EFFORT (100MB/s)

Without TTEL		
Exp No	Message Size	NoC QoS
3	2	Best Effort 100MB/s
	4	Best Effort 100MB/s
	8	Best Effort 100MB/s
	16	Best Effort 100MB/s

Table 5 Experiment setup with TTEL (BEST EFFORT 100 MB/s)

TTEL		
Exp No	Message Size	NoC QoS
4	2	Best Effort 100MB/s
	4	Best Effort 100MB/s
	8	Best Effort 100MB/s
	16	Best Effort 100MB/s

Table 6 Experiment setup without TTEL (ISOCHRONOUS 50 MB/s)

Without TTEL		
Exp No	Message Size	NoC QoS
5	2	Isochronous 50MB/s
	4	Isochronous 50MB/s
	8	Isochronous 50MB/s
	16	Isochronous 50MB/s

Table 7 Experiment setup with TTEL (ISOCHRONOUS 50 MB/s)

TTEL		
Exp No	Message Size	NoC QoS
6	2	Isochronous 50MB/s
	4	Isochronous 50MB/s
	8	Isochronous 50MB/s
	16	Isochronous 50MB/s

Table 8 Experiment setup without TTEL (BEST EFFORT 50 MB/s)

Without TTEL		
Exp No	Message Size	NoC QoS
7	2	Best Effort 50MB/s
	4	Best Effort 50MB/s
	8	Best Effort 50MB/s
	16	Best Effort 50MB/s

Table 9 Experiment setup with TTEL (BEST EFFORT 50 MB/s)

TTEL		
Exp No	Message Size	NoC QoS
8	2	Best Effort 50MB/s
	4	Best Effort 50MB/s
	8	Best Effort 50MB/s
	16	Best Effort 50MB/s

The above tables indicate the different sets of experiments carried out on versal NoC. The experiment was done for various message sizes (Burst Length), Traffic class and read-write Bandwidth. For each traffic class and read-write bandwidth, the NoC was simulated with TTEL and without TTEL.

Two processors (core 1 and core 2 as depicted in Figure 14 (With TTEL) are used to send the data to the DDR which is connected to the NoC, and Figure 15 (Without TTEL). Both cores send the data to the VERSAL NoC at the same time, to evaluate the delay when two messages are accessing resources in the VERSAL NoC. A comparison of jitter is done in these experiments with TTEL added in the VERSAL NoC and without TTEL in the VERSAL NoC.

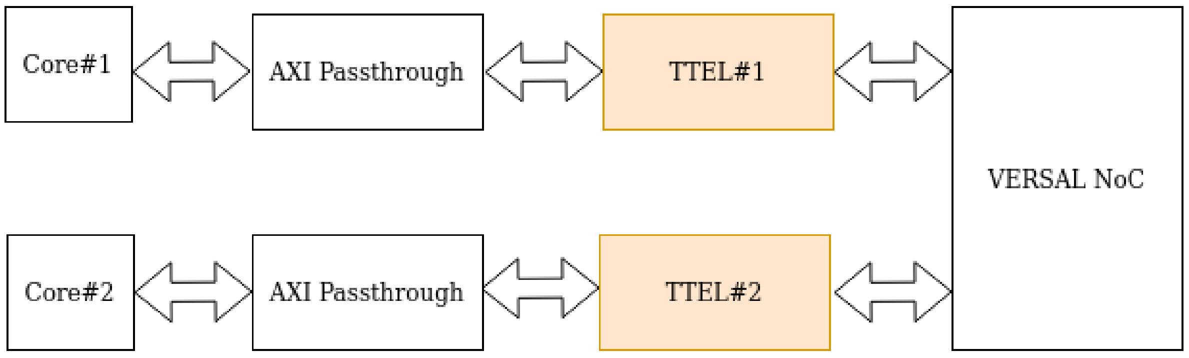


Figure 14 Block Diagram of VERSAL NoC with TTEL

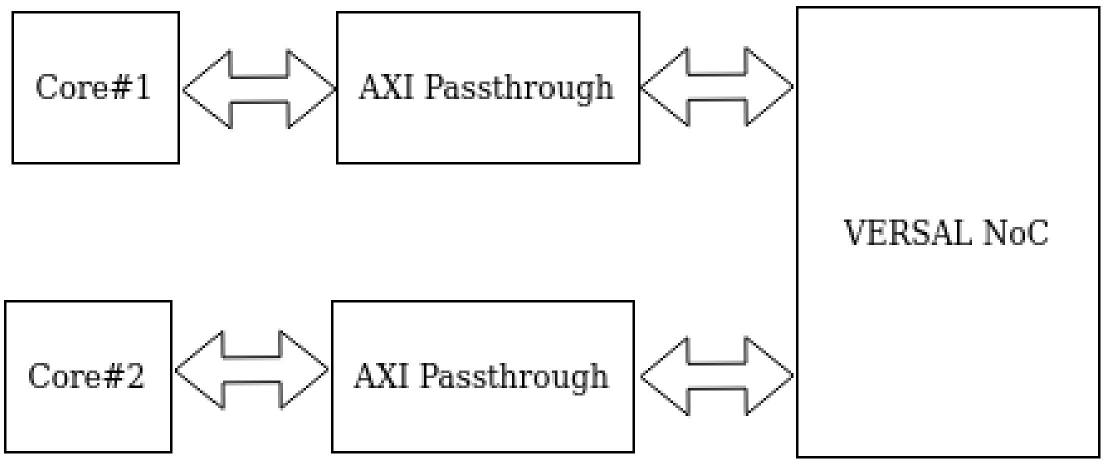


Figure 15 Block Diagram of VERSAL NoC without TTEL

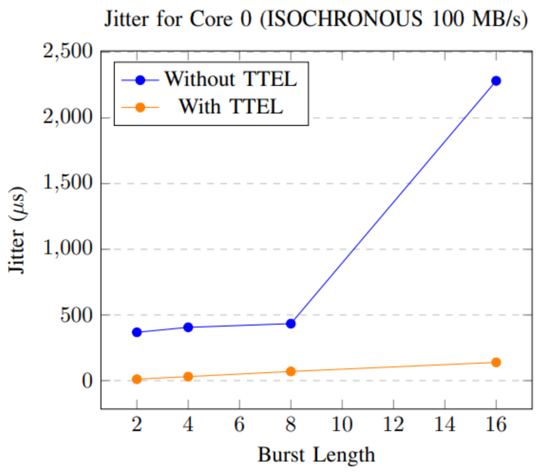


Figure 16 Experiment 1

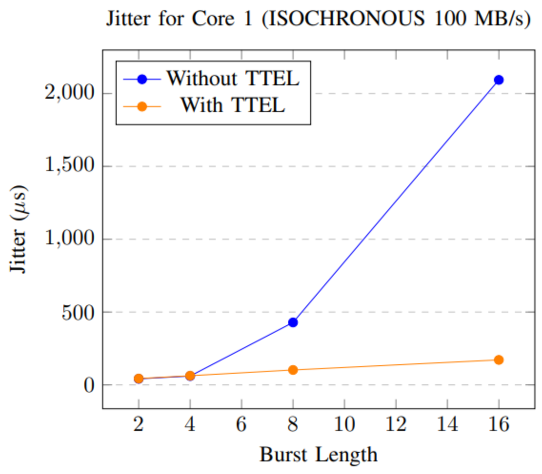


Figure 17 Experiment 2

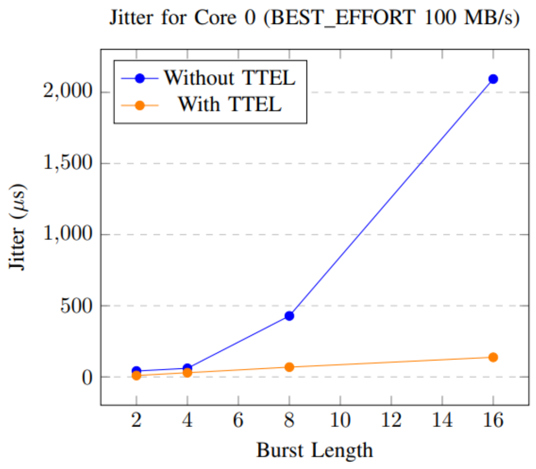


Figure 18 Experiment 3

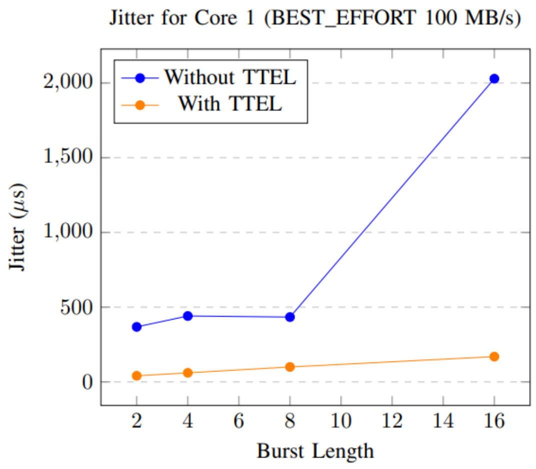


Figure 19 Experiment 4

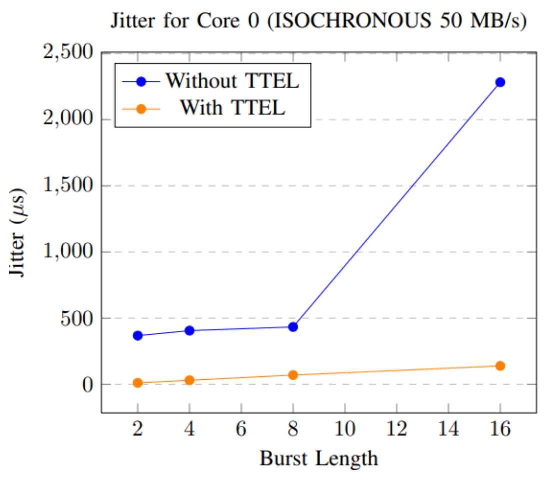


Figure 20 Experiment 5

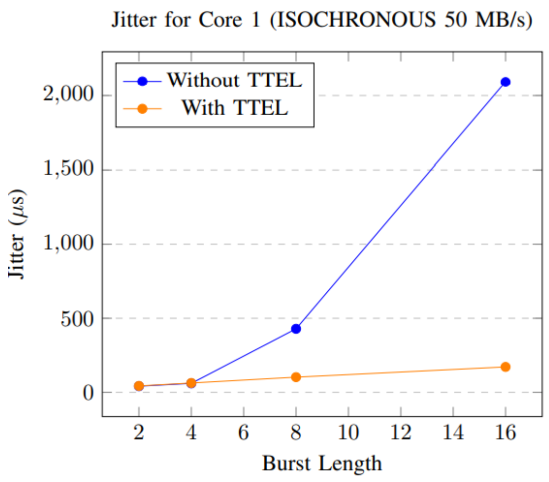


Figure 21 Experiment 6

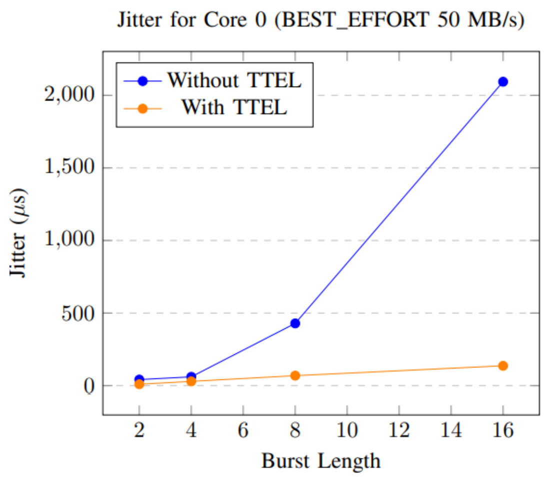


Figure 22 Experiment 7

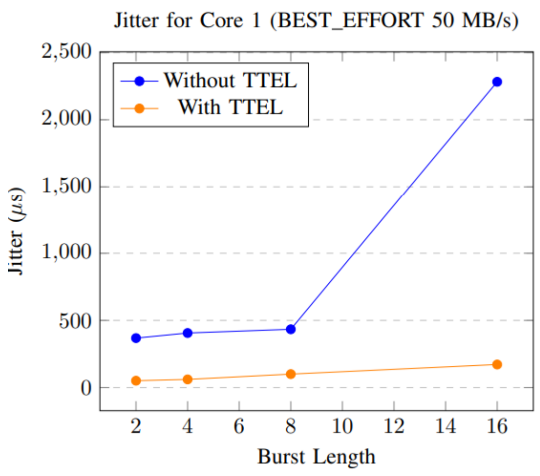


Figure 23 Experiment 8

From the performance analysis done by plotting the graph of Jitter for different cores with TTEL and without TTEL, one could clearly say that the design with TTEL outperformed the design without TTEL. The value of jitter appears to increase with the increase in the burst length for both cases. However, the values of jitter were high for the design without TTEL in all cases. The maximum latency for the message to reach the NoC in the case of design with TTEL remained the same and was independent of burst length. However, it was not the case with the designs without TTEL, where the maximum latency increased with the burst length. Thus, going through the graphs and tables, one could clearly say that TTEL improves the functionality of the hardened NoC.

[SIEG-TTEL1] Obermaisser, R., Ahmadian, H., Maleki, A., Bebawy, Y., Lenz, A., & Sorkhpour, B. (2019). Adaptive time-triggered multi-core architecture. *Designs*, *3*(1), 7.

[SIEG-TTEL2] Xilinx. Versal noc. https://www.xilinx.com/support/documentation/ip_documentation/axi_noc/v1_0/pg313-network-on-chip.pdf.Accessed 2022-29-01.

[SIEG-TTEL3] Hamidreza Ahmadian and Roman Obermaisser. Time-triggered extension layer for on-chip network interfaces in mixed-criticality systems. In 2015 Euromicro Conference on Digital System Design, pages 693–699, 2015.

5.2 Adaptive Time-Triggered NoC

5.2.1 Component Description

Adaptive Time-Triggered NoC is an extension of Time-Triggered NoC with adaptability features. The TTNoC is used in safety critical systems since it offers significant benefits for the safety arguments of dependable systems. However, adaptation is a key factor for fault recovery in Cyber-Physical Systems (CPS). This component introduces the Adaptive Time-Triggered Network-on-Chip, which supports:

- Adaptation using multi-schedule graphs while preserving the key properties of time-triggered systems including implicit synchronization, temporal predictability, and avoidance of resource conflicts.
- Fault tolerance techniques using Seamless redundancy mechanism. The seamless redundancy mechanism in the TTNoC, ensures that transient faults occurring during the transmission of critical message between two cores, can be masked.

5.2.2 Architectural Building Blocks of ATTNoC

The Adaptive TTNoC offers inherent temporal predictability and fault containment for processing cores. Figure 24, gives an overview of the Adaptive Time-Triggered Network-on-Chip. The architecture has tiles, which are interconnected by a NoC. The NoC consists of routers, each of which is connected to other routers and to tile using communication links. A tile includes three parts: cores (cf. gray area in Figure 24) for the application services, adaptation logic (cf. orange area in Figure 24) for switching between schedules, and Network Interface (NI) (cf. blue area in Figure 24) for accessing the NoC.

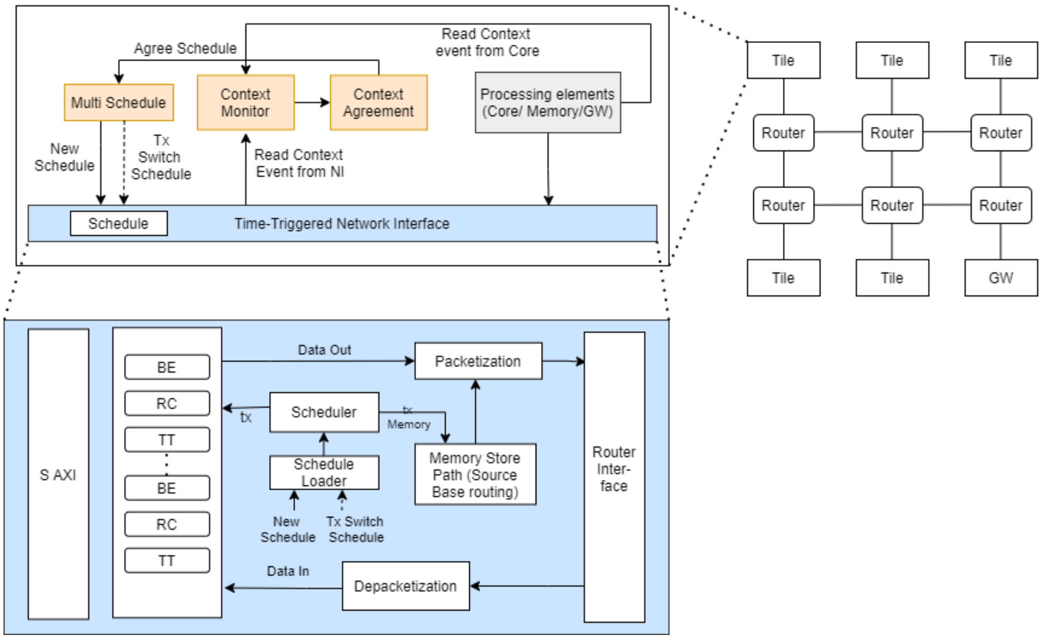


Figure 24 Adaptive Time-Triggered Network-on-Chip (NoC) **[SIEG-ATTNoC1]**

The cores of a tile can be heterogenous or homogenous with bare-metal application software or state machines implemented in hardware. The adaptive logic is responsible for the adaptation (switching between schedules) in the ATTNoC. This logic has 3 elements including: Multi schedule memory, context monitor, and context agreement.

Context monitor: responsible for observing the behavior of the cores and NI connected to the core. An example is a failure in the core, NI or links.

Context agreement: Context agreement unit ensures that all operational tiles possess identical context information.

Schedule memory: Memory that stores multiple schedules. This schedule memory generates one schedule based on the agree context feed to the module.

The Network Interface: is used as a bridge to connect the core and router. The NI is extended with Time-Triggered traffic, that enables the NoC to inject messages using time-triggered traffic. Figure 24 illustrates a block diagram of the implemented TTNI (cf. blue area), which includes a core interface (CI), Schedule loader, Scheduler, Packetization, Depacketization, Memory store the possible path for source base routing, and Router interface.

Core Interface (CI): facilitates communication between the TTNI and Core. The CI is a memory interface that has input and output ports, that stores data from the NoC to core, respectively. The core interface has state ports, used for the periodic transmission of messages with state semantics. In addition, the core interface can achieve concurrency of event-triggered and time-triggered messages.

Schedule loader: is an interface used to configure the TTNI with a predefined schedule.

Scheduler: is the main block in TTNI, that has the actual schedule, that runs the NoC.

Packetization: Receive the messages from the core, and encoded the messages before sending them to the router of the NOC.

Depacketization: Receive the messages from the router of the NoC, and decoded the messages.

Memory store Path: Since the ATTNoC, used a source base routing, it has a memory that stores all possible paths.

5.2.3 Fault tolerance techniques using Multiple Schedule

Multiple schedules are used in the ATTNoC, that allow the systems to operate, even if in presence of fault in the ATTNoC. The ATTNoC allows for schedule change in response to a context event (fault in the NoC). In addition, it also considers synchronization, temporal predictability, and avoidance of resource conflicts.

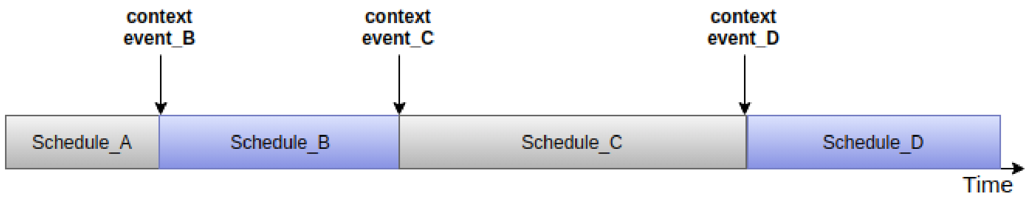


Figure 25 Switching between Schedules

The Adaptation Unit is a building block inside the TTNI, which is responsible for switching schedule when context event occurs. The Adaptation Unit is illustrated in Figure 26 This unit manages the time triggered dispatching of adaptation messages and adaptation in the TTNI in the following steps:

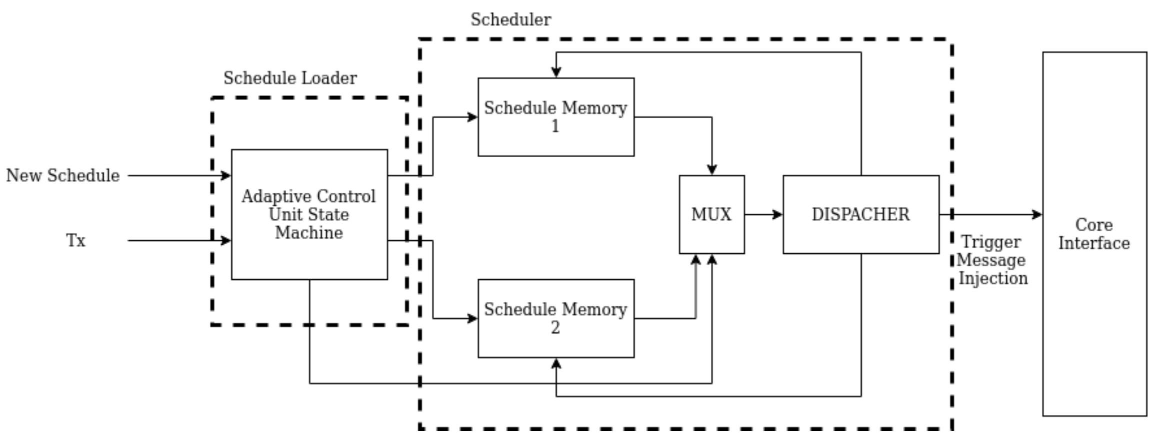


Figure 26 Adaptive Unit in TTNI

Schedule memory selection: The adaptive control unit (ACU) is a state machine, known as a schedule loader, and uses two schedule memories known as ScheduleMem1, and ScheduleMem2 located in the scheduler as depicted in Figure 26. The system's first schedule is saved in the first schedule memory computed off-line and loaded when it boots up. Then, the ACU stores the new schedule in a second memory when a global context event occurs. Finally, after three clock cycles of write schedule operation, the dispatcher selects the proper schedule. The ACU loads the next adapted schedule to the first schedule memory and repeats the process on the occurrence of another context event. The ACU controls the selection of the correct schedule memory through the multiplexer.

Triggering step: The dispatcher initiates injection for the corresponding ports in the core interface based on the new schedule loaded in the active schedule memory.

Figure 27 represents a state machine used for the Adaptive Control Unit.

States = { Initial , WaitEvent , WriteScheduleMem1, WriteScheduleMem2 }

variable : count : {3 bit}, ramUsed : {1 bit}

input : rst , ctx {1 bit }

outputs: We1 , We2 , tx_mux : {1 bit} , WriteAdd1 , WriteAdd2 : {3 bit} , WriteData1, WriteData2 : {32 bit}

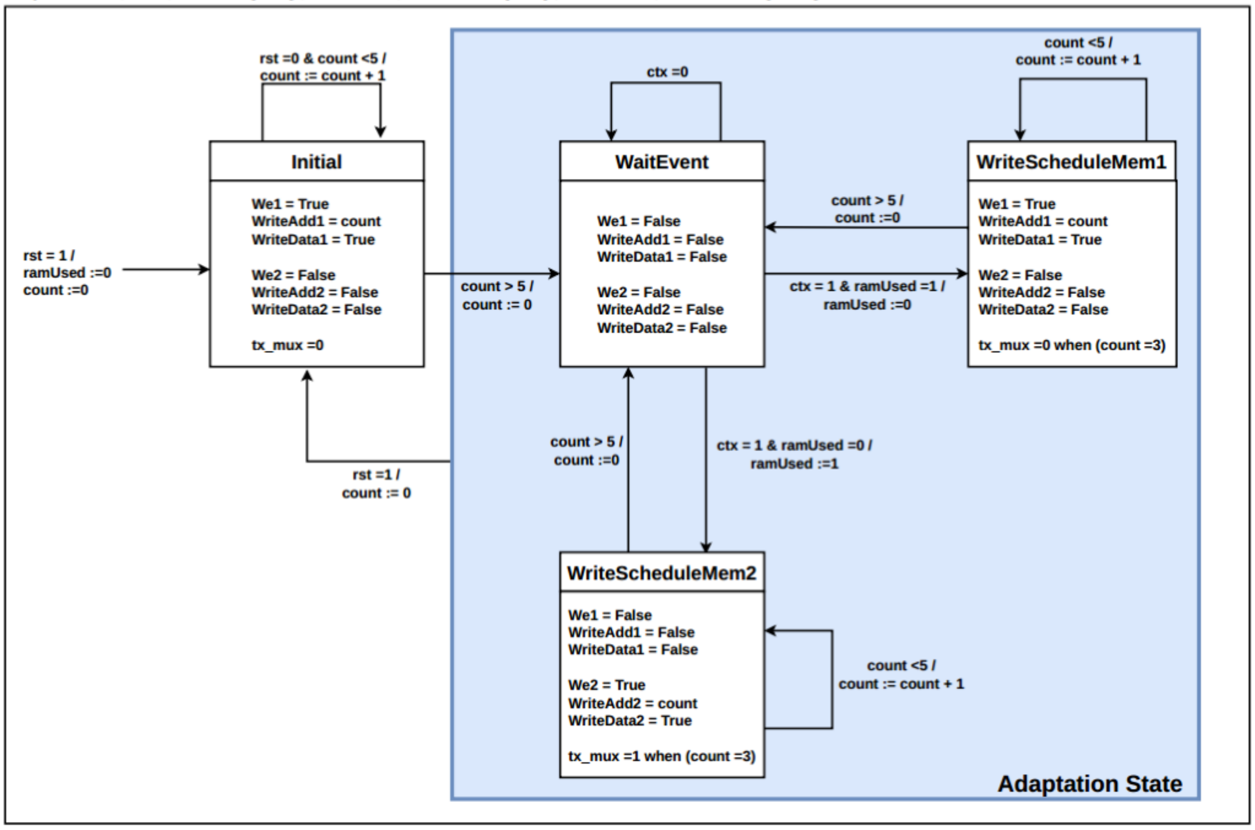


Figure 27 State Machine for Adaptive Control Unit

From Figure 27, the ACU has 4 states: the Initial, WaitEvent, WriteScheduleMem1, and WriteScheduleMem2 state.

Initial State: Schedule memory 1, receives the initial schedule.

Wait Event State: After five clock cycles, the initial state transitions to the WaitEvent State. Transition to the next state, WriteScheduleMem1 and WriteScheduleMem2, involves a context event and the active memory used. The ACU transits to the WriteScheduleMem1 if ScheduleMem2 is used and WriteScheduleMem2 if ScheduleMem1 is active.

WriteScheduleMem1 State: In this state, the ACU writes the schedule to ScheduleMem1.

WriteScheduleMem2 State: In this state the ACU writes the schedule to ScheduleMem2.

Each schedule loaded in the schedule memory is made up of a circular linked list. Each link list is associated with a period of the periodic system, with unique entries. Each port has one ID in each entry of the circular linked list. An example of one TTNI communication schedule is shown in Figure 28.

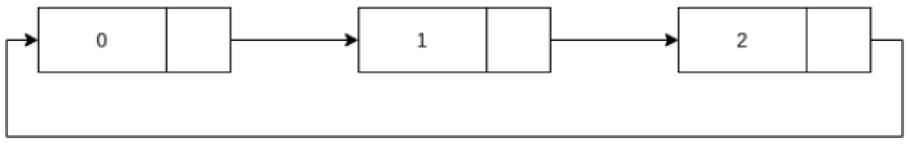


Figure 28 Example of Circular Linked List Schedule

Each of the linked list schedule entries has the following fields :

Next : Points to the next entry of the circular linked list.

Instant : Denotes the phase of message injections located at a given port.

PortId : Describes the ID of the port whose messge is injected into the NoC.

5.2.4 **Fault tolerance techniques using Seamless redundancy**

The seamless redundancy mechanism is a fault tolerance used in the ATTNoC. The seamless redundancy mechanism is integrated in the TTNI, and allows the ATTNoC, to exchange messages between two cores using seamless path. The seamless redundancy mechanism is designed to tolerate a transient fault in the ATTNoC. The ATTNoC has the capabilities to switch schedule when permanent fault occurs in the ATTNoC. However, for transient fault, switching schedule is not a good option, since the fault occurs only within a few clocks cycle and then recover. The figure below illustrates the seamless redundancy mechanism in ATTNoC.

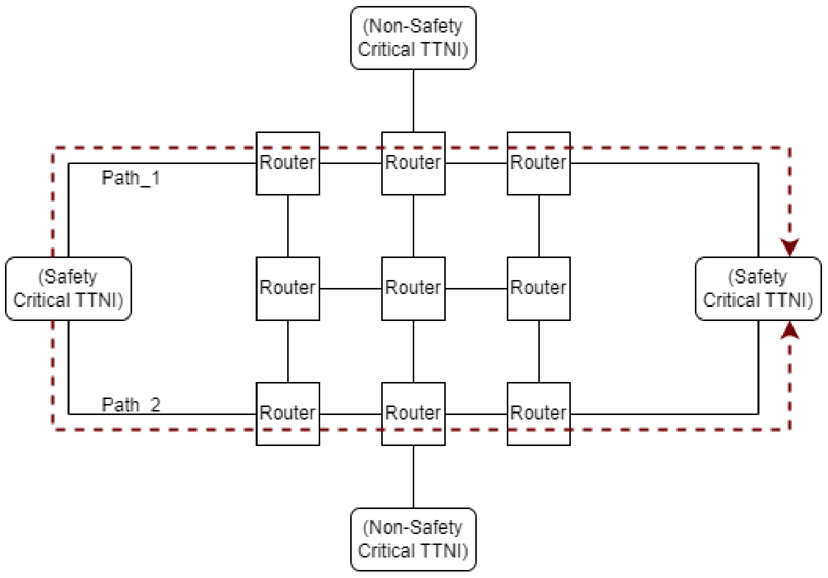


Figure 29 Seamless Redundancy in TTNoC

The ATTNoC has two types of NI to support mixed-criticality/safety-critical systems: Safety-Critical TTNIs (SCTTNIs) and Non-Safety Critical TTNIs (NSCTTNIs) or TTNIs.

SCTTNIs are connected to two different routers and can operate in two modes. The seamless redundancy mode and non-seamless redundancy mode. When the network interfaces need to send critical messages, the seamless redundancy mode is engaged. However, when sending non-critical messages, the non-seamless redundancy modes are initiated. In non-critical messages, only one path is selected instead of two paths.

NSCTTNIs is known as TTNIs, which is connected to one router and support only a non-seamless redundancy mode. The architecture of TTNIs is depicted in Figure 24(cf. blue area in Figure 24).

The SCTTNIs (depicted in Figure 30) is an extension of TTNIs, that has two packetization and two depacketization, respectively. Both packetization and depacketization are connected to two different routers to transfer duplicate messages into two seamless paths. On the other hand, the redundancy controller is used to control the operation of redundancy mode. The redundancy controller duplicated the messages from the core interface whenever seamless redundancy modes is required, otherwise, it activates one of the packetization blocks. The redundancy controller operates according to a schedule. Redundancy controller is responsible for message's selection when both depacketization receive data from source NI. It selects only one message based on the information from the scheduler.

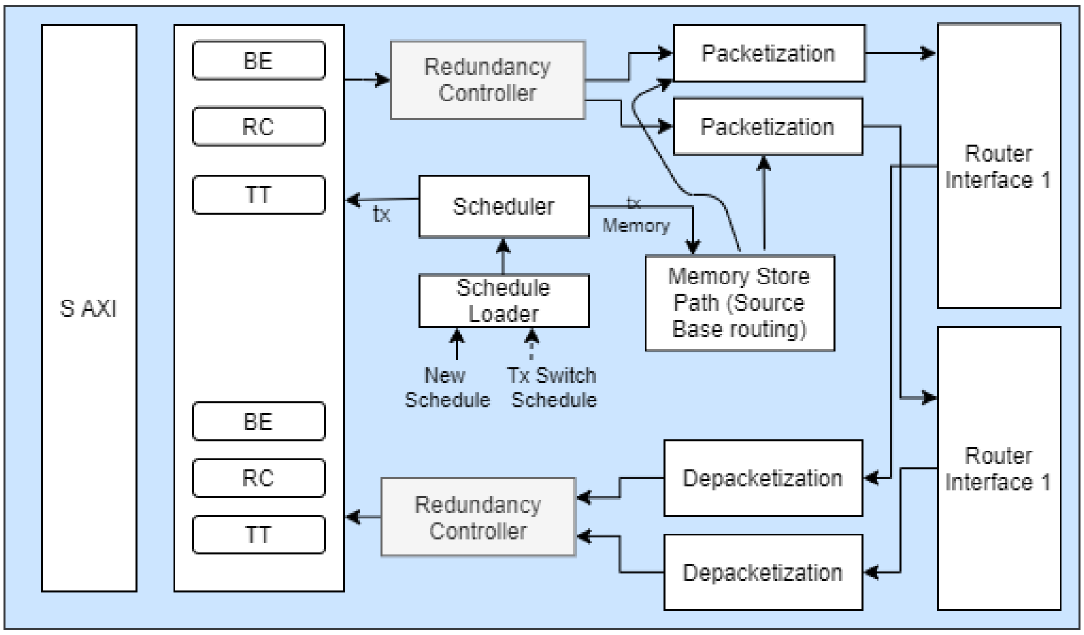


Figure 30 SCNIs Architecture

5.2.4.1 Testing and evaluation

The Fractal Node's intended redundancy mechanism was simulated to evaluate its behaviour. The simulation was carried out using the Vivado Design Suite-Hlx Edition-2020.1. In addition, the Versal ACAP VCK 190 Evaluation Board was used to model the design.

The simulation was performed for two different scenarios, redundancy when there was no error or failure and redundancy when there was an error or failure in the NoC.

Scenario 1: Redundancy Without Errors or Failures

The redundancy mechanism in TTNoC is evaluated using a 3x3 mesh topology. The TTNoC IP with redundancy was deployed in this case, and the results were recorded. Figure 31 and Figure 32, depicts the simulation results.

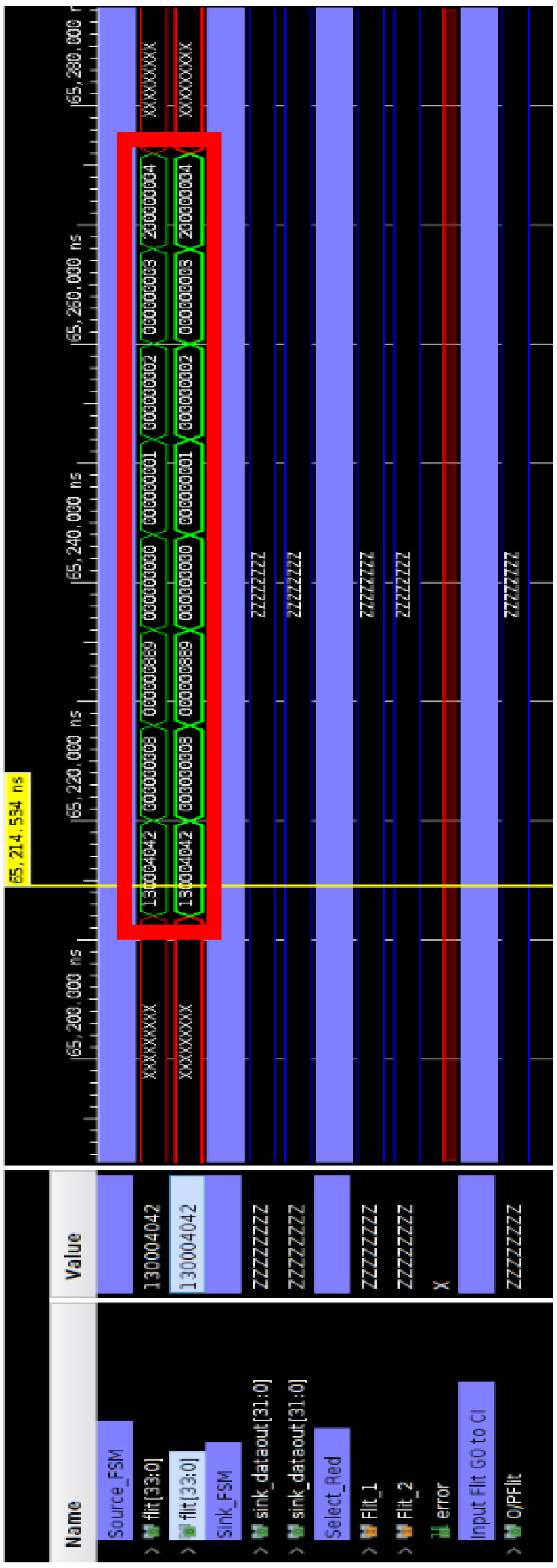


Figure 31 Simulation Result of Scenario 1, NI sender

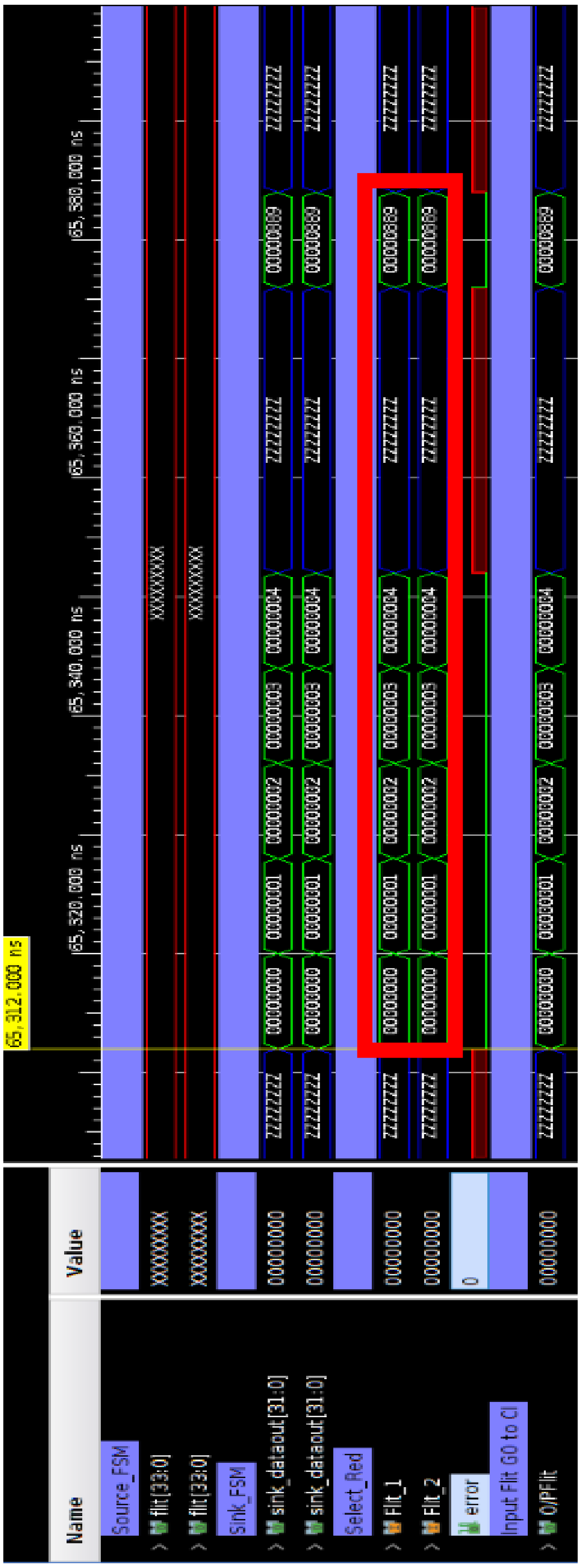
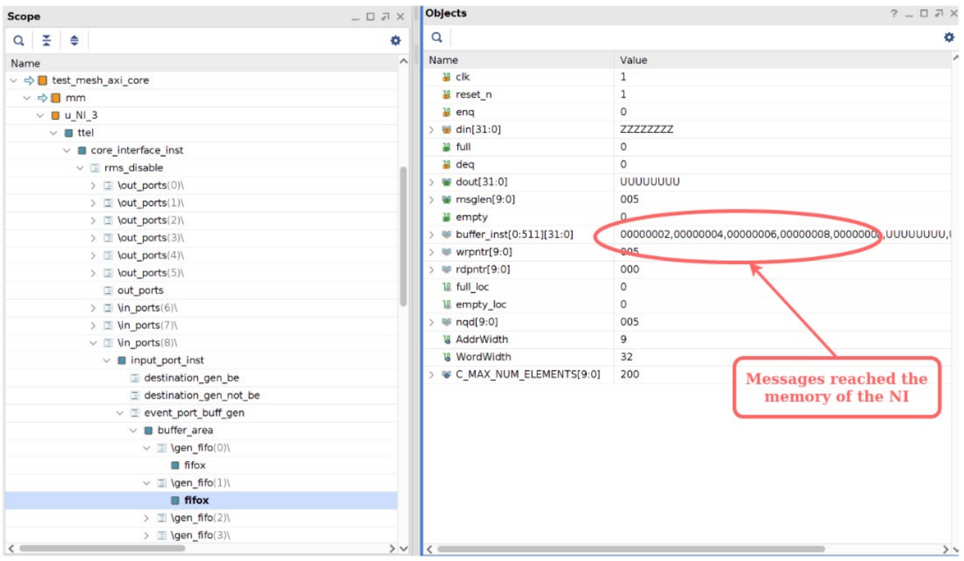


Figure 32 Simulation Result of Scenario 1, NI receiver

Figure 31, and Figure 32 showed the waveform of the NI when there is no presence of fault within NoC. Figure 31, represents the behavior of the messages in the NI sender, and Figure 32 represents the behavior of the received messages in the NI receiver. As shown in Figure 31, the NI sender duplicates the messages (Figure 31, under "source_FSM", the flits are duplicated on the NI sender). Figure 32 showed that both messages reached the targeted NI (Figure 32, under "Select_Red", both messages reached the NI as seen in both signals "Flit_1, and Flit_2"). In the waveform, in Figure 32 under "Select_Red", both data from Signal "Flit_1, and Flit_2" are the same. As shown in the waveform (In Figure 32 under "Select_Red"), the signal "error" equal to zero, indicates that both messages are not corrupted, and reached the NI. Figure 33, represents the successful original message returned on the buffer of the core interfaces of NI.



```
Name
test_mesh_axi_core
  mm
    u_NI_3
      ttel
        core_interface_inst
          rms_disable
            \out_ports(0)\
            \out_ports(1)\
            \out_ports(2)\
            \out_ports(3)\
            \out_ports(4)\
            \out_ports(5)\
            out_ports
            \in_ports(6)\
            \in_ports(7)\
            \in_ports(8)\
              input_port_inst
                destination_gen_be
                destination_gen_not_be
                event_port_buff_gen
                  buffer_area
                    \gen_fifo(0)\
                      fifox
                    \gen_fifo(1)\
                      fifox
                    \gen_fifo(2)\
                    \gen_fifo(3)\
```

Name	Value
clk	1
reset_n	1
enq	0
din[31:0]	ZZZZZZZZ
full	0
deq	0
dout[31:0]	UUUUUUUU
msglen[9:0]	005
empty	0
buffer_inst[0:511][31:0]	00000002,00000004,00000006,00000008,0000000[illegible],UUUUUUUU,
wrpntr[9:0]	005
rdpntr[9:0]	000
full_loc	0
empty_loc	0
nqd[9:0]	005
AddrWidth	9
WordWidth	32
C_MAX_NUM_ELEMENTS[9:0]	200

Figure 33 Messages return to the core Interface

Scenario 2: Redundancy with Errors or Failures

Failure (corrupted coming data) is introduced in one of the routers to evaluate the behavior of TTNoC with a seamless redundancy mechanism, as shown in Figure 34.

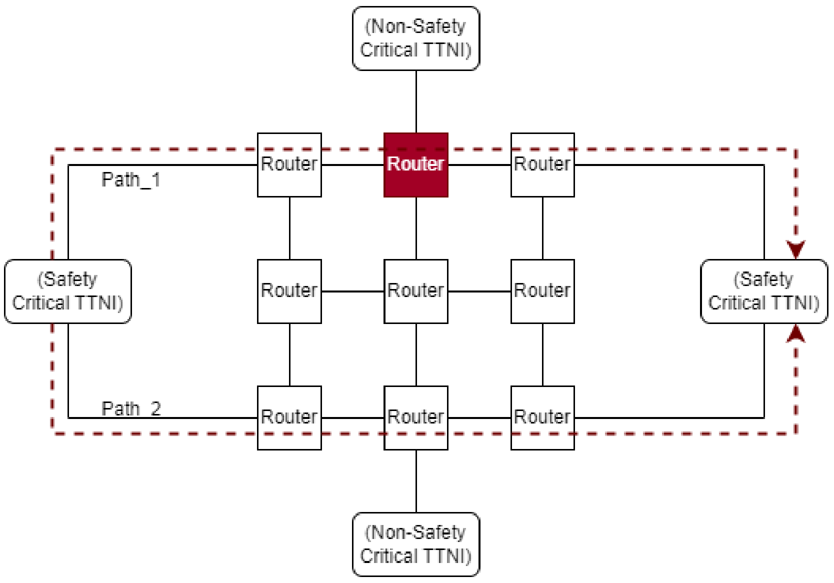


Figure 34 Seamless redundancy in TTNoC with faulty Router

Figure 35 and Figure 36, shows the results of the behavioral simulation of TTNoC with failures that occur in a router.

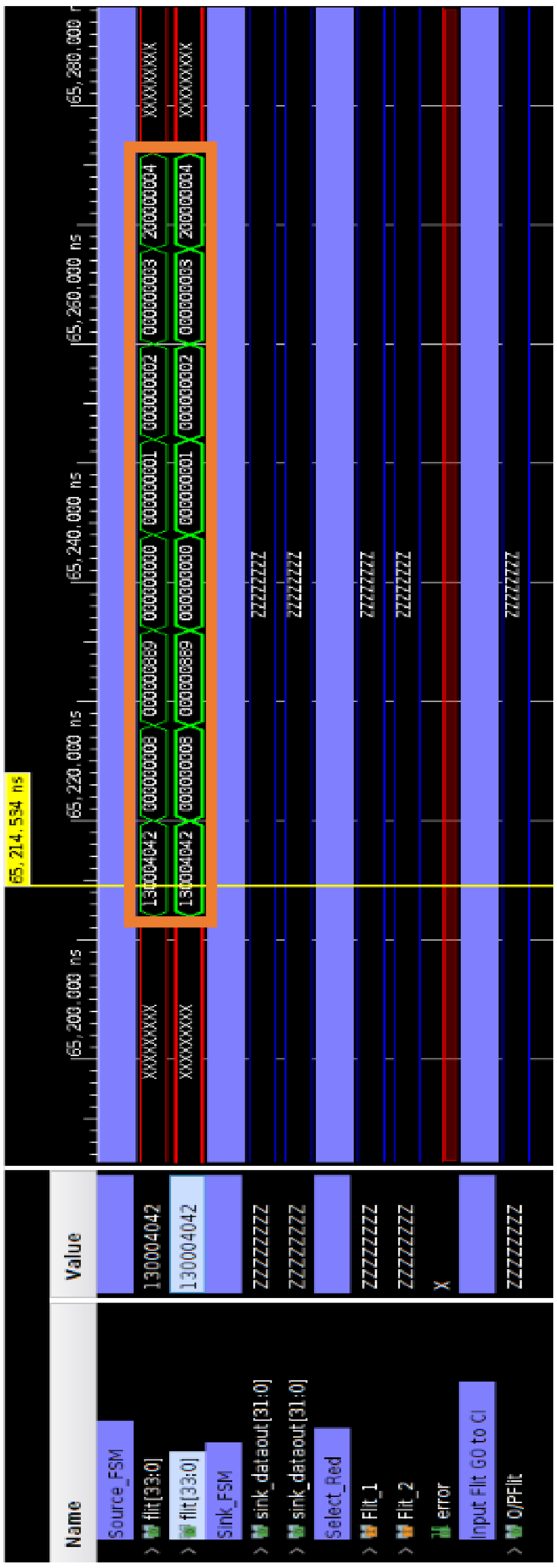


Figure 35 Simulation result of Scenario 2, NI Sender

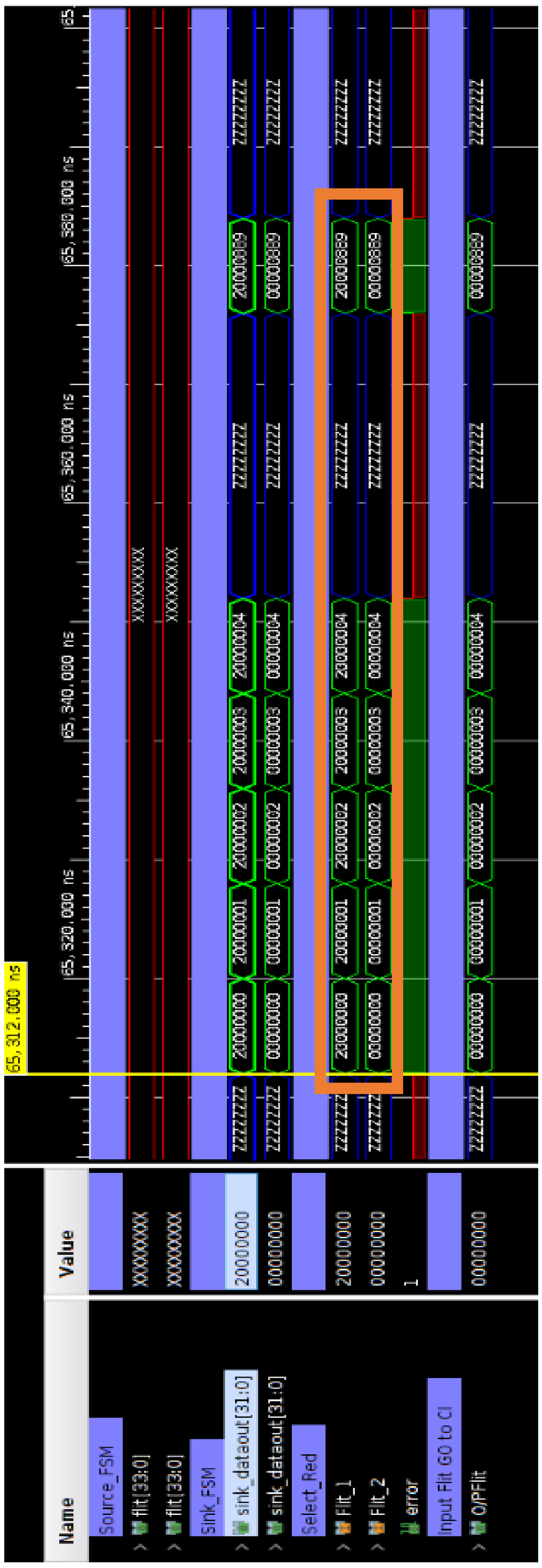


Figure 36 Simulation result of Scenario 2, NI Receiver

The Figure 35 and Figure 36, showed the waveform of the NI. Figure 35, represent the behavior of the messages in the NI sender, and the Figure 36, represent the behavior of the received messages in the NI receiver. As shown in Figure 35, the NI sender duplicate the messages (Figure 35, under "source_FSM", the flits are duplicated on the NI sender). The Figure 36 showed that, the both messages reached the targeted NI (Figure 36, under "Select_Red", the both messages reached the NI as seen in the both signal "Flit_1, and Flit_2"). As seen, in the waveform, in the Figure 36 under "Select_Red", the both data from Signal "Flit_1, and Flit_2" are different. The reason for this is due to the faulty routers, as depicted in Figure 34. However, the "Redundancy controller" select the original messages based on the schedule, and here in Figure 36, we can see that, the signal "O/PFlit" which is the output signal of the redundancy controller (module responsible to select the both messages at the receiver NI) gives the non-corrupted (original) messages.

[SIEG-ATTNoC1] Obermaisser, R., Ahmadian, H., Maleki, A., Bebawy, Y., Lenz, A., & Sorkhpour, B. (2019). Adaptive time-triggered multi-core architecture. *Designs*, *3*(1), 7.

6 Software diverse redundancy library (BSC)

6.1 Component description

Dual Core LockStep (DCLS) is generally needed by high-integrity applications requiring support to make residual the risk of a single fault leading the system to a failure. While redundancy is effective against many fault types, redundancy on its own is not enough if a fault affects redundant components similarly. For instance, two redundant cores executing the same program synchronized could experience a fault in their common clock input signal or power supply, which could lead both of them to the same erroneous output, which would not be detected employing comparison. DCLS imposes staggered execution across redundant cores (e.g., 2-3 cycles of delay for one of the cores w.r.t. the other). Upon a common fault, both cores have a different internal state and experience different errors that can be detected through comparison. However, as discussed in our past work **[BSC-SDR1]**, such an approach is generally too expensive and inflexible since redundant cores cannot be used independently, hence wasting half of the potential performance of the platform even if DCLS is not needed.

Recently, we have proposed a solution based on a software monitor able to enforce staggered execution and hence diversity across redundant user processes (see **[BSC-SDR1]**)**.** However, such a solution has only been prototyped to prove the feasibility of the approach, but a standard interface has not been offered, relieving end-users from the burden of having to replicate input data, create redundant processes and compare results.

This section presents software diverse redundancy library component, our library implementing diverse redundancy support with software-only means, hence compatible with any multicore lacking native DCLS, that relieves end users from the burden to manage the process. As shown in this section, such a library is feasible, provides a standard interface for end-users, and is successfully deployed on a Linux-based RISC-V multicore – it was already proven compatible with ARM multicores in **[BSC-SDR1]**. Such a library has already been tested on the RISC-V multicore and, it will be offered as an open-source component with a permissive license in https://bsccaos.github.io/ along with a number of already public safety-related components.

6.2 Design and implementation

The concept. WP4T43-03 implements a software monitor able to keep a given staggering between two redundant processes. The staggering is measured in the number of instructions, and it is a parameter for WP4T43-03. The monitor checks the staggering across the head and trail processes at a given frequency – also a WP4T43-03 parameter – and, if the current staggering is below the corresponding threshold, the monitor stops the trail process. If the staggering is above the threshold and the trail process is stopped, the monitor activates the trail process so that it can resume its execution. Access to the instruction counts and stopping/resuming the trail process is performed using the WP3T34-02 component.

The staggering between the head and tail processes must be sufficiently large so that, if the head process gets completely stalled and the trail one runs at full speed right after the monitor checks that the staggering is enough, the monitor must be in time to stop the trail process in the next checking interval. Such staggering is completely platform dependent since it is determined by the peak performance of the cores in the platform, and the overheads imposed by the operating system to retrieve instruction counts from the head and trail cores to the core where the monitor runs, as well as to stop the trail process if needed. In general, such threshold needs to be determined empirically, but recommendations in **[BSC-SDR1]** provide guidance on how to do it.

```
void matrix_multiply_wrapper(void * argv_input[], void * argv_output[] ){
    int *matA = (int * )argv_input[0];
    int *matB = (int * )argv_input[1];
    int *matC = (int * )argv_output[0];

    int rows = *(int *)argv_input[2];
    int cols = *(int *)argv_input[3];

    matrix_mutilply(matA, matB, matC, rows, cols);
}
```

Figure 37. Matrix multiplication wrapper

The interface. To allow the WP4T43-03 to generate redundant processes replicating input and output data and comparing results, it needs to receive information in a particular format, which requires end-users to create a wrapper for their software to be protected. In particular, such wrapper needs to be invoked with a vector with pointers to the input data and another with pointers to the output data. This is, for instance, illustrated in Figure 37 for the example of matrix multiplication. The wrapper, `matrix_multiply_wrapper`, calls the matrix multiplication function (`matrix_multiply`) unfolding input and output data vectors, as shown in the figure.

The WP4T43-03 library is used as illustrated in the figure below for the example of the matrix multiplication. In particular, the monitor, `protect_default`, needs receiving the following five parameters:

- A pointer to the application wrapper, `matrix_multiply_wrapper` in the example.
- The vector with the pointers to the input data to use, `argv_input`.
- A vector, `input_size`, with the size of each input data item in `argv_input`.
- The vector with the pointers to the output data to produce, `argv_output`.
- A vector, `output_size`, with the size of each output data item in `argv_input`.

Note that if a particular data item is to be used as input/output data, this needs to be managed accordingly by the application wrapper. In this case, as in any other case, WP4T43-03 will create independent copies of the input and output data for each of the redundant wrapper invocations. The wrapper should use pointers as needed to operate on the copy of the data in the output vector. This typically will imply dismissing the data in the input vector and passing the pointer to the output vector to the application as both input and output data.

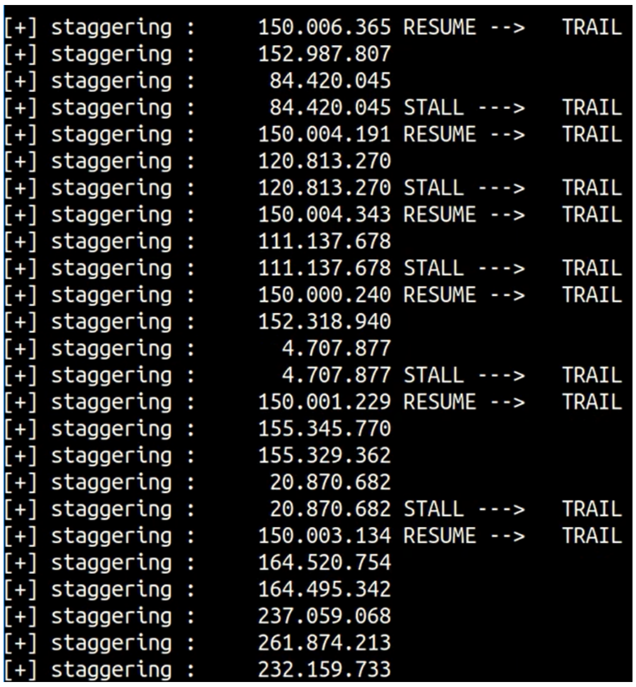
```
void (*ptr)( void *[], void *[]) = &matrix_multiply_wrapper;
void * argv_input[] = { (void *)matA, (void *)matB, (void *)&rows, (void
   *)&cols, NULL};
void * argv_output[] = { (void *)matC, NULL};
int * input_size[] = {(int *)matAbytes, (int *)matBbytes, (int *)&rows-
   bytes, (int *)&colsbytes, NULL};}
int * output_size[] = { (int *)matCbytes, NULL};
bool pass_flag = protect_default(ptr, argv_input, input_size, argv_output,
   output_size) ;
```

Figure 38. Call to the monitor passing as argument the matrix multiplication wrapper

6.3 Testing and evaluation

To showcase WP4T43-03, we have created a demo of this technology in which the matrix multiplication described before is executed with diverse redundancy. In particular, the matrix multiplication has been set sufficiently large so that it takes more than 1 minute to run for the sake of the demo. We have deployed WP4T43-03 on the RISC-V NOEL-V platform. For demo purposes, we print the actual staggering (in number of instructions) every time the monitor is executed in its own core. However, since printing on screen is a slow process subject also to some execution time variability on top of a regular Linux operating system, we have set the staggering threshold to be large enough: 150 million instructions. Note that, in general, staggering should be in the order of 100µs or 1ms, as shown in **[BSC-SDR1]**, which would require staggering between a few hundreds of thousands or a few millions of instructions in most multicores. Nevertheless, WP4T43-03 is agnostic to the actual threshold provided and, if set too short, it could be the case that negative staggering values were observed, meaning that the trail process caught up with the head one so that diversity could be lost.

Figure 39, shows the staggering observed during some consecutive executions of the WP4T43-03 monitor (`protect_default` function). We observe that, in the third interval, the staggering drops down to ~84M instructions. Then, the monitor stalls the trail process. For some intervals, the staggering remains below the threshold so that trail process remains stalled (staggering not plotted). Eventually, the staggering is sufficient (5th line in the figure), and the trail process resumed. In the next interval, the staggering falls below the threshold again, and the trail process is stalled. This sequence of events repeats several times until, after the last time the trail process is resumed in the figure, the staggering starts growing and stabilizing above 200 million of instructions. Hence, during that period, no trail process stalling occurs. Overall, our example illustrates how WP4T43-03 monitors the progress of both processes created and enforces staggering.



```
[+] staggering :     150.006.365 RESUME -->   TRAIL
[+] staggering :     152.987.807
[+] staggering :      84.420.045
[+] staggering :      84.420.045 STALL --->   TRAIL
[+] staggering :     150.004.191 RESUME -->   TRAIL
[+] staggering :     120.813.270
[+] staggering :     120.813.270 STALL --->   TRAIL
[+] staggering :     150.004.343 RESUME -->   TRAIL
[+] staggering :     111.137.678
[+] staggering :     111.137.678 STALL --->   TRAIL
[+] staggering :     150.000.240 RESUME -->   TRAIL
[+] staggering :     152.318.940
[+] staggering :       4.707.877
[+] staggering :       4.707.877 STALL --->   TRAIL
[+] staggering :     150.001.229 RESUME -->   TRAIL
[+] staggering :     155.345.770
[+] staggering :     155.329.362
[+] staggering :      20.870.682
[+] staggering :      20.870.682 STALL --->   TRAIL
[+] staggering :     150.003.134 RESUME -->   TRAIL
[+] staggering :     164.520.754
[+] staggering :     164.495.342
[+] staggering :     237.059.068
[+] staggering :     261.874.213
[+] staggering :     232.159.733
```

Figure 39. Example of use of WP4T43-03

Summary. Enabling diverse redundancy on COTS processors becomes increasingly important to meet the safety requirements of high-integrity applications on platforms delivering enough performance. While we provided proof of concept of a feasible software-only solution recently, it was just prototyped for a handcrafted example. In this section, we present a standard interface offered in the form of a user-friendly library: the WP4T43-03 component. We plan to offer this library as an open-source component in the forthcoming months.

[BSC-SDR1] S. Alcaide, L. Kosmidis, C. Hernandez and J. Abella, "Software-only based Diverse Redundancy for ASIL-D Automotive Applications on Embedded HPC Platforms," 2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2020, pp. 1-4, doi: 10.1109/DFT50435.2020.9250750.

7 Performance monitoring services (BSC)

7.1 Component description

Performance monitoring is a key tool to diagnose abnormal timing behavior and optimize it. Often, processors provide per-core performance monitoring units (or statistics units) able to count a number of events related to the core itself such as instructions executed, execution cycles, instruction breakdowns, cache hits and misses, stall cycles in different buffers, and the like. Some of those events are useful to diagnose timing misbehavior, which in the case of time-critical systems is particularly relevant if related to inter-task interference. However, such events often provide insufficient information for a precise diagnostic since they neither provide detailed information about the access to shared interfaces, nor about the mutual interference that different cores can cause on each other.

This section presents WP4T43-01 component, our service for timing monitoring for time-critical tasks running in edge devices. In particular, this service builds on the edge-oriented monitoring unit presented in WP3 (WP3T31-01 component) and its driver (WP3T34-01 component). WP3T43-01 is a tiny layer whose operation consists of configuring WP3T31-01 component, and use it to collect specific statistics relevant for timing interference by interfacing such monitoring unit with its driver, WP3T34-01.

7.2 Design and implementation

The concept. This service builds upon configuring WP3T31-01 to measure a specific set of events, typically related to the core where the time-critical task is intended to run. Then, the unit is reset and enabled right before the critical task whose behavior needs monitoring is about to run. When the task execution finishes, event counters are read and system level decisions on the user application side could be taken based on the amount of shared resource access performed and timing interference experienced.

Typically, such service is planned to be linked to tasks executing periodically (e.g., every 100ms) with the aim of detecting abnormal timing behavior during operation, and/or for optimization purposes during system design by, for instance, rescheduling conflicting tasks producing high mutual contention. Alternative implementations of the service based on interrupts are also feasible and offer higher flexibility, but their integration in the corresponding use case is more challenging.

```
pmu_counters_disable();
pmu_register_events(crossbar_event_table, EVENT_COUNT);
pmu_counters_reset();
pmu_counters_enable();

/* END USER FUNCTION INSERTED HERE */
foo();

pmu_counters_disable();
pmu_counters_print(crossbar_event_table, EVENT_COUNT);
```

Figure 40. Code sketch of the time monitoring service

Figure 40, shows the skeleton of the WP4T43-01 component. As shown, the hardware unit (WP3T31-01) is first disabled with the appropriate driver call, `pmu_counters_disable`, which is part of the driver (WP3T34-01). Once the unit is disabled, we configure it setting the appropriate events to be counted (`pmu_registers_events`), whose configuration is passed as parameters with the struct `crossbar_event_table` (see an example in Figure 41. Example of statistics unit configuration). Then, counters are reset and the unit enabled to start counting. At this point, everything is ready to run the time-critical user task to be monitored (`foo` in the figure above). Eventually, when the user task ends its execution, the unit is disabled to stop counting further activity, and counters can be either read or, as shown in the figure, directly printed.

```
const crossbar_event_t crossbar_event_table [] = {
    {CROSSBAR_OUTPUT_1,     EVENT_0,
        "\033[0;34m Constant High (Clock cycles) \033[0m"},
    {CROSSBAR_OUTPUT_2,     EVENT_6,
        "\033[0;32m Core 0: Data cache miss \033[0m"},
    {CROSSBAR_OUTPUT_3,     EVENT_2,
        "\033[0;34m Instruction count Core 0 pipeline 0 \033[0m"},
    {CROSSBAR_OUTPUT_4,     EVENT_4,
        "\033[0;34m Core 0: Instruction cache miss \033[0m"},
    {CROSSBAR_OUTPUT_5,     EVENT_13,
        "\033[0;32m Core 1: Data cache miss \033[0m"},
    {CROSSBAR_OUTPUT_6,     EVENT_49,
        "\033[0;33m Contention caused to core 0 due to core 1 AHB accesses \033[0m"},
    {CROSSBAR_OUTPUT_7,     EVENT_20,
        "\033[0;32m Core 2: Data cache miss \033[0m"},
    {CROSSBAR_OUTPUT_8,     EVENT_54,
        "\033[0;33m Contention caused to core 0 due to core 2 AHB accesses \033[0m"},
    {CROSSBAR_OUTPUT_9,     EVENT_27,
        "\033[0;32m Core 3: Data cache miss \033[0m"},
    {CROSSBAR_OUTPUT_10,    EVENT_59,
        "\033[0;33m Contention caused to core 0 due to core 3 AHB accesses \033[0m"},
    {CROSSBAR_OUTPUT_11,    EVENT_34,
        "\033[0;32m Core 4: Data cache miss \033[0m"},
    {CROSSBAR_OUTPUT_12,    EVENT_64,
        "\033[0;33m Contention caused to core 0 due to core 4 AHB accesses \033[0m"},
    {CROSSBAR_OUTPUT_13,    EVENT_41,
        "\033[0;32m Core 5: Data cache miss \033[0m"},
    {CROSSBAR_OUTPUT_14,    EVENT_69,
        "\033[0;33m Contention caused to core 0 due to core 5 AHB accesses \033[0m"}
    };
```

Figure 41. Example of statistics unit configuration

Note that the service is a very tiny layer keeping the integration with the user application simple and easy, while hiding all complexity into the driver.

Configuration. For completeness, Figure 41 shows an example of the configuration of the WP3T31-01 component (`crossbar_event_table` struct). The configuration is set by indicating how each element of the input crossbar of the unit (first element of each triplet) is mapped to a specific event counter (second element of each triplet). The third element of the triplet is the textual description of the event set for debugging purposes. The number of potential events that can be monitored is large, including cache events, bus access events, contention cycles, etc. and they can be counter on a per core basis, and on a per core pair basis when they involve two cores (e.g., contention caused by core A on core B).

7.3 Testing and evaluation

Internal testing has consisted of the execution of a battery of tests with multicore workloads whose behavior was known beforehand, e.g., combining the execution of sensitive and non-sensitive to interference tasks to measure their behavior when running against tasks creating null, medium and high interference. Apart from the combinations of different benchmarks, we had to repeat each experiment multiple times to collect results about different events since the number of possible events is much larger than the number of counters set able to count them. Overall, several hundred experiments were run.

In all cases, results for the counters matched expectations. While those batteries of tests are huge, in Figure 42, we show an example of the outcome of 2 tests for illustrative purposes. We first show the values of the counters just after reseting them, after the first experiment without any type of interference, and after the second experiment with moderate interference.

```
* Before enabling counters:
PMU_COUNTER[ 0] =          0     Constant High (Clock cycles)
PMU_COUNTER[ 1] =          0     Core 0: Data cache miss
PMU_COUNTER[ 2] =          0     Instruction count Core 0 pipeline 0
PMU_COUNTER[ 3] =          0     Core 0: Instruction cache miss
PMU_COUNTER[ 4] =          0     Core 1: Data cache miss
PMU_COUNTER[ 5] =          0     Contention caused to core 0 due to core 1 AHB accesses
PMU_COUNTER[ 6] =          0     Core 2: Data cache miss
PMU_COUNTER[ 7] =          0     Contention caused to core 0 due to core 2 AHB accesses
PMU_COUNTER[ 8] =          0     Core 3: Data cache miss
PMU_COUNTER[ 9] =          0     Contention caused to core 0 due to core 3 AHB accesses
PMU_COUNTER[10] =          0     Core 4: Data cache miss
PMU_COUNTER[11] =          0     Contention caused to core 0 due to core 4 AHB accesses
PMU_COUNTER[12] =          0     Core 5: Data cache miss
PMU_COUNTER[13] =          0     Contention caused to core 0 due to core 5 AHB accesses
==========
Experiment "(no ub)"
PMU_COUNTER[ 0] =      49558     Constant High (Clock cycles)
PMU_COUNTER[ 1] =         96     Core 0: Data cache miss
PMU_COUNTER[ 2] =       7058     Instruction count Core 0 pipeline 0
PMU_COUNTER[ 3] =        224     Core 0: Instruction cache miss
PMU_COUNTER[ 4] =        135     Core 1: Data cache miss
PMU_COUNTER[ 5] =          0     Contention caused to core 0 due to core 1 AHB accesses
PMU_COUNTER[ 6] =        161     Core 2: Data cache miss
PMU_COUNTER[ 7] =          0     Contention caused to core 0 due to core 2 AHB accesses
PMU_COUNTER[ 8] =         79     Core 3: Data cache miss
PMU_COUNTER[ 9] =          0     Contention caused to core 0 due to core 3 AHB accesses
PMU_COUNTER[10] =        147     Core 4: Data cache miss
PMU_COUNTER[11] =          0     Contention caused to core 0 due to core 4 AHB accesses
PMU_COUNTER[12] =        147     Core 5: Data cache miss
PMU_COUNTER[13] =          0     Contention caused to core 0 due to core 5 AHB accesses
-- End of current test
==========
Experiment "ub_ld_l1hit"
PMU_COUNTER[ 0] =      80218     Constant High (Clock cycles)
PMU_COUNTER[ 1] =        273     Core 0: Data cache miss
PMU_COUNTER[ 2] =      14158     Instruction count Core 0 pipeline 0
PMU_COUNTER[ 3] =        457     Core 0: Instruction cache miss
PMU_COUNTER[ 4] =        223     Core 1: Data cache miss
PMU_COUNTER[ 5] =          0     Contention caused to core 0 due to core 1 AHB accesses
PMU_COUNTER[ 6] =        238     Core 2: Data cache miss
PMU_COUNTER[ 7] =          0     Contention caused to core 0 due to core 2 AHB accesses
PMU_COUNTER[ 8] =        240     Core 3: Data cache miss
PMU_COUNTER[ 9] =          0     Contention caused to core 0 due to core 3 AHB accesses
PMU_COUNTER[10] =        219     Core 4: Data cache miss
PMU_COUNTER[11] =          0     Contention caused to core 0 due to core 4 AHB accesses
PMU_COUNTER[12] =        219     Core 5: Data cache miss
PMU_COUNTER[13] =          0     Contention caused to core 0 due to core 5 AHB accesses
-- End of current test
```

Figure 42. Example trace of validation tests for the timing monitoring service

8 Safety services for PULP systems (ETH)

8.1 Component description

PULP-based systems offer powerful and efficient systems as the basis for IoT end nodes. To ensure safe operation, a variety of reliability features have been integrated into the existing platform, ensuring its correct operation, even in hazardous and critical environments. These features are outlined below and include Error-Correcting Codes to protect memory within the SoC, and a Watchdog. Furthermore, reliable processing within the RISC-V cores can be ensured with a Triple-core lock-step implementation, capable of correcting any faults in a single core, if needed.

8.2 Design and implementation

ECC Memory. To reliably protect against errors in the memory, due to radiation or hardware faults, Error-Correcting Codes (ECC) are implemented for all SRAMs within the SoC.

To efficiently encode and decode data in hardware, Hsiao **[ETHZ-PULP1]** codes are used. These require minimal overhead both in area and timing to allow single error correction and double error detection (SECDED) within the protected section. As the protected section, a full 32-bit word is used, requiring 7 additional bits to protect the data, resulting in a bit width of 39 for the memory words, both in the larger SoC memory, as well as the fast, tightly-coupled memory.

As the PULP architecture allows for sub-word access to memory, the ECC encoder needs to be augmented to support partial writes, correctly modifying the correction bits. The hardware encoder at the memory bank loads the stored word and recomputes the correction bits based on all bits within the word, as shown in Figure 43. While this requires an extra cycle, this is implemented to delay the subsequent access to the memory bank, reducing the overhead.

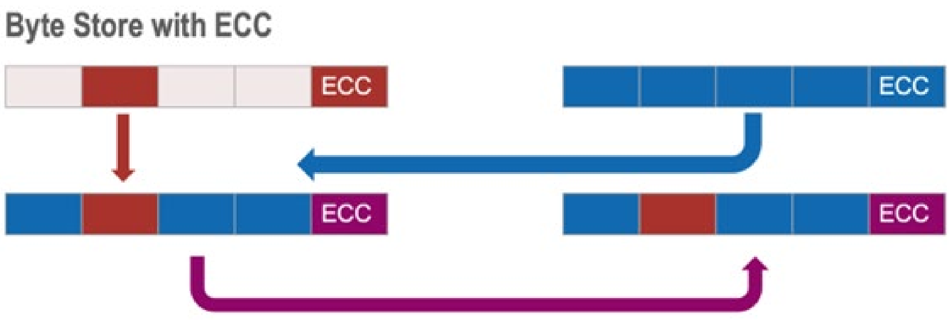


Figure 43 Byte store with ECC

Furthermore, to avoid latent errors in the system, a hardware scrubber is also implemented, continually scanning the memory bank and correcting any correctable errors found.

ODRG. To ensure correct execution in the presence of errors, a PULP-based system is augmented with On-Demand Redundancy Grouping (ODRG) **[ETHZ-PULP2]**.

While multicore systems usually use cores in parallel to process different data more quickly, they can also be used to process identical data, subsequently comparing their results to detect potential computation errors. ODRG adds architectural modifications to allow three cores to work in parallel on the same data. It uses a majority voter to detect differences in their results and recover from potential errors. Figure 44 shows an ODRG architecture where three cores are grouped into a single ODRG unit. Such a unit provides two different operation modes: (i) redundant mode and (ii) performance mode. The ODRG unit is available open-source at https://github.com/pulp-platform/redundancy_cells.

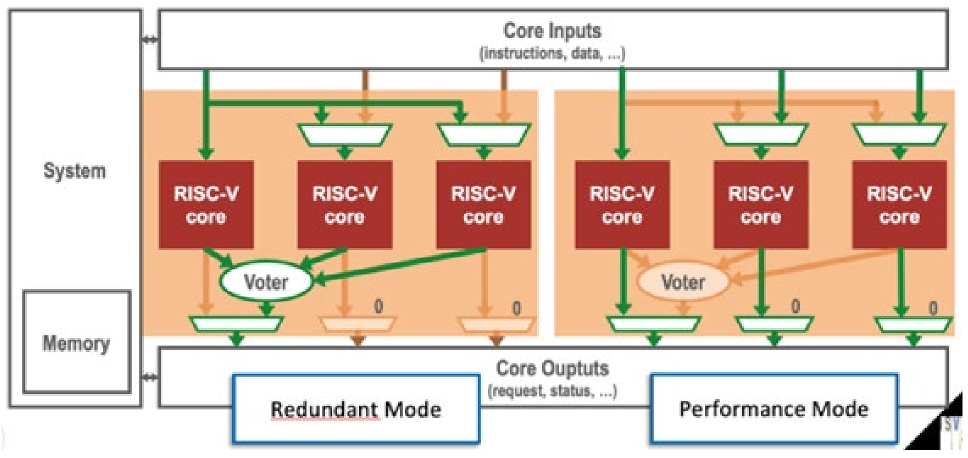


Figure 44 On-Demand Redundancy Grouping : two different operation mode available (i) Redundant mode , (ii) Performance mode

The redundant mode trades performance for reliability features. In such a configuration, the three cores contained in the ODRG group receive the same inputs. Their outputs are then compared bit by bit using a majority voter to detect and correct potential errors. As mismatches indicate different internal states of the cores, once an error is detected, the ODRG unit triggers a re-synchronization sequence managed by an FSM. The FSM sends an interrupt to all the cores. It saves the internal state of the core, copying all internal registers, including relevant CSRs and the entire register file, excluding the stack pointer which is saved inside the ODRG unit. During this phase, the majority voter is active, allowing correcting internal errors within a core. Afterward, the corrected internal states are reloaded, and execution restarts.

The performance mode allows the three cores to operate independently, enabling higher performance as each core can process different data.

Watchdog. To recover from any uncaught errors, a watchdog was designed for PULP, complying with the RISC-V draft specification for watchdogs **[ETHZ-PULP3]**. The base specification for a two-stage watchdog was implemented, with additional features, namely a prescaler for configurable timeout duration, multiple clock inputs to avoid unreliable clocks, and external warnings to speed up timeout when other elements detect an error. This memory-mapped watchdog is integrated into PULP and warns the fabric controller using an interrupt.

8.3 Testing and evaluation

ECC Memory. The ECC memory encoder was implemented in the PULP system and tested in simulation. Using a variety of memory intensive compute benchmarks, it was found that the overhead in execution time due to additional cycles for sub-word accesses remained below 1%, even for an 8-bit matrix-matrix multiplication. Analyzing a representative compute benchmark (Coremark), it is found that around 90% of store operations are word stores, which do not suffer any penalty.

Furthermore, the ECC memory and scrubber were verified to correct errors in simulations with fault injection.

Synthesizing the design showed limited impact in timing for the memory within the PULP cluster, with a 22% overhead in area for the individual memory banks due to the additional stored bits.

ODRG. The ODRG-protected PULP cluster was verified in simulation using fault injection to show corrective behavior. Once an error occurred, re-synchronization required around 700 cycles. Switching modes from soft-error tolerant to performance showed a speedup of up to 2.96x in tasks such as matrix-matrix multiplication, requiring around 40’000 cycles to switch.

Synthesizing the PULP cluster with ODRG implemented, limited impact in timing was observed, with a 1% increase in overall area for the protection unit.

[ETHZ-PULP1] M. Y. Hsiao, “A Class of Optimal Minimum Odd-weight-column SEC-DED Codes,” IBM Journal of Research and Development, vol. 14, no. 4, pp. 395–401, Jul. 1970, doi: 10.1147/rd.144.0395.

[ETHZ-PULP2] M. Rogenmoser, N. Wistoff, P. Vogel, F. Gürkaynak, and L. Benini, “On-Demand Redundancy Grouping: Selectable Soft-Error Tolerance for a Multicore Cluster,” arXiv, arXiv:2205.12580, May 2022. doi: 10.48550/arXiv.2205.12580.

[ETHZ-PULP3] https://github.com/riscv-non-isa/riscv-watchdog

9 FPGA-based Fault Injection (UPV)

9.1 Component description

FPGA-based fault injection (FFI) is a technique for dependability assessment and verification of FPGA designs. It consists in analyzing the behaviour of FPGA prototype under the deliberate introduction of faults into that prototype. In the framework of FRACTAL project FFI experiments are carried out by means of the DAVOS-FFI tool. This tool forms a part of DAVOS, an open-source fault injection toolkit (developed by the UPV) that is publicly available at **[UPV-FFI1].** In particular, DAVOS-FFI tool supports following dependability-driven processes:

- Robustness assessment of FPGA prototypes against permanent and transient hardware faults;
- Verification of fault-tolerance mechanisms and safety-related components;
- Identification of vulnerabilities and fault propagation paths;
- Dependability benchmarking of IP cores.

DAVOS FFI tool emulates faults in FPGA prototype by manipulating the content of its configuration memory at runtime, this method is known as dynamic partial reconfiguration (DPR). The main advantage of this approach is low intrusiveness, i.e.the FPGA design under test (DUT) is kept unchanged, as it doesn't require any design modification/instrumentation, which is important for obtaining credible and representative experimental results.

The supported fault models include bit-flips in user memories (registers, Block RAMs, and LUT RAMs), as well as bit-flips in configuration memory (CM). Bit-flips in user memories have transient nature, since after corrupting the stored data or control bits, they can be recovered by the normal circuit operation. Whereas the bit-flips in CM affect the structure of the circuit itself (logic and routing), manifesting at the netlist level as the permanent (in the absence of CM scrubbing) stuck, short, open and bridging. During an FFI experiment faults are randomly sampled from the fault space after the methodology in **[UPV-FFI2]**.

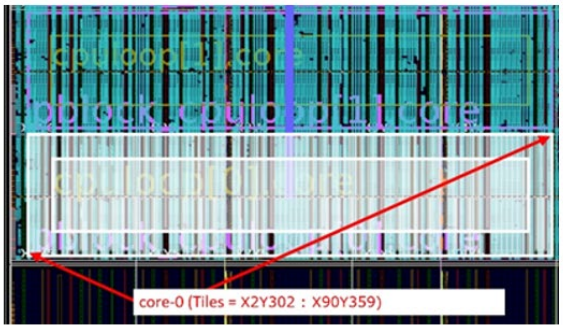
The outcomes of each individual injection run are described in terms of failure modes, which describe all the different ways the DUT may react to the injected fault. Failure modes are determined by tracing the DUT outputs (processing results), its internal state, and auxiliary flags (error detection or alarm signal), and by comparing them to the fault-free trace. The generic failure modes determined after this approach are listed in Table 10. At the end of FFI experiment (after the injection of all sampled faults) several robustness metrics are calculated. These include the percentage of observed failure modes (that are interpreted as the probability of a given fault type to a cause a certain DUT failure), number of critical bits, failure rate, etc.

Table 10 Definition of common failure modes observed during FFI experiments.

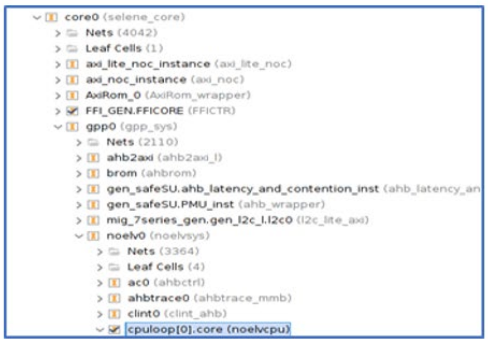
Processing results (outputs)	Internal State (registers)	Alarm signal (error detection)	Failure Mode
✓	✓	—	Masked
✓	X	—	Latent error
X	*	✓	Signaled Failure
X	*	—	Silent Data Corruption
✓	✓	✓	False Alarm
—	*	—	Hang / Crash

Notation: ✓ (valid), X (invalid), — (absent), * (any/wildcard)

To improve the efficiency of FFI experiments, FFI tool targets only the so-called "essential bits", i.e. a subset of CM cells that are actually used in a given bitstream to configure the logic and routing of the circuit in FPGA. In turn, a subset of essential bits that (when flipped) cause a DUT failure, are referred to as critical bits. The number of critical bits in an FPGA design can be estimated by FFI experiment, and consequently can be used to estimate the failure rate of and FPGA prototype in FIT units.



pblock="Core0:Tiles:X2Y302:X90Y359"



```
v core0 (selene_core)
  > Nets (4042)
  > Leaf Cells (1)
  > axi_lite_noc_instance (axi_lite_noc)
  > axi_noc_instance (axi_noc)
  > AxiRom_0 (AxiRom_wrapper)
  > ✓ FFI_GEN.FFICORE (FFICTR)
  v gpp0 (gpp_sys)
    > Nets (2110)
    > ahb2axi (ahb2axi_l)
    > brom (ahbrom)
    > gen_safeSU.ahb_latency_and_contention_inst (ahb_latency_an
    > gen_safeSU.PMU_inst (ahb_wrapper)
    > mig_7series_gen.gen_l2c_l.l2c0 (l2c_lite_axi)
    v noelv0 (noelvsys)
      > Nets (3364)
      > Leaf Cells (4)
      > ac0 (ahbctrl)
      > ahbtrace0 (ahbtrace_mmb)
      > clint0 (clint_ahb)
      v ✓ cpuloop[0].core (noelvcpu)
```

dut_scope= "cpu/core0/gpp0/noelv0/cpuloop[0].core"

Figure 45 Injection modes of the DAVOS-FI tool: area-based (left) and scope/ hierarchy-based (right)

To perform more selective (fine-grained) FFI experiments, DAVOS-FFI tool allows to filter the targeted essential bits by the area, and/or by the design scope, as it is illustrated in Figure 45. The area-based injection mode (shown in the left plot of Figure 45) targets essential bits located within a rectangular area on the FPGA floorplan (pblock); it is configured in terms of the Tile coordinates (Bottom-Left and Top-Right corners). The netlist-based injection mode (shown in the right plot of Figure 45) targets those essential bits that correspond to the selected DUT component (part of the design tree).

9.2 Design and implementation

DAVOS-FFI tool comprises two main components: an on-chip FFI controller at the board side, and an FFI application at the host side, as it is depicted in Figure 46.

The FFI controller is in charge of manipulating the content of the configuration memory through the internal configuration access port (ICAP), in order to inject and remove faults on the request of the host application. This FFI controller is located in the same FPGA chip as the design under test (DUT). This allows direct access to the CM of the target FPGA and minimizes the fault injection latency (time overhead). The controller itself is implemented on the basis of Xilinx's Microblaze IP (a small-footprint soft-core processor), which runs a custom board-side FFI application.

The FFI controller is isolated in a separate region (Pblock) on the FPGA floorplan, in order to prevent unexpected interferences with the DUT, as well as to avoid any side-effects of injected faults on the FFI controller itself. The integration of FFI controller and its floor-planning are automated in the SELENE platform by means of TCL scripting (as a part of the SELENE makefiles).

After initializing the FFI controller, the board-side application enters the control loop in which it awaits and executes the incoming FFI commands. These commands are loaded to the dedicated memory buffer by the host application (through the debug-jtag link of the Xilinx XSCT service). Currently there are four commands recognizable by the board application: (a) flip the value of a given CM bit at a given clock cycle, (b) set a given CM bit (or CM word) to the specified value, (c) recover a given CM bit to its initial value, (d) trigger hard reset signal of the DUT. After executing each command, the board application sends a status message to the host (through the memory buffer), to confirm the successful injection/recovery, or to signal any error of an FFI controller.

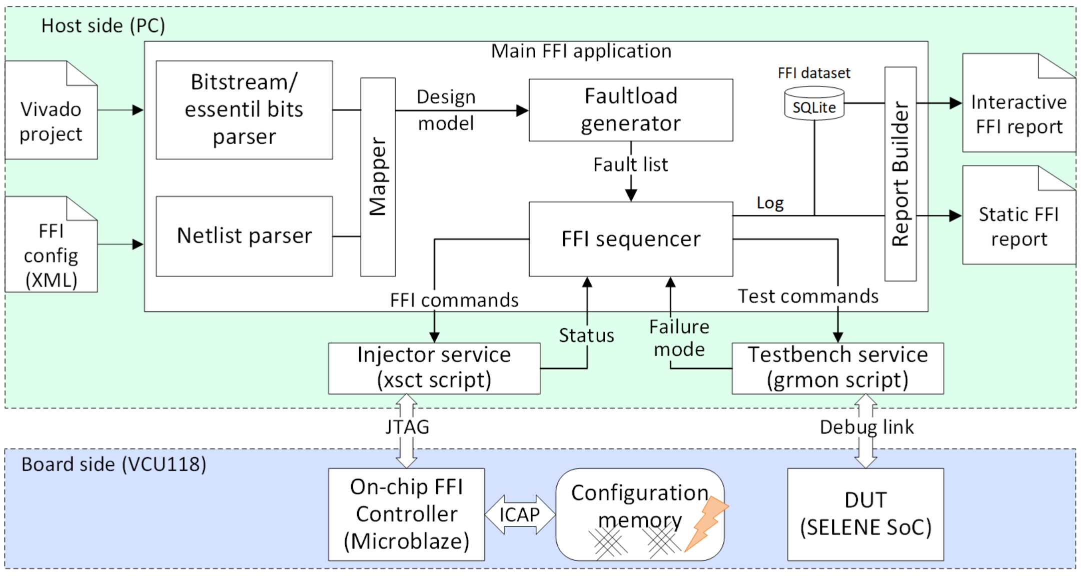


Figure 46 Architecture of the DAVOS-FFI tool

The host side includes the main FFI application and two support services: the injector service and a testbench service.

The injector service implements a communication bridge between the board-side FFI controller and a host application. It translates the inject/recover commands received (through the socket) from the main FFI application into the commands recognized by the FFI controller, submits them via JTAG link, and forwards back the status messages from the FFI controller. It is also in charge of loading the bitstream to the FPGA, initializing the FFI application at the board side, and uploading the fault list to the board side. The injector service is implemented in the form of TCL script for the Xilinx command-line tool (XSCT).

The testbench service is in charge of determining the effects of injected faults on the DUT behavior. After receiving the test command from the main FFI application this service runs the selected workload on the DUT, traces its processing results and its internal state (when detection of latent errors is activated), determines the failure mode by comparing these traces to the reference, and returns the resulting failure mode in the status message to the main FFI application. The testbench service for the SELENE platform is implemented in the form of a TCL script for the GRMON tool.

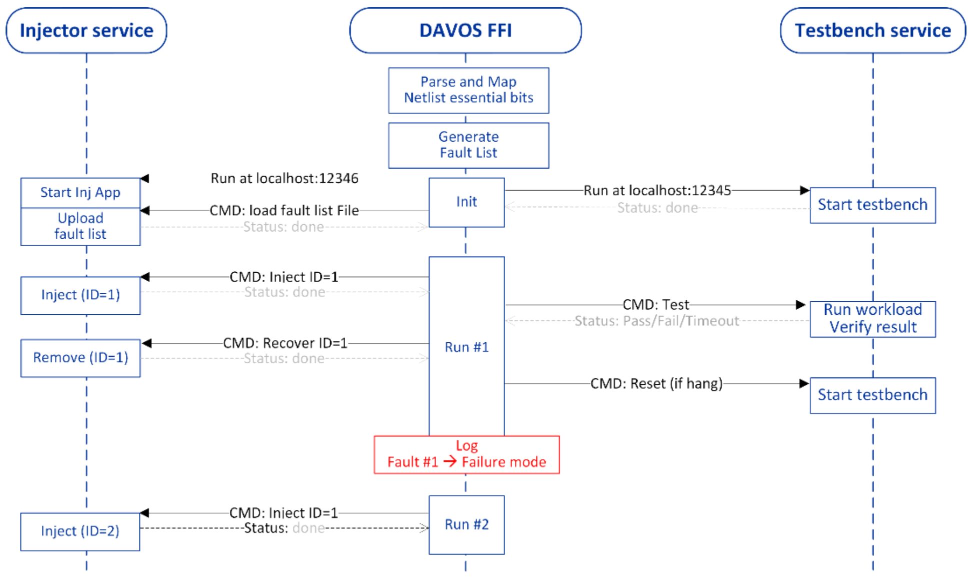


Figure 47 The workflow of DAVOS-FFI tool

The main host-side application is the core of the DAVOS FFI tool. Its workflow (depicted in Figure 47) includes:

- Parsing the design netlist, bitstream (essential bits), and floorplan, and mapping them within an internal DUT model (depicted in Figure 48);
- Locating relevant essential bits attending to the configured filters (floorplan area and hierarchical DUT scope), and generating the fault list;
- Initializing the support services (injector service and testbench service) and establishing a connection to them via sockets;
- Controlling the execution of individual FFI runs:
    - Selecting the next fault configuration from the fault list, and issuing the "inject Fault-ID" command to the injector service,

- Issuing the test command to the testbench service in order to determine the effect of injected fault (failure mode);
- Removal of injected fault (recover command to the injector service);
- Resetting the DUT to its initial state through the testbench service;
- Logging the failure mode of each injection run to the database.

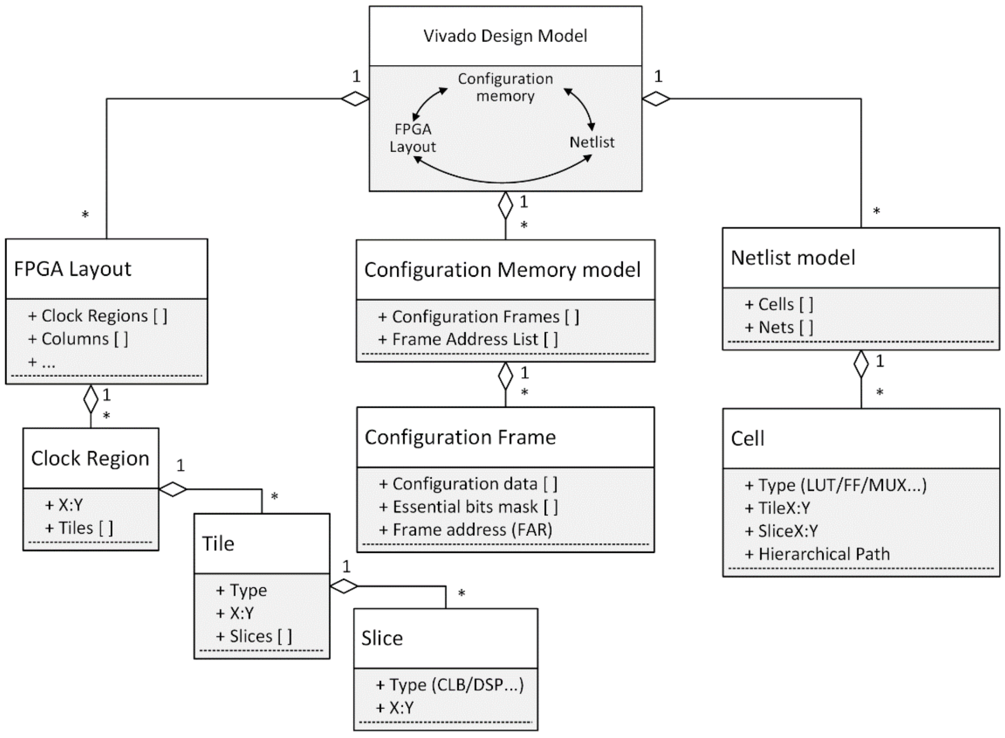


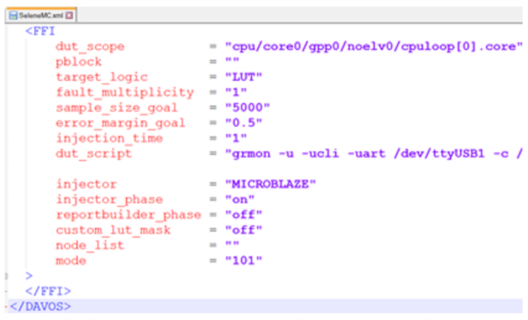
Figure 48 Fragment of FPGA design model used in DAVOS for the mapping of essential bits

9.3 Testing and evaluation

To illustrate the operation of DAVOS-FFI tool two simple FFI experiments have been carried out, both targeting core-0 of the SELENE GPP in two different injection modes:

- Scope-based mode: injecting 5000 bit-flips into the LUTs of the core-0;
- Area-based mode: injecting 5000 bit-flips into the essential bits (TYPE-0 frames covering both logic and routing) of the core-0;

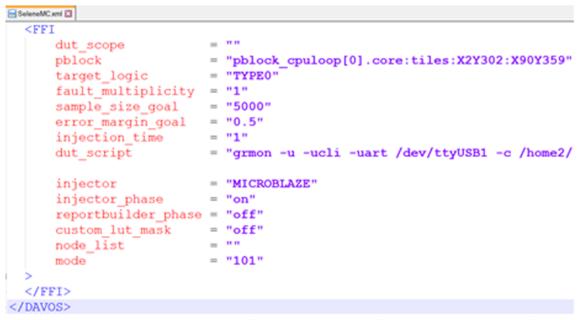
The configuration XML files of the DAVOS-FFI tool for these two experiments are illustrated in Figure 49(a) and Figure 49(b) respectively.



```
SeleneMC.xml
  <FFI
      dut_scope            = "cpu/core0/gpp0/noelv0/cpuloop[0].core"
      pblock               = ""
      target_logic         = "LUT"
      fault_multiplicity   = "1"
      sample_size_goal     = "5000"
      error_margin_goal    = "0.5"
      injection_time       = "1"
      dut_script           = "grmon -u -ucli -uart /dev/ttyUSB1 -c /

      injector             = "MICROBLAZE"
      injector_phase       = "on"
      reportbuilder_phase  = "off"
      custom_lut_mask      = "off"
      node_list            = ""
      mode                 = "101"
  >
  </FFI>
</DAVOS>
```

(a) DAVOS-FFI configuration for injecting bit-flips into LUTs within the design scope of core-0



```
SeleneMC.xml
  <FFI
      dut_scope            = ""
      pblock               = "pblock_cpuloop[0].core:tiles:X2Y302:X90Y359"
      target_logic         = "TYPE0"
      fault_multiplicity   = "1"
      sample_size_goal     = "5000"
      error_margin_goal    = "0.5"
      injection_time       = "1"
      dut_script           = "grmon -u -ucli -uart /dev/ttyUSB1 -c /home2/

      injector             = "MICROBLAZE"
      injector_phase       = "on"
      reportbuilder_phase  = "off"
      custom_lut_mask      = "off"
      node_list            = ""
      mode                 = "101"
  >
  </FFI>
</DAVOS>
```

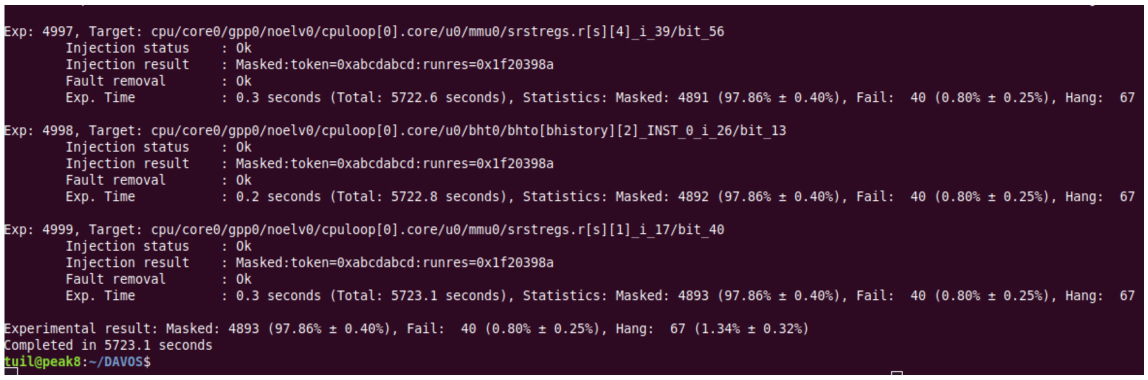
(b) DAVOS-FFI configuration for injecting bit-flips into the TYPE0 CM frames (logic and routing) within the pblock of core-0

Figure 49 Configuration of DAVOS-FFI tool for (a) scope-based injection targeting LUTs of core-a, and for (b) area-based injection targeting essential bits of core-0

The DUT is executing an integer matrix multiplication kernel (running on the core-0), followed by a hash sum calculation for the resulting matrix. The testbench script adapted for the selected workload is available in the DAVOS repository (DAVOS/testconfig/host_grmon.do). This script is linked to the DAVOS configuration in the *dut_script* XML attribute ("-c" argument of the GRMON tool).
After configuring the FFI tool (customizing the testbench script and SeleneMC.xml), the FFI experiment is started by invoking the following command from the DAVOS installation folder:

```
DAVOS/> python FFI_Tool.py testconfig/SeleneMC.xml
```

The first fault injection experiment (scope-based, targeting LUTs) has taken roughly 1.5 hours per 5000 injection runs, as it can be seen from the fragment of an FFI trace in Figure 50. The second experiment (area-based, targeting essential bits) has taken roughly 2.5 hours per 5000 injection runs. The higher run time in the second experiment is explained by the higher number of 'hang' outcomes, which require complete DUT recovery (taking up to 50 seconds when it involves reloading of FPGA bitstream). Whereas injection runs with a 'Masked' outcome are much faster, taking 0.3 seconds on average.



```
Exp: 4997, Target: cpu/core0/gpp0/noelv0/cpuloop[0].core/u0/mmu0/srstregs.r[s][4]_i_39/bit_56
        Injection status    : Ok
        Injection result    : Masked:token=0xabcdabcd:runres=0x1f20398a
        Fault removal       : Ok
        Exp. Time           : 0.3 seconds (Total: 5722.6 seconds), Statistics: Masked: 4891 (97.86% ± 0.40%), Fail:  40 (0.80% ± 0.25%), Hang:  67

Exp: 4998, Target: cpu/core0/gpp0/noelv0/cpuloop[0].core/u0/bht0/bhto[bhistory][2]_INST_0_i_26/bit_13
        Injection status    : Ok
        Injection result    : Masked:token=0xabcdabcd:runres=0x1f20398a
        Fault removal       : Ok
        Exp. Time           : 0.2 seconds (Total: 5722.8 seconds), Statistics: Masked: 4892 (97.86% ± 0.40%), Fail:  40 (0.80% ± 0.25%), Hang:  67

Exp: 4999, Target: cpu/core0/gpp0/noelv0/cpuloop[0].core/u0/mmu0/srstregs.r[s][1]_i_17/bit_40
        Injection status    : Ok
        Injection result    : Masked:token=0xabcdabcd:runres=0x1f20398a
        Fault removal       : Ok
        Exp. Time           : 0.3 seconds (Total: 5723.1 seconds), Statistics: Masked: 4893 (97.86% ± 0.40%), Fail:  40 (0.80% ± 0.25%), Hang:  67

Experimental result: Masked: 4893 (97.86% ± 0.40%), Fail:  40 (0.80% ± 0.25%), Hang:  67 (1.34% ± 0.32%)
Completed in 5723.1 seconds
tuil@peak8:~/DAVOS$
```

Figure 50 Fragment of FFI trace at the end of first experiment (targeting LUTs)

The detailed FFI results are exported into a csv file, and saved into the DAVOS SQLite database. Figure 51 illustrates the resulting distribution of failure modes for both experiments. As it can be seen from the results, the SELENE GPP is more sensitive to the bit-flips in the Type-0 CM bits (essential bits) than to the bit-flips in LUTs. This is explained by the fact that essential bits include configuration of routing resources (among others), and the routing faults (short/open/bridging) are usually more critical than LUT faults.

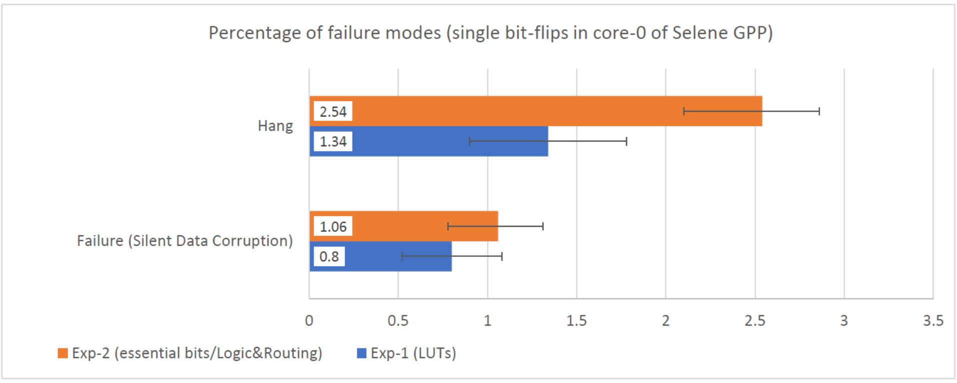
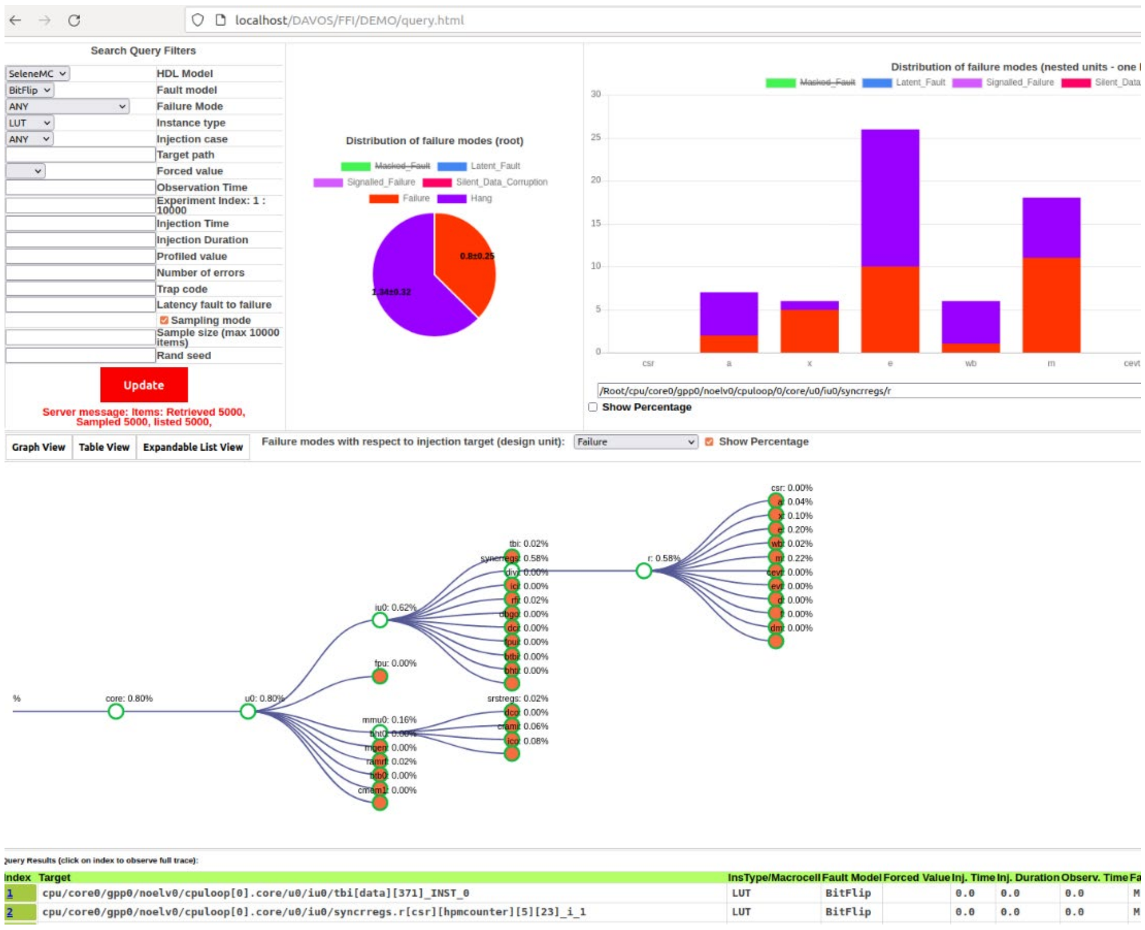


Figure 51 Estimated percentage of failures (SDC and Hangs) resulting from bit-flips in configuration bits of core-0 (LUT in exp-1, and essential bits in exp-2)



Search Query Filters

SeleneMC	HDL Model
BitFlip	Fault model
ANY	Failure Mode
LUT	Instance type
ANY	Injection case
	Target path
	Forced value
	Observation Time
	Experiment Index: 1 : 10000
	Injection Time
	Injection Duration
	Profiled value
	Number of errors
	Trap code
	Latency fault to failure
✓	Sampling mode
	Sample size (max 10000 items)
	Rand seed

Update

Server message: Items: Retrieved 5000, Sampled 5000, listed 5000,

/Root/cpu/core0/gpp0/noelv0/cpuloop/0/core/u0/iu0/syncrregs/r

Show Percentage

Failure modes with respect to injection target (design unit): Failure

✓ Show Percentage

Query Results (click on index to observe full trace):

Index	Target	InsType/Macrocell	Fault Model	Forced Value	Inj. Time	Inj. Duration	Observ. Time	Fa
1	cpu/core0/gpp0/noelv0/cpuloop[0].core/u0/iu0/tbi[data][371]_INST_0	LUT	BitFlip		0.0	0.0	0.0	M
2	cpu/core0/gpp0/noelv0/cpuloop[0].core/u0/iu0/syncrregs.r[csr][hpmcounter][5][23]_i_1	LUT	BitFlip		0.0	0.0	0.0	M

Figure 52 Querying and visualizing FFI results by means of DAVOS web interface

Finally, the results from the DAVOS database can be queried and visualized by means of the DAVOS web interface, as it is depicted in Figure 52. Its top-left widget is used for querying FFI results attending to a set of filters (fault model, targeted logic, etc.). The top-middle widget illustrates the overall percentage of each failure mode (for the entire sample). The Top-Right widget illustrates the contribution of each design sub-module into these percentages, for instance it can be seen that execute (E) and memory access (M) stages of the pipeline are those DUT components that contribute the most to the observed SDC failures. The interactive widget below illustrates the sensitivity of each module in the design tree, highlighting the weak/critical points of the design. Finally, the table on the bottom of the page lists the details of each individual injection run: injection point, injection time, fault model, failure mode, fault to failure latency.

[UPV-FFI1] Universitat Politècnica de València (2022). DAVOS - a fault Injection toolkit for dependability assessment, verification, optimization and selection of hardware designs
[Online] https://gitlab.com/selene-riscv-platform/DAVOS/

[UPV-FFI2] Tuzov I, de Andrés D, Ruiz JC. Accurate robustness assessment of HDL models through iterative statistical fault injection. In 2018 14th European Dependable Computing Conference (EDCC) 2018 Sep 10 (pp. 1-8). IEEE.

10 Safety Analysis of the NOEL-V Platform redundant acceleration scheme (UPV)

10.1 Component description

The redundant acceleration scheme (component WP3T32-06 described in D3.5) provides reliable/fail-safe HW acceleration of convolutional neural networks (CNN). This redundancy scheme (depicted in Figure 53) includes N replicated accelerators, an HW/SW voter, and a control/monitoring process running on the NOELV CPU. The processing is performed in steps. At the beginning of each step the accelerators remain in the idle state, waiting for the availability of input data. The CPU (monitor) loads the next portion of data into the data buffer of each accelerator, adjusts the processing parameters (stored in the memory-mapped control registers), and activates the start flag of each accelerator. While the computation is ongoing, the monitor checks the completion of each accelerator by polling their ready flags. Once the computation is completed, the processing results are read out from each accelerators' data buffer. The correctness of processing results is checked by majority voting in software or in hardware. Finally, the voting results are interpreted by the monitor for the diagnosis of errors and for performing any required corrective actions.

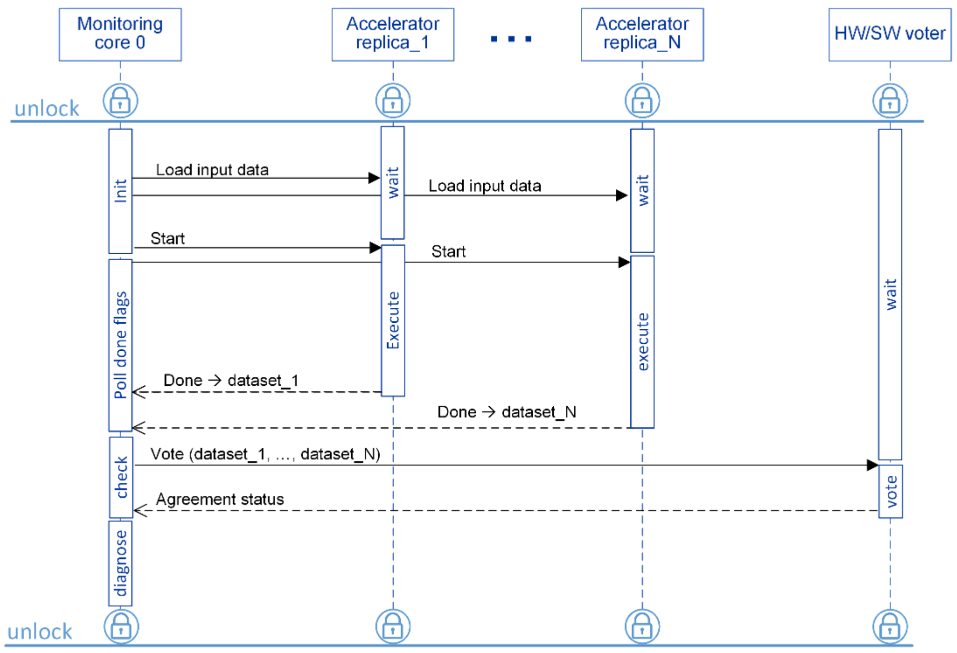


Figure 53 Acceleration subsystem with N-modular redundancy and majority voting

10.2 Design and implementation

One of the main metrics considered by functional safety certification standards (IEC61508, ISO26262) is the probability of dangerous failure (DF). DF is a failure mode in which the safety instrumented system is unable to provide a correct service and cannot reach the safe state. The definition of DF in the case of the redundant accelerator's subsystem is determined by the particular application scenario. For instance, accelerators may be used for computer vision in autonomous driving (e.g., detection of obstacles on the road), in which case any failure of accelerators subsystem may be treated as dangerous.

At the same time, standards also distinguish between detected and undetected DF. Detected DF are diagnosed and alerted by the system, so that the user or higher-level system are aware of system degradation. Undetected DF escape the diagnostics, thus being the most hazardous failures. In such a way, the features of redundant accelerators scheme can be described from the viewpoint of (i) the ability of the system to provide correct service in presence of faults in its components/replicas (fault tolerance), and (ii) the percentage of failures that are detected and alerted (diagnostic coverage). At the same time, these features can also be described with respect to the different fault models, as follows.

10.2.1 **Transient faults of varied multiplicity affecting single replica.**

T1. Transient faults affect the system during a short period of time, being usually abstracted by such fault models as bit-flips in registers, or pulses in combinational logic. Transient faults can be corrected (recovered) by a normal circuit behaviour, or propagated through the system, causing different kinds of failures.

In the non-replicated acceleration model (Table 11) this may cause (a) an undetected failure, such as silent data corruption (SDC), being it the most critical (unsafe) failure mode, or (b) a hang of acceleration subsystem, which is treated as detectable failure (using a common watchdog timer).

In the double modular redundancy with 2oo2 voting model (Table 12) failure of any one of the replicas is detected, providing an alarm signal from the acceleration subsystem as a safety measure. On the failure detection, the failing replica is recovered (reset), while the erroneous result may be skipped, or recomputed if needed by the application.

In the triple modular redundancy with 2oo3 voting model (TMR,Table 13) a failure of any one of the replicas is detected and corrected by majority voting. The TMR scheme is suitable for hard real-time applications, since it imposes little to no time overheads for error correction, albeit at the cost of high area overhead. On the detection of replica failure, the acceleration subsystem itself can be switched to the degraded mode (double modular redundancy) until the failing replica is recovered (reset).

10.2.2 **Multiple transient faults affecting multiple replicas.**

The double modular redundancy scheme allows error detection in both replicas, unless the erroneous result produced by them is exactly the same. This latter situation (same erroneous result) may occur in the case of common-cause faults affecting similarly both replicas. This unsafe failure mode can be avoided by diversification of replicas. In our case this diverse redundancy can be achieved by implementing accelerators with different convolution types, as current CNN accelerators support tree types of convolutions: direct convolution, Winograd's algorithm, and DepthWise Separable convolution. Alternatively, can be employed a time-staggering mechanism, that allows different fault manifestation across replicas even without hardware-level diversification.

A triple modular redundancy mode (TMR), similarly to the dual redundancy, allows detection of errors in multiple replicas, as long as their erroneous results are different. A failure of multiple replicas degrades the TMR scheme to a state in which the real-time correction of results (by majority voting) is not possible. The recovery from this degraded state thus requires to reset all failing replicas and recompute the erroneous result (if required by the application).

Table 11 Diagnosis and recovery actions in the case of single replica

Processing Result	Diagnosis (Failure Mode)	Recovery Action
✓	Correct (masked)	—
X	Silent Failure (Silent Data Corruption)	—
—	Hang (Timeout)	Reset and recompute

Notation: ✓ (correct), X (incorrect), — (timeout)

Table 12 Diagnosis and recovery actions in the case of two replicas

Processing Result		Voter agreement	Diagnosis (Failure Mode)	Recovery Action
Rep1	Rep2			
✓	✓	✓	Correct (masked)	—
*	X	X	Signaled failure	Reset and recompute
X	*	X		
*	—	X	Hang (timeout)	Reset and recompute
—	*	X		

Table 13 Diagnosis recovery actions in the case of three replicas (TMR)

Processing Result			Voter agreement	Diagnosis (Failure Mode)	Recovery Action
Rep1	Rep2	Rep3			
✓	✓	✓	✓	Correct (masked)	—
✓	✓	X/—	✓	Correct, degraded mode (replica fail / timeout)	Reset replica and continue
✓	X/—	✓	✓		
X/—	✓	✓	✓		
✓	X/—	X/—	X	Signaled failure (multiple replicas failure / timeout)	Reset and recompute
X/—	✓	X/—	X		
X/—	X/—	✓	X		

10.2.3 **Permanent faults affecting one or several replicas**

Similarly, to the transient faults, the permanent ones are detectable by both dual and triple modular redundancy (TMR) schemes. A permanent fault affecting a single replica is also correctable by a TMR scheme. However, automatic recovery of a system from a degraded mode would require an additional hot-spare (reserved) component that can be activated instead of the degraded replica. In spite of even higher hardware overheads, this hot spare scheme may be reasonable in some applications with limited manual maintenance (e.g., satellites).

10.3 Testing and evaluation

The aforementioned safety attributes (probability of dangerous failures, and diagnostic coverage) can be estimated by means of FPGA-based fault injection experiments. The DAVOS FFI tool (described in Section 9 of this document) is used for this purpose. The system under study is a double modular redundancy scheme, a 2oo2 adaptation of the acceleration subsystem depicted in Figure 53.

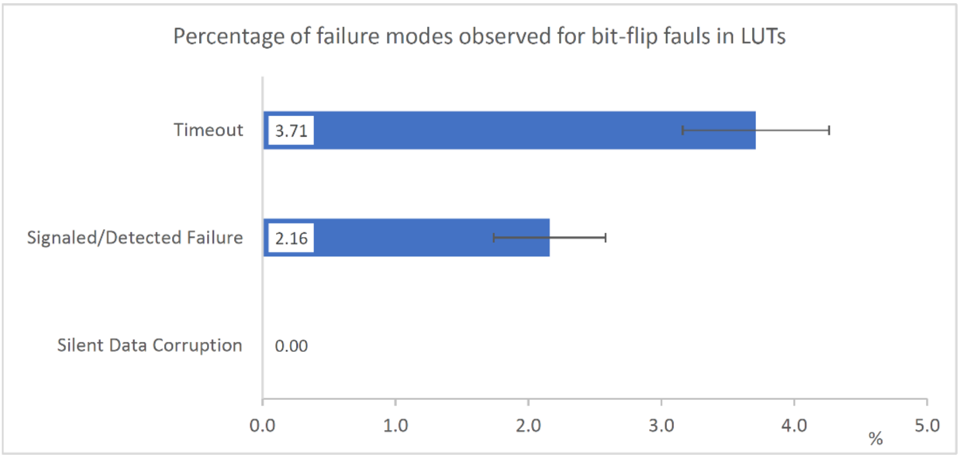


Figure 54 Percentage of failure modes of 2oo2 redundancy scheme estimated by FFI experiments

The workload executed by the accelerators is generated by the NOELV processor on the basis of randomized data. The fault load applied to the DUT comprises a sample of 5000 single bit-flips in the essential bits of accelerator’s subsystem. The observed failure modes (described in Table 12) include (i) masked fault (absence of fault manifestation), (ii) signaled failure, i.e., incorrect processing result alerted as a mismatch between the two replicas by a 2oo2 voter, (iii) hang/timeout of one or both accelerator replicas, and (iv) silent data corruption which is expected to be completely covered (mitigated) by the 2oo2 redundancy scheme.

The resulting percentage of failure modes is depicted in Figure 54. As it can be seen from the results, the 2oo2 redundancy scheme successfully detects all failures of its replicas: (i) the incorrect processing results from the replicas are reported as signaled failures (2.16%), and hangs of the replicas, reported as timeouts (3.71%). In such a way, despite the 2oo2 scheme doesn’t perform on-the-fly correction of results, it effectively signals all considered failures (100% diagnostic coverage). It is also worth noting that the timeout detection is not specific to the redundancy scheme, i.e., if assuming the same percentages of timeouts and failures for the simplex accelerator (1oo1) then the diagnostic coverage still would amount to roughly 63% (percentage of timeouts within the total set of failures). It should be noted, however, that the latter estimation is based on speculative assumption, and should be verified by further FFI experiments.

Under the conservative assumption that both detected failures and hangs are dangerous failures (DF), the DF failure rate can be estimated by the following equation:

$\lambda_{DF} = \lambda_{FPGA} \times N_{eb} \times (P_{dec} + P_{hang})$, where:

λ_{FPGA} – constant failure rate (upset rate of FPGA configuration memory cells) at the sea-level taken from the device reliability report for a given FPGA. For the used FPGA part (Xilinx Virtex Ultrascale+) it equals 5 FIT per megabit of configuration memory **[UPV-ACC1]**;

N_{eb} – the number of essential bits (in Megabits) used by the DUT in a given implementation; this parameter can be retrieved from the FFI report generated by the DAVOS FFI tool, as it reports the number of essential bits in the components targeted in FFI experiment. For the considered DUT it reports 49.5 Mb of essential bits per one accelerator replica, thus resulting in a rough estimation of 99 Mb for the entire redundant acceleration subsystem;

P_{dec} and P_{hang} are the probability of an upset to lead to the detected failure, and to the hang/timeout of acceleration subsystem respectively.

In such a way, the DF rate for the considered acceleration subsystem can be estimated as 29.05 FIT, which can be translated into the probability of dangerous failure per hour of 2.9×10^{-8}. With respect to the IEC51508, for a continuously used high-demand system this number corresponds to the safety integrity level SIL-2 **[UPV-ACC2]**.

[UPV-ACC1] Xilinx Inc. Device Reliability Report. UG116 (v10.16), June 29, 2022

[UPV-ACC2] Smith, D. J., & Simpson, K. G. (2010). Safety critical systems handbook: a straight forward guide to functional safety, IEC 61508 (2010 Edition) and related standards, including process IEC 61511 and machinery IEC 62061 and ISO 13849. Elsevier.

11 Safety concept based on ISO26262 (VIF,AVL)

11.1 Introduction to vehicle safety and the relevant ISO standard

The advent of electronics and software development in the automotive industry has led to transforming motorized vehicle from pure mechatronics system to full embedded control system units, using latest advanced technologies for an optimal control of the vehicle.

One of these technologies is the usage of artificial intelligence, for the control of vehicle components. Considering that the control of a vehicle requires the usage of many sensors and actuators for each component depending on external condition and driver commands; the complexity of vehicle controls has exponentially increased in the past years.

A data-driven control strategy is a solution to reduce development costs (e.g., in terms of eliminating the need for manual calibration) and increase control accuracy of a vehicle (e.g., due to self-adaptation).

However, unlike a rule-based control strategy, a data-driven control strategy is the outcome of a machine learning process and as such may be a non-deterministic function.

To ensure the safety in the vehicle these non-deterministic functions need to be under supervision of deterministic safety function which are defined in the safety concept.

During the concept phase, the hazard analysis and risk assessment (based on the item definition) is generated.

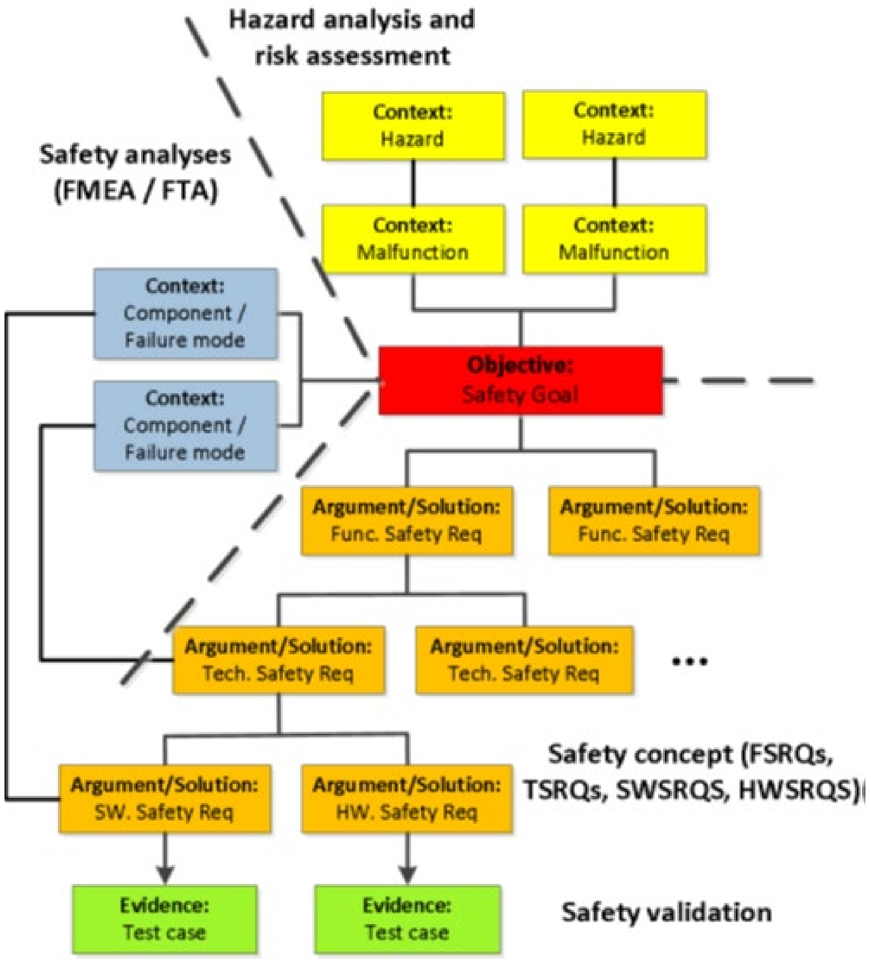

Hazard analysis and risk assessment
Context: Hazard
Context: Hazard
Safety analyses (FMEA / FTA)
Context: Malfunction
Context: Malfunction
Context: Component / Failure mode
Objective: Safety Goal
Context: Component / Failure mode
Argument/Solution: Func. Safety Req
Argument/Solution: Func. Safety Req
Argument/Solution: Tech. Safety Req
Argument/Solution: Tech. Safety Req
...
Safety concept (FSRQs, TSRQs, SWSRQS, HWSRQS)
Argument/Solution: SW. Safety Req
Argument/Solution: HW. Safety Req
Evidence: Test case
Evidence: Test case
Safety validation

Figure 55 Safety concept

The main goal of the hazard analysis and risk assessment is to analyze the functionalities of the system in its intended context and to identify and classify possible hazards. Main outcome here are safety goals – which represent safety objectives during the project.

The main purpose while developing the safety concept is to systematically refine the safety argumentation by providing functional requirements, then technical requirements, and finally software (resp. hardware) requirements to mitigate the risks.

This assumption is violated when a training set is used in place of a specification since such a set is necessarily incomplete, and it is not clear how to create assurance that the corresponding hazards are always mitigated. Thus, an ML component violates the assumption. Furthermore, the training process is not a verification process since the trained model will be "correct by construction" with respect to the training set, up to the limits of the model and the learning algorithm.

11.2 UC2

11.2.1 Description of the work-flow using the standard

During the ISO 26262 lifecycle the output of each phase must be verified against the requirements in input to the same phase. In addition to this, a validation is requested at the end of the design process to ensure that the final implementation of the item, integrated in the vehicle, is compliant with the initial safety goals and so the system can provide the adequate level of safety. Therefore, each phase of the design process needs to be verified against input requirements while the whole item is finally validated against the top-level safety requirements without any intermediate step

As a consequence of this approach, verification and validation activities are tightly linked to requirements throughout all the development process. The following section shows those links referring to the V model proposed by ISO 26262

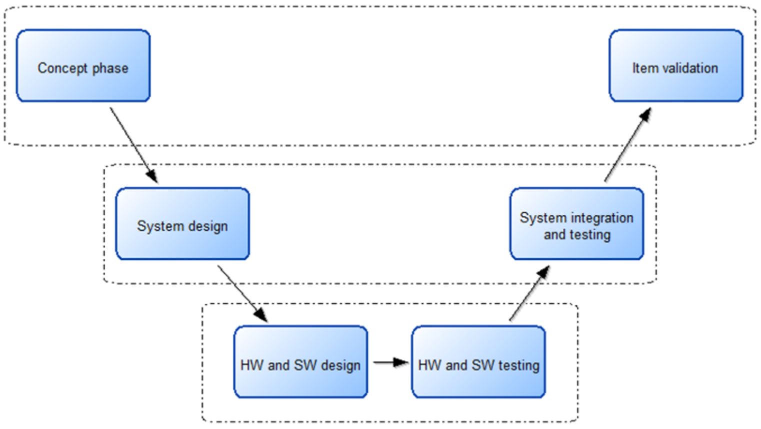


Figure 56 Work flow using the standard

We have then here a deterministic SW component which has to be verified according to verification method described in the Norm ISO 26262

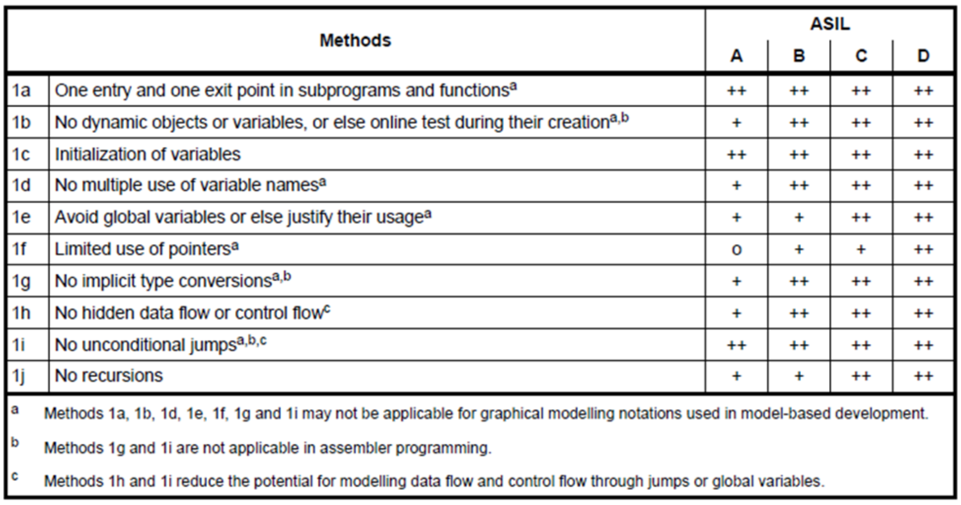
We will here focus on the SW part of the safety concept as in the UC2 of Fractal project (Air Path control) we are mainly interested of the Control SW of an Engine component.

Once SW safety requirements are defined, they can be implemented and tested.

We have then here a deterministic SW component which has to be verified according to verification method described in the Norm ISO 26262.

Following tables extracted from the ISO 26262 Norm are describing which SW implementation guideline as well as SW verification method need to be used depending on the ASIL Rating of each safety goal. While some of these remain applicable to ML components and others could readily be adapted, many remain that are specifically biased toward the assumption that code is implemented using an imperative programming language.

Table 14 Design principles for software unit design and implementation



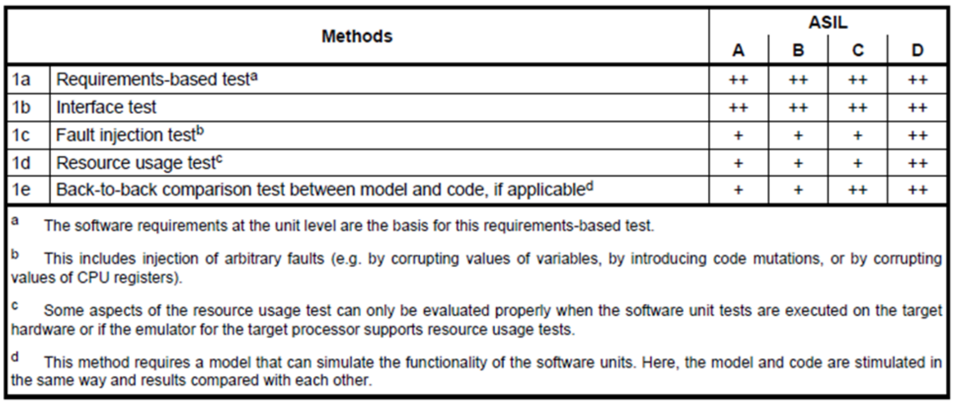
	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	One entry and one exit point in subprograms and functions[a]	++	++	++	++
1b	No dynamic objects or variables, or else online test during their creation[a,b]	+	++	++	++
1c	Initialization of variables	++	++	++	++
1d	No multiple use of variable names[a]	+	++	++	++
1e	Avoid global variables or else justify their usage[a]	+	+	++	++
1f	Limited use of pointers[a]	o	+	+	++
1g	No implicit type conversions[a,b]	+	++	++	++
1h	No hidden data flow or control flow[c]	+	++	++	++
1i	No unconditional jumps[a,b,c]	++	++	++	++
1j	No recursions	+	+	++	++

[a] Methods 1a, 1b, 1d, 1e, 1f, 1g and 1i may not be applicable for graphical modelling notations used in model-based development.

[b] Methods 1g and 1i are not applicable in assembler programming.

[c] Methods 1h and 1i reduce the potential for modelling data flow and control flow through jumps or global variables.

Table 15 Methods for software unit testing



	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Requirements-based test[a]	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[b]	+	+	+	++
1d	Resource usage test[c]	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable[d]	+	+	++	++

[a] The software requirements at the unit level are the basis for this requirements-based test.

[b] This includes injection of arbitrary faults (e.g. by corrupting values of variables, by introducing code mutations, or by corrupting values of CPU registers).

[c] Some aspects of the resource usage test can only be evaluated properly when the software unit tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

[d] This method requires a model that can simulate the functionality of the software units. Here, the model and code are stimulated in the same way and results compared with each other.

To cater with AI, we first need to perform an ASIL decomposition. The AI can be regarded to be covered by classical QM as there ia an non-AI safety monitor in place. Since AI is covered by QM, the safety requirement no longer applys and it is a matter of quality.

Once the SW has been designed, implemented, verified, and validated according to the Norm 26262, we can consider that after its integration in the vehicle, the component it is supervising, is safe. It means that outputs from a control algorithm which could generate a potential safety violation, is recognized and a safety reaction is triggered.

A safety reaction shall be triggered as less as possible, as it impacts the drivability of the vehicle. As example, a safety reaction inhibits some control function, which could limit the drivability of the vehicle.

Therefore, it is mandatory that during the training phase of the neural network, boundary conditions which are the output of the Safety concept (in our case, the safety monitor), are considered.

Safety functions must then be incorporated during the learning process of the model.

The picture below shows the integration of the developed Safety SW within the learning procedure without reinforcement learning.

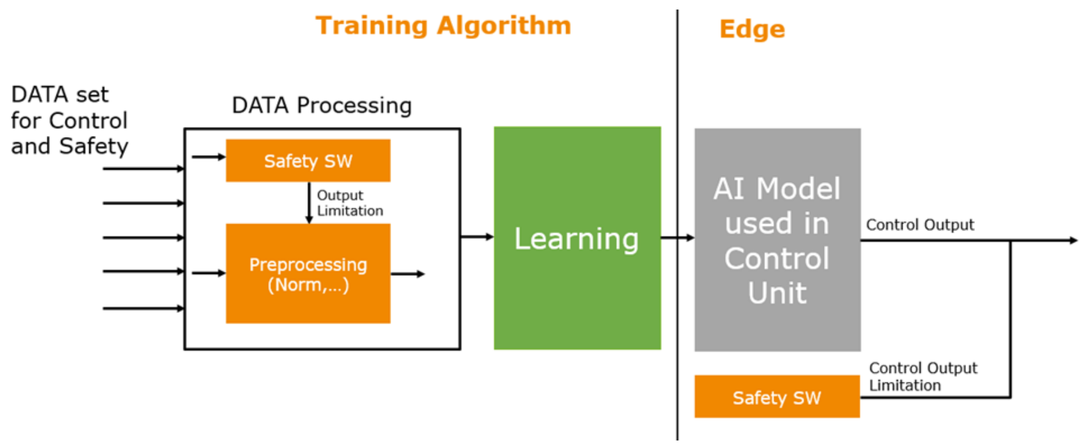


Figure 57 Integration of the developed Safety SW within the learning procedure without reinforcement learning.

As you can see output used to control actuator of the vehicle must be limited by safety function before the data preprocessing occurs. For example, thermal runaways of the battery may constitute a potential hazard.

The trained AI model learns to know all control limitations due to safety. By consequences, the AI model is providing control commands which consider safety range. The probability that the ML model is violating safety requirement is then reduced.

The regular Safety SW shall still be executed in the control ECU according to the norm ISO-26262.
The picture below shows how the integration of the developed Safety SW could be performed using reinforcement learning

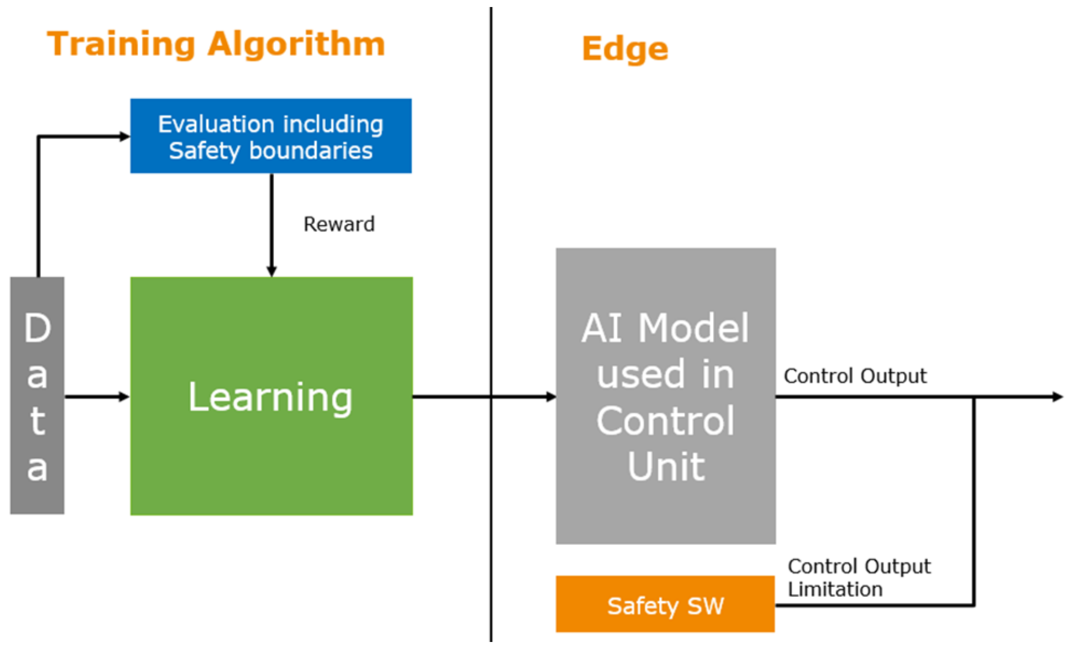

Training Algorithm
Edge
Evaluation including Safety boundaries
Reward
Data
Learning
AI Model used in Control Unit
Control Output
Control Output Limitation
Safety SW

Figure 58 Integration of the developed safety SW could be performed using reinforcement learning

In any case, the Safety SW shall be implemented on the Control Unit to directly supervise safety relevant components.

11.3 Safety concept with UC7 (SPIDER autonomous robot)

11.3.1 Description of the work-flow using the standard

The general safety activities of the concept phase according ISO 26262 can be seen in the following Figure 59.

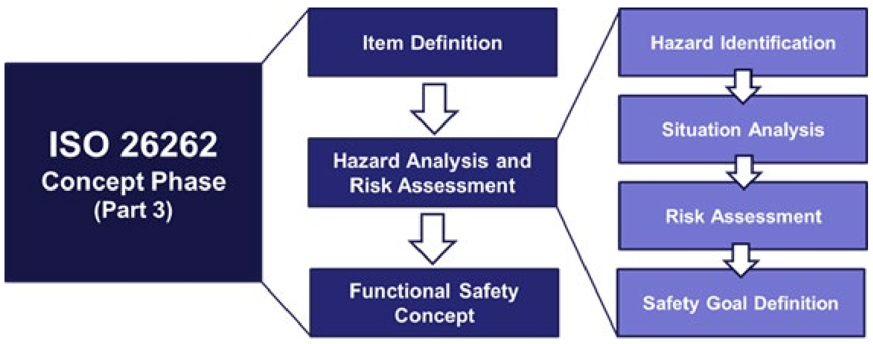


Figure 59 Overview of safety activities in concept phase of ISO 26262

11.3.2 Item Definition

The first objective of the Item Definition is to define and describe the item SPIDER, its dependencies on, and interaction with, the environment and other related items. The second objective is to support an adequate understanding of the item so that the activities in subsequent phases can be performed.

The document serves to provide sufficient information about the item to the persons who conduct the subsequent subphases like "Initiation of safety lifecycle", "Hazard analysis and risk assessment" and "Functional safety concept".

A part of the Item definition will be presented in chapter 11.3.4.

11.3.3 **Hazard Analysis and Risk Assessment**

In the first step of the hazard analysis and risk assessment (HARA) the malfunctions of the item and their potential hazards on vehicle level are identified and a situations analysis is used to formulate a proper set of situations for the possible hazards events.

Finally, the identified situations and identified hazards are combined in an assessment matrix to derive the hazards events. These hazards events are classified with the risk parameter severity S, exposure E and controllability C. The ASIL for each hazardous event of the item is specified.

As the last step of the hazard analysis and risk assessment is the definition of the safety goals for each hazardous event of the electric power train and the association of the ASILs. The HARA for the Spider will be presented in more detail in chapter 11.3.5.

11.3.4 **Description of use-case 7**

Figure 60 shows the item SPIDER (in blue) and its surroundings in which the SPIDER shall operate.

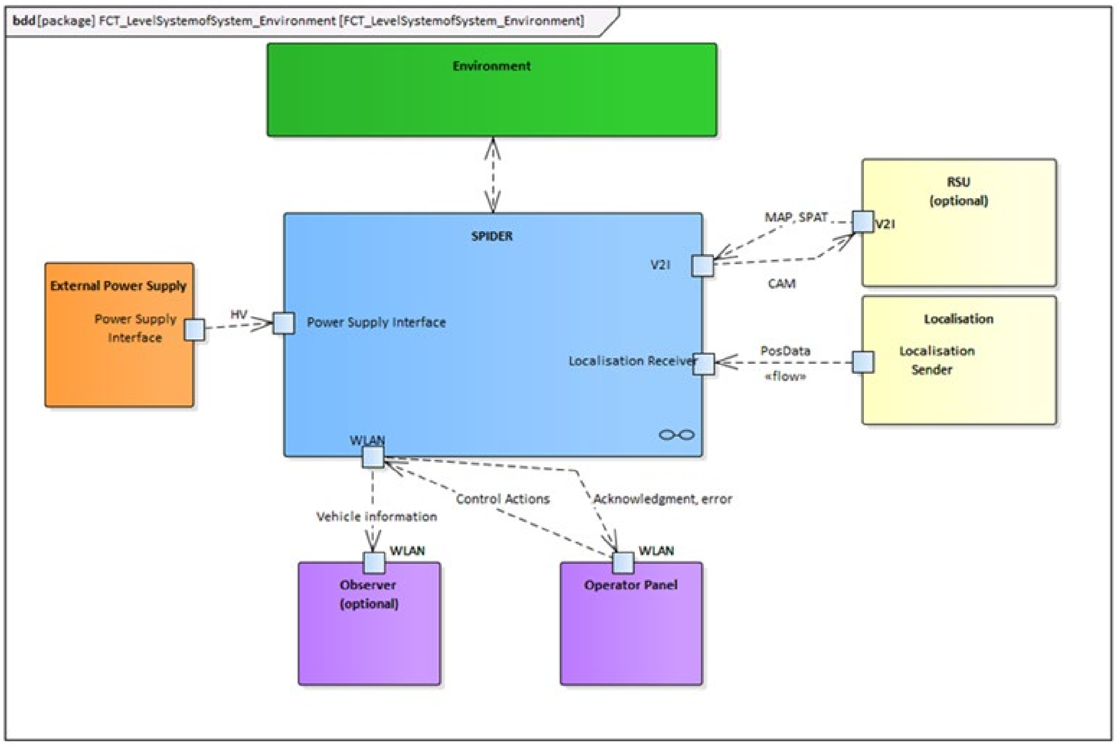


Figure 60: Overview of system block diagram of SPIDER

Overview of SPIDER functions:

Automated omnidirectional movement (aka. Path Tracking Function)

Four single automated controlled steering servo motors allow a pseudo omnidirectional movement of the SPIDER along a predefined global path and four single automated controlled electric motors allow to accelerate or decelerate the SPIDER to a predefined velocity.

Collision Prevention/Avoidance (aka. Collision Avoidance Function)

A collision with any person and any object is prevented by stopping the SPIDER.

Base safety feature of SPIDER.

Determination of position

Based on the data from the localization (by GNSS), SPIDER determinates his own position within a tolerance of +/-150 mm.

Base feature of SPIDER for evaluation of proving ground tests.

Manual omnidirectional movement

Operator Panel is used for manual control commands of the operator to the SPIDER.

Four single manual controlled steering servos allows a pseudo omnidirectional movement of the SPIDER and four single manual controlled electric motors allows to accelerate or decelerate the SPIDER.

SPIDER data recording

Storage of sensing data and the possibility for download for offline processing

Base feature of SPIDER for evaluation of proving ground tests.

The two main functions for the safe operation of the spider are described in more detail below.

Collision Avoidance Function

The task of the collision avoidance function is to detect obstacles and initiate measures to avoid a collision. Four Lidar sensors, which are mounted on the outer corners of the robot, enable an all-round view with at least double redundancy at about 50 meters. In a preprocessing step the point cloud data from the sensors is filtered and fused using the Point Cloud Library (PCL) **[SC-UC7-1]**. The resulting fused point cloud is mapped onto a two-dimensional grid, called cost map, with occupancy values for each grid cell. Based on the cost map, an algorithm calculates the distance to the closest obstacle in the movement direction, and triggers an emergency brake if the obstacle distance is inside a defined danger zone around the robot.

Path Tracking Function

The path tracking function (PTF) is intended to follow a predefined global path in a precise manner. A global path is an ordered list of waypoints, which shall be touched by the robot. A waypoint is defined by the coordinates, target speed and orientation of the vehicle. When the PTF is activated by the human operator, the function calculates a trajectory to the next waypoint from the current location of the robot. From this trajectory the required speed and direction is computed and forwarded to the motion control unit of the SPIDER.

11.3.5 Hazard analysis and risk assessment

11.3.5.1 Hazard identification

By conducting a hazard and operability study (HAZOP) the malfunctions of the SPIDER and their potential hazards on vehicle level were identified. With the help of guide words possible deviations of the function were identified. These deviations are the malfunctions of the spider.

The following table gives an overview of commonly used guide words and common interpretations of them.

Table 16 HAZOP guide words

No.	Guide Word	Notes
1	None	Does not happen what is expected
2	Reverse	What happened is opposite to expectation
3	More	Go beyond the expected maximum value
4	Less	Go below the expected minimum value
5	As well as	Meets the expectation but unwanted thing happens in addition
6	Part Of	A part of the expectation happens
7	Other tham	The expectation happens other than expected

As the next step the consequence of these malfunctions on vehicle level and the resulting hazards of these malfunctioning behaviour on vehicle level were identified.

The top level hazard is the potential source of harm caused by malfunctioning behaviour of the item and could lead to harm (the physical injury or damage to the health of persons).

Some result of the HAZOP for the control unit/main functions of the SPIDER is shown in Table 17.

Table 17 HAZOP for control unit and main functions

Function	Guideword	Malfunction/Deviation	Malfunctioning behaviour on vehicle level/Consequences	Hazard-potential source of harm
Emergency stop	No	No emergency stop	No emergency stop when emergency stop required	No emergency stop lead to collison with objects and persons
	More			
	Less			
	As well as			
	Part of			
	Reverse			
	Other than	Wrong emergency stop	Wrong emergency stop activation	No emergency stop lead to collison with objects and persons
	Early			
	Late			
	Before			
	After			
Sensor data storage	No	No sensor data storage	none	None
	More			
	Less			
	As well as			
	Part of			
	Reverse			
	Other than	Wrong sensor data storage	none	None
	Early			
	Late			
	Before			
	After			
Manual operation	No	No manual operation	No manual Control of AGV	Unintendend omnidirectional movement lead to collison with objects and persons
	More			
	Less			
	As well as			
	Part of			
	Reverse			
	Other than	Wrong manual operation	Wrong manual Control of AGV	Unintendend omnidirectional movement lead to collison with objects and persons
	Early			
	Late			
	Before			
	After			

Different malfunctions of the SPIDER can lead to the same hazard, which can be seen in Table 18: Hazards resulting from an item's malfunctions, and thus result in recurrent hazards. The final list of hazards, that must be considered in the HARA is presented in Table 19: Situations for possible hazardous events (the recurrent hazards are deleted).

Table 18: Hazards resulting from an item's malfunctions

Haz001	Leak of toxic gases,fire or explosion close to persons
Haz002	Unintendend omnidirectional movement lead to collison with objects or persons
Haz003	Loss of AGV control lead to collison with objects or persons
Haz004	No emergency stop lead to collison with objects and persons

11.3.5.2 *Situation analyse*

For every item usage the reasonable combinations of location, road conditions and traffic were identified with the focus on worst case scenarios. Table 19 shows a set of situations for possible hazardous events.

Table 19: Situations for possible hazardous events

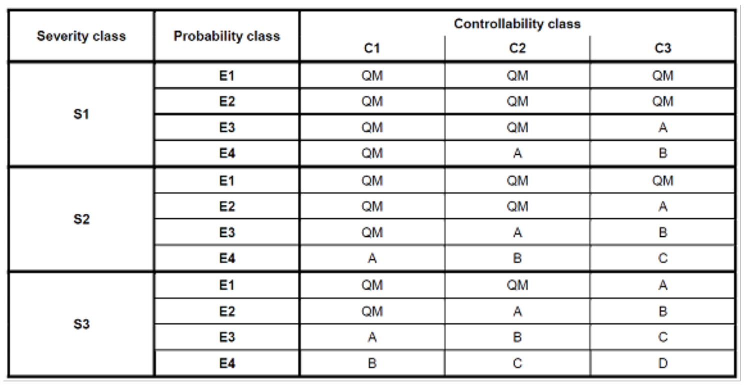
DM1.1	PARKING	LOC3.1	City	EC1.0	Normal enviroment	WC1.0	No special weather	MA1.0	No special Maneuver
DM1.2	VEHICLE CHARGING	LOC3.1	City	EC1.0	Normal enviroment	WC1.0	No special weather	MA1.0	No special Maneuver
DM1.3	SERVICE	LOC3.1	City	EC1.0	Normal enviroment	WC1.0	No special weather	MA1.0	No special Maneuver
DM2.1	DRIVE (Driver only)	LOC3.1	City	EC1.0	Normal enviroment	WC1.0	No special weather	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.0	Normal enviroment	WC1.0	No special weather	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.3	High/Motorway	EC1.0	Normal enviroment	WC1.0	No special weather	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.1	Obstacle at street	WC1.0	No special weather	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.3	High/Motorway	EC1.1	Obstacle at street	WC1.0	No special weather	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.0	Normal enviroment	WC1.1	Fog	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.3	High/Motorway	EC1.0	Normal enviroment	WC1.1	Fog	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.1	Obstacle at street	WC1.1	Fog	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.3	High/Motorway	EC1.1	Obstacle at street	WC1.1	Fog	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.0	Normal enviroment	WC1.2	Snowing	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.3	High/Motorway	EC1.0	Normal enviroment	WC1.2	Snowing	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.1	Obstacle at street	WC1.2	Snowing	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.3	High/Motorway	EC1.1	Obstacle at street	WC1.2	Snowing	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.0	Normal enviroment	WC1.4	Rain	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.3	High/Motorway	EC1.0	Normal enviroment	WC1.4	Rain	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.1	Obstacle at street	WC1.4	Rain	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.3	High/Motorway	EC1.1	Obstacle at street	WC1.4	Rain	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.0	Normal enviroment	WC1.10	Sun	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.3	High/Motorway	EC1.0	Normal enviroment	WC1.10	Sun	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.1	Obstacle at street	WC1.10	Sun	MA1.0	No special Maneuver
DM2.4	DRIVE (Fully Automated)	LOC3.3	High/Motorway	EC1.1	Obstacle at street	WC1.10	Sun	MA1.0	No special Maneuver
DM1.2	VEHICLE CHARGING	LOC3.1	City	EC1.0	Normal enviroment	WC1.14	Low temp. (-20°C<te	MA1.0	No special Maneuver
DM2.1	DRIVE (Driver only)	LOC3.1	City	EC1.0	Normal enviroment	WC1.0	No special weather	MA1.1	Braking to Standstill
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.0	Normal enviroment	WC1.0	No special weather	MA1.1	Braking to Standstill
DM2.4	DRIVE (Fully Automated)	LOC3.3	High/Motorway	EC1.0	Normal enviroment	WC1.0	No special weather	MA1.1	Braking to Standstill
DM2.4	DRIVE (Fully Automated)	LOC3.1	City	EC1.1	Obstacle at street	WC1.0	No special weather	MA1.1	Braking to Standstill

11.3.5.3 *Risk assessment*

The identified situations and the identified hazards were combined finally in an assessment matrix to get the hazardous events.

The hazardous events were classified in accordance with the risk matrix of ISO 26262 (s) and the ASIL for every item malfunction was defined.

Table 20: Risk matix for ASIL determination



Severity class	Probability class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

11.3.5.4 Safety goal definition

Based on the classification, the Safety Goals (SG) and their associated ASILs are identified. Safety goals are top-level safety requirements for the item. They lead to the functional safety requirements needed to avoid an unreasonable risk for each hazardous event. Safety goals are not expressed in terms of technological solutions, but in terms of functional objectives.

The safety goal is defined based on the hazard event. The safety goal shall describe how the hazard could be prevented for the driving situation of the hazard event.

If one single SG has two different ASILs (or class QM) the most restrictive one is assigned to that SG for further analysis purposes. The derived SGs for the SPIDER are presented in the Table 21.

11.3.6 **Result for functional safety concept**

In this use case the focus is on two safety critical functions of the SPIDER, the collision avoidance function, and the path tracking function (see chapter 11.3.4).

The collision avoidance function must fulfil

- SG010 "Prevent no or wrong emergency stop activation when emergency stop is required in any driving situation" ASIL C,
- SG022 "Prevent false negative object detection in any driving situation" ASIL C.

The path tracking function (PTF) must fulfil

- SG013 "Prevent wrong path following of SPIDER in any driving function" ASIL C.

To demonstrate how to use the results from the safety concept we derived the following functional safety requirements for the collision avoidance function to fulfil SG010 and SG022.

For SG010:

- FSR0071 Receive EmergencyStop activation: The EmergencyStopActivation through the RemoteControl shall be received
- FSR0072 Activate emergency stop: If EmergencyStop requested, the emergency stop shall be activated

For SG022:

- FSR0008 EmergencyStop: If the SPIDER is in the <EmergencyStop> the
    - target velocity on every wheel shall be 0 rotations per minute
    - The emergency brake shall be activated.

The resulting requirements are convoyed to WP8 as additional use-case related requirements. From those use-case requirements KPIs will be derived to assure the compliance to the safety goals.

Table 21: Safety Goals

ASIL	Safety Goal_ID	Safety Goal (Top Level)	Safe State	Fault tolerant Time
ASIL C	SG001	Prevent unintended asymmetrical braking torque for the four wheels in all driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG002	Prevent unintended asymmetrical acceleration torque for the four wheels in all driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG003	Prevent unintended asymmetrical steering torque for the four wheels in all driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG004	Prevent deep discharging of battery cells in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG005	Prevent higher acceleration of SPIDER than required in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)

ASIL C	SG006	Prevent no or incomplete power supply for SPIDER- Systems (PV & LV) and computation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG007	Prevent contradicting steering in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG008	Prevent no or lower deceleration instead of required deceleration in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG009	Prevent no or wrong or incomplete automated (no lateral or longitudinal) control of SPIDER if automated control is required in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG010	Prevent no or wrong emergency stop activation when emergency stop is required in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG011	Prevent no or wrong manual control of SPIDER if manual control is required in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG012	Prevent no steering or stronger or weaker steering than required of the SPIDER when steering is required in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG013	Prevent wrong path (following of SPIDER in any driving situation)	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG014	Prevent overheating of battery cells in	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)

		any driving situation		
ASIL B	SG015	Prevent deep discharging of battery cells in any parking situation.	Deactivation of battery system	FTT <= 1s
ASIL A	SG016	Prevent no or wrong manual control of SPIDER if manual control is required in any service situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL A	SG017	Prevent no steering or stronger or weaker steering than required of the SPIDER when steering is required in any service situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL A	SG018	Prevent contradicting steering in any service situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL B	SG019	Prevent deep discharging of battery cells in any charging situation	Deactivation of battery system	FTT <= 1s
ASIL B	SG020	Prevent overheating of battery cells in any charging situation	Deactivation of battery system	FTT <= 1s
ASIL C	SG021	Prevent unwanted in verse acceleration of SPIDER in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG022	Prevent false negative object detection in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG023	Prevent wrong acceleration of SPIDER in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG024	Prevent wrong deceleration of SPIDER in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)
ASIL C	SG025	Prevent wrong emergency management of SPIDER in any	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)

		driving situation to avoid overheating of battery cells		
ASIL C	SG026	Prevent wrong activation of operation mode in any driving situation	Safe/ emergency mode of SPIDER	FTT <= 200 ms (1.9 sec deceleration time)

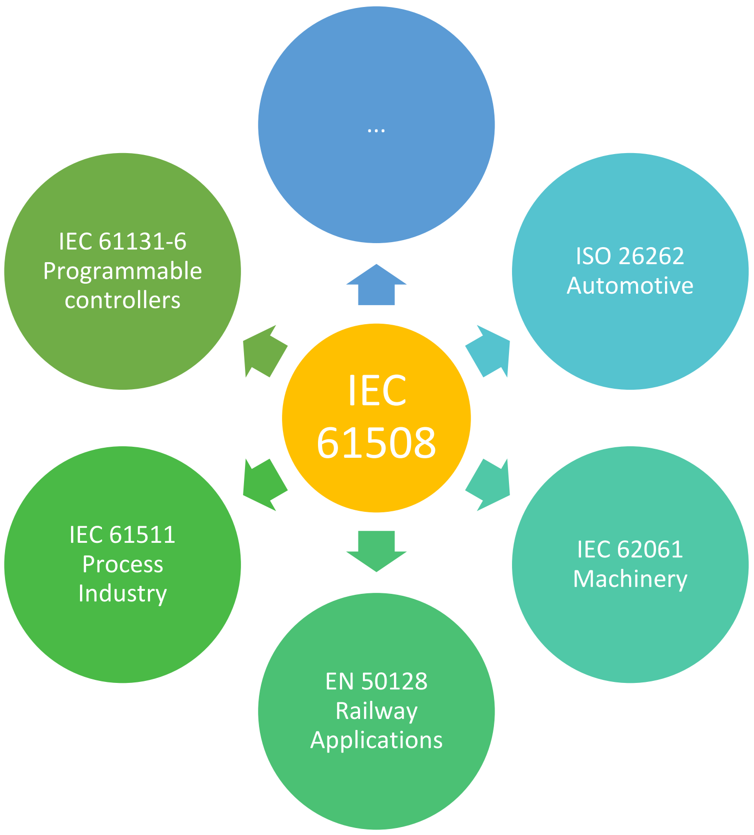
[SC-UC7-1] Rusu, Radu Bogdan, and Steve Cousins. "3d is here: Point cloud library (pcl)." 2011 IEEE international conference on robotics and automation. IEEE, 2011

12 Safety concept based on IEC 62061 (BEEA)

12.1 Introduction for industrial applications of the standard IEC 61508 and corresponding additional standards

The IEC standard 61508 is a general standard applicable to all industries for functional safety. Many standards are derived from this ISO standard, on the base of which functional safe systems are created in a wide range of industries, like shown in Figure 61.

The complexity of modern machines is increasing and with it the requirements of safe functions for human protection. To satisfy these requirements, fail-safe PLCs are growing in popularity, also in embedded systems. The logic programming can be more complex and still providing the necessary overview in comparison to common electromechanical parts, like safety relays. A further reason could be flexibility, as corrections of the logic remain only in the software part in most cases.



...
IEC 61131-6 Programmable controllers
ISO 26262 Automotive
IEC 61508
IEC 61511 Process Industry
IEC 62061 Machinery
EN 50128 Railway Applications

Figure 61 Examples of functional safety standards derived from IEC 61508

E.g. chapter 11 deals with standard ISO 26262, the application in the automotive industry, chapter 12 apply the standard IEC 62061 as an application example in use case 8 – swarm intelligence in a shuttle system. The goal is the implementation of a functional safety service in a fractal edge node and the connection to an existing fail-safe PLC. The procedure will be clarified and the preliminary result explained in section 12.2.4.

12.2 Safety concept of UC8 (Shuttle system with swarm intelligence)

In state-of-the-art applications the safety services of a closed system like the shuttle system are connected between each component. That means, to gain access to the system, safe wireless communication and safety control units in every component of the system are required to prevent unsafe operation states and avoid threats towards persons entering the system. The new concept concentrates on the implementation of fractal components to separate the safety functions between the access control and the shuttle.

12.2.1 Description of the workflow using the standard

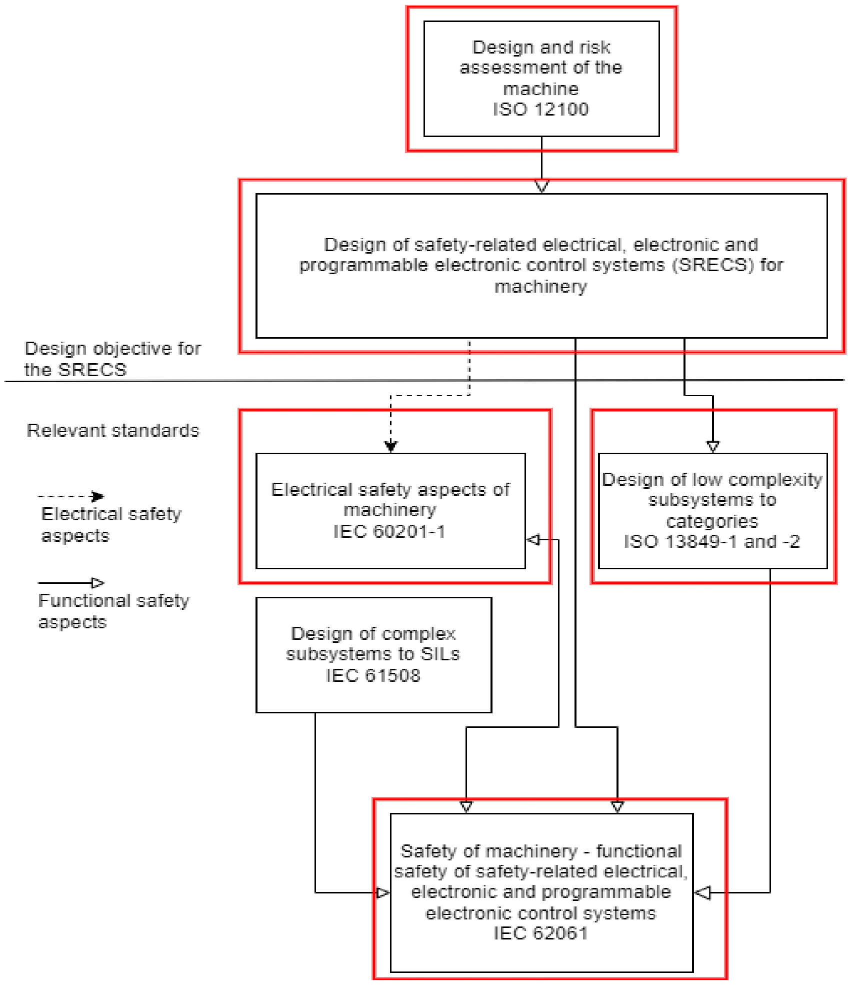

Design and risk assessment of the machine ISO 12100
Design of safety-related electrical, electronic and programmable electronic control systems (SRECS) for machinery
Design objective for the SRECS
Relevant standards
Electrical safety aspects
Functional safety aspects
Electrical safety aspects of machinery IEC 60201-1
Design of low complexity subsystems to categories ISO 13849-1 and -2
Design of complex subsystems to SILs IEC 61508
Safety of machinery - functional safety of safety-related electrical, electronic and programmable electronic control systems IEC 62061

Figure 62 Relationship of IEC 62061 to other relevant standards

The safety concept for UC8 has followed the typical design process and starts with the risk assessment like shown in **Fehler! Verweisquelle konnte nicht gefunden werden.** **[BEEA-UC1]**. Necessary packages are highlighted in red boxes. During the risk assessment, the safety functions are defined, and their requirements are determined. As UC8 wants to implement a low complex safety function, the leading standard to determine the requirements are ISO 13849-1 and -2 with respect to electrical safety aspects from IEC 60204-1. To summarize, the risk assessment provides the required safety functions and the required performance levels for each determined function. After determining all safety functions, the first safety concept for the system will be designed under consideration of the output from the risk assessment. The verification/ certification of new developed safety-related functions requires always a nationally recognized testing laboratory e.g., the TuV Rheinland in Germany and is mandatory for standardization in the product development. As a European manufacturer of machinery, each product must be CE marked and the risk assessment, as well as the safety concept are inherent parts of it.

12.2.2 **Risk assessment**

Beginning with the risk assessment, design principles for safety of machinery is applied in UC8 from the harmonized standard DIN EN ISO 12100, which is based on the ISO 12100 and is available in German translation. For this approach a table with all possible hazards based on the safety requirements of the standard DIN EN ISO 3691-4 is applied and examined. The standard DIN EN ISO 3691-4 (as a type C standard) is about safety requirements and verification of driverless industrial trucks and their systems. AGV's such as the shuttles also fall under this standard. Risks in the assessment follows always the same procedure, that means iterative steps for each hazard will be done, until the reduction of the risk is low enough or require a complete redesign of the product. To measure the risk potential, there are four parameters (similar to a FMEA) used, degree of possible harm (Table 22), likelihood of occurrence (Table 23), frequency of exposure (Table 24) and number of persons at risk (Table 24).

Table 22 Degree of Possible Harm

DPH		Degree of Possible Harm
0,1	S1	Scratch or bruise
0,5	S1	Laceration or mild ill-effect
2	S1	Break of minor bone or minor illness (temporary)
4	S2	Break of major bone or major illness (temporary)
6	S2	Loss of one limb, eye, hearing (permanent)
10	S3	Loss of two limbs or eyes (permanent)
15	S3	Fatality

Multiplied together, this gives the potential risk (Table 25) of a hazard, which are followed by measures to reduce the estimated risk. E.g., values bigger than 25 requires a technical solution, values bigger than 50 should be considered for a redesign if the constructive and technical solutions don't provide a strong reduction of the danger.

Table 23 Breakdown by likelihood of occurence

LO		Likelihood of Occurrence	
0,033	P1	Almost impossible	Only in extreme circumstances
1	P1	Highly unlikely	Though conceivable
1,5	P1	Unlikely	But could occur
2	P1	Possible	But unusual
5	P2	Even chance	Could happen
8	P2	Probable	Not surprising
10	P2	Likely	To be expected
15	P2	Certain	No doubt

After every measure, the new risk potential will be calculated, if the new potential is still too high, the next step of measure must be iterated.

There are three types of measures, the first one is called constructive measure and contains measures, which can change the risk by mechanical design, specification of the product or physical properties/ limitations of single parts in the product.

The technical measure is used, when the risk potential is too high after application of the first measure. It contains electrical, electronic and programmable electronic control systems and is associated with the performance level.

In the last step of measures, the informative measures are applied, were safety signs and instruction manuals may can finalize the hazard to an acceptable risk index, if the risk is already below 25.

Table 24 Breakdown by frequency of exposure and number of persons at risk

FE		Frequency of Exposure	**NP**	**Number of persons at risk**
0,5	F1	Annually	1	1-2 persons
1	F1	Monthly	2	3-7 persons
1,5	F1	Weekly	4	8-15 persons
2,5	F2	Daily	8	16-50 persons
4	F2	Hourly	12	50+ persons
5	F2	Constantly		

Depending on the risk index, new measures must be taken into the assessment, if the risk potential is still too high after these three steps of measures. A redesign must be considered for values over 50, when measures aren't feasible in the current realization.

Table 25 Breakdown by risk potential value

R*	R-Index	Risk
0-5	1	very low
5-25	2	low
25-50	3	middle
50-500	4	high
Over 500	5	not acceptable

As the list of hazards is very long, in this chapter only the resulting safety functions will be presented. In the test setup of UC8 will be two safety functions realized, to split the state-of-the-art solution. Both safety functions will be tested during the research project. The presented tables were translated from the German language.

12.2.2.1 Safety function 1 – Lift access

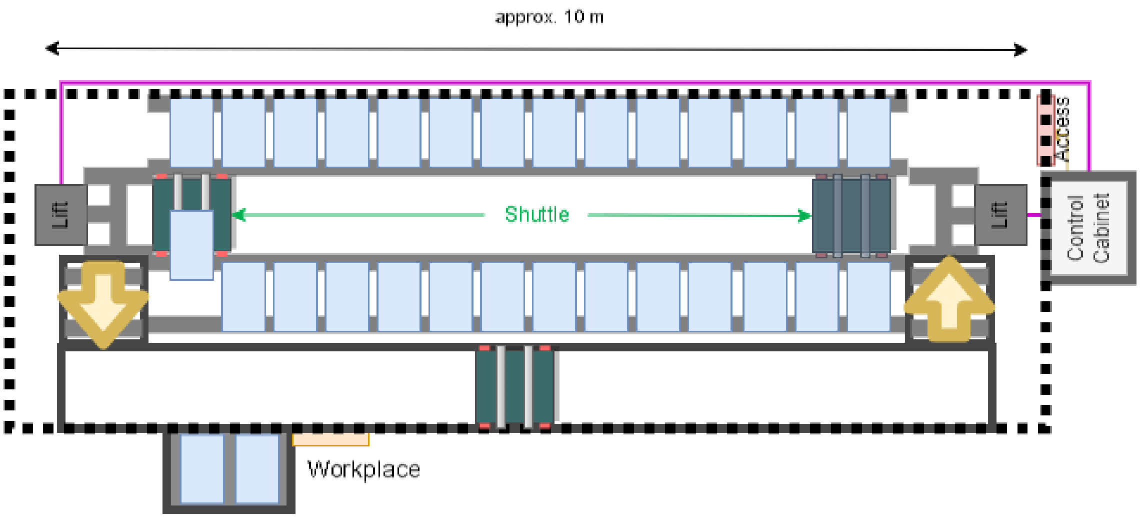


Figure 63 UC8 test setup - schematic top view

As the lifts are on both sides in the test setup, like shown in Figure 63, access to the rack still requires deactivating at least one of them. There are multiple hazards, which has to be covered by this safety function. As shown in Table 26, all three measures were carried out, to accomplish a safe operation with the hoist system of the lifts.

Table 26 Iterative procedure of a hazard by kinetic energy for the lifts

risk assessment					***protection target, protective action, residual risk***	***risk assessment, after implementation of the protektive target***					***risk assessment for electric control***			**Performance level**	
Lo	FE	DPH	NP	R		Lo	FE	DPH	NP	R	S	F	P	PLr	PL
10	1,5	6	1	90	Constructive measures: A fence around the lift equipment shall protect from severe injuries.	5	1,5	6	1	45				FALSE	
5	1,5	6	1	45	Technical measure: Access monitoring by means of access door/ door switch will put the lift in a safe condition. Any movement of the lift motor is restricted. The access door will not be released until the release process is complete.	1	1,5	6	1	9	S2	F1	P2	d	
1	1,5	6	1	9	Informative measure: Only trained staff is allowed to enter the system. Warning notices before each access. Access concept is described in the operator manual.	1	1,5	6	1	9				FALSE	

The safety function must fulfill PL “d” and additional informative measures are still mandatory. In detail, the first hazard where this function occurs is about the acceleration/ deceleration (kinetic energy) with the possible consequences of being run over or an impact on the human body. Mechanical measures could be e.g., a fence as a solution to restrict direct access to the machine. As we still need access for maintenance purposes, a controlled environment must be created by safety components and defined access points. This includes an access process, which requires a safe door lock and the corresponding request by key or pushbutton.

12.2.2.2 Safety function 2 – person detection in the AGVs

The second safety function will be realized in the shuttle equipment. As the standard requires person detection for the AGVs, the assessment targets the reduction of the required performance level from “d” to “c” by applying constructive measures. Table 27, shows the evaluation of the kinetic energy from the shuttle. In the first step of measures, the approach for the shuttle is a light-weighted construction with physical limitations in travel speed and motor torque to prevent in the base design sever hazards to the human body. In the second step, the technical measure is accomplished by the person detection and an emergency stop circuit in case the shuttle is nearby for manual operations by the operator. The behavior of the AGV is defined in the standard for person detection and will be briefly listed in the safety concept of chapter 12.2.4.

Table 27 Iterative procedure of a hazard by kinetic energy for the shuttle

risk assessment					***protection target, protective action, residual risk***	***risk assessment, after implementation of the protektive target***					***risk assessment for electric control***			**Performance level**	
Lo	FE	DPH	NP	R		Lo	FE	DPH	NP	R	S	F	P	PLr	PL
8	1,5	4	1	48	Constructive measure: Travel speed max. 0.8 m/s (target speed 0.6 m/s) Container dimensions are limited to max. 600x400x320 mm Container weight limited to max. 35 kg Motor power designed for 0.6 m/s with a maximum total weight of 70 kg Slip behavior of the drive system in case of mechanical resistance AGV can only travel along the rack in guided rails Mechanical end in both directions on the rails	2	1,5	4	1	12				FALSE	
2	1,5	4	1	12	Technical measures: The person detection system is designed to prevent injuries caused by a collision. In addition, emergency stop switches are fitted on both sides to switch off the power section of the AGV via a safety circuit.	2	1,5	2	1	6	S2	F1	P1	c	
2	1,5	2	1	6	Informative measures: Only trained staff is allowed to enter the system. Warning notices before each access. Operation of AGVs is described in the operator's manual.	1,5	1,5	2	1	4,5				FALSE	

12.2.3 Hardware architecture

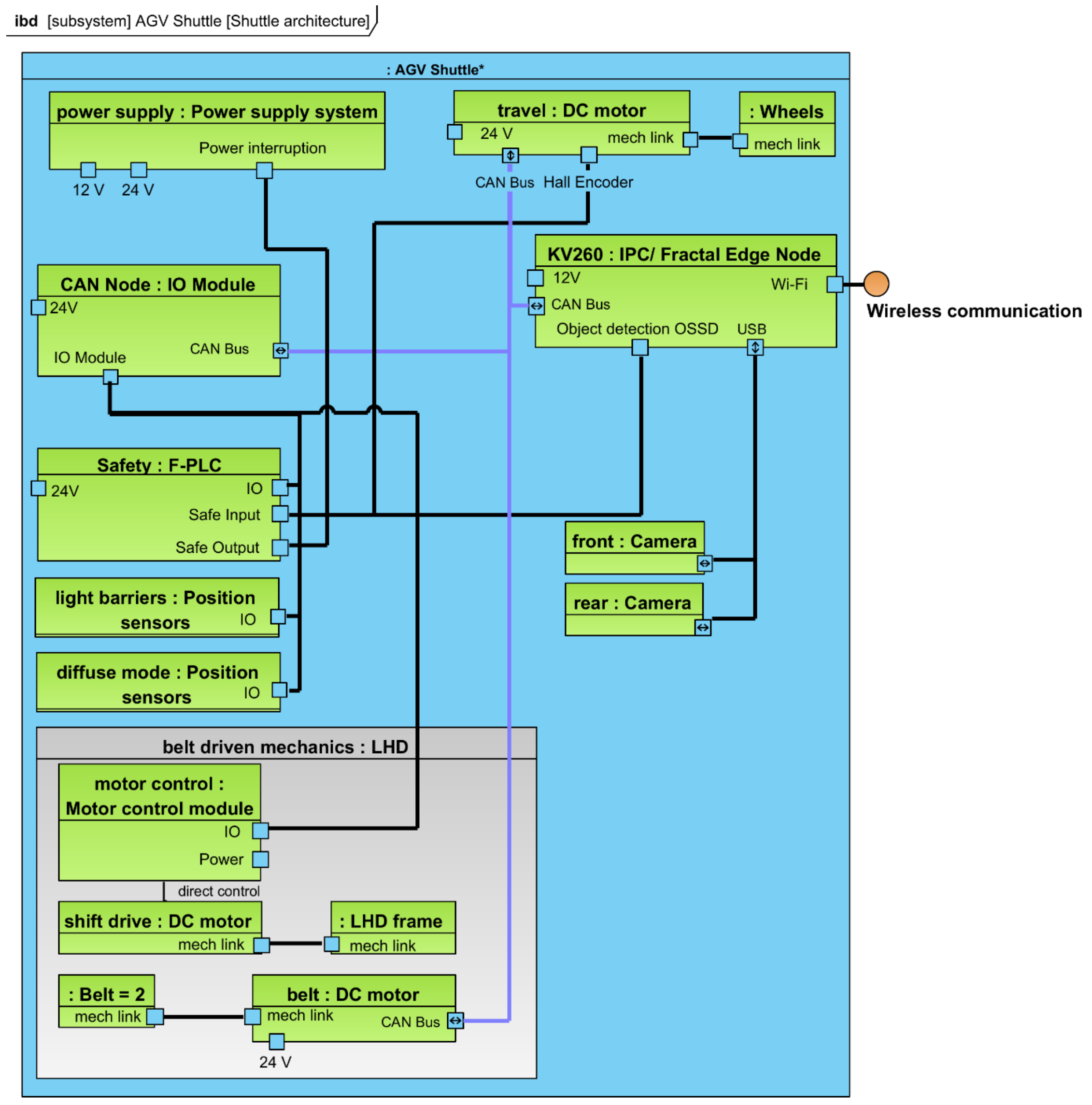
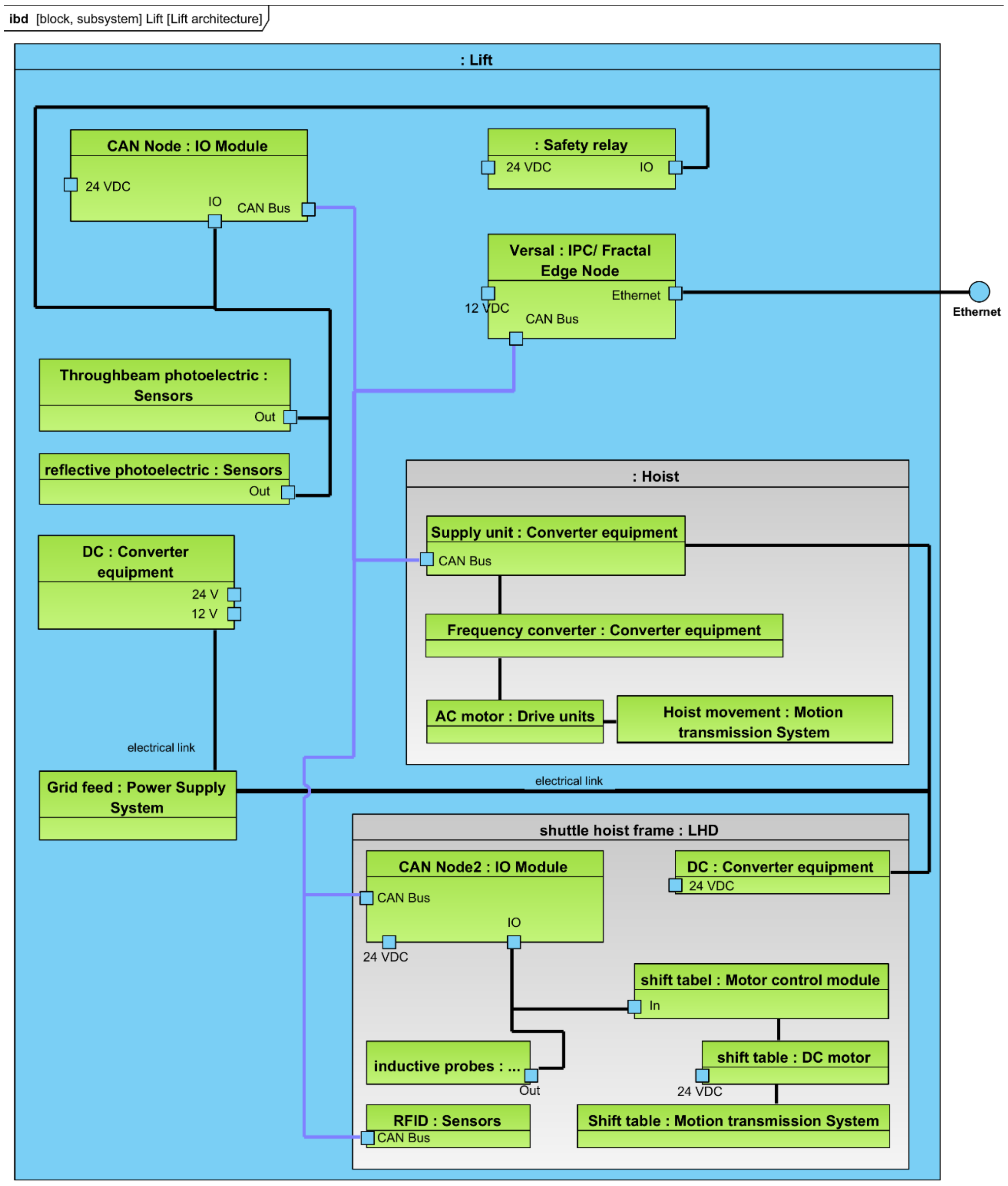


Figure 64 Final shuttle architecture

The resulting hardware architecture based on the risk assessment is shown for the shuttle in Figure 64. The AGV shuttle block consists of all important components in green blocks linked together with the corresponding interfaces, especially the CAN Bus for internal communication. For the connectivity with other nodes, wireless communication e.g., Wi-Fi is planned. On both sides of the chassis cameras will be mounted for person detection and the IPC replaced by the Kria platform from Xilinx, where the fractal components will be deployed and the necessary control services.

Analogously to the shuttle architecture, Figure 65 illustrates the architecture of the lift with the corresponding interfaces. The only change in the setup prior to the fractal project is the upgrade from the IPC to the Versal platform and the changed safety concept.



ibd [block, subsystem] Lift [Lift architecture]
: Lift
CAN Node : IO Module
24 VDC
IO
CAN Bus
: Safety relay
24 VDC
IO
Versal : IPC/ Fractal Edge Node
12 VDC
Ethernet
CAN Bus
Ethernet
Throughbeam photoelectric : Sensors
Out
reflective photoelectric : Sensors
Out
DC : Converter equipment
24 V
12 V
electrical link
Grid feed : Power Supply System
: Hoist
Supply unit : Converter equipment
CAN Bus
Frequency converter : Converter equipment
AC motor : Drive units
Hoist movement : Motion transmission System
electrical link
shuttle hoist frame : LHD
CAN Node2 : IO Module
CAN Bus
IO
24 VDC
DC : Converter equipment
24 VDC
shift tabel : Motor control module
In
inductive probes : ...
Out
shift table : DC motor
24 VDC
RFID : Sensors
CAN Bus
Shift table : Motion transmission System

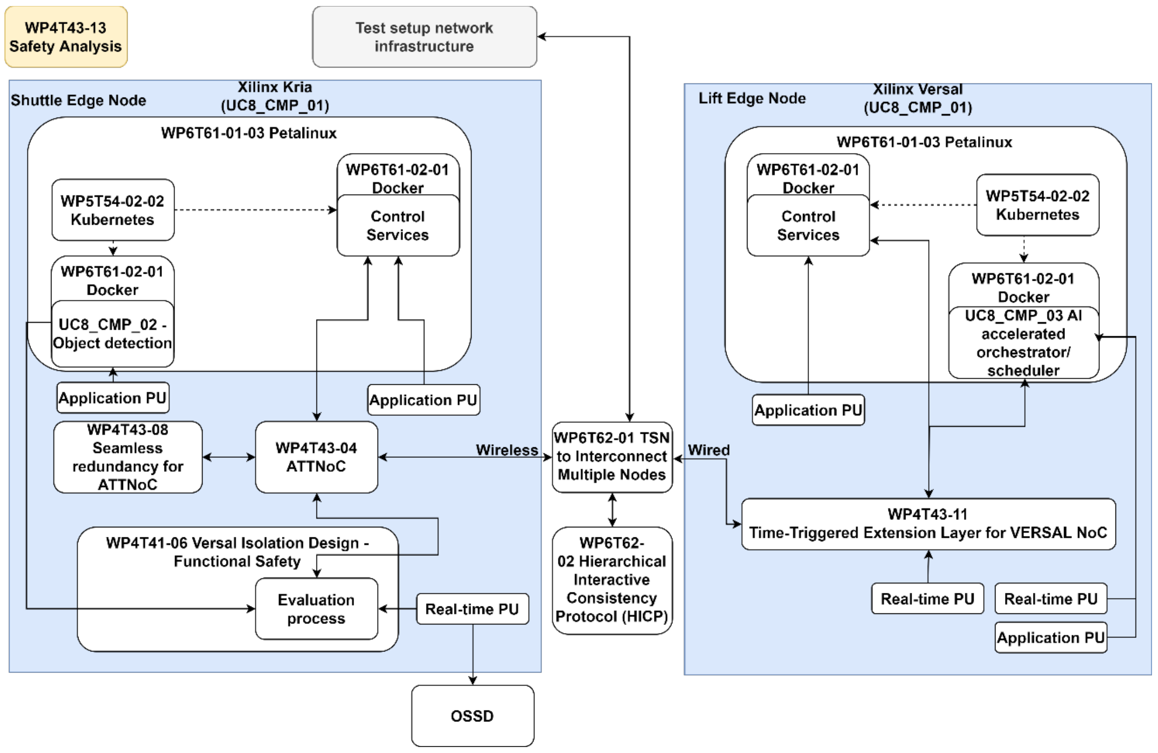
Figure 65 Final lift architecture

12.2.4 **Safety concept UC8**

The safety concept for UC8 will be presented in this subchapter. The approach is to explain the concept and to show the traceability to the developed fractal components related to the safety services, either the Kria platform or the Versal platform.

12.2.4.1 Traceability fractal components

The planned fractal components related to the safety concept are shown in Figure 66, Component WP4T43-13 – Safety Analysis is the base for the safety concept and is shown in a separate block without direct connections as a reference. For on- and off-chip communication are network on chip solutions planned, WP4T43-04 – ATTNoC will be implemented in the shuttle edge nodes and WP4T43-11 – TT-Extension Layer for Versal in the lift edge node. The physical connection between the edge nodes is accomplished wireless for the shuttle edge nodes and wired for the lift edge node.



WP4T43-13 Safety Analysis
Test setup network infrastructure
Shuttle Edge Node
Xilinx Kria (UC8_CMP_01)
WP6T61-01-03 Petalinux
WP6T61-02-01 Docker
Control Services
WP5T54-02-02 Kubernetes
WP6T61-02-01 Docker
UC8_CMP_02 - Object detection
Application PU
Application PU
WP4T43-08 Seamless redundancy for ATTNoC
WP4T43-04 ATTNoC
Wireless
WP4T41-06 Versal Isolation Design - Functional Safety
Evaluation process
Real-time PU
OSSD
WP6T62-01 TSN to Interconnect Multiple Nodes
WP6T62-02 Hierarchical Interactive Consistency Protocol (HICP)
Wired
Lift Edge Node
Xilinx Versal (UC8_CMP_01)
WP6T61-01-03 Petalinux
WP6T61-02-01 Docker
Control Services
WP5T54-02-02 Kubernetes
WP6T61-02-01 Docker
UC8_CMP_03 AI accelerated orchestrator/ scheduler
Application PU
WP4T43-11 Time-Triggered Extension Layer for VERSAL NoC
Real-time PU
Real-time PU
Application PU

Figure 66 Traceability of fractal components related to safety services in UC8

Safety related telegrams shall be used to extend the system reaction regarding specific events, like the access request from maintenance staff or the detection of human bodies in range.

12.2.4.2 Description

The access to the system is limited to one defined point, the location is near the control cabinet like shown in Figure 63. A safety door lock for the system access will be used, which is connected to a safety relay in the control cabinet. This safety relay handles also the emergency stop circuit. The first safety function is fulfilled by this logic.

From the shuttle perspective, the object detection more specifically the human body detection, gives more flexibility in operating mode. The shuttles are still allowed to stay in operational mode as targeted from the risk assessment, even if a request is made for system access. To accomplish this safety function, an evaluation service will be deployed in an isolated part of the FPGA and generates outputs to the F-PLC of the shuttle under the defined rules of behavior. The outputs are used to send information about the detected state, either the detected person is in a warning or danger zone. The F-PLC is responsible for monitoring important IOs, evaluate them and switching off moving parts in an event of error, like the OSSD from the Kria platform.

In the state of closed system, the cameras of the shuttles are turned off to safe power during operational tasks. While requesting access to the system, a telegram will be generated to turn on the evaluation with the person detection model. During this process, the lift will be set in a safe position before the access will be granted by the door lock. The shuttle's behavior will be defined as follows:

- Person detected in danger zone:
    - Turn off immediately moving parts
    - Send information about detected person with coordinates and status
- Person detected in warning zone:
    - Reduce target speed to 0.3 m/s
    - Send request to reschedule tasks
    - Send information about detected person with coordinates and status
- Person out of both zones:
    - Reduce target speed to 0.3 m/s
    - Stay in operational mode
    - Send information about detected person with coordinates and status
- No person detected:
    - Stay in operational mode

The telegram "Send information about detected person with coordinates and status" is used to reschedule shuttle tasks and therefore avoid collisions with the maintenance staff. This approach is aimed to hold the targeted throughput of containers in the system, even when interrupts in the system are occurring.

[BEEA-UC1] IEC 62061:2012 Safety of machinery - Functional safety of safety-related control systems

13 Conclusions (SIEG)

In this document, the preliminary implementation for safety services described in the previous deliverable D4.1 has been elaborated. Several building blocks and components that contribute to meeting the T4.3 objectives have been reported. Specifically, they are:

- Time-Triggered Network-on-Chip for VERSAL platform and non VERSAL Platform, including fault tolerance techniques for the Time-Triggered NoC
- Software diverse redundancy library
- Performance monitoring services
- Safety services for PULP-Systems
- FPGA Based fault injection
- Safety analysis of the NOEL-V Platform redundant acceleration scheme.
- Safety concept based on ISO 26262 and safety concept based on IEC 62061.

In particular, the explanation, design, implementation, and testing of these components have been reported.

14 List of figures

Figure 1 Fractal System Architecture 7
Figure 2 Document Organization...10
Figure 3 FRACTAL Big Picture 11
Figure 4 Time-Triggered On-Chip Communication location in the big picture 12
Figure 5 Software diverse redundancy library in the big picture 13
Figure 6 Performance monitoring services in the big picture...............................14
Figure 7 Safety services for PULP systems in the big picture 14
Figure 8 FPGA based fault Injection in the big picture.......................................15
Figure 9 Safety analysis of the NOEL-V Platform redundant acceleration scheme in the big picture...16
Figure 10 Safety concept based on ISO26262 in the big picture..........................16
Figure 11 TTEL on VERSAL NoC 17
Figure 12 Block Diagram of VERSAL NoC **[SIEG-TTEL2]**..................................18
Figure 13 Time-Triggered Extension Layer for VERSAL NoC **[SIEG-TTEL3]** 19
Figure 14 Block Diagram of VERSAL NoC with TTEL..22
Figure 15 Block Diagram of VERSAL NoC without TTEL 22
Figure 16 Experiment 1 23
Figure 17 Experiment 2 23
Figure 18 Experiment 3 23
Figure 19 Experiment 4 23
Figure 20 Experiment 5 24
Figure 21 Experiment 6 24
Figure 22 Experiment 7 24
Figure 23 Experiment 8 24
Figure 24 Adaptive Time-Triggered Network-on-Chip (NoC) **[SIEG-ATTNoC1]**....26
Figure 25 Switching between Schedules 27
Figure 26 Adaptive Unit in TTNI...27
Figure 27 State Machine for Adaptive Control Unit 28
Figure 28 Example of Circular Linked List Schedule 29
Figure 29 Seamless Redundancy in TTNoC..30
Figure 30 SCNIs Architecture 31
Figure 31 Simulation Result of Scenario 1, NI sender 32
Figure 32 Simulation Result of Scenario 1, NI receiver 33
Figure 33 Messages return to the core Interface 34
Figure 34 Seamless redundancy in TTNoC with faulty Router.............................35
Figure 35 Simulation result of Scenario 2, NI Sender 36
Figure 36 Simulation result of Scenario 2, NI Receiver 37
Figure 37. Matrix multiplication wrapper 40
Figure 38. Call to the monitor passing as argument the matrix multiplication wrapper 41
Figure 39. Example of use of WP4T43-03 42
Figure 40. Code sketch of the time monitoring service 44
Figure 41. Example of statistics unit configuration 44

Figure 42. Example trace of validation tests for the timing monitoring service 46
Figure 43 Byte store with ECC 47
Figure 44 On-Demand Redundancy Grouping : two different operation mode available (i) Redundant mode , (ii) Performance mode 48
Figure 45 Injection modes of the DAVOS-FI tool: area-based (left) and scope/ hierarchy-based (right) 51
Figure 46 Architecture of the DAVOS-FFI tool 52
Figure 47 The workflow of DAVOS-FFI tool...53
Figure 48 Fragment of FPGA design model used in DAVOS for the mapping of essential bits 54
Figure 49 Configuration of DAVOS-FFI tool for (a) scope-based injection targeting LUTs of core-a, and for (b) area-based injection targeting essential bits of core-0 55
Figure 50 Fragment of FFI trace at the end of first experiment (targeting LUTs) ...55
Figure 51 Estimated percentage of failures (SDC and Hangs) resulting from bit-flips in configuration bits of core-0 (LUT in exp-1, and essential bits in exp-2) 56
Figure 52 Querying and visualizing FFI results by means of DAVOS web interface.56
Figure 53 Acceleration subsystem with N-modular redundancy and majority voting 58
Figure 54 Percentage of failure modes of 2oo2 redundancy scheme estimated by FFI experiments...61
Figure 55 Safety concept 64
Figure 56 Work flow using the standard...65
Figure 57 Integration of the developed Safety SW within the learning procedure without reinforcement learning..67
Figure 58 Integration of the developed safety SW could be performed using reinforcement learning 68
Figure 59 Overview of safety activities in concept phase of ISO 26262 68
Figure 60: Overview of system block diagram of SPIDER 70
Figure 60 Examples of functional safety standards derived from IEC 61508 80
Figure 61 Relationship of IEC 62061 to other relevant standards 81
Figure 62 UC8 test setup - schematic top view 84
Figure 63 Final shuttle architecture 87
Figure 64 Final lift architecture 88
Figure 65 Traceability of fractal components related to safety services in UC8 89

15 List of tables

Table 1 Summary of the components developed in T4.3 8
Table 2 Experiment setup without TTEL, ISOCHRONOUS (100MB/s)....................20
Table 3 Experiment setup with TTEL, ISOCHRONOUS (100 MB/s) 20
Table 4 Experiment setup without TTEL, BEST EFFORT (100MB/s) 20
Table 5 Experiment setup with TTEL (BEST EFFORT 100 MB/s)...........................21
Table 6 Experiment setup without TTEL (ISOCHRONOUS 50 MB/s) 21
Table 7 Experiment setup with TTEL (ISOCHRONOUS 50 MB/s)..........................21
Table 8 Experiment setup without TTEL (BEST EFFORT 50 MB/s)........................21
Table 9 Experiment setup with TTEL (BEST EFFORT 50 MB/s) 21
Table 10 Definition of common failure modes observed during FFI experiments....51
Table 11 Diagnosis and recovery actions in the case of single replica 60
Table 12 Diagnosis and recovery actions in the case of two replicas....................60
Table 13 Diagnosis recovery actions in the case of three replicas (TMR) 60
Table 14 Design principles for software unit design and implementation 66
Table 15 Methods for software unit testing 66
Table 16 HAZOP guide words 72
Table 17 HAZOP for control unit and main functions...73
Table 18: Hazards resulting from an item’s malfunctions 74
Table 19: Situations for possible hazardous events 74
Table 20: Risk matix for ASIL determination 75
Table 21: Safety Goals 76
Table 22 Degree of Possible Harm 82
Table 23 Breakdown by likelihood of occurence 83
Table 24 Breakdown by frequency of exposure and number of persons at risk......83
Table 25 Breakdown by risk potential value 84
Table 26 Iterative procedure of a hazard by kinetic energy for the lifts................85
Table 27 Iterative procedure of a hazard by kinetic energy for the shuttle 86
Table 28 List of abbreviations...95

16 List of Abbreviations

Table 28 List of abbreviations

ACAP	Adaptive Compute Acceleration Platform
ACU	Adaptive Control Unit
AGV	Automated Guided Vehicles
ATTNoC	Adaptive Time-Triggered Network-on-Chip
AXI	Advanced eXtensible Interface
CI	Core Interface
CNN	Convolutional Neural Network
DCLs	Dual Core LockStep
DF	Dangerous Filure
DPR	Dynamic Partial Reconfiguration
DUT	Design Under Test
EBU	Egress Bridging Unit
ECC	Error-Correcting Codes
FFI	FPGA-based Fault Injection
FPGA	Field Programmable Gate Array
GW	Gateway
HNoC	Horizontal NoC
IP	Intellectual Property
IPC	Industrial Personal Computer
ISO	International Organization for Standardization
LHD	Load Handling Device
LUT	Look Up Table
MPSOC	Multiprocessor System-on-a-Chip
NIDB	NoC Inter-Die-Bridge
NMU	NoC Master Unit
NOC	Network on Chip
NPS	NoC Packet Switches
NSCTTNIs	Non-Safety-Critical Time-Triggered Network Interface
NSU	NoC Slave Unit
ODRG	On-Demand Redundancy Grouping
PL	Performance Level
PLC	Programmable Logic Controller
PMU	Performance Monitoring Unit
PULP	Parallel Ultra-Low Power
QoS	Quality of Service
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RMI	Reconfiguration and Monitoring Interface
SCTTNIs	Safety-Critical Time-Triggered Network Interface
SECDED	Single Error Correction and Double Error Detection

SLRs	Super Logic Regions
SOC	System on Chip
SoC	System-on-Chip
SW	Software
TMR	Triple Modular Redundancy
TT	Time-Triggered
TTEL	Time-Triggered Extension Layer
TTNI	Time-Triggered Network Interface
UC	Use case
VNoC	Vertical NoC
WI-FI	Wireless Fidelity
XSCT	Xiling Command-Line Tool