Universidade do Minho
Escola de Engenharia

Emanuel José Rodrigues da Silva

Safety-Critical Applications in a Multicore Environment

Junho de 2021

Safety-Critical Applications in a Multicore Environment

Emanuel Silva

UMinho | 2021

Universidade do Minho
Escola de Engenharia

Emanuel José Rodrigues da Silva

Safety-Critical Applications in a Multicore Environment

Dissertação de Mestrado
Mestrado em Engenharia Eletrónica Industrial eComputadores
Sistemas Embebidos e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Jorge Cabral

Junho de 2021

Acknowledgements

First of all I would like to thanks my family, especially my parents and my brother Tiago. Without the support and the understanding of my parents this road would be absolutely much more difficult. My brother Tiago was always there setting the right path for me, calling me to reason a bunch of times even when the times went darker. For sure, those were the pillars in which I have been relying all these years.

Secondly I would like to thanks Professor Jorge Cabral for all the technical support throughout this year. It was an overwhelming year for both of us but even though we were able to work around and met a few times in order to trade some knowledge.

Last but not least, I would like to give a huge thanks to Bosch team who make this project available and give me the opportunity to work with an amazing team. It was a real pleasure to work with the Eng. 31 team.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Resumo

A indústria automóvel acompanha a evolução tecnológica exponencial a que estamos sujeitos nos dias de hoje. As mais recentes novidades nos automóveis são resultado de melhorias eletrónicas (*hardware* & *software*) levando a um aumento na complexidade dos sistemas Elétricos e Eletrónicos [1]. Toda esta complexidade está passível a falhas que podem surgir pelas mais diversas razões, nomeadamente causas naturais ou deficiências de conceção/fabrico. Infelizmente é impossível eliminar todas as falhas da equação, contudo um sistema altamente confiável deve ter em conta os mais diversos cenários e reportar caso detete algo inesperado. Só assim podemos estabelecer um sistema seguro (*safe*). Como é que se pode medir este tipo de segurança? Um sistema seguro é aquele que não compromete a integridade física do seu utilizador ao colocá-lo em situações de risco. A industria automóvel tem este parâmetro de segurança bem presente na conceção dos seus projetos assim como nas tecnologias que usa, e o *steer-by-wire* não é exceção. O *steer-by-wire* é a inovação que vem revolucionar a próxima geração de veículos [2]. O propósito desta tecnologia consiste na substituição de componentes hidráulicos por componentes totalmente elétricos/eletrónicos no âmbito do sistema de direção do automóvel [3].

Com esta dissertação pretende-se analisar os mecanismos de *lockstep* presentes nos microcontroladores modernos e tirando partido deles, desenvolver uma arquitetura capaz de reagir a falhas fazendo uso de uma abordagem de tolerância a falhas em ambientes de multiprocessador. Esta arquitetura será usada no projeto SPA (Sensor de Posição Angular) e este projeto faz parte da tecnologia steer-by-wire. Qualquer mecanismo de tolerância a falhas é baseado em redundância [4], e é em redundância que se baseia *o lockstep* assim como se deve basear a arquitetura que se pretende desenvolver. Toda a arquitetura desenvolvida deverá a assegurar os requisitos de segurança que são característicos à tecnologia.

Keywords: *Lockstep*; Multicores; Segurança Critica; Sistemas Embebidos; Tolerância a Falhas.

Abstract

The automotive industry has been following the exponential evolution in technology present in our days. Most of the advances seen in vehicles are the result of better electronics (hardware and software), which consequently increase the complexity of those systems [1]. No complexity is fault-free, which can arise due to many reasons, from natural causes to a mistake in the assembly process. Eliminating all the events that could lead a system to a failure state is at very least challenging if not impossible; however, with the usage of fault-tolerant approaches, the system can be surprisingly compliant with safety requirements. But how can safety be measured? A safe system is one that must not harm people, not even put them in dangerous circumstances. The automotive industry takes this safety parameter very seriously in the conception of its projects or technology that they use, and the *steer-by-wire* is no exception. The *steer-by-wire* is the innovation coming to revolutionize the next generation of vehicles [2]. The technology consists of replacing the mechanical parts of a vehicle with systems totally electric/electronic ones at the steering system of a car [3].

This dissertation aims to analyse the *Lockstep* mechanism present in recent microcontrollers and by making use of it develop a fail-operational architecture using a fault-tolerant approach for automotive applications in a multiprocessor environment. The architecture developed intends to be used within the APS (Angular Position Sensor) project, which is part of *steer-by-wire* technology. Any fault-tolerant mechanism is based on redundancy [4] and it is in redundancy that lockstep is based as must be the architecture that intends to be developed. The fail-operational architecture to be developed must be compliant with safety requirements.

Keywords: Fault-Tolerance; Fail-operational; Embedded Systems; Multicore; Lockstep;

Table of Contents

List of Figures

List of Tables

Acronyms List

ABS Anti-lock Braking System.

ADC Analog to Digital Converter.

ADAS Advanced Drivers Assistant Systems.

ADR Adaptive Data Rate.

AES Advanced Encryption Standard.

API Application Programming Interface.

APS Angular Position Sensor.

AUTOSAR Automotive Open System Architecture

BIST Built-In Self-Test.

CAN Controlled Area Network.

CAN-FD Controlled Area Network with Flexible Data.

CPU Central Processing Unit.

CRC Cyclic Redundancy Check.

DCLC Dual Core Lockstep.

DMR Dual Modular Redundancy.

DSP Digital Signal Processor

ECC Error Correction Code.

ECUAL Electronic Control Unit Abstraction Layer.

ECU Electronic Control Unit.

EIM Error Injection Module.

ERM Error Reporting Module.

ESP Electronic Stability Control.

FCCU Fault Collection and Control Unit.

FPGA Full Programmable Gate Array.

FPU Floating Point Unit

GND Ground.

GPIO General Purpose Input Output.

HAL Hardware Abstraction Layer.

IDE Integrated Development Environment.

ISO International Standards Organization.

ISR Interrupt Service Routine.

MCAL Microcontroller Abstraction Layer.

MCU Microcontroller Unit.

MMR Multiple Modular Redundancy.

MTTF Mean Time to Failure.

MTTR Mean Time to Repair.

PIT Programmable Interrupt Timer.

RTOS Real Time Operating System.

SoC System on Chip.

SWC Software Component.

SWD Serial Wire Debug.

TMR Triple Modular Redundancy.

TCLS Triple Core Lockstep.

USB Universal Serial Bus.

Chapter 1

Introduction

The automotive industry has evolved, and as a result, the product that the industry delivers has been growing in complexity to provide a unique experience to the user. Every year the industry releases cars with a new set of specifications which consequently design a path for the incoming ones. In this context and focusing on the next vehicle, autonomous driving is the next step to achieve but until then there is a long way to go and a set of technologies/projects must first be developed. All those projects/technologies must be developed according to the safety and security requirements created by the International Organization of Standards (ISO), to provide a fully reliable system.

Steer-by-wire is a technology that has been developed as long as autonomous driving has been innovating. Autonomous driving relies on the success of *steer-by-wire* (among many other technologies) as it is impossible to drive a car without any steering system. Besides, *steer-by-wire* technology has great advantages such as reducing weight, and with that being done, consumes and pollute gases will be reduced as well. For that, all the hydraulic parts at the steering system used nowadays must be replaced which is the core idea of the *steer-by-wire* technology. To be more precise, the technology consists of the elimination of the steering column that links the steering wheel to the front axles of the car and by replacing all the mechanical actuators with electric/electronics circuitry [3]. The source of this concept comes from the x-by-wire, with origin in 1972 by NASA [5]. The “x” in x-by-wire name stands for the system in which is intended to replace the hydraulic/mechanical parts for the electronic/electric ones. Since there is a replacement of those classic components, and normally they are the ones ensuring the well-functioning of the system, even when some undesirable scenario happens (e.g. assistance steering failure), these systems must guarantee that a system failure does not lead to a state in which human life or surrounding environment are endangered. Additionally, they must ensure that a single failure of one component does not lead to a failure of the whole system [6]. These systems must base their entire architecture in redundancy, with the purpose of mitigating possible faults that could outcome from the

lifecycle of it. For that reason, the concepts of safety and fail-operational must be enhanced in these types of systems.

At the processor level, there are several mechanisms that could be used to improve the safety of a system. Lockstep is one of those mechanisms. It consists of replicated processors executing the same set of instructions and compare its outputs with the purpose of finding inconsistencies between them. If the output of an instruction is different between the locked processors, then it means that a fault has occurred and must be treated in such a way that the overall system is capable to tolerate the fault.

Based on these fault-tolerant mechanisms and having in mind that the concepts of safety and fail-operational must be enhanced, this dissertation aims to develop an architecture considering all the previously mentioned. The architecture developed is to be used in APS (Angular Position Sensor), which is a sensor integrating the steer-by-wire technology. The APS is the sensor placed in the steering wheel/steering column of a vehicle and is responsible to get the intention of the user when he is moving the wheel.

1.1 Contextualization

In a world where changes are constant, technological advances are a certainty, new necessities are inevitable. The necessity that comes out of the technology advance in automotive industry is to provide a fully reliable system (vehicle) which must not cause any harm in their users or surrounding environment. As one can imagine, a vehicle (the system) relies on the proper functioning of other subsystems such as the steering subsystem, braking subsystem, traction subsystem, and many others. Most of the innovations in a vehicle raises many questions regarding safety and security that must be answered. In the scope of any subsystem, safety and security measures must be fulfilled in a way not to compromise the system. Safety and security are closely interrelated concepts that pertain to the protection of lives and assets. While safety is protection against hazards (accidents that are unintentional), security is a state of feeling protected against threats that are deliberate and intentional [7]. Regarding this matter, the International Standards Organization (ISO) is developing guidelines and demands to respect those measures in the automotive field [8] that any system engineer in the automotive industry must respect.

Within this dissertation scope, safety is of extreme importance since the goal is to develop an architecture to be applied in the automotive field. As stated above, safety refers to protection against potential hazards, in other words, faults, and errors that could be causing those threats should be eliminated from the system. With that in mind, this dissertation aims to achieve a fail-operational

architecture. The system in which the architecture is applied, when facing faults, still needs to provide its supposed function, mitigating/tolerating faults that could lead a system to an undesirable state. It is important to note that fault tolerance is unreachable without any type of redundancy [9]. Fault tolerance approaches at the processor level are widely used in industries such as automotive and avionics [10]. For instance, in avionics, there are plenty of configurations liable to be used from TMR(Triple Modular Redundancy) to DMR (Dual Modular Redundancy). Although, TMR is the most famous among them. The proposed architecture will be using redundant hardware and lockstep mechanism to be compliant with fault-tolerant requirements avoiding common mode failures and providing the proper function of the system (APS system) even when faults are induced at the system. The Lockstep mechanism is a configuration of multicore processors where two cores (DMR) or more (TMR) are working as one. In other words, cores execute the same set of instructions and compare their outputs identifying possible inconstancies between them and reporting the case if necessary. Usually, this report is made by triggering an ISR (Interrupt Service routine). The configuration used in this dissertation was DMR, so two processor cores were used in lockstep.

1.2 Motivation

Increasingly safety is nowadays a requirement in most of the daily basis commercial systems. The requirement, leads to an increase in the demand for fail-operational systems, boosting the development of new architectures and new systems with fault-tolerant capability. It opens opportunities to research distinct techniques and approaches for implementing high reliability and safety systems, which was the main motivation to embrace this research journey.

Another interesting point is that we are in the golden age of processor architectures with the appearance of processors with built-in lockstep. Thus, new approaches must be thought of to use this fault tolerance mechanism.

All together with the high demands on the industry for these types of features give the necessary motivation to pursue and develop new architectures enabling the system to be fully fail-operational.

1.3 Objectives

After taking into consideration the motivation, this dissertation aims to develop a fail-operational architecture to be used within the APS project. This fail-operational architecture covers the acquisition and calculation of the angle of the steering wheel. Therefore, the main goals of this dissertation are as follows:

- Analysis of architectures and fault tolerant mechanisms in the case of study;
- Analysis of software architectures;
- State machine design in a way to react to possible faults;
- Software implementations according with certification ASIL-D of standards ISO 26262;
- Driver elaboration liable to be used in AUTOSAR software architecture;
- Tests and validation of the architecture;

1.4 Dissertation Structure

This document is structured in six chapters, and its structure follows a logical order according to the development process that occurred during this Master's Thesis.

The first chapter introduces the current technological concepts, referring to the context and the motivation for the development of this project, as well as its objectives.

The second chapter explores the concepts which are the basis of this project, and thus gives a more in-depth overview of safety and dependability in systems. It is also mentioned in this chapter The State of the Art for the different kinds of lockstep mechanisms that exist and are considered for use in this project.

The third chapter gives an overview of the system and a further selection of which components were chosen and the reasoning for their choices.

The fourth chapter is divided into two sections corresponding to the hardware and software implementations. It focuses on how this project was developed and explains the path taken.

Chapter five describes the tests that were made, along with some considerations about the obtained results.

Chapter six presents the main conclusions relative to this project, as well as future improvements to the proposed architecture.

Chapter 2

State of the Art

To develop a fail-operational architecture, some technological concepts need to be understood. It is crucial to fully understand the relevance of safety, dependability concept, the redundancy, lockstep mechanisms, and the different existing implementations.

After knowing what a lockstep configuration is composed of and its architecture, it is then possible to make further studies on which type of configuration to use.

This chapter presents a technological overview and discussion on the topics previously mentioned.

2.1 Embedded Systems

Embedded systems are usually everywhere executing several daily tasks, despite often being unnoticed. They can have different sizes and complexities, from a television or even a printer to a smart-watch. Usually, interacting with the outside world through sensors (input), actuators (output), and in some cases, using an interface to the user like an LCD or even a LED, an embedded system characterizes for having hardware and software combined to accomplish a specific task. In many circumstances, it could have mechanical parts to help it with that particular job [11]. Due to their assignment, an embedded system must be application-oriented; this means that engineers that design them must project it with the strictly necessary resources for their application to optimize variables like cost, power efficiency, weight, and performance. However, as an embedded system has requirements and constraints that must be respected, it is necessary to have a trade-off between management and usage of resources. Examples of embedded systems with strong constraints are the ones that have time restrictions, classified as soft and hard real-time. The soft real-time embedded systems are the ones that, if missing a deadline, there is not much harm for the system in what it was incorporated it is somehow allowed to miss a deadline as it is in the case of video streaming. However, in hard real-time, a missed deadline could lead to a catastrophic event as it happens in the Antilock Braking System (ABS) of a car.

Despite everything mentioned, embedded systems must be reliable, assuring the well-functioning in all situations; resilient, backing to a safe state even after the occurrence of a fault; safety and secure, being fault-tolerant and having secure communications. The simple embedded systems usually are programmed without layers of abstraction, in other words, directly in their logic hardware (bare-metal programming). However, with the rising complexities in systems, the usage of an operating system has become fundamental.

2.1.1 Elements of an Embedded System

At the architectural level, an embedded system represents the interaction between hardware and software elements, whose details are hidden in a way to have only information about the behavioural and relational levels. These elements can be internally implemented in the embedded system device or externally implemented interacting with the internal elements as well as with the external environment [12].

An embedded system is generally composed of basic elements necessary to the execution of code, internals peripherals, communication interfaces, and the respective software. In a generic form, an embedded system has the following elements:

- **Central Processing Unit**: Responsible for the execution of code, making the logic and control operations as the entrance and exit of data;
- **Random Access Memory (RAM)**: It is a volatile memory of quick access used to store temporarily the variables needed for the execution flow of code;
- **Flash Memory**: It is a non-volatile memory with access being slower than the access to RAM. Used to store data permanently like the code responsible for the boot of the system, operating system code, programs, and file system;
- **Communication Peripherals**: An embedded system frequently uses communication protocols like the Universal Serial Bus (USB), RS232, and Ethernet for which there is peripheral existent;
- **Input & Output Devices**: As referenced before, an embedded system interacts with the outside world and may or may not have an interface with the user. Some peripherals like Analog-to-Digital-Converter (ADC), audio controllers, General Purpose Input Output controllers (GPIO), among many others, are also in an embedded system.

2.2 Functional Safety

The complexity of today's systems is, for the most, due to electronic and electric (E/E) systems. E/E systems have a major role to play in our daily lives making our tasks easy to accomplish. However, those systems when badly designed could have a terrible effect, possibly causing harm and injuries to those who are using them. The vehicle of today is a product of that complexity and innovation in E/E systems. It is believed that most of the innovation that we see in newer cars is based on the innovation experienced at ECUs that a car contains [13]. Nonetheless, it is important to highlight those electronic devices are not perfect and fault-proof. As addressed previously, when the system is made over a faulty design a high probability exists that it could cause serious damage, especially in a vehicle. With that in mind, the International Organization of Standards was in an inevitable necessity to create some standards regarding functional safety in E/E systems in the automotive domain, the ISO 26262 [8].

2.2.1 Introduction to safety standards

As previously mentioned, the ISO26262 [8] is an international standard focusing on the safety of automotive electrical/electronic systems. Divided into 12 documents, covering the entire product development lifecycle, and designed to ensure that systems developed for road vehicles are composed with an appropriate level of rigor required for their intended application. The standard applies additional constraints to the process of development, focused on the system safety aspects. Safety means one must not harm others. A safe system is one that does not cause harm to people. Of course, no system can be made completely safe, so safety is about an attempt to reduce the potential for harm to an acceptable level. ISO26262 takes a risk-based approach to manage potential harm (often referred to as residual risk), based on three factors:

- **Severity**: the potential harm;
- **Exposure**: the probability of occurrence;
- **Controllability**: the ability of the system to avoid the specified harm.

In other words, risk, as defined in the standards is a combination of the probability of occurrence of harm and the severity of that harm.

Thus, the standards organize the risk into four Automotive Safety Integrity Levels (ASILs). ASIL A is the lowest level while level D is the highest one. For instance, a system classified as ASIL A is Cruise Control. This one may cause inconvenience or minor injury to the driver, which means that the severity is low. On the other extreme, on the ASIL D level, is the electric steering system. This system has the potential to cause significant harm by providing the wrong level of assistance, feedback, or even

completely incorrect output. Later in this document a detailed insight of the ASIL classification is approached.

For this dissertation, the author will just be considering 6 of those 12 parts that compose ISO26262, meaning that some of the subjects covered by the standards will not be addressed. For example, the conception of hardware is not a concern for this dissertation since all the hardware that will be used has already been developed and it is certified as a specific level of rigor for the task that is assigned. The only hardware that it is intended to develop is concerning the communication protocol and it is going to be addressed later in this document. With that said, the parts considered are [14], [15], [16], [17], [18], [19]:

- **Part 1**: defines the language of ISO262626;
- **Part 2**: is an over-arching guide focusing on the management of safety requirements, both from a project and organizational point of view;
- **Part 3**: focuses on what the standards call the concept phase. This phase is considered with initial project definition, establishing the safety requirements and criteria for the project and initiating the safety lifecycle;
- **Part 4**: is concerned with system level development, that is, detailed requirements analysis, system synthesis, functional and logical allocation, and system evaluation, validation and verification;
- **Part 6**: focuses on the software aspects of system design and implementation;
- **Part 9**: gives requirements and guidance with respect to safety analyses. In particular with all aspects related to ASIL-oriented requirements.

2.2.2 ASIL classification and decomposition

To have a better understanding of how the ASIL and safety goals are determined, Figure 2-1 and Table 2-1 can be used. As addressed in the previous chapter, in [8] the guidelines to develop a system in a way to decrease the residual risk of it are described. According to the standards, the residual risk (RR) is a product of the potential harm (C) with the probability of occurrence (E) with the severity (S). Residual Risk can be defined as RR = C*E*S. This equation can be better understood by referring to Figure 2-1 since it depicts the process of hazard analysis and risk assessment. The letters used in the equation stands for the three parameters, in which the residual risk is calculated, Exposure, Controllability, and Severity. Regarding these parameters there are different levels that describe different situations that a system can face.

- **Severity**: describes the extent of the harm from S0 (no harm to any person) to S3 (severe injuries, survival uncertain);
- **Controllability**: represent the probability that the driver, passengers or surrounding environment can avoid the specific harm, from C0 (no harm to any person) to C3 (difficult to control or uncontrollable);
- **Exposure**: describes the probability of being in that particular situation from E0 (unlikely to be) to E4 (highly probable).

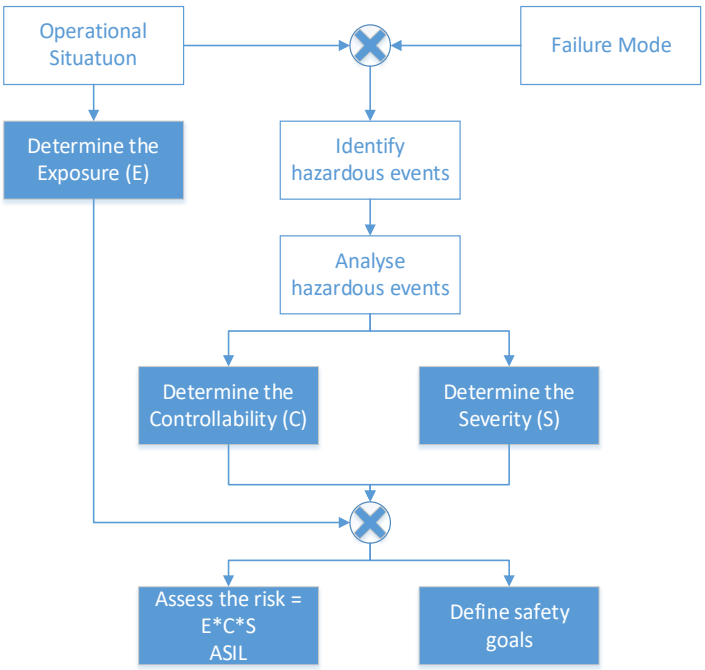

Figure 2-1: Process of the hazard analysis and risk assessment

Table 2-1 has all the combinations possible in a way to establish which are the best safety goals for a specific system. As it can be seen in the table ASIL has a classification from A to D describing the safety goals for a system but besides those four levels, there is also another unit described as QM (Quality Management). The level QM denotes that it is not required to cope with the requirements of the [8], quality management is sufficient.

Severity Class	Exposure Class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
S1	E2	QM	QM	QM
S1	E3	QM	QM	A

S1	E4	QM	A	B
S2	E1	QM	QM	QM
S2	E2	QM	QM	A
S2	E3	QM	A	B
S2	E4	A	B	C
S3	E1	QM	QM	A
S3	E2	QM	A	B
S3	E3	A	B	C
S3	E4	B	C	D

Table 2-1: ASIL determination

Despite specifying restrict rules, in [8] also exists a bit of flexibility in the ASIL classification. According to [19], the standards allow the designer to benefit from a sufficiently independent redundant architecture. ASIL decomposition is a measure to comply with systematic failures by decomposing a single safety requirement into two sufficiently independent requirements and by implementing those requirements in two independent architectural elements. The benefit is the resulting two requirements founded have lower ASIL classification than the initial one. The key principle is if two independent architectural elements are performing the same function then the probability of both failing simultaneously is lower even if their safety integrities are lower than that of the original requirement.

In [19],the following decomposition schemes:

- ASIL D
  - ASIL D = ASIL B (D) + B (D)
  - ASIL D = ASIL C (D) + A (D)
  - ASIL D = ASIL D (D) + QM (D)
- ASIL C
  - ASIL C = ASIL B (C) + A(C)
  - ASIL C = ASIL C (C) + QM (C)
- ASIL B
  - ASIL B = ASIL A (B) + A (B)
  - ASIL B = ASIL A (B) + QM (B)
- ASIL A
  - ASIL A = ASIL (A) + QM (A)

This decomposition can be applied to any level of the system. Hardware components can have this procedure as well as the software components. This can even be used recursively.

2.3 Dependability and Security

Dependability is the ability to deliver service that can justifiably be trusted. However, this definition leads to a problem related to the trust's definition, and that is where the definition of security enters [20]. Security and dependability are highly correlated as is going to be addressed throughout this chapter. Therefore, those concepts must be understood before any development on a safety critical application due to the importance that they have. A dependable and secure system has attributes with which the system is measured, it has threats that compromise the functionality of the system and, finally, it has the means to eliminate these threats. In Figure 2-2, an overview of the taxonomy is depicted in a graph. Throughout this chapter, every element of the tree represented will be explained.

2.3.1 Attributes

The dependability attributes define the properties that a system is expected to have [21]. According to [20], the attributes are composed by: (1) availability, readiness for correct service; (2) reliability, the ability of the system to deliver a correct service continuously; (3) safety, absence of catastrophic consequences to the system external environment, both for the user and the environment; (4) integrity: absence of improper system alterations; (5) maintainability, ability to undergo modifications and repairs. When addressing security, an additional attribute has a great prominence, confidentiality. Confidentiality is defined by the absence of unauthorized disclosure of information. It can be concluded that dependability attributes are a subset of abilities that a system must have to provide the correct service. In [22] is said that dependability attributes are a subset of the non-functional properties that must be specified, analysed, and verified during the system development process. For that reason and according

to [21] and [23], this subchapter is going to focus on the three main attributes, reliability, availability and safety.

2.3.1.1 Reliability

According to the literature, reliability describes the ability of a system to function under stated conditions for a specified period of time [21]. It can be described by a mathematical function given a period of time and given that at the initial the system was in working conditions. The function describes the probability of the system to operate without a failure in the interval. By expressing this attribute in a

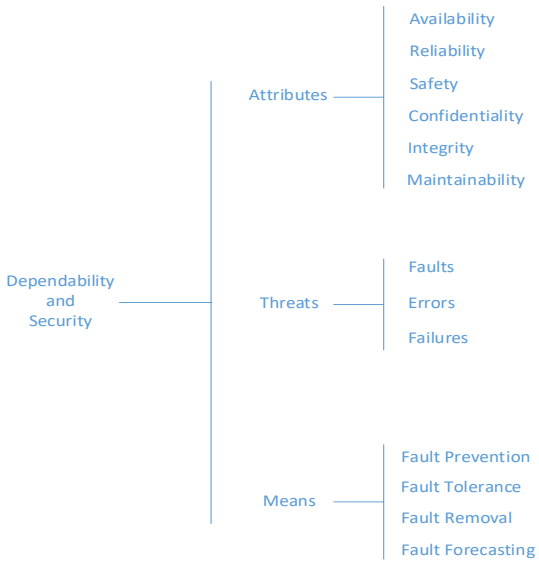

Figure 2-2: Dependability and security taxonomy

mathematical way is it possible then to express the Mean Time to Failure (MTTF). The MTTF is the expectation of the time at which the system will fail. If a system is highly reliable it means that MTTF is larger when compared to MTTR then availability is close to 100% [9].

2.3.1.2 Availability

Availability of a system, likewise reliability, can be expressed by a mathematical function. Similarly, it expresses a probabilistic function that attempts to guess whether the system is functioning or not at the instant of time specified [21]. As stated above for high values of availability, meaning that the system for the most of instants is working as it should, MTTF values should also be larger. Therefore, for smaller MTTF values, availability varies significantly with repair time, the MTTR. The Mean Time to Repair (MTTR), is the expectation of the time to restore a failed system to correct operation. [9] [24]. For that reason, there is another concept that it is important to have in mind, which is undoubtedly related with availability, it is the downtime per year. It represents the amount of time in a year where the system is inoperable.

Said that, it can be deduced that a safety-critical must have a high availability and low downtime per year. Table 2-2 represents the availability and downtime per year for different systems ratings.

System Rating	Availability	Downtime
Routine	99%	3.65 days
Essential	99.9%	8.77 hours
Critical	99.999%	5.26 minutes
Safety-Critical	99.99999%	3.16 seconds

Table 2-2: Availability percentage for different system types

2.3.1.3 Safety

In [20] is considered that safety refers to the absence of catastrophic consequences on the users and the environment. For that reason, systems where the functionalities have a critical safety variable this concept is very important. There is a great deal of systems that have safety critical ratings, such as, avionics, power plants and pacemakers. The reason why is mostly because of the definition made in [20]. These systems when facing a fault must not compromise the user that is why when they have these safety-critical ratings they must also have a definition of safe state. The definition of the term safe state leads to the different behavioural models a system can incorporate in the presence of failures. These terms are quite controversial since they have many interpretations. Here the terms according to [13] are going to be presented.

- **Fail-operational systems** remain functional in case of a subsystem failure.
- **Fail-silent systems** enter a state that does not interfere with other safety related systems in case of a failure.
- **Fail-safe** is, if after one or several failures, the system is brought to an active or passive safe state.

2.3.2 Threats

The threats represent in a system context, situations that could lead the system to an erroneous state. These threats are usually referred as errors, fails and, failures. They can come from the most various causes, from an error at the assembly process to an error happening at the run time

2.3.2.1 Fault, Error, Failure

Regarding the definition of faults and failures, this thesis will focus on the definitions of the [14] and the recursive interpretation of the [19]:

- **Fault** is an abnormal condition that can cause an element or an item to fail.
- **Error** is discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition.
- **Failure** represents termination of the ability of an element, to perform a function as required.

The recursive definition mentioned, that the terms fault and failure can be used recursively, is shown in Figure 2-3.

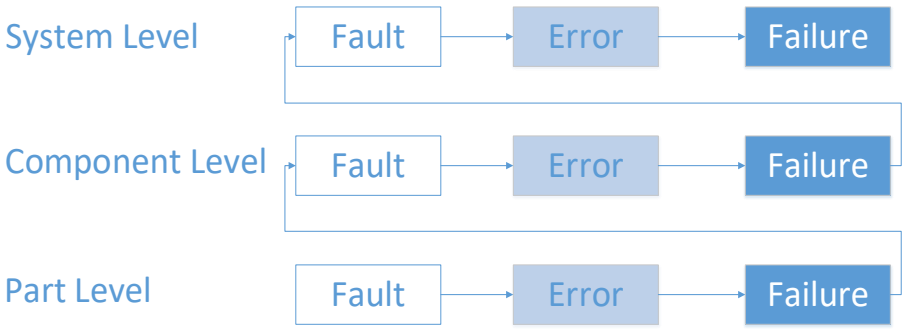

Figure 2-3: Recursive definition of faults and failures

Faults and failures can be categorized based on their origin as systematic- or random HW faults. Systematic faults and failures are deterministic. They can be eliminated only by improving the design or the production process. Typical causes for systematic failures are poor design (e.g. wrong SW specification) or manufacturing-related issues (e.g. contaminated soldering). Random HW failures on the other hand occur in an unpredictable manner. They are mainly caused by ageing or environmental factors. It is obvious that SW is prone to systematic failures only.

Another classification of faults is based on their temporal behaviour. Permanent faults occur and remain until removed or repaired. Transient faults, on the other hand, occur and disappear subsequently. Transients can occur in HW elements for example due to cosmic radiation. Related to the terms of faults and failures are the often misunderstood, hazard and hazardous event. A hazard is defined in [8] as potential source of harm. Only those failures can be hazards, where there are operational situations, in which the failure can lead to an accident. A hazardous event is exactly this failure (= hazard) combined with this unfavourable operational situation. For example, unintended gear change from the 5th to the 2nd is harmless when the vehicle is coasting at 60 km/h in a straight line. But the very same failure mode can lead to an accident when occurring during high speed cornering.

2.3.3 Means

Dependability means are the methods and techniques enabling the development of a dependable system. As it can be seen in Figure 2-2, there are four different means. During this chapter it is intended to address all of them but with more focus on fault tolerance since this dissertation will make use of that mean. It is important to note that the different means represented in Figure 2-2 take their role in different phases of the product lifetime. For that reason, before getting into the concepts, a table is presented where each mean makes sense to enter. In Table 2-3 the system lifetime is divided in two groups where pre-service consists in the development phase and the in-service represent the phase where the system is already commercialized and it is in operational conditions. Each mean has correspondent assignment in the table. If a "tickle" is presented in front of the mean, could be said that the appliance of that mean make sense at that phase of lifetime. On the other hand, if it has an "x" it means that the mean is not applicable on the phase.

Means	System Lifetime Phases	
	Pre-Service	In-Service
Fault Preventing	Design & Implementation	×
Fault Removal	Test & Debug	Preventive Maintenance
Fault Forecasting	✓	✓
Fault Tolerance	×	✓

Table 2-3: Dependability Means and their use cases [25]

2.3.3.1 Fault Preventing

Fault prevention means to prevent the occurrence or introduction of faults [20]. It is achieved by quality control techniques during specification, implementation, and fabrication stages of the process. For hardware, this includes design reviews, component screening, and testing. For software, this includes structural programming, modularization, and formal verification techniques [26]. A rigorous design review and well elaborated set of tests may eliminate specification faults. If it is efficiently tested many of its faults and component defects are avoided.

2.3.3.2 Fault Removal

Fault removal means to reduce the number and severity of faults [20]. It is performed during the development phase as well as during the operational life of a system. During the development phase,

fault removal involves three steps: verification, diagnosis, and correction. Fault removal during the operational life of the system consists of corrective and preventive maintenance.

Verification is the process of checking whether the system meets a set of given conditions. If it does not, the other two steps follow: the fault that prevents the conditions from being fulfilled is diagnosed and the necessary corrections are performed. In preventive maintenance, parts are replaced, or adjustments are made before failure occurs. The goal is to increase the dependability of the system over the long term by staving off the ageing effects of wear-out. In contrast, corrective maintenance is performed after the failure has occurred in order to return the system to service as soon as possible [21].

2.3.3.3 Fault Forecasting

Fault Forecasting means to estimate the present number, the future incidence, and likely consequences of faults [20]. It is done by performing an evaluation of the system behaviour with respect to fault occurrences or activation. The evaluation can be qualitative which aims to rank the failure modes or event combinations that lead to system failure or quantitative, which aims to evaluate in terms of probabilities the extent to which some attributes of dependability are satisfied. Simplistic estimates merely measure redundancy by accounting for the number of redundant success paths in a system. More sophisticated estimates account for the fact that each fault potentially alters ability of the system to resist further faults [21].

2.3.3.4 Fault Tolerance

According to the threats approached in 2.3.2.1 a fault is a malfunction in one component. That malfunction can lead the affected component malfunctioning which may propagate to a failure. Fault tolerance means to avoid service failures in the presence of fault [20]. Fault tolerance is achieved by using some kind of redundancy. The redundancy allows a fault either to be masked or detected. This terms of detection and masking will be addressed in detail in this subchapter while the importance of redundancy in fault tolerance is approached on the next chapter. First, before getting to know what detection and masking are, it is important to highlight that redundancy alone does not make a system fault tolerant. For instance, two components duplicated and connected in parallel do not make a system fault tolerant, for that it is necessary to have monitoring and processing analysing the results and selecting the correct one. For that reason, masking and detection are important.

Fault masking is the process of ensuring that only correct values get passed to the system output despite the presence of a fault. This is done by preventing the system from being affected by errors by either correcting the error, or compensating for it in some way [27]. Since the system does not show the impact of the fault, the existence of the fault is invisible to the user/operator. For example, a memory protected

by an error-correcting code (ECC) corrects the faulty bits before the system uses the data. Another example of fault masking is triple modular redundancy with the majority voting [28].

Fault detection is the process of determining that a fault has occurred within a system. Examples of techniques for fault detection are acceptance tests and comparison. An acceptance test is a fault detecting mechanism that can be used for systems having no replicated components. Acceptance tests are common in software systems [29]. The result of a program is subjected to a test. If the result passes the test, the program continues execution. A failed acceptance test implies a fault [30]. Comparison is an alternative technique for detecting faults, used for systems with duplicated components. The output results of two components are compared. A disagreement in the results indicates a fault.

Fault location is the process of determining where a fault has occurred. A failed acceptance test cannot generally be used to locate a fault. It can only tell that something has gone wrong. Similarly, when a disagreement occurs during the comparison of two modules, it is not possible to tell which of the two has failed. Fault containment is the process of isolating a fault and preventing the propagation of its effect throughout the system. This is typically achieved by frequent fault detection, by multiple request/confirmation protocols and by performing consistency checks between modules.

Once a faulty component has been identified, a system recovers by reconfiguring itself to isolate the faulty component from the rest of the system and regain operational status. This might be accomplished by having the faulty component replaced by a redundant backup component. Alternatively, the system could switch the faulty component off and continue operation with a degraded capability.

2.4 Redundancy in Fault tolerance

There are various approaches to achieve fault tolerance. Common to all these approaches is a certain amount of redundancy. For our purposes, redundancy is the delivery of functional capabilities that would be unnecessary in a fault-free environment [31]. This can be a replicated hardware component, an additional check bit attached to a string of digital data, or a few lines of program code verifying the correctness of the program results. The idea of incorporating redundancy in order to improve the reliability of a system was pioneered by John von Neumann in [32]. Two kinds of redundancy are possible [33]: space redundancy and time redundancy. Space redundancy provides additional components, functions, or data items that are unnecessary for fault-free operation. Space redundancy is further classified into hardware, software, and information redundancy, depending on the type of redundant resources added to the system. In time redundancy the computation or data transmission is repeated and the result is compared to a stored copy of the previous result. The reason why this is called time

redundancy is due to the fact that this specific technique need significantly more time to have the outputs produced. In order to clarify those concepts, each one is going to be addressed in the next subchapters. Either way, it can be said beforehand that redundancy is not for free, it adds additional resources to systems to improve its reliability. Extending system about one more computer, or doubled available memory, may drastically complicate the system design. Performance, weight, size of the system may be affected, as well as the cost of design and implementation. Appropriate redundancy method must be selected to achieve the goals of the system [34].

2.4.1 Space Redundancy

As previously mentioned, this technique integrates the categories of hardware, software, and information redundancy. All of these categories are achieved by replicating or adding additional components that are unnecessary for the execution of a certain task.

In hardware redundancy the hardware components are replicated in order to cooperate among themselves. Both have the purpose of achieving the same task and they are only added to mitigate the fault scenarios. Taking the computer system example, the hardware redundancy is made through two or more independent computers with their own processor, memories and peripherals. The computers systems are able to cooperate in three ways, according to its implementation: Statically; dynamically or in a hybrid way. In a static implementation all the computers are working in parallel and comparing the results calculated. If inconsistencies appear it means that a fault occurred in one of the subsystems, and the correct output is chosen with a predefined algorithm [35]. In dynamic form only one computer subsystem is working and if it fails then another computer is started to continue the system task. The hybrid implementation uses a combination of both previously addressed implementation, static and dynamic approaches.

Software redundancy uses replicated code for the same function that is intended to deliver. This technique of software redundancy replicates the code to get a compiled redundant machine code that owns different instruction for the same purpose. This software redundancy technique is also known as instruction redundancy. This name is attributed due to the additional instructions that are added to the binary code whenever a spare code is added to the application code. In the spatial redundancy, one part of the code is replicated, e.g., variables or functions. After the replicated code is executed the replicas are compared and checked if they are all equals. If it is not the case, then an error has occurred.

Information redundancy consists of using additional information which is not required to perform the task in question. For instance, the Error Correction Code (ECC) is an additional value attached to the

information content of a memory and it is not required to execute the software task. Nevertheless, it allows to check data correctness and may also be used to correct corrupted data. Another great example of information redundancy is the data backup that everyone is familiar with by using the so famous cloud. The same information is duplicated to reduce the possibility of data loss.

As time redundancy, space redundancy is suitable to deal with transient faults since it is very unlikely to occur in the same memory region twice. For example, if a variable is replicated when it is affected by an error, it is possible to detect and correct it through comparison with the other redundant variables.

2.4.2 Time Redundancy

This technique consumes additional time to get a correct and valid result. In the time redundancy, one specific part of the code is re-executed more than once. Execution results are stored and at the end of all program executions, the stored results are compared. The outputs are verified if they match the execution went as it should, if they do not, it is because an error has occurred. This kind of redundancy in software is suitable to deal with transient faults since they do not occur (or are very unlikely to occur) in the same location twice and cause the same error consecutively. So, the re-execution of the same code should not produce the same transient fault which will be mitigated.

2.5 Lockstep

Lockstep is a fault tolerance technique that uses hardware redundancy at the processor level. Its implementation can differ by the number of replicated cores used as well as by the error recovery techniques that the cores employ. The configuration of redundancy used at this level is static, which means the working principle consists of, having each processor used running at the same time, or in some cases with a delay of one or few clock cycle in the same instruction, then the outputs produced are compared in order to identify possible inconsistencies between cores without even the possibility of changing the implementation. When the inconsistencies appear, it means that some error has occurred. Therefore, since a fault tolerance mechanism is being used, which is the lockstep the error must be treated in order not to propagate it to the whole system and not change the fault into a failure.

During this chapter the different implementations of lockstep that were found interesting are going to be presented for the understanding of this dissertation and after getting to know the implementations the error recover techniques will also be presented.

2.5.1 Lockstep Implementations

Regarding lockstep implementations they can be differed by the way the checker (the unit responsible for finding inconsistencies) is performed and by the number of cores that they use, as it was mentioned previously. Besides, it can have a loosely- or tightly coupled- implementation. In the tightly-coupled hardware lockstep, the processors are running synchronously and the outputs are compared instruction by instruction. The comparisons are continuously being made. An error is detected before it propagates to the outside of the system (causing a failure). This type of lockstep is more robust as the granularity is small. However, it is expensive to implement [36]. In a loosely-coupled implementation, the checker is made with less frequency when compared to the tightly implementation. This does not ensure that each instruction is executed properly, but errors are caught before they are allowed to leave the processors and propagate to other devices. This can also lead to performance boosts over tightly-coupled lockstep designs because fewer comparisons are performed [37]. Due to this, the error detection is weaker. In what concerns the way the checker is performed, the type of the lockstep should be chosen according to the type of application, its safety-critical requirements, and the hardware system constraints.

Now, when it comes to the number of CPU cores used in this technique, there is a great deal of possible implementations combining architectures of lockstep. For the purpose of this dissertation only the Triple Modular Redundancy (TMR) also known as Triple Core Lockstep (TCLS) and the dual modular redundancy (DMR), often called Dual Core Lockstep (DCLS), will be addressed.

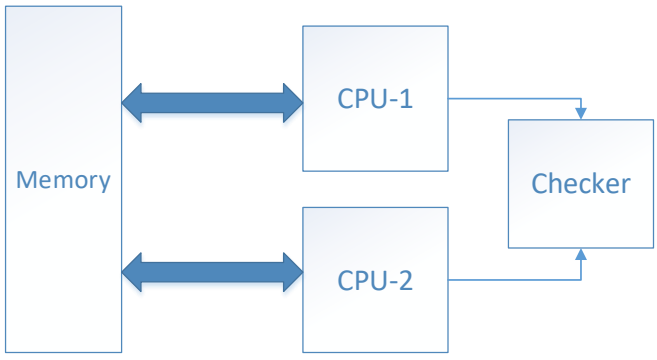

Figure 2-4: Dual Core Lockstep (DCLS or DMR) implementation

2.5.1.1 Triple Modular Redundancy

In the TMR configuration (Figure 2-5) three identical CPUs execute the same code in lockstep and a majority vote of the outputs masks any possible single CPU fault. The memory and communication sub-system faults can be masked employing ECC techniques. Its unique capability of masking any single fault, at the cost of an additional CPU, it offers a 100% error detection coverage within a single clock period. It is possible to "see" the TMR as operating in degraded mode when it is working with just two healthy CPUs [4].

2.5.1.2 Dual Modular Redundancy

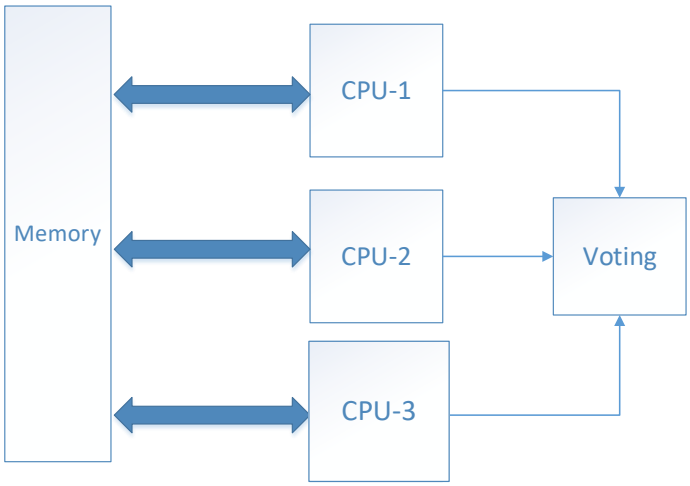

Figure 2-5: Triple modular redundancy (TMR or TCLS) implementation

In dual modular redundancy the configuration is quite different from the previously addressed one. In this case there is no democracy, there is no majority, just the concept of right and wrong values. As it can be seen in Figure 2-4 the outputs produced by both cores are feeding a checker unit which is responsible for identifying the inconsistencies if they appear. If such a scenario occurs it means that the system is facing some kind of fault and if not treated as soon as possible it can propagate to an error and consequently to a failure as seen in section 2.3.2.1. The disadvantage of this implementation is that it does not permit to identify which was the erroneous CPU core so if one is under fault, the system is faulty and must recover. In this implementation there is no fail degraded mode.

2.5.2 Error Recover Techniques

In this phase, depending on the outcome of BIST (Built in Self Test), some action for recovery or containing the error are taken. In the DMR, if a hard error has occurred, the processors are stopped, a fatal error is signalized, and recovering the system from it is impossible. So, the system stops working and switches to a safe state. When a hard error occurs in a system with Multiple Modular Redundancy (MMR), the erroneous processor is disabled, and the other health processors keep their execution as

long as another error in the remain executing processors does not occur. After a error, if the MMR was composed of three redundant instances (TMR), the MMR starts working like a DMR redundant system. In the case the BIST does not detect any hard error, it means that a soft error has occurred. So, in the system with DMR technique, both processors are recovered to a state without any error, since it is unknown what the erroneous processor is. In the opposite side, in the MMR technique, the recovery is made to the erroneous processor only. The system keeps executing with the remaining health processors, and when the erroneous processors are recovered, the MMR changes to its fully functional state without having any execution interruption.

2.6 Conclusion

This chapter gave an overview of all the relevant topics that needed to be considered about safety and dependability. A particular emphasis was given to the fault-tolerance means definition and the importance of redundancy due to the fact that this project will take advantages of some fault-tolerance mechanisms such as lockstep.

The different types of lockstep configurations that currently exist and are relevant for this dissertations scope were also presented and may be addressed to better understand the concept. In addition to all that was mentioned the main advantages and disadvantages of each implementation were also shown.

Chapter 3

System Specification

After having a theoretical insight about concepts and technologies that this dissertation will focus on and having an overview of the lockstep mechanism, it is possible to define the application that it is intended to develop as well as the components that will be part of the system.

Primarily, it is important to outline the application scenario that this dissertation covers which is the calculation of an angle and possible faults that could outcome from the hardware used for that matter. However, on the long run, it is intended to have a functional generic architecture that allows the use of different applications without significant changes in its structure.

This chapter presents the design and specification of the system and its architecture, as well as all the requirements and constraints for the architecture that is intended to develop.

A general overview of the whole architecture is given, taking into consideration the technological study made in Chapter 2.

3.1 Use Case

As mentioned in previous sections, the aim of this dissertation is to develop an architecture that could still perform its intended function even under faults, achieving in this way the fail-operational behaviour. The goal is to keep it generic, enabling it to use in a variety of application scenarios, despite the fact that the main use case for this architecture is to be applied in safety critical applications as it is the steer-by-wire. Steer-by-wire must not allow faults to propagate into failures within the system since this could lead the car to cause serious harm for the people that are using it as well as for the surrounding environment. That is the reason why systems like this have the necessity to have its safety enhanced. Safety critical applications such as steer-by wire must have their subsystems in which they rely on (e.g. APS) with the

highest availability and reliability improving in that way the safety which is considered critical for that matter.

The architecture that it is intended to develop has the aim to be used on APS covering the calculation

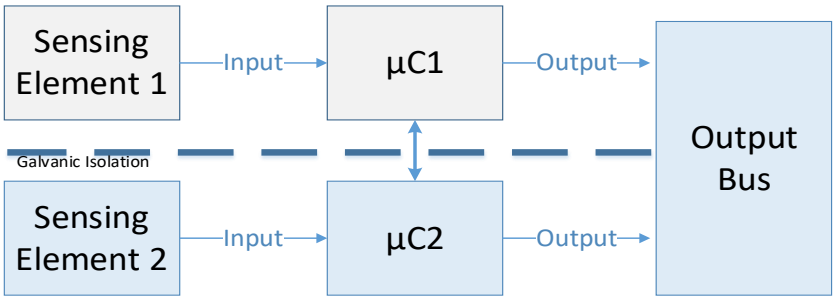

Figure 3-1: Application use case

of the angle of the steering wheel. This angle once calculated is sent to the electronic control unit (ECU) that is responsible for sending the right outputs to the actuators, the motor that controls the movement of car wheels. It can be imagined that if the subsystem responsible for calculating the angle fails, or even the actuator subsystem, all the steering system would consequently fail too. This, due to the fact that the entire steering system depends on its subsystems and there is no redundancy ensured, as it would be if the steering system was conventional (mechanic redundancy ensured by the steering column). In the conventional steering system, if APS would have failed the steering column would ensure the intention of the driver. Since this column links the steering wheel to the axis, the intention would still be transmitted to the car wheels, despite having some functionalities that would not be available anymore such as steering assistance. In order to solve the problem, which is the elimination of the steering column, the architecture that is going to be developed to perform the calculation of the angle of the steering wheel, is going to have the redundancy necessary to ensure the function even when faults occur. An overview of the architecture is shown in Figure 3-1.

3.2 System Requirements

To properly design an architecture, first, all of its requirements and constraints must be defined beforehand, as it is crucial to fulfil them in the decision-making process as well as in the implementation process. With that being said, Table 3-1 is presented. It describes all the conditions that the architecture must comply with to have the system fail operational as it is intended. It is important to note that some concepts addressed at the table are extremely correlated with the theoretical background approached in Chapter 2. The table is divided into 5 sections. First, the requirements are addressed in a general view of the architecture. That being comprehended and having defined what the subsystem is, then the requirements related with the subsystem are pointed out. As addressed throughout this document,

redundancy is necessary thus, redundancy for the software and hardware were also described in the table. Finally, the requirements for the subsystem communication are shown.

ID	Description
1. Architecture	
1.1.	The architecture in order to achieve a fail operational behaviour shall have redundancy, both at software and hardware level.
1.2.	The redundant hardware used, which define a subsystem, shall have galvanic isolation between them.
1.3.	The architecture shall be compliant with ASIL D classification.
1.4.	Each subsystem shall be able to communicate to the outer system using CAN/CAN-FD.
1.5.	A CAN message shall be sent within a fix period of 10 milliseconds.
2. Subsystem	
2.1.	Each subsystem shall have fault-tolerant mechanisms.
2.2.	Each subsystem shall have a communication protocol to exchange data between themselves.
2.3.	The microcontrollers used in each subsystem shall have an ASIL D classification.
3. Redundant Hardware	
3.1.	Two processing units defining the subsystems shall compose the architecture.
3.2.	The signals acquired by both processing units shall have different sources.
3.3.	Each subsystem shall have lockstep mechanism in it.
4. Redundant Software	
4.1.	The software developed shall take advantageous of homogeneous redundancy.
4.2.	The software developed should be able to fit in AUTOSAR architecture.
4.3.	The software shall be developed according with V-model of ISO 26262
5. Subsystem communication	
5.1.	The communication protocol that will be used for communication between subsystems shall be automotive certified
5.2.	The communication protocol that will be used for communication between Microcontrollers shall work with Galvanic Isolation Scenarios
5.3.	The communication protocol that will be used for communication between Microcontrollers shall have multi master
5.4.	The communication protocol that will be used for communication between Microcontrollers shall have error correction codes implemented in order to identify faults

Table 3-1: System Requirements

3.3 System Architecture

As mentioned in the previous sections, the aim of this dissertation is to develop a fail-operational architecture that still provides its supposed function, even when facing fault. Taking into account all the requirements described in 3.1 and the study presented in Chapter 2 it is possible, at this phase, to define the architecture as well as its components. For better understanding, Figure 3-2 can be referred. It depicts the full system stack containing all the different layers division between hardware and software as the elements that compose those layers.

The hardware layer contains the modules that are present in microcontroller unit (MCU) and were

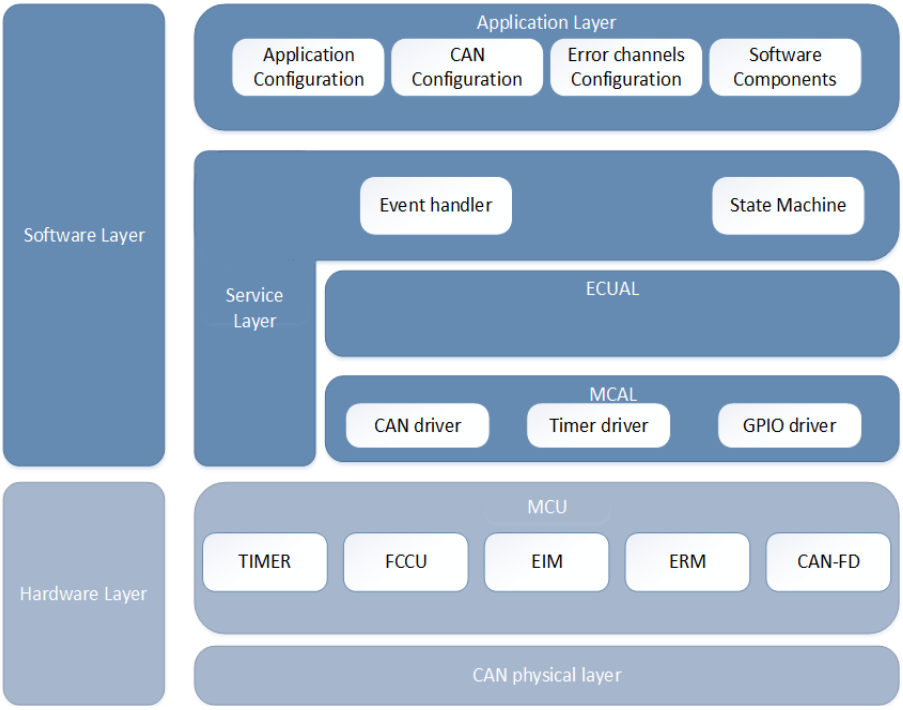

Figure 3-2: System Stack

used for the purpose of the project and of course the MCU. Below this MCU layer the CAN physical layer needed to have the communication performed is presented. All the other modules present in hardware layer are used for both communication and diagnosis purposes.

The software layer contains the modules needed to be developed for the good functioning of this architecture. All the software that is developed followed the guidelines demanded by the AUTOSAR architecture. AUTOSAR architecture is a standard created in consortium of automotive companies to standardize the software developed by those companies. In the MCAL layer all the drivers responsible for managing directly over the hardware are presented. Some of those drivers, such as CAN-FD, were already developed and for this project were only used from an user point of view. At the ECUAL there is no module represented since no module that fits in this layer was developed. Either way, this layer

intended to act like an interface between the upper layers and the MCAL. In this project the communication between the upper layers and the MCAL are going to be made through Service Layer, with exception of CAN-FD module that uses the ECUAL as an interface. Most of the modules of this project are going to be used for diagnostic purposes so it makes more sense to organise the software in that way. At the service layer there is going to be placed the state machine that is intended to be designed to keep the system under control as the event handler. This last one will be the module responsible to react to faults when they occur. In the upper layer, there is all the software responsible for the application. This layer is composed mostly of configurations made at the higher level where the amount of channels that are going to be used to report the errors are defined; CAN-FD channels and initiating all the modules necessary for the project. It is also at this level that the software components are present for the use case application. Nevertheless, for the purpose of this dissertation they are not going to be developed despite being referenced later in implementation to depict how the software will interact.

3.4 Hardware Specification

Having the system requirements and system architecture addressed is now time to go deep into the hardware level. This section intends to describe hardware components that this Master's Thesis will focus on. Considering that at the hardware level is represented essentially by the MCU, and MCU's peripherals, this section will have a top-down approach. First the microcontroller will be addressed followed by MCU's peripherals.

3.4.1 Microcontroller

As stated in 3.1, for each microcontroller used in this system's scope lockstep must be included. With that constraint, the choice of a microcontroller turned out to be very limited. The market does not have a wide choice when it comes to a microcontroller that has lockstep mechanisms certified for the automotive. For that reason and after some negotiations with NXP semiconductors the chosen microcontroller is a S32K2TV. This microcontroller is an evaluation board that NXP provides for the

purpose of this dissertation. It is believed that this board will not be released in the market. However, some variants of it will.

Figure 3-3: S32K2TV Board

The lockstep that this board has, is based in DMR. Said that, and since it is needed to have two cores for fulfilling this feature, the cores that are being used with that purpose are two ARM Cortex M33. These processors are massively used in safety critical applications due to its security specifications and high-performance modules that it has, such as, trust-zone memory, Digital Signal Processing (DSP) and, Floating Point Unit (FPU). Nevertheless, the M33 processors that this board has included in do does not have a trust-zone memory. In S32K2TV as it was said previously, there are two cores running in two-cycle delayed lockstep, meaning, the operation that one core is performing will be executed on the second one with two clock cycles delay.

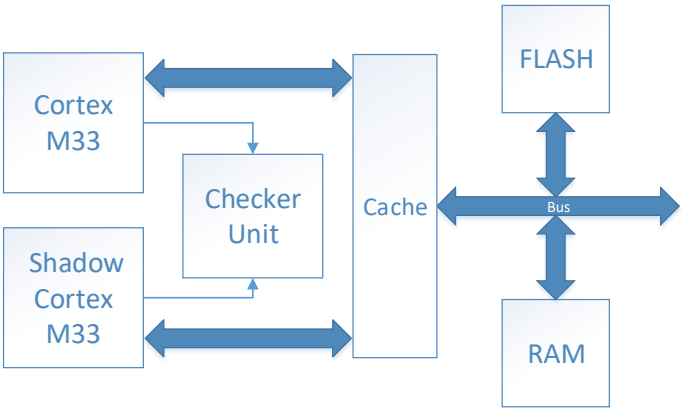

Figure 3-4: MCU's Lockstep

The CM33 core does not contain an inner cache, information about caching is only to signal these settings to any outer lever of cache which may be present in the memory system. This device implements two 8kb instances of outer L1 caches: Code cache and System cache. The code and system caches are accessible through the code and system buses of Cortex M33 core, respectively. The delayed lockstep is

due to the fact that the cores have access to this resource once at a time. In Figure 3-4: MCU's Lockstep a simplistic representation of the lockstep included in the microcontroller is shown.

This microcontroller besides lockstep feature additionally have a bunch of specification that are extremely important to highlight. Here, it is going to be detailed the ones that were found crucial to the implementation of this dissertation. Said that, the features that this board has and were extremely important are [38]:

- ARM Cortex-M33/M7 core, 32-bit CPU
  - M7 supports up to 320 MHz frequency with
  - 2.14DMIPS / MHz
  - M33 supports up to 160 MHz frequency with
  - 1.5DMIPS / MHz
  - ARM Core based on the ARMv7 and ARMv8
  - Architecture and ThumbR-2 ISA
  - Integrated Digital Signal Processor (DSP)
  - Configurable Nested Vectored Interrupt Controller (NVIC)
  - Single Precision Floating Point Unit (FPU)
- Analog mixed signal
  - Up to three 12-bit Analog-to-Digital Converters
  - (ADC) with up to 32 channel analogic inputs per
  - module
  - One Temperature Sensor
  - Up to three Analog Comparators (CMP), with each comparator having an internal 8-bit DAC
  - One 12-bit Digital to Analog Converter (DAC)
- Communications interfaces
  - Up to 20 serial communication interface (LINFlexD) modules, with UART and DMA support
  - Up to ten Low Power Serial Peripheral Interface (LPSPI) modules with DMA support and low power availability
  - Up to two Low Power Inter-Integrated Circuit (LPI2C) modules with DMA support and low power availability
  - Up to eight FlexCAN modules (with optional CAN-FD support)

  - FlexIO module for flexible and high-performance serial interfaces
  - One Ethernet module
  - 2-ch FlexRay module
  - Up to three Serial Audio Interface (SAI) modules
  - One Secured Digital Host Controller (SDHC)
- Reliability, safety, and security
  - Hardware Security Engine (HSE_B)
  - Up to three Internal Software Watchdog Timers (SWT)
  - Error-Correcting Code (ECC) on all memories
  - Error Detection Code (EDC) on data path
  - Cyclic Redundancy Check (CRC) module
  - 120-bit Unique Identification (ID) number
  - Extended Cross domain Controller (XRDC), providing protection for master core access rights

When it comes to the last pack of features related with safety and security most of them will be addressed more in detail in the next sub-section 3.4.2.

3.4.2 MCU's peripherals

This sub-section will present the main modules regarding safety that will be used in order to study and validate the lockstep implemented in evaluation board. It is important to note that despite the modules being used to validate the lockstep mechanism implemented, they are also useful to check the good functioning of modules such as memory and buses of the system.

3.4.2.1 Fault Collection and Control Unit (FCCU)

The FCCU offers a hardware channel to collect faults and to place the device into a safe state when a failure in the device is detected. No CPU intervention is requested for collection and control operation. The distinctive characteristics are:

- Collection of fault information from safety relevant modules on the device.
- Collection of tests results;
- Configurable fault control
- Configurable internal reactions for each NCF
  - No reaction
  - IRQ

  - Short functional reset
  - Long functional reset
  - NMI
- External reactions via configurable output pins
- Watchdog timer for the configuration phase
- Lockable configuration
- One of the fault output signals is high to indicate operational state. The above is not true in case the error out protocol is configured to be a toggling protocol.
- After power on the output signals have high impedance. They show operational state only after configuration by software
- In case of a failure event or on software request for error pin indication, the pin are set to faulty state for a minimum time even if software tries to release it before
- Management of noncritical faults
- HW and SW fault recovery management
- Fault Collection
- Fault Injection

The FCCU circuitry is checked at the start up by the self-checking procedure. The FCCU is operative with the default configuration immediately after the completion of the self-checking procedure. Internal and external reactions are statically defined or programmable. The default configuration can be modified in CONFIG state. The FCCU is designed to function when the clock system is faster than the clock safe.

Regarding the fault recovery management there are two definitions that must be clear:

HW recoverable fault: The fault indication is an edge-triggered and level-sensitive signal that remains asserted until the fault cause is deasserted. That is, if logical 0 on the fault signal indicates fault, then the status flags are valid as long as the fault line stays at 0. The status is automatically cleared when the fault signal goes to 1. Typically, the fault signal is latched external to the FCCU in the module where the fault occurred. The FCCU state transitions are consequently executed on the state changes of the input fault signal. No SW intervention is required to recover the fault condition.

SW recoverable fault: The fault indication is a signal asserted without a defined time duration. The fault signal is latched in the FCCU. The fault recovery is executed following a SW recovery procedure (status/flag register clearing).

HW recoverable is an option to exclude the handling of error sources by FCCU management SW, in case it is known that the fault is recoverable by itself when the fault condition gets corrected.

3.4.2.2 Error Injection Module (EIM)

EIM is integrated between the memory controller and memory array to enable a controlled way for error injection. Each memory controller has its own EIM channel. This module permits the error injection to be activated in a global way or just for a particular channel. It is used mainly for diagnostic purposes and it provides support for inducing single-bit or multi-bit inversions on read data when accessing peripheral RAMs. Injecting faults on memory can be used to exercise the Single Error Correct – Dual Error Correct (SEC-DEC) ECC function of the related system. This module supports 20 error injection channels and also protection against accidental enable reconfiguration error injection function via two-stage enable mechanism. Each of the error injection channels is assigned to a single memory array interface and intercept the assigned memory read data bus and check bit bus, then inject errors by inverting the value transmitted for selected bits on each bus line.

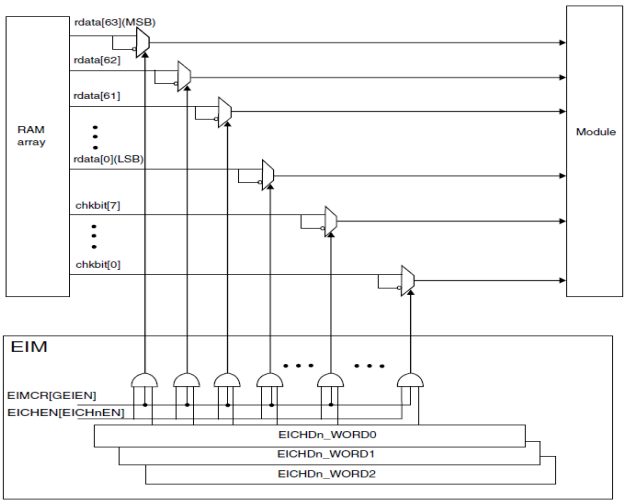

Figure 3-5: EIM module

3.4.2.3 Error Reporting Module (ERM)

The Error Reporting Module (ERM) provides information and optionally interrupt notification on memory error events associated with ECC (Error Correction Code) and parity. The module collects ECC events on memory accesses for platform local memory arrays, such as flash memory, system RAM or peripheral RAMs. It can also record the count value of the number of correctable error events. The diagnostic information is provided by ERM per logical memory:

- For each memory, ERM has status register to detect if it is a single bit or a multi bit ECC
- Faulty system address of the last recorded ECC event on memory n can be read through ERM memory Error address register (EARn)

- ERM configuration register (CR) configures the interrupt notification capability for each supported memory channel
- ECC counter for correctable error count in ERM is resettable through functional reset and by writing all zeros to the COUNT field.

3.5 Conclusion

This chapter started by describing a general architecture capable to implement the angular position sensor following an AUTOSAR-like approach. After having the big picture it was then possible to establish the requirements for the project but take into account the purpose of the architecture as the study made in Chapter 2. Having the requirements defined it was further possible to analyse and build a system stack, which allowed to better decide on the components. As soon as the microcontroller was chosen and the modules necessary for the diagnosis purposes were defined, a brief overview of each was given. The configuration used as the metrics used for the diagnosis purposes can be referred in the annexes as in the next chapter.

When all the components were selected, it was then possible to proceed to the software development that would support them.

Chapter 4

Implementation

After having defined all the system specifications and components to be used according with requirements, it was then possible to proceed to the implementation to achieve fail operational architecture and fulfil the objectives of this dissertation.

This chapter aims to address the process of the implementation, explaining what was done both at hardware and software level.

The hardware was configured considering that the aim for the architecture is to achieve fail operational system as addressed in 2.4. Redundancy must be fulfilled in the hardware. The modules that were used were replicated and, since redundancy does not solve all the problems, other techniques must be developed as well.

Regarding software development various scenarios of fault injections were approached, arbitration processes in order to define which subsystem was "talking" to the outside bus by default and a state machine was designed in order to define the behaviour for each subsystem as it is going to be presented. Despite everything mentioned, a Kalman filter was also developed with the purpose of acting as tiebreaker when it came to a decision of which subsystem had the correct value when the values differed from each other

4.1 Hardware Configuration

This subsection will describe how the hardware chosen was used during the project of this dissertation. As can be seen in Figure 3-2, the hardware used is mostly constituted by the MCU and its modules. For the purpose of this project the sensing elements and the acquisition of data is not a concern, what is the real interest to explore is the functioning of each board and its modules. With that done and well documented it is then possible to iterate the process and have the acquisition process. Nevertheless, for the study processes a dummy acquisition was performed to validate the software as it is going to be seen later.

4.1.1 Angle Position Sensor

In order to satisfy most of the requirements described in chapter 3.2 the hardware have disposal depicted in Figure 4-1. This disposition of hardware wish to solve the requirements addressed in Table 3-1, in particular the points concerning the architecture (1), the subsystem (2), and the hardware (3). In addition, as stated throughout this dissertation the architecture aims to cover the calculation of an angle which is later used in steer-by-wire. The angle position sensor would have an hardware architecture identical to the one depicted in Figure 4-1.

As we can see in Figure 4-1, there are two entities clearly separated by the dash line. Those entities represent each redundant subsystem of the APS project. Each subsystem is composed by a MCU, and its modules necessary such as ADC, CAN-FD and others modules responsible for injection errors that were addressed in 3.4.2.1, 3.4.2.2, and in 3.4.2.3. Besides the MCU, each subsystem has its own sensing element and its own CAN transceiver. The reason why this was done this way, was to fulfil the requirements regarding the redundancy.

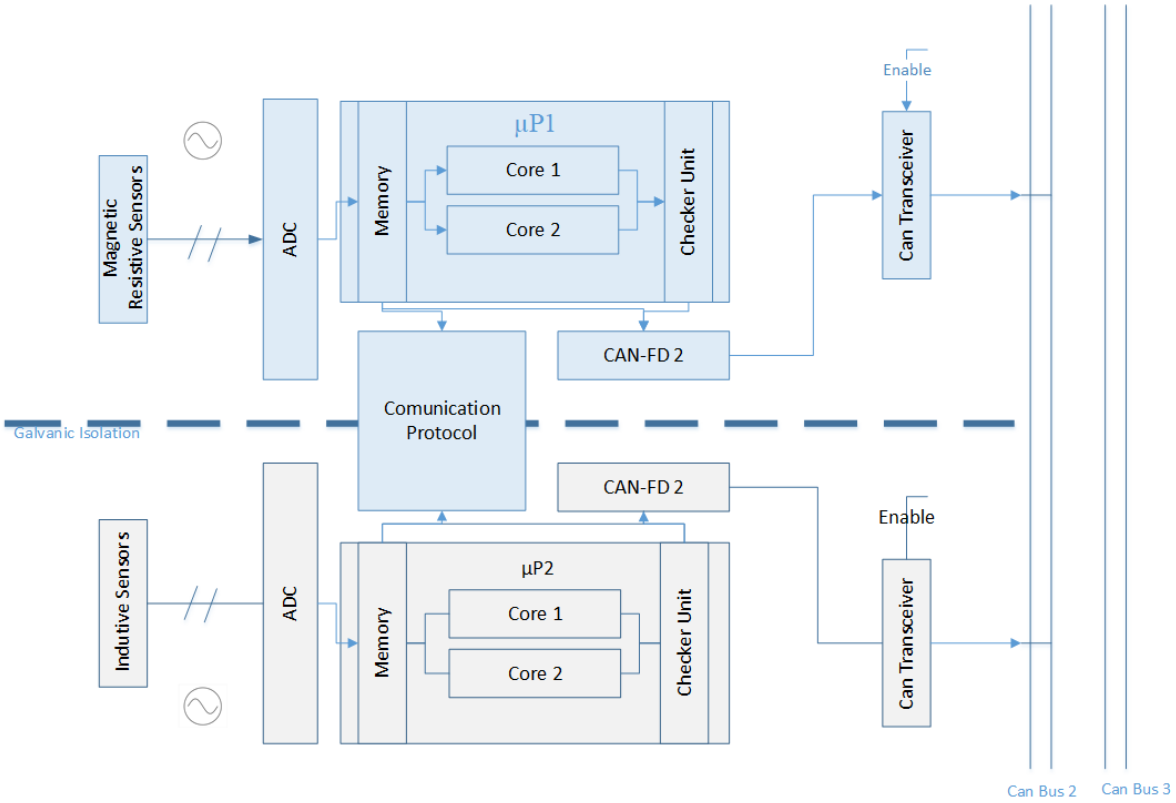

Figure 4-1: Hardware Disposal

4.1.2 Subsystem

The subsystem as addressed previously is defined by its own MCU, modules of the MCU, sensing elements and by the transceiver. Nevertheless, this alone does not satisfy the requirements established in Table 3-1 regarding the subsystem. The fault tolerance mechanism that each subsystem must implement is ensured by the lockstep already approached in 3.4.1. The lockstep allows to identify possible faults that could happen during the processing time at hardware level enabling the possibility to mitigate them before those faults turn into system failures. With that mechanism the requirement with the number of 2.3 is fulfilled.

4.1.3 Hardware Setup

Figure 4-2 is showing the real hardware setup used for the purpose of the project. As stated in the previous sections the architecture is composed of two subsystems which are illustrated in the figure by the two boards. Between them there are galvanic isolation granted and with that there is an implementation of a protocol communication that must work with galvanic isolation as well. The circuit

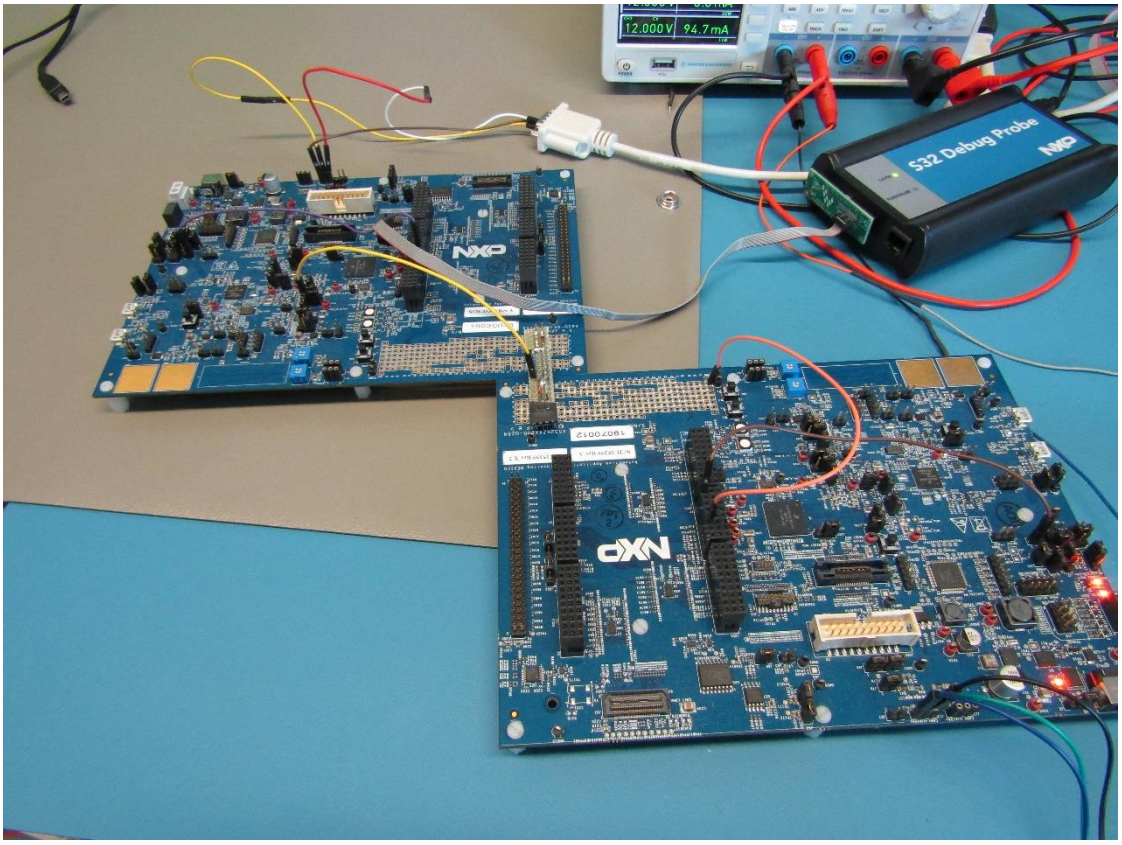

Figure 4-2: Hardware Setup

used for that purposed can be seen in Figure 4-2.

The connections made in each board depicted in the figure by the wires are connections for the purpose of arbitration and reset processes. Those processes will be addressed more in detail in the next chapter.

4.2 Software Implementation

The software implementation was made according to the existing hardware. The main goal for it was to achieve a fail operational behaviour enabling the system to still provide the output even when any undesired event occurs in it. To achieve that feature, some problems were in need to be taken care of. In a first stage a use case (APS) where the fail operational architecture could fit was designed, then a state machine was developed to define the behaviour for each redundant subsystem. After having all these software components developed it came out the biggest challenge that should be overtaken. Since two redundant subsystems are being used and each one is sharing data with the other to identify possible inconsistencies, there is no easy way to identify the correct and the wrong value in the process. If it were being used three subsystems, then we could implement a voter and mask any possible fault that could outcome from the calculous of any subsystem. However since there are only two subsystems, the approach to solve this problem was to develop a predictive Kalman filter. It estimates the next value and compare with the real one calculated by both subsystems.

This subsection aims to explain the process of all this software implementation. First the use case is going to be presented in which the software developed can be applied, then the state machine and finally the developed Kalman filter.

The software developed for the purpose of this project aim to be fitted in an AUTOSAR architecture. For that reason, the modules that were developed during this master thesis were divided according to the standard demands.

4.2.1 Software Components

As can be seen in Figure 4-3 the software is divided by blocks representing software components (SWC). To note that the diagram represents the software organized for the APS system. The dash line, once again, distinguishes the subsystem, just like presented in the hardware disposal.

Since the subsystems are in fact mirrored from one another, only the subsystem 1 will be explained. Explaining each software component of the subsystem 1, which is the one represented above the dashed line, the SWC-Acquisition is the part of the software that is responsible for the acquisition of data. That data is acquired from the sensing elements, which in this specific case of the subsystem 1 are the magnetic resistive sensor. The SWC-pre-processing is the component responsible for the noise cleaning associated with the data acquired. At this point of execution is expected to have an exchange of data between subsystems just to check if there is some type of inconsistence among the data acquired. That being done, it is now time to calculate the angle. That is the role of the SWC-Calculation. During this time, if for some reason an error occurred triggered by hardware or even by an inconsistency between data, flags are exchanged in order to have either way an output to the outer system. So, the process is: if an error occurs, flags are exchanged, then the error is treated which is the role for the SWC-Error Detection. In the error detection the main goal is to identify the fault caused and inform the SWC-Recover what is the best reaction to have according to the fault triggered. On the other hand, if at the calculous phase there is no fault triggered the software is ready to enter the final stage of the program.

In the last phase, the angle calculated is exchanged between subsystems with the purpose to have a match, if it does not, means that something along the process went wrong. But the challenge is to validate one of two different angles. That is why this system was in need to have the Kalman filter predicting the

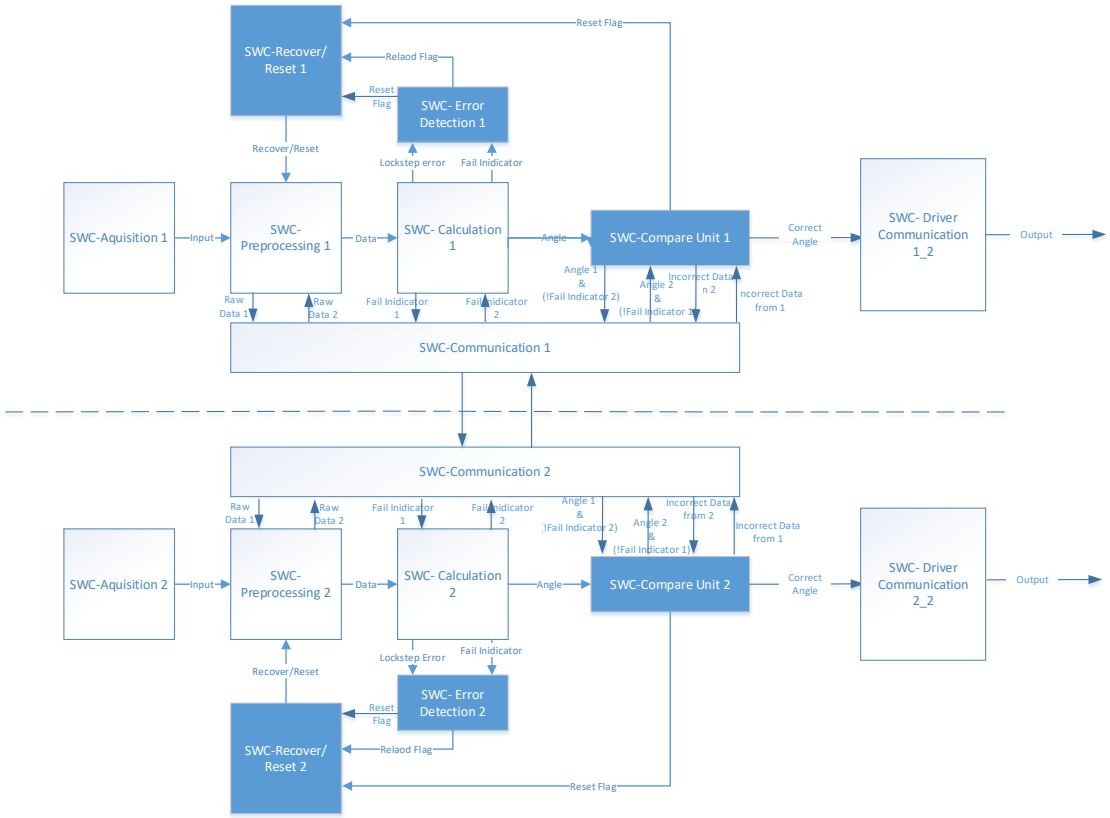

Figure 4-3: Software block diagram

next value of the angle. The one calculated that is closer to the value predicted is the one that is going to be chosen to transmit. All this process of comparing and deciding which is the correct angle is the aim of the SWC-Compare Unit. Finally, after the angle is chosen, the software is in condition to have the angle transmitted to the external bus. That is the responsibility of the SWC-Driver communication.
It is important to note that for the sake of this project only the blocks highlighted will be covered.

4.2.2 Sequence Diagrams

To have a clear picture on how the software components would interact between themselves a group of sequence diagrams that depicts the various scenarios was elaborated. First, a scenario that depicted the fault free environment was approached, then the fault that occurred at the lockstep mechanism was depicted and finally a scenario where the data is miscalculated.

4.2.2.1 Fault free scenario

In a fault free environment both subsystems are acquiring and calculating data, and the correct output is sent to the output driver, by default, through subsystem 1. During this process there are at least two exchanges of data between subsystems, in a first phase the raw data is exchanged and in a final phase the calculated angle is exchanged to make the identification of errors that could come from the calculation possible. In this particular scenario depicted in Figure 4-4 the process went as it should, no error was found.

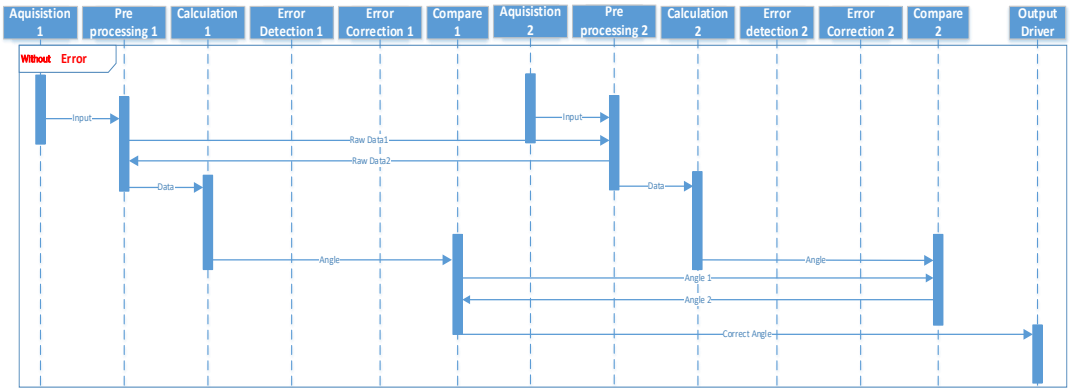

Figure 4-4: Sequence diagram, fault free environment

4.2.2.2 Fault at one subsystem

In this case it was emulated the scenario where a fault has occurred in subsystem 2. As addressed previously, throughout the well-functioning of the application it is intended to have two exchanges of data between the subsystems, the first one consisting in the raw data acquired by each subsystem and the second data traded is the angle already calculated. The scenario that Figure 4-5 has depicted, has indeed two types of data exchanged between subsystems but the last one is not the angle calculated.

In an initial phase both subsystems are acquiring data and trading the raw data but then a fault has been triggered at subsystem 2. The process that should outcome from that is to send a message through the internal bus to subsystem 1 informing that subsystem 2 is under a fault and that the fault free subsystem must ensure the communication to the output bus.

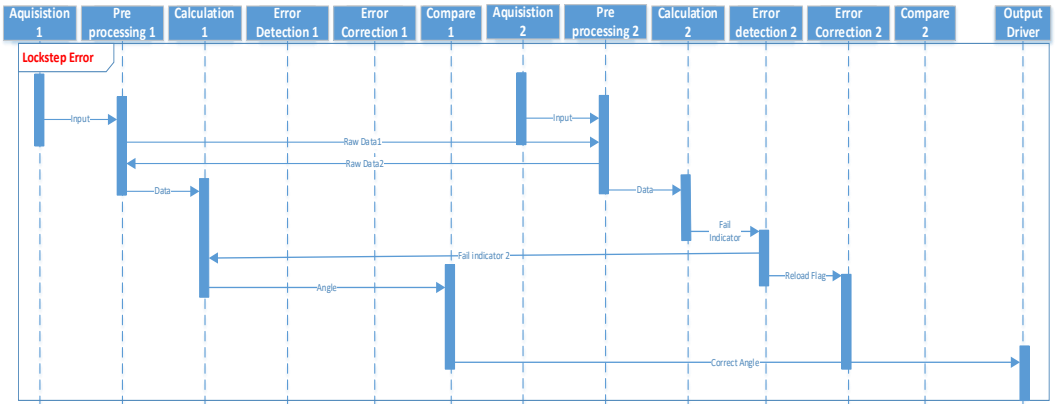

Figure 4-5: Sequence diagram, fault at one subsystem

4.2.2.3 Miscalculated Data

In this last scenario the case where an inconsistency at the calculous algorithm has been found will be addressed. Here, the error is only found in the last phase of the process. Once again, on the initial phases both subsystems are working properly, acquiring, exchanging and calculating data but in the last phase when the angles are compared and if by some reason the angles differ too much among themselves it means that the system is facing a fault. This decision is made at the SWC compare unit with the help of Kalman filter as it is going to be seen later.

Figure 4-6 illustrates the scenario where the miscalculated angle is coming from the subsystem 1. Said that, and when identified by SWC-compare unit of subsystem 2 an extra message is sent to subsystem 1 through the internal bus to inform that the angle calculated is wrong. The next step is to ensure the communication through subsystem 2 which has the correct angle calculated.

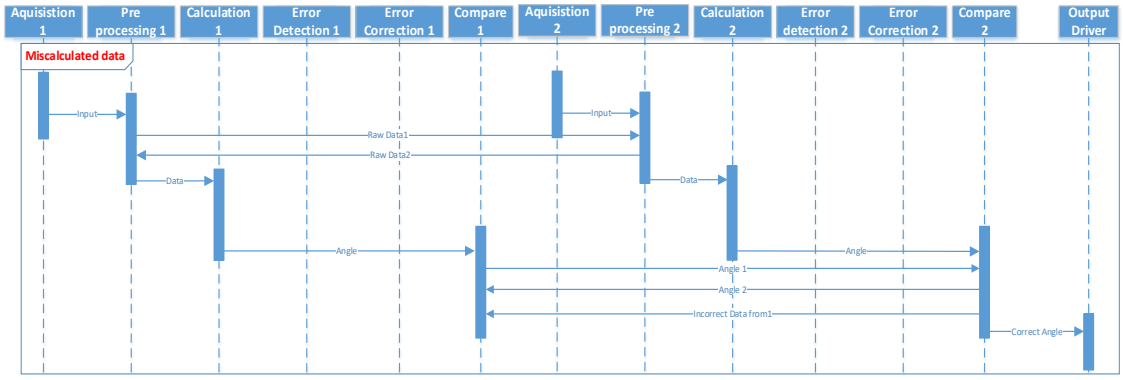

Figure 4-6: Sequence diagram, miscalculated data

4.2.3 Software Workflow

In this phase some flowcharts are presented to better understand the implementation achieved. There are some specific topics approached that were crucial to the implementation and validation of the system in cause. Since the major point of this dissertation is to study and present a solution when it comes to the error detection and error recovery, some errors were induced at the system to better validate it.

4.2.3.1 Main Cycle

Starting from the main cycle where the application is running cyclically in Figure 4-7 it is presented in what consists this main cycle. Figure 4-7 is quite explanatory, but in a brief description the main cycle starts by doing an initialization of all the modules and then it checks if there is a button pressed. That button is used for validating the system with errors. If the button is pressed then the function responsible for injecting the errors is called. Whether the button is pressed or not the application still needs to be in run mode, that is why the function Run is called cyclically.

4.2.3.2 Initialization Process

The initialization process is called once during the well-functioning of this software as can be seen in the main cycle. Basically, this process is responsible for the initialization of all the modules needed for the application. Figure 4-8 represents the flow of execution when this function is called. To note that most of the functions called within this process will not be addressed since they consist in a configuration. On the other hand, the sub-processes such as arbitration and check_ status are going to be explained in detail later.

4.2.3.3 Arbitration Process

The arbitration process arises from the necessity to have a homogeneous software functioning,

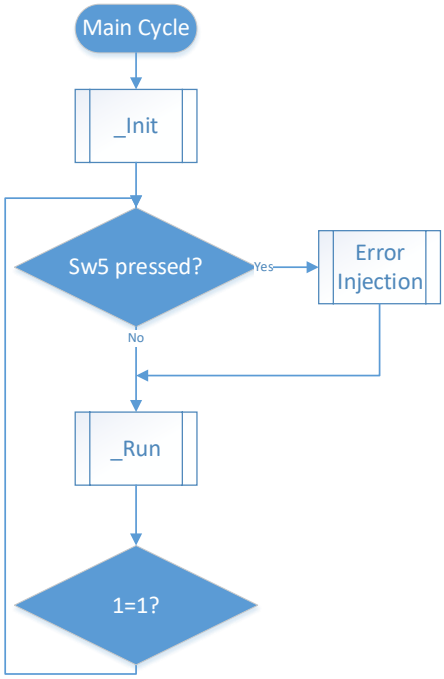

Figure 4-7: Main Cycle flowchart

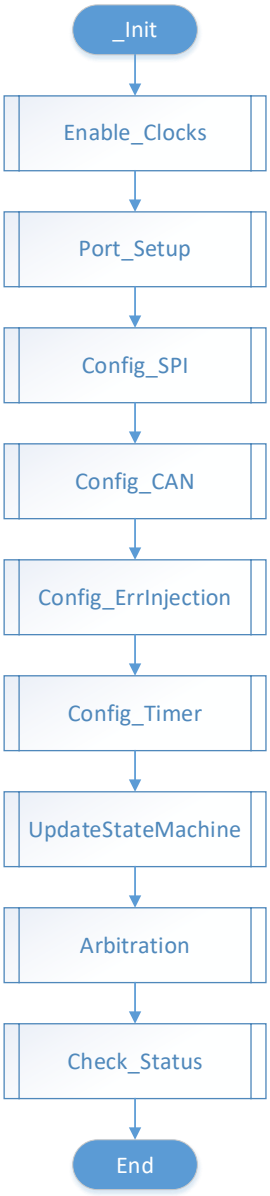

Figure 4-8: Initialization process flowchart

meaning that the software flashed into the two subsystems must be equal in both sides. Nevertheless,

when communicating internally the IDs for CAN message must be different for both subsystems. Therefore, the software, despite being homogeneous, has a condition defining the IDs for messages exchanged among subsystems.

That condition is no more than an arbitration process. This process consists of reading a value of a pin previously connected to 5V or GND depending on which subsystem. If that value is read as one (5V) then it defines the CAN_NODE_A. On the other hand, if that value is read as zero (GND) it does not define the CAN_NODE_A, as can be seen in Figure 4-9.

4.2.3.4 Check Status Process

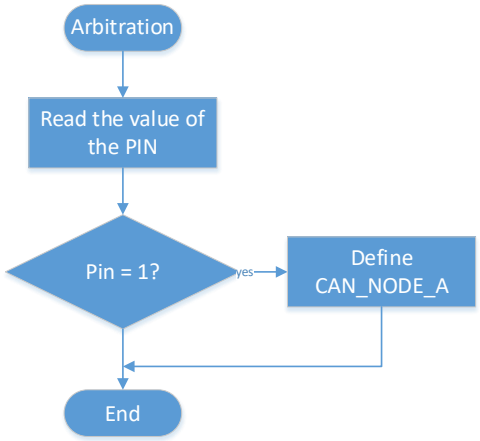

Figure 4-9: Arbitration flowchart

The check status is a process made at the end of the initialization. This is mainly to understand the status of the other subsystem and synchronize both subsystems. The flow chart can be referred in Figure 4-10. There are two different paths for the flow of execution, depending on which subsystem this process is running. If the CAN_NODE_A is defined then it means that this system will, by default, communicating to the outer system with a fix period of 10ms. Then a timer is enabled to define that period. In this case the timer used was the Programmable Interrupt Timer (PIT). After that the NODE_A sends a message internally to the other subsystem to understand the system state. If it is in fail-operational, it means that NODE_A is waking up from an error recovery (as it is going to be shown later), so the next step is to send a recovery command in order to inform the other subsystem that NODE_A is already fully functional. Being or not the other subsystem in fault state, this NODE_A subsystem is the one communicating by default, so a message to the outer system is ensured in either scenario. That is, if NODE_A is in normal state the message to outer bus is sent by it, if is in fault state is the NODE_B who do it.

The other path is for the subsystem in which the CAN_NODE_A is not defined. In this path where NODE_B is defined, the first step is quite similar to the other path, in which the NODE_A is defined. The main difference is that in this phase this subsystem does not need to ensure the communication with the external bus being NODE_A running properly.

4.2.3.5 Error Injection Process

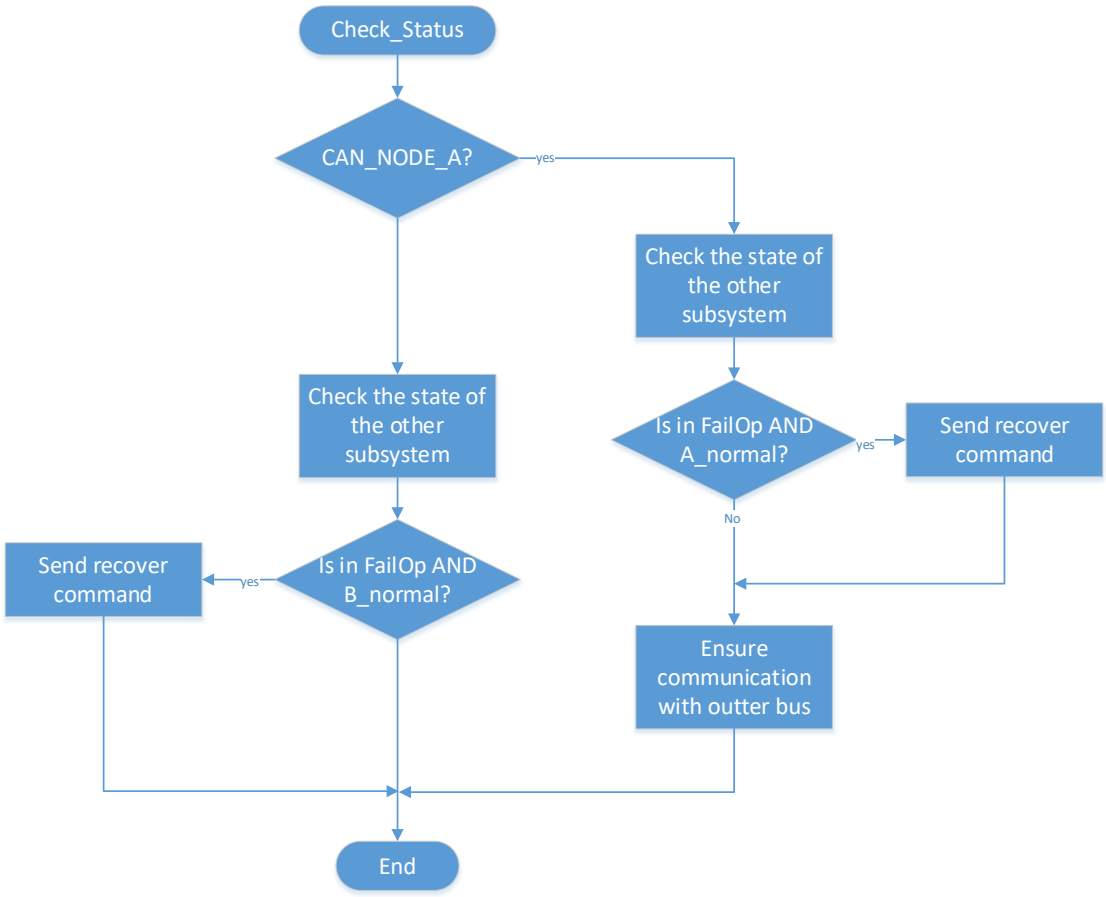

Figure 4-10: Check status flowchart

The scenarios developed for the error injection as can be seen in the main cycle are asynchronous. Of course, the aim for this kind of implementation was done only to understand how the system would react in case of an error. In real application there is no need to trigger an error manually, the hope is that when an error has occurred the system knows what to do. With this implementation the purpose was to define what to do in such case. As addressed in 3.4.2.2., there are 17 channels available for error injection. In this phase is important to stress out the scenarios regarding memory and lockstep, error 1 and error 2 respectively represented in Figure 4-11.

Error 1 is made by taking into account the address of memory that is going to be read in the next instruction and then with the help of EIM some bits are inverted in order to have inconsistencies between

the data read and the CRC calculated. This type of error is reported by an ISR caused by the ECC. And the ISR is mapped into the ERM.

Error 2 is caused into the lockstep. The EIM intersects the bus shared by both cores and then invert some bits with the purpose of having an inconsistency in the outputs produced by both cores. This event, depending on how the FCCU module is configured, can trigger an ISR or a functional reset. For the purpose of this dissertation, the configuration of the FCCU was done to trigger an ISR if such fault occurs and then within the ISR the reset is activated by the developer. This implementation is much more attractive since it allows the trade of information with the other subsystem, having more control of the system as a whole.

The implementation regarding the memory injection is made according to Figure 4-12.

To better understand how the modules were configured and in order to inject and to listen the errors injected the annexes can be referred as well as chapters 3.4.2.1, 3.4.2.2 and, 3.4.2.3.

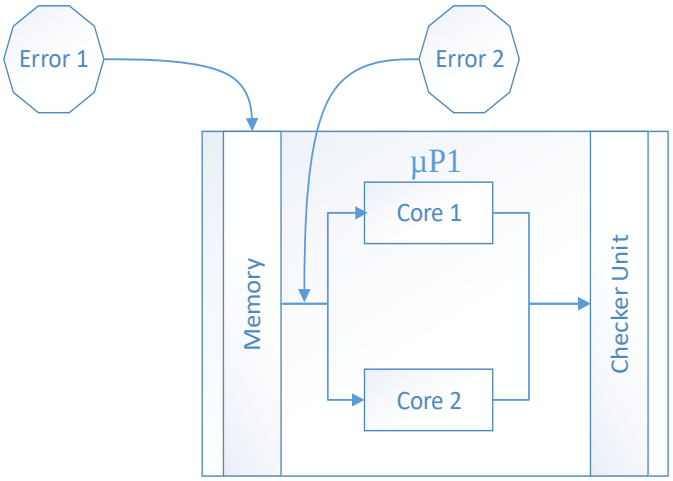

Figure 4-11: Error Injection Scenarios

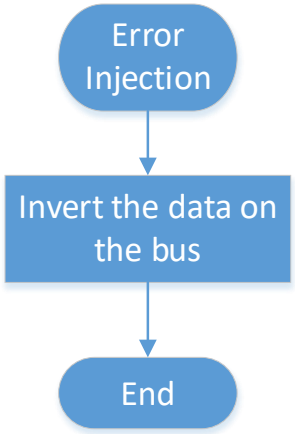

Figure 4-12: Error Injection flowchart

4.2.3.6 Run Process

The run process is made according to the necessities of the application. Since there is a great deal of messages being traded at the internal bus, the first action to take in this process is to check if, in fact, there are any messages to be read. Therefore, when the software checks that there exist messages on the bus, actions must be taken. If it has received an 'S', which is basically a request, it must reply with the status of the system. If the message is different from the mentioned 'S', then it means that the one receiving a message must update the system state machine. And these are the scenarios in which there are internal messages on the bus.

Since software is redundant there is a extreme necessity to understand in which system it is running. Thus, it verifies if CAN_NODE_A is defined. If it is, the software checks whether the subsystem is in a

normal state or not and if the system is normal or in fail operational state. It also verifies if there is any message to be sent over the external bus. The reason why the system sends messages in fail operational sate is due to the fact that the system state machine is only updated to the fail operational state when it receives a message indicating that the other subsystem is facing a fault. Since the CAN_NODE_A assures by default the external communication, it communicates with the outer system both in normal state and in fail operational state of the system. On the other hand, if the CAN_NODE_A is not defined, then the software checks only if the system is in fail operational state and whether the B subsystem is in normal state as well as the need to send any messages over the external bus. As it was said previously in the normal state of the system, the subsystem ensuring the communication with the "external" world is the one where the CAN_NODE_A is defined. If, for some reason in B subsystem the state machine is in fail operational state is, undoubtedly, due to the fact that the A subsystem is facing a fault, so it is the time for B to ensure the communication.

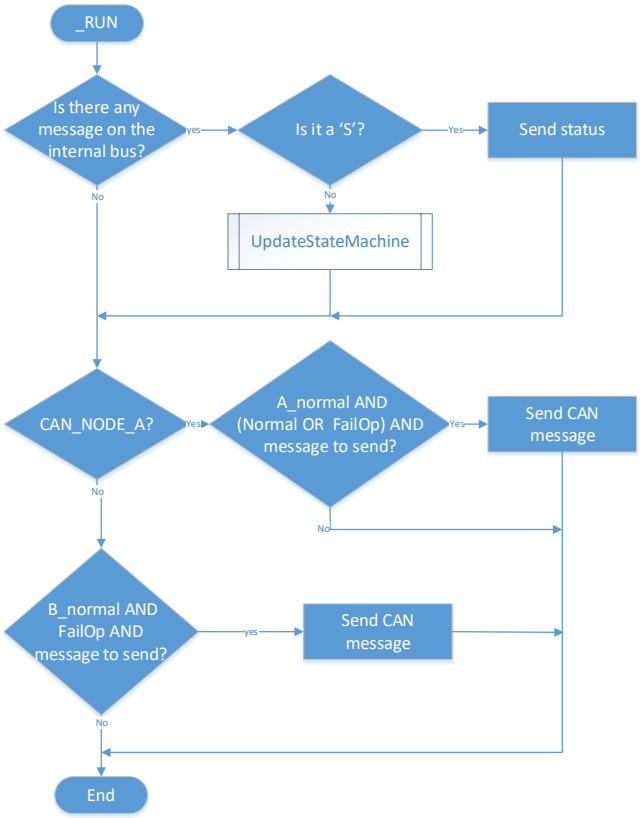

Figure 4-13: Run process

4.2.4 State Machine

The state machine developed for the purpose of this project aims to track the system state. The state machine in question is composed of three different states. The normal state fail operational state and the failure state. Each state represents an action to take during the flow of the program. For instance, the normal state represents the good functioning of the system where no fault has occurred nor inconsistencies at the acquisition or at calculous level. The fail operational state means that a fault might have occurred at the hardware and possibly leading to some inconsistencies while the system is still operational due to the use of redundancy. So the process is: a subsystem that is facing a fault transmits to the other one suggestive data that indicates that the previous one is under a fault. The receiving one updates the state machine while the faulty one is trying to recover from that fault by a system reset or even within an ISR defined. The other very unlikely and undesirable scenario is the failure scenario. In this case, the system as a whole is facing a fault without even the possibility to exchange data between subsystems. In other words, both subsystems at this state are facing a fault. The proper reaction to have in these types of situations is undoubtedly the power cycles or even the system reset.

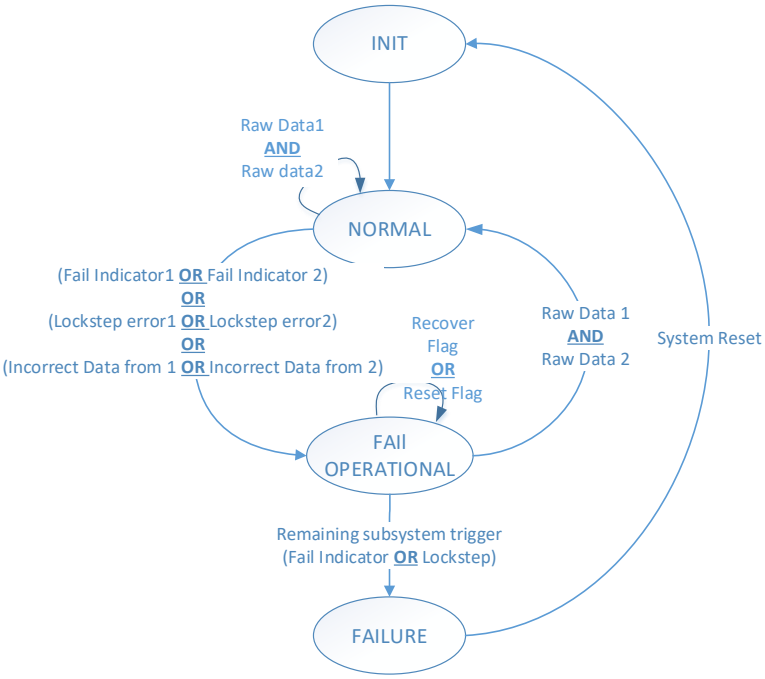

Figure 4-14: System state machine

Besides the state machine already mentioned and depicted in Figure 4-14, there is another that emulates the behaviour of each subsystem. That one is represented in Figure 4-15. In a brief description of the state machine it is necessary just to highlight that the arrows that are coming in at the normal and at the failure state represent the internal communication where both subsystems are trading data among themselves. Once again, the normal state is where the application is running in the proper way, acquiring data, sending data and calculating the angles based on data acquired. If for some reason the redundant subsystem detects a fault then it sends that information to the other subsystem and transit to the failure state where the fault is trying to be mitigated or recovered. If that occurs the subsystem is in conditions to go back to the normal state, if not, it means that the fault is unrecoverable therefore the best reaction to have is resetting the subsystem.

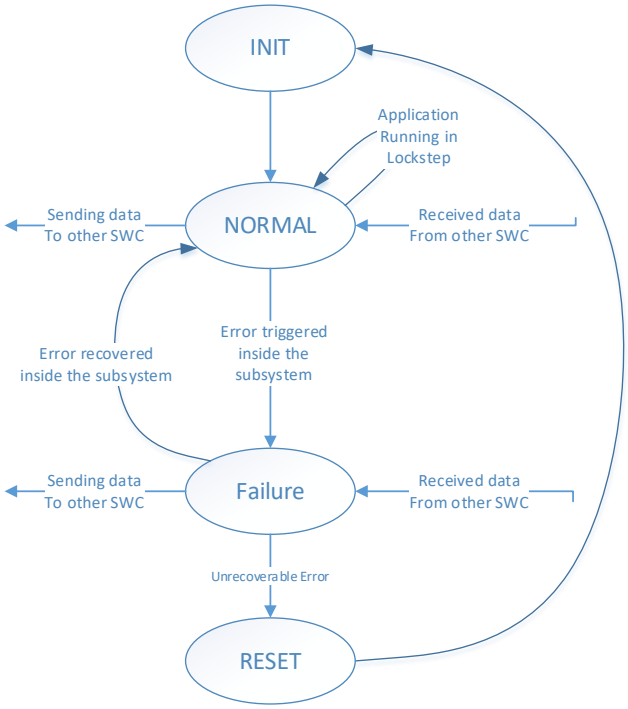

Figure 4-15: Subsystem state machine

4.2.5 Kalman Filter

Kalman Filter is one of the most important and common estimation algorithms. The Kalman Filter produces estimates of hidden variables based on inaccurate and uncertain measurements. The Kalman Filter also provides a prediction of the future system state, based on the past estimations [39]. With that stated, the Kalman filter developed was used on the SWC-Compare-Unit. This filter aims to calculate the next value that is going to be considered in order to identify possible inconsistencies between calculated angles. If the reader imagines a scenario where there are two entities that supposedly produce the same

output, in case of one differing from the other, there is no easy way to identify the erroneous one. The Kalman filter aims to solve that problem. The goal is to use this Kalman filter in each subsystem meaning that each subsystem should be tuning their own filter parameters according to the values in their calculation block. At every calculous performed the filter is being feed with that result. If the values calculated for each subsystem differ from each other, then the decision of which subsystem should transmit the angle to the outer bus is made according to the estimations of the Kalman filter. But it is important to mention that this decision is only valid if the estimated values from the filter are accurate enough. Taking the scenario where the values calculated differ from each other the Kalman filter aims to act as a tiebreaker and the value of the angle closer to the valid estimation is sent through CAN. If the Kalman filter is not tuned to a point where is possible to trust its predictions and if the angle values at that specific point in time differ a more rudimentary way of making a decision is used. A threshold value for the angle is used. That is, from one sample to another the angle should not overtake the value defined and if it does it means the value is the wrong one.

The filter was developed in the third order, in other words, the angle value was predicted based on the variation of speed and acceleration of it. The problem associated with this filter is that most of its estimations are created based on their previous value meaning that the initial estimations are not as

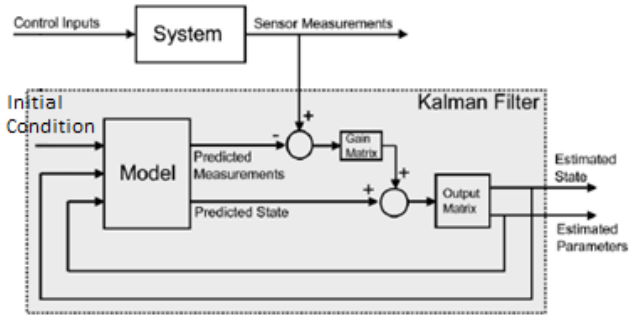

Figure 4-16: Block diagram of Kalman filter

accurate as their older ones. The parameters of the Kalman filter are being tuned at real time. Despite this step back, the implementation can work around the problem since the data acquired are made at a so high speed that for each 10ms of a message being sent to the outer bus there are at least five samples acquired which gives the Kalman filter the inputs necessary for it to tune its values for a valid estimation. Figure 4-16 gives a good overview of the Kalman filter used. The Kalman filter is shown in the shaded part of the diagram and the system where the filter is collecting its values for a proper estimation is also represented but in the outer part of the diagram.

4.3 Conclusion

This chapter gave an overview of the implementation achieved during this project. The implementation was made according to the chapter of system specification since most of the requirements were respected and were taken into consideration during the implementation. The errors induced were performed only to validate the architecture developed as well as to validate the fault tolerant mechanism which is lockstep. Those scenarios induced also allow the possibility to validate the state machines designed to track the states of both: system and subsystem. The Kalman filter is developed to validate the calculation algorithm.

Chapter 5

Tests and Results

This chapter contains the tests made to the architecture. As it was intended to develop a fail-operational architecture the tests were performed to validate it. Tests were divided in a way to validate each specification crucial for the architecture. In other words, a subset of functionalities which were needed for the system were tested distinctly. With that being done, then make sense to evaluate the system as a whole and check if the requirements were respected. It can be said beforehand that in order to have a fully functional fail-operational architecture the costs are much higher due to the amount of redundant resources needed. But when a system is designed there should be always a trade-off between the costs and the functionality. Since the system is rated as safety critical, the functionality is the priority.

5.1 Tests

This chapter will describe each test performed and later evaluate its results. To each subchapter created within this chapter corresponds to a test performed. Table 5-1 shows the tests that were in need to be performed as well as the results that were expected to have from each one.

Test	Expected Result	Real Result
Send flags through the internal communication bus, from both subsystems	Have an ISR for each internal message received	
Have a CAN-FD message in the external bus within a fix period of 10ms	Validate the message and the timing of the message in the CANoe software	

Make use of module EIM in order to induce faults at the subsystem	An ISR triggered for each error induced and flags exchanged between subsystems	
For each fault detected at the system, the state machine must transit from its state	Signalize the event of transition, through a CAN-FD message and through a LED interface	
Use Kalman Filter to predict the value of the angle calculated	Measure the error associated with the prediction having error values lower than 3%	

Table 5-1: Test Table

5.1.1 External Communication test

The external communication was made with the help of CANoe tool. CANoe is the comprehensive software tool for development, test and analysis of individual ECUs and entire ECU networks. It supports network designers, development and test engineers throughout the entire development process – from planning to system-level test. [40] For the scope of this project the software tool was used to validate the messages sent to the external bus from both subsystems. For that there was a necessity to emulate an external bus which was later interpreted by the CANoe software. The tool used for that matter, was a VN1610 which consist of a CAN interface with the possibility to emulate a CAN/CAN-FD bus.

With all the tools needed to test this communication, two test scenarios were elaborated. The first one was made in order to verify the time window in which the messages were being received at the CANoe. The other test is to check if the external communication is ensured even when the whole system is in fail degraded.

5.1.1.1 Time Window

In this scenario what must be taken care of is that the messages must appear in the external bus within a 10ms time window. Therefore, with the help of CANoe which has a possibility to see Δt of the messages that are being received was possible to check if the time constraints were being respected. Figure 5-1 shows the messages sent by one of subsystems and in the time column we can see the amount of time that pass since the last message received. As can be concluded the time slots were being respected. It was taken two print screens two see that there is an error associated, which is roundly to 1ns.

Time	Chn	ID	Name	Event Type	Dir	DLC	Data
0.010001	CAN 2	600		CAN FD Frame	Rx	10	03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03

Time	Chn	ID	Name	Event Type	Dir	DLC	Data
0.009999	CAN 2	600		CAN FD Frame	Rx	10	03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03

Figure 5-1: External communication test, time windows

5.1.1.2 Fail degraded

With this particular test what was expected to verify is that during a fault induced and correspondent recovery, the system ensure the communication with the external bus. In other words, the system still communicates even in fail-degraded mode. Fail-degraded due to the fact that one of the redundant subsystems is facing a fault and for that reason the system has lost some of its features. This means that the system state machine is on fail-operational state.

Once again, using the CANoe software was possible to see the messages sent from different channels which consist of messages being sent from both subsystems. This test has also a particularity to have messages received at different CAN-FD IDs. This is to inform the entity that is external to the system that the system is facing a fault. Despite the system still being operational, as it should, it will inform the external "world" that is operating under fail degraded mode. As it can be seen the angle message which is the message sent with ID of 600 was sent both by channel 1 and 2. This message ID represent the angle was being sent only in channel 2 but with a fault induced at the system the angle still need to be transmitted so that is why the channel 1 which is the channel of the other subsystem. Between changes of channels, an error message is sent to signalize the fail-degraded mode of the system since during this process of fault and fault recovery there is only one subsystem operational.

Time	Chn	ID	Name	Event Type	Dir	DLC	Data
0.009999	CAN 2	600		CAN FD Frame	Rx	10	03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
0.000000	CAN 1	601	STM_Error_messages	CAN FD Frame	Rx	10	0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
0.009999	CAN 1	600	Angle_Message	CAN FD Frame	Rx	10	03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
0.000213	CAN 2	601	STM_Error_messages	CAN FD Frame	Rx	10	05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05

Figure 5-2: External communication, fail degraded test

5.1.2 Error Injection Test

This test was performed with the help of the existing modules available in the microprocessor to induce faults at the different peripherals of the microprocessor. To be more detailed the modules used are the ones already approached in 3.4.2.1; 3.4.2.2; 3.4.2.3. These modules turn out to be very useful not only to induce faults at subsystems but also to study what the possible reactions are to be taken

when a fault occurs. Furthermore, these modules are also used for detecting the faults through ISRs mapped into the memory region.

At this point it is important to highlight that these tests are not as visual as the others already addressed, so this test will mainly be explained over text since there is no image that could depict a fault induced at the microcontroller. In brief reasons, two different tests will be explained: the ones induced at the lockstep mechanism and the ones induced at the memory region.

5.1.2.1 Lockstep

This test was performed with the help of modules such as EIM and FCCU as addressed in 4.2.3.5. Nevertheless, at this point the two possible scenarios that FCCU allows for it to have were exploited, the ISR reaction and the functional reset.

With a functional reset, as mentioned in 4.2.3.5, there is a lack of control of the application. That is, when the error occurs the functional reset is performed without any possibility to track the problem or to understand what happened. For that reason, this implementation is less attractive for the purpose of this project.

In the ISR scenario, as soon as the error is injected, the ISR is triggered. Inside it, there are plenty of paths to be taken. The error techniques that this lockstep architecture uses so much could be applied, but according to the documentation of the microcontroller when this type of error occurs the recommended reaction to have is a functional reset. Nevertheless, the author explores other solutions with this reaction. A scenario taken was trying to clean all the flags that were causing the ISR but in that scenario the program counter (PC) was still tied in the ISR. So, the solution was to trigger an ISR and within it, trigger a reset. Between processes the subsystem must inform the other one that a fault has occurred in order to have more control of the application.

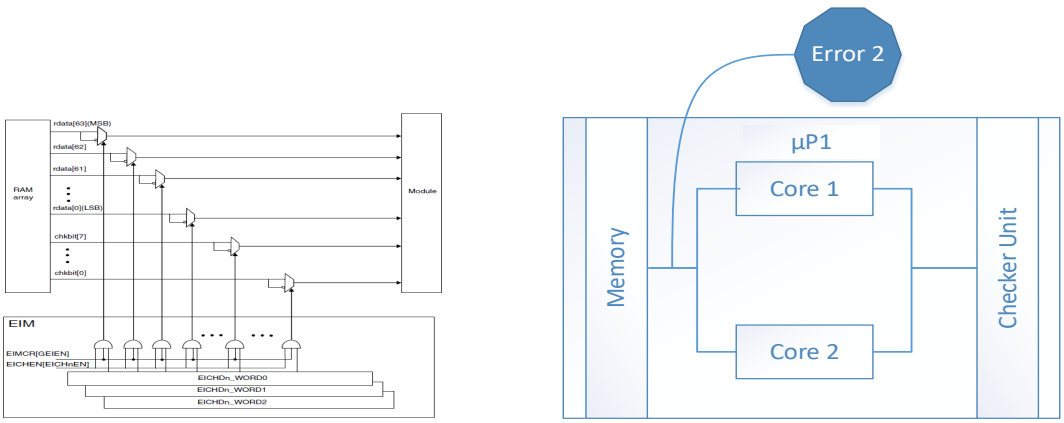

Figure 5-3: Lockstep Error Injection

5.1.2.2 Memory

The error related with memory was injected through EIM module once again, and the module that was responsible to "listen" to the errors was the ERM with the help of ECC mechanisms. In a first phase it was induced at the memory errors composed by more than one bit flip, which is not compliant with the ECC mechanism. According to the documentation the ECC can correct up to 1 bit flip detected at the memory buses but more than that trigger an undefined behaviour, possibly causing hard faults. For this type of errors a channel at the ERM was also configured to trigger an ISR if such error has occurred. Within the ISR the author thought of reconstructing the signal by reverse engineering the value of CRC. Nevertheless, there is nothing published indicating the CRC algorithm, thus making it almost impossible to be done. Within ISR the possibility to clean the flags asserted at the error injection time was also tested in order to verify if the PC was able to move forward. In this case two scenarios occur depending on how many bits were flipped with the error injection. If it was just one, cleaning the flags that were causing the error injection were enough to keep the program flowing. If it was more than one bit flipped, as mentioned above, it causes undefined behaviour which the developer cannot control, and cleaning the flags does not make it better.

It is important to highlight that memory errors are critical since all the application depends on it, being by code memory or data memory. Once corrupted, if it is not possible to restore its content then it is impossible to trust it again. For that reason the best reaction to have when these events occur is to trigger a reset such as the reaction to have when a lockstep error occurs.

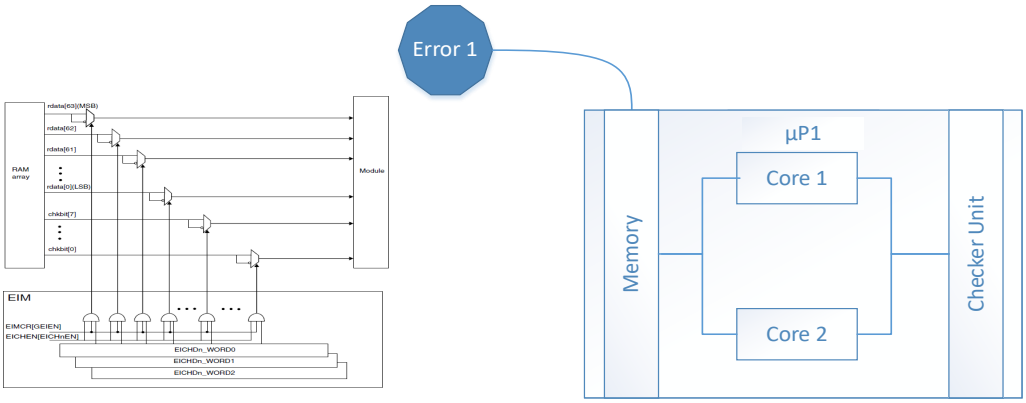

Figure 5-4: Memory Error Injection

5.1.3 State Machine Test

In this test it is intended to test the transitions of the state machine according to the scenario in which the system is facing. As addressed in 4.2.4, two state machines were designed, where one emulates the

behaviour of each subsystem and the other is used to discover in which state the entire system is. With that in mind, the test will take into consideration the state machine of the system and it is going to be explained with the help of the state machines designed for describing the behaviour of each subsystem.

5.1.3.1 Normal state

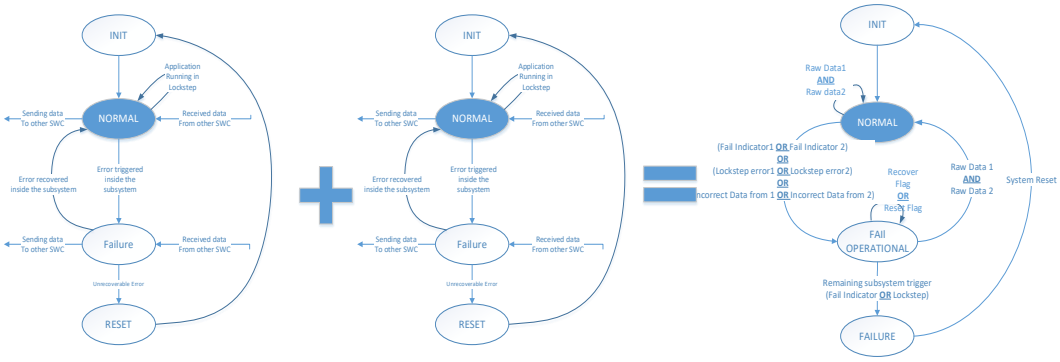

Figure 5-5: State machine in normal state

The normal state of the state machine is where the application is running as it should. Each subsystem is able to exchange data between them, there is no error injected, the outputs are being sent by the subsystem in which CAN_NODE_A is defined, and both subsystems are at normal state as can be seen in Figure 5-5.

On the left side of the equal signal the state machines are defining the behaviour of each subsystem, both at normal state. On the right side of the signal there is the result, the state machine that defines the behaviour for the entire system. For this specific scenario, the only thing that was done was to run the application. As it can be seen, the transition from INIT to Normal is unconditional.

5.1.3.2 Fail Operational State

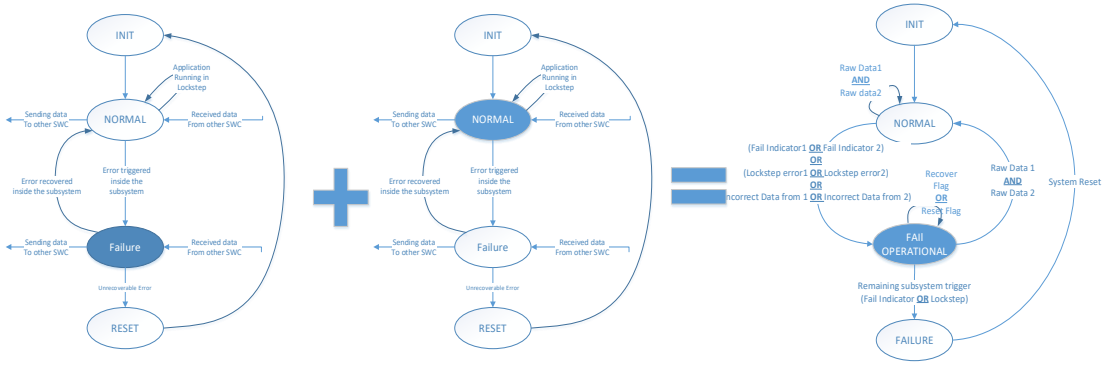

Figure 5-6: State machine in fail operational state

For this test, a fault was induced at one subsystem in order to trigger in it a transition from the normal to the failure state. With that scenario the entire system transits from normal to fail operational state as it can be seen in Figure 5-6.

The scenario has two possible outcomes for the system. The first one is: the faulty subsystem is able to recover from the fault, by a recover or a reset, and then the system goes to normal state again. The other scenario, extremely undesirable, is where both subsystems are under faults. Nevertheless, throughout this chapter, after inducing the fault, the subsystem was able to recover from it making the system to transit to normal state again.

5.1.3.3 Failure State

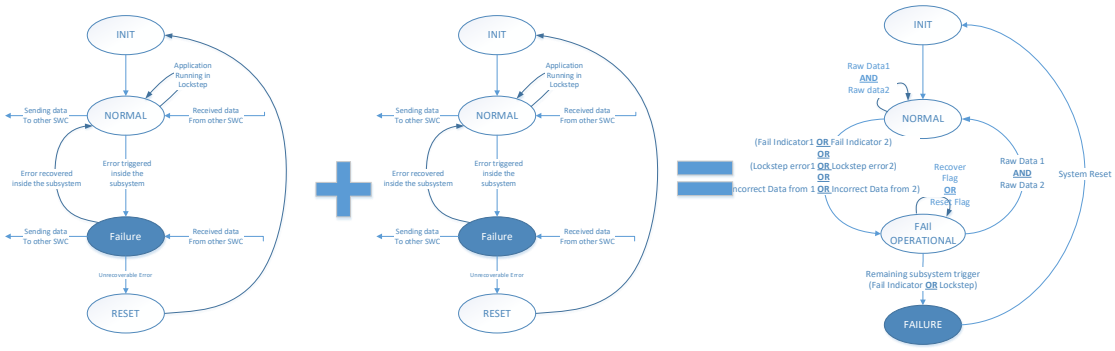

Figure 5-7: State machine in failure state

This is the most critical scenario of this system. The scenario where a fault at one subsystem occurs and followed by it there is another fault at the other subsystem. In this particular scenario there is not much to do other than to perform a system reset.

The subsystems in this particular scenario ensure the reset at each one, but before that the outside "world" will be informed that a power on reset is necessary due to errors at the same time in the subsystems. This is done by a repeated CAN-FD message informing the error.

5.1.4 Kalman Filter Test

As already stated in this document the Kalman filter was developed with the purpose of being a tiebreaker. As two subsystems are running in parallel with the same purpose and with the goal to compare its outputs if they differ it is extremely difficult to identify which is the best value to transmit to the external CAN bus being compliant with a fail operational architecture. For that reason Kalman filter is believed to be a suitable solution.

In these tests what was intended to have as a result was the best tuned model of the filter. For that, the filter was studied with a set of datasets and the parameters were tuned to have a better response from the Kalman filter. The datasets that were used are angles already calculated as it should be in the real life application. Nevertheless, the data used in this purpose are not the best possible ones since it is a little bit noisy. Throughout these tests data acquired from magnetic resistive sensors sensing

elements were used and then the various scenarios that emulate the real life application were taken into account. The values can differ by rotating to right and left directions.

5.1.4.1 Rotating wheel in one direction

In both tests the Kalman filter has two purposes, the first is to clean the signal from all its noisy acquisitions since the data with which the angle is calculated is not treated as it should and the second one is to predict the next value that is going to be calculated. For each iteration the prediction that came from the filter is compared to the real value and it is evaluated how accurate the prediction is. Taking that scenario, the filter is tuning its parameters in order to fit the data the best as it could.

This test takes the scenario where the wheel is rotating in one direction only. The angle can vary its values between -2800deg and 2800deg and the discontinuity point that the signal has is due to the fact

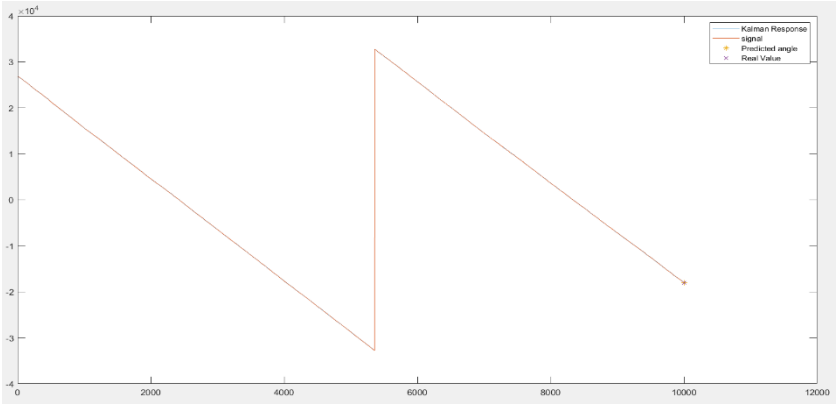

Figure 5-8: Kalman filter test, rotating in one direction

that the angle calculated has reached its negative limit and then it should continue from a positive value and decrease once again. Figure 5-8 also depicts the cleaning and prediction of the Kalman filter. On the right top corner of the image is the legend of each signal represented in the image. The real signal appears to be the only signal represented in the image but by taking a closer look it is going to be seen that this image has much more to say.

The signal and the Kalman response are overlapped as it was expected since the filter response aims to follow the real signal. There might be small deviations from the real signal and the Kalman response but that is due to the cleaning that the filter is doing. In the last points of the signal the prediction value is represented and the real value where the error associated with this is going to be evaluated.

In Figure 5-9 the last points of the signal are depicted but with zoom in in order to take a closer look. Here the existent deviations of the filter and the real signal are clearer. As it can be seen the filter

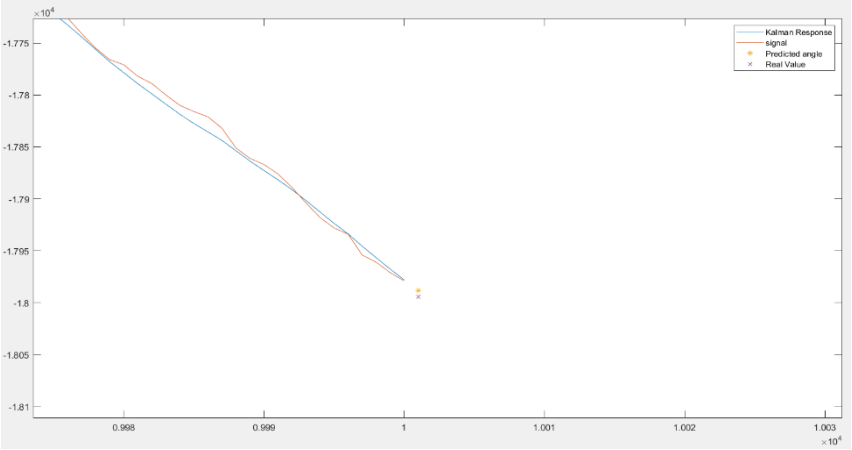

Figure 5-9: Kalman filter test, rotating in one direction zoom in

response takes the signal in a much more linear form when compared to the real signal. In the last points, represented by an '*' and an 'x' there is the prediction and the next value calculated respectively. As it can be seen the points are very close from one to another and in a signal where the acquisition is made in a more careful way the points would be even closer.

Quantifying the error between these two signals the average of the error associated with the prediction is about 0.08% (Error= [RealValue-PredictedValue]/RealValue*100).

5.1.4.2 Rotating wheel in both directions

In this test what was intended to verify was if the Kalman filter was able to follow the signal when suddenly the wheel rotated in another direction. In order to perform this test and due to the lack of datasets where the wheel rotating in the two senses is depicted, the data is simulated. In other words, the data present and represented in Figure 5-10 was manipulated in order to have emulated that scenario depicted. Said that, it was expected that the response of the filter was totally overlapped by the signal but, as there is a gross change of the signal due to the change of direction in the wheel that is why in Figure 5-10 the blue line, which represents the Kalman filter response, is seen more clearly when compared to the previous test. In addition, in this test less samples were used when compared to the previous ones which also have impact on how the signals are depicted.

Either way the results coming from this test were also quite satisfactory. The filter was not only able to follow the signal as well as it also had more acceptable error values very. The average of the error associated in this test was about 0.07%.

Despite the great challenge that it was to develop this Kalman filter it is believed the filter has extremely satisfactory results and this implementation can be used, without a doubt, in future implementations to act as a prediction algorithm.

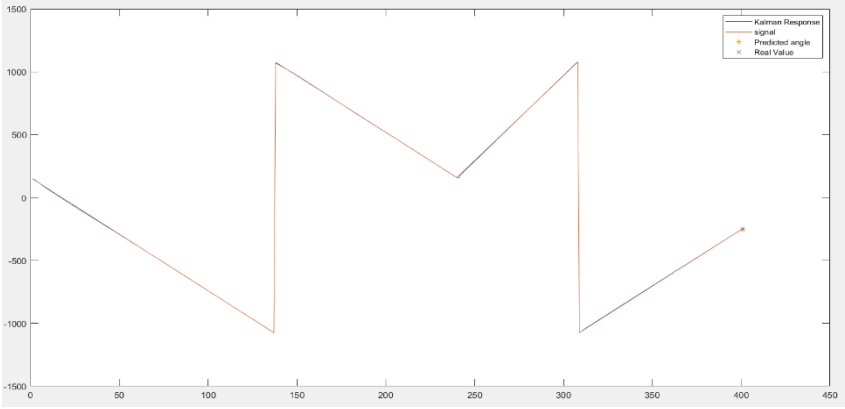

Figure 5-10: Kalman filter test, rotating in both directions

5.2 Results

This subchapter aims to evaluate the real result of each test that was intended to perform in order to validate the project developed.

Test	Expected Result	Real Result
Send flags through the internal communication bus, from both subsystems	Have an ISR for each internal message received	Messages sent and received successfully within the ISRs.
Have a CAN-FD message in the external bus within a fix period of 10ms	Validate the message and the timing of the message in the CANoe software	CAN messages are being sent to the external bus with a time frame of 10ms±1ns
Make use of module EIM in order to induce faults at the subsystem	An ISR triggered for each error induced and flags exchanged between subsystems	Error successfully injected ant the status flags were able to be exchanged between subsystems
For each fault detected at the system, the state machine must transit from its state	Signalize the event of transition, through a CAN-FD message and through a LED interface	Event signalized; state exchanged.

Use Kalman Filter to predict the value of the angle calculated	Measure the error associated with the prediction having error values lower than 3%	It was able to predict the next value. Values of error lower than expected

Chapter 6

Conclusion and Future Work

After the whole implementation, conclusions can be drawn from the developed work and the results obtained. This chapter aims to make an overview of the results obtained and what can be concluded from them, as well as suggest further improvements that can be made in future implementations.

6.1 Conclusion

With this project it was possible to explore fault-tolerant techniques and get a deep knowledge of fail operational architectures while having in mind the norms that ISO 26262 have designed for these types of systems. It is important to highlight that this project was developed under the V model. For each part developed a set of tests was designed in order to avoid systematic failures in the future. This is of extreme importance since the aim of the architecture developed is to avoid faults, which consequently lead to failures. So, the processes that come before the implementation such as the analysis and design must be performed with the purpose of avoiding faults as well. It is important that these processes be revised by other people besides the one responsible for doing it. This is to ensure redundancy in the process of revising, analysing and designing thus being compliant with V model mentioned.

There are three concepts that could summarize the architecture developed. Those concepts were already approached in the state of the art chapter as background knowledge. Nonetheless, as a conclusion, they are going to be referred once again since the architecture developed does not make sense without having them in context. Redundancy, Fail Operational and Functional safety are the crucial terms for the understanding of this project. As can be concluded, and as stressed throughout the document, the terms are extremely correlated between themselves especially in safety critical applications. It is impossible to have a fail operational architecture without having redundancy satisfied at some level. On the other hand, the purpose for having a fail operational architecture is to be compliant with the functional safety classification imposed by the ISO 26262.

The built-in lockstep mechanism that each subsystem has, among others, was a great challenge in this project. To note that these mechanisms are gaining the interest of the industry mostly due to its safety achievements. In this project it was no different. It was the intention to study how the mechanism in question would react in case of fault. That was possible to achieve with the help of modules that the microcontroller has. With that being done, the design of the best reaction to have when such fault occurs was done in order to still have a valid output in the outer bus, being compliant with the fail operational requirements.

It is believed, in what concerns the requirements, that this project fulfilled them all. In what regards the tests, it is safe to conclude that most of them had the expected result as can be seen and regarding Kalman filter the values of error were even better than was expected.

Obviously, that in order to have a full fail operational architecture there is still a long way to go but the first step has, undoubtedly, already been taken.

6.2 Future Work

Despite the development achieved during this project, it is assuredly possible to improve some points or even follow another path of implementation. During this subchapter those points in which the system developed could improve are going to be presented.

Integrate the calculous algorithm

As stated in previous chapters the use case of this architecture is the calculous of an angle. During the implementation the goal was to keep the architecture the most generic possible enabling it to be used in different types of applications. Nevertheless, it is important to have the application for which the architecture was designed implemented in order to study performances of the application. With that, it is possible to take more reasoned decisions.

Acquisition block

The acquisition block is highly correlated with the previous point. In order to have the calculous implemented it is necessary to have the data in which that calculous will rely on. To have valid data, the acquisition block implemented, and all block related with it, such as the pre-processing one is needed. Once again this is of extreme importance since it allows a performance study of the application. With that done it is possible to analyse how many acquisitions and calculations are possible to have within a 10ms window which is the time that the system possesses to put a CAN message in the external bus.

Communication redundancy

All the design of the architecture was made to avoid common mode failures and for that redundancy at both hardware and software levels were used. However, in what concerns communication there is no total redundancy ensured. In the external communication, the entity that could be transmitting to the external bus is replicated but the bus is not redundant so if any problem occurs with the external bus there is no possibility to transmit a message. In this undesirable scenario, despite the fact that the system works as it should, this problem is interpreted as fault by other systems that will be dependent on this one. Another common mode failure that this architecture has is the internal communication since it has not redundancy ensured. There is only one bus between the subsystems so the same scenario addressed for the external bus could also happen in here. A possible solution is to implement another protocol between subsystems. In that way, if the internal communication bus used was under a fail the subsystems could use the other protocol.

Heterogeneous redundancy

Throughout the implementation of this project the use of homogeneous redundancy for software was highlighted in order to be compliant with the requirements established. Despite the software developed it was made according to the V model imposed by ISO26262 which ensures the redundancy necessary for the processes that came before the implementation. This type of redundancy could lead to permanent faults of the system due to a possible fault in design or even in the implementation phase.

For that reason, the author finds it relevant to study the possibility to use heterogeneous redundancy. With this, the software working at both subsystems would be different from each other but with the same goal which, in this case, is the calculation of the angle. This implementation also must be made according to the V model to ensure the redundancy necessary for revising and implementing the software in question.

Chapter 7

Annexes

```
/*
 * Err_Cfg.h
 *
 *  Created on: 10/07/2020
 *      Author: EMS1BRG
 */

#ifndef ERR_CFG_H_
#define ERR_CFG_H_

#include "S32K2TV.h"
#include "NVIC.h"




/*Enable/Disable channels for error injection*/
#define       ERM_CHANNEL_NOT_INIT   0
#define       ERM_CHANNEL_INIT      1

/*Channels for ERM*/
```

```
#define ERM_CH0 ERM_CHANNEL_NOT_INIT /*Error injection in SRAM0
Error Collected in ERM */
#define ERM_CH1 ERM_CHANNEL_NOT_INIT /*Error injection in SRAM1
Error Collected in ERM*/
#define ERM_CH2 ERM_CHANNEL_NOT_INIT /*Error injection in SRAM2
Error Collected in ERM*/
#define ERM_CH3 ERM_CHANNEL_NOT_INIT /*Error injection in DMA
TCD Error Collected in ERM*/
#define ERM_CH4 ERM_CHANNEL_NOT_INIT /*Error injection in CM33_0
instruction cache tag Error Collected in ERM*/
#define ERM_CH5 ERM_CHANNEL_NOT_INIT /*Error injection in CM33_0
instruction cache data Error Collected in ERM*/
#define ERM_CH6 ERM_CHANNEL_NOT_INIT /*Error injection in CM33_0
data cache tag Error Collected in ERM*/
#define ERM_CH7 ERM_CHANNEL_NOT_INIT /*Error injection in CM33_0
data cache tag Error Collected in ERM*/
#define ERM_CH8 ERM_CHANNEL_NOT_INIT /*Error injection in CM33_1
instruction cache tag Error Collected in ERM*/
#define ERM_CH9 ERM_CHANNEL_NOT_INIT /*Error injection in CM33_1
instruction cache data Error Collected in ERM*/
#define ERM_CH10 ERM_CHANNEL_NOT_INIT /*Error injection in CM33_1
data cache tag Error Collected in ERM*/
#define ERM_CH11 ERM_CHANNEL_NOT_INIT /*Error injection in CM33_1
data cache data Error Collected in ERM*/
#define ERM_CH12 ERM_CHANNEL_NOT_INIT /*Error injection in CM7
instruction cache tag Error Collected in ERM*/
#define ERM_CH13 ERM_CHANNEL_NOT_INIT /*Error injection in CM7
instruction cache data Error Collected in ERM*/
#define ERM_CH14 ERM_CHANNEL_NOT_INIT /*Error injection in CM7 data
cache tag Error Collected in ERM*/
#define ERM_CH15 ERM_CHANNEL_NOT_INIT /*Error injection in CM7 data
cache data0 Error Collected in ERM*/
```

```
#define ERM_CH16 ERM_CHANNEL_NOT_INIT /*Error injection in CM7 data cache data1 Error Collected in ERM*/
#define ERM_CH17 ERM_CHANNEL_INIT /*Error injection in CM33 Lockstep Error Collected in FCCU*/
#define ERM_CH18 ERM_CHANNEL_NOT_INIT /*Error injection in DMA Lockstep Error Collected in FCCU*/
#define ERM_CH19 ERM_CHANNEL_NOT_INIT /*Error injection in EDC checking Error Collected in FCCU*/



/**
 * Function to enable and inject errors at modules enabled
 */
extern void Err_SetEIM(void);

/**
 * Function to initialize ERM(Error Reporting Module)
 */
extern void Err_InitERM(void);

/**
 * Function to initialize FCCU(Fault Collection and Control Unit) to report errors
 */
extern void Err_InitFCCU(void);

#endif /* ERR_CFG_H_ */




/*
```

```
 * Err_Cfg.c
 *
 *  Created on: 10/07/2020
 *      Author: EMS1BRG
 */

#include "Err_Cfg.h"



void Err_SetEIM(void)
{
        /* Check if there if ERM_CH0 is wanted - SRAM0 Error Injection*/
        #if ERM_CH0 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD0_WORD0.B.CHKBIT_MASK |= 1;        /*Bits exchanged at Read Time*/
            EIM.EICHD0_WORD1.B.B0_3DATA_MASK |= 1;        /*Bits    exchanged    at    Read
Time*/
            //EIM.EICHD0_WORD2.B.B4_7DATA_MASK |= 1;     /*Bits    exchanged    at    Read
Time*/
            EIM.EICHEN.B.EICH0EN |= 1;                  /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                     /*Enable Module*/
            Err_InitERM();
        #endif

        /* Check if there if ERM_CH1 is wanted - SRAM1 Error Injection*/
        #if ERM_CH1 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD1_WORD0.B.CHKBIT_MASK |= 1;        /*Bits exchanged at Read Time*/
            EIM.EICHD1_WORD1.B.B0_3DATA_MASK |= 1;        /*Bits    exchanged    at    Read
Time*/
            EIM.EICHD1_WORD2.B.B4_7DATA_MASK |= 1;        /*Bits    exchanged    at    Read
Time*/
            EIM.EICHEN.B.EICH1EN |= 1;                  /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                     /*Enable Module*/
```

```
    #endif


    /* Check if there if ERM_CH2 is wanted - SRAM2 Error Injection*/
    #if ERM_CH2 !=ERM_CHANNEL_NOT_INIT
        EIM.EICHD2_WORD0.B.CHKBIT_MASK |= 1;       /*Bits exchanged at Read Time*/
        EIM.EICHD2_WORD1.B.B0_3DATA_MASK |= 1;       /*Bits exchanged at Read
Time*/
        EIM.EICHD2_WORD2.B.B4_7DATA_MASK |= 1;       /*Bits exchanged at Read
Time*/
        EIM.EICHEN.B.EICH2EN |= 1;                /*Enable Channel to induce errors*/
        EIM.EIMCR.B.GEIEN |= 1;                   /*Enable Module*/
    #endif
    /* Check if there if ERM_CH3 is wanted*/
    #if ERM_CH3 !=ERM_CHANNEL_NOT_INIT
        EIM.EICHD3_WORD0.B.CHKBIT_MASK |= 1;       /*Bits exchanged at Read Time*/
        EIM.EICHD3_WORD1.B.B0_3DATA_MASK |= 1;       /*Bits exchanged at Read
Time*/
        EIM.EICHD3_WORD2.B.B4_7DATA_MASK |= 1;       /*Bits exchanged at Read
Time*/
        EIM.EICHEN.B.EICH3EN |= 1;                /*Enable Channel to induce errors*/
        EIM.EIMCR.B.GEIEN |= 1;                   /*Enable Module*/
    #endif
    /* Check if there if ERM_CH4 is wanted*/
    #if ERM_CH4 !=ERM_CHANNEL_NOT_INIT
        EIM.EICHD4_WORD0.B.CHKBIT_MASK |= 1;       /*Bits exchanged at Read Time*/
        EIM.EICHD4_WORD1.B.B0_3DATA_MASK |= 1;       /*Bits exchanged at Read
Time*/
        EIM.EICHD4_WORD2.B.B4_7DATA_MASK |= 1;       /*Bits exchanged at Read
Time*/
        EIM.EICHEN.B.EICH4EN |= 1;                /*Enable Channel to induce errors*/
        EIM.EIMCR.B.GEIEN |= 1;                   /*Enable Module*/
    #endif
```

```
        /* Check if there if ERM_CH5 is wanted*/
        #if ERM_CH5 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD5_WORD0.B.CHKBIT_MASK |= 1;       /*Bits exchanged at Read Time*/
            EIM.EICHD5_WORD1.B.B0_3DATA_MASK |= 1;      /*Bits exchanged at Read
Time*/
            EIM.EICHD5_WORD2.B.B4_7DATA_MASK |= 1;      /*Bits exchanged at Read
Time*/
            EIM.EICHD5_WORD3.B.B8_11DATA_MASK |= 1;     /*Bits exchanged at Read
Time*/
            EIM.EICHEN.B.EICH5EN |= 1;                  /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                     /*Enable Module*/
        #endif
        /* Check if there if ERM_CH6 is wanted*/
        #if ERM_CH6 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD6_WORD0.B.CHKBIT_MASK |= 1;       /*Bits exchanged at Read Time*/
            EIM.EICHD6_WORD1.B.B0_3DATA_MASK |= 1;      /*Bits exchanged at Read
Time*/
            EIM.EICHD6_WORD2.B.B4_7DATA_MASK |= 1;      /*Bits exchanged at Read
Time*/
            EIM.EICHEN.B.EICH6EN |= 1;                  /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                     /*Enable Module*/
        #endif
        /* Check if there if ERM_CH07 is wanted*/
        #if ERM_CH7 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD7_WORD0.B.CHKBIT_MASK |= 1;       /*Bits exchanged at Read Time*/
            EIM.EICHD7_WORD1.B.B0_3DATA_MASK |= 1;      /*Bits exchanged at Read
Time*/
            EIM.EICHD7_WORD2.B.B4_7DATA_MASK |= 1;      /*Bits exchanged at Read
Time*/
            EIM.EICHD7_WORD3.B.B8_11DATA_MASK |= 1;     /*Bits exchanged at Read
Time*/
            EIM.EICHD9_WORD4.B.B12_15DATA_MASK |= 1;/*Bits exchanged at Read Time*/
```

```
            EIM.EICHEN.B.EICH7EN |= 1;                /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                   /*Enable Module*/
        #endif
        /* Check if there if ERM_CH8 is wanted*/
        #if ERM_CH8 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD8_WORD0.B.CHKBIT_MASK |= 1;      /*Bits exchanged at Read Time*/
            EIM.EICHD8_WORD1.B.B0_3DATA_MASK |= 1;    /*Bits exchanged at Read
Time*/
            EIM.EICHD8_WORD2.B.B4_7DATA_MASK |= 1;    /*Bits exchanged at Read
Time*/
            EIM.EICHEN.B.EICH8EN |= 1;                /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                   /*Enable Module*/
        #endif
        /* Check if there if ERM_CH9 is wanted*/
        #if ERM_CH9 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD9_WORD0.B.CHKBIT_MASK |= 1;      /*Bits exchanged at Read Time*/
            EIM.EICHD9_WORD1.B.B0_3DATA_MASK |= 1;    /*Bits exchanged at Read
Time*/
            EIM.EICHD9_WORD2.B.B4_7DATA_MASK |= 1;    /*Bits exchanged at Read
Time*/
            EIM.EICHD9_WORD3.B.B8_11DATA_MASK |= 1;   /*Bits exchanged at Read
Time*/
            EIM.EICHD9_WORD4.B.B12_15DATA_MASK |= 1;/*Bits exchanged at Read Time*/
            EIM.EICHEN.B.EICH9EN |= 1;                /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                   /*Enable Module*/
        #endif
        /* Check if there if ERM_CH10 is wanted*/
        #if ERM_CH10 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD10_WORD0.B.CHKBIT_MASK |= 1;     /*Bits exchanged at Read Time*/
            EIM.EICHD10_WORD1.B.B0_3DATA_MASK |= 1;  /*Bits exchanged at Read Time*/
            EIM.EICHD10_WORD2.B.B4_7DATA_MASK |= 1;   /*Bits exchanged at Read
Time*/
```

```
            EIM.EICHEN.B.EICH10EN |= 1;              /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                  /*Enable Module*/
        #endif
        /* Check if there if ERM_CH11 is wanted*/
        #if ERM_CH11 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD11_WORD0.B.CHKBIT_MASK |= 1;    /*Bits exchanged at Read Time*/
            EIM.EICHD11_WORD1.B.B0_3DATA_MASK |= 1;  /*Bits exchanged at Read Time*/
            EIM.EICHD11_WORD2.B.B4_7DATA_MASK |= 1;    /*Bits exchanged at Read
Time*/
            EIM.EICHD11_WORD3.B.B8_11DATA_MASK |= 1;/*Bits exchanged at Read Time*/
            EIM.EICHD11_WORD4.B.B12_15DATA_MASK |= 1;  /*Bits exchanged at Read
Time*/
            EIM.EICHEN.B.EICH11EN |= 1;              /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                  /*Enable Module*/
        #endif
        /* Check if there if ERM_CH12 is wanted*/
        #if ERM_CH12 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD12_WORD0.B.CHKBIT_MASK |= 1;    /*Bits exchanged at Read Time*/
            EIM.EICHD12_WORD1.B.B0_3DATA_MASK |= 1;  /*Bits exchanged at Read Time*/
            EIM.EICHD12_WORD2.B.B4_7DATA_MASK |= 1;    /*Bits exchanged at Read
Time*/
            EIM.EICHEN.B.EICH12EN |= 1;              /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                  /*Enable Module*/
        #endif
        /* Check if there if ERM_CH13 is wanted*/
        #if ERM_CH13 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD13_WORD0.B.CHKBIT_MASK |= 1;    /*Bits exchanged at Read Time*/
            EIM.EICHD13_WORD1.B.B0_3DATA_MASK |= 1;  /*Bits exchanged at Read Time*/
            EIM.EICHD13_WORD2.B.B4_7DATA_MASK |= 1;    /*Bits exchanged at Read
Time*/
            EIM.EICHD13_WORD3.B.B8_11DATA_MASK |= 1;/*Bits exchanged at Read Time*/
```

```
            EIM.EICHD13_WORD4.B.B12_15DATA_MASK |= 1;  /*Bits exchanged at Read
Time*/
            EIM.EICHEN.B.EICH13EN |= 1;              /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                  /*Enable Module*/
        #endif
        /* Check if there if ERM_CH14 is wanted*/
        #if ERM_CH14 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD14_WORD0.B.CHKBIT_MASK |= 1;     /*Bits exchanged at Read Time*/
            EIM.EICHD14_WORD1.B.B0_3DATA_MASK |= 1;  /*Bits exchanged at Read Time*/
            EIM.EICHD14_WORD2.B.B4_7DATA_MASK |= 1;     /*Bits exchanged at Read
Time*/
            EIM.EICHD14_WORD3.B.B8_11DATA_MASK |= 1;/*Bits exchanged at Read Time*/
            EIM.EICHD14_WORD4.B.B12_15DATA_MASK |= 1;  /*Bits exchanged at Read
Time*/
            EIM.EICHEN.B.EICH14EN |= 1;              /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                  /*Enable Module*/
        #endif
        /* Check if there if ERM_CH15 is wanted*/
        #if ERM_CH15 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD15_WORD0.B.CHKBIT_MASK |= 1;     /*Bits exchanged at Read Time*/
            EIM.EICHD15_WORD1.B.B0_3DATA_MASK |= 1;  /*Bits exchanged at Read Time*/
            EIM.EICHD15_WORD2.B.B4_7DATA_MASK |= 1;     /*Bits exchanged at Read
Time*/
            EIM.EICHD15_WORD3.B.B8_11DATA_MASK |= 1;/*Bits exchanged at Read Time*/
            EIM.EICHD15_WORD4.B.B12_15DATA_MASK |= 1;  /*Bits exchanged at Read
Time*/
            EIM.EICHEN.B.EICH15EN |= 1;              /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                  /*Enable Module*/
        #endif
        /* Check if there if ERM_CH16 is wanted*/
        #if ERM_CH16 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD15_WORD0.B.CHKBIT_MASK |= 1;     /*Bits exchanged at Read Time*/
```

```
            EIM.EICHD15_WORD1.B.B0_3DATA_MASK |= 1;  /*Bits exchanged at Read Time*/
            EIM.EICHD15_WORD2.B.B4_7DATA_MASK |= 1;     /*Bits exchanged at Read
Time*/
            EIM.EICHD15_WORD3.B.B8_11DATA_MASK |= 1;/*Bits exchanged at Read Time*/
            EIM.EICHD15_WORD4.B.B12_15DATA_MASK |= 1;  /*Bits exchanged at Read
Time*/
            EIM.EICHEN.B.EICH15EN |= 1;              /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                  /*Enable Module*/
        #endif
        /* Check if there if ERM_CH17 is wanted*/
        #if ERM_CH17 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD17_WORD1.B.B0_3DATA_MASK      |=      0b1111111111111111111;
    /*Bits exchanged at Read Time*/
            EIM.EICHEN.B.EICH17EN |= 1;              /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                  /*Enable Module*/

        #endif
        /* Check if there if ERM_CH18 is wanted*/
        #if ERM_CH18 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD18_WORD1.B.B0_3DATA_MASK |= 1111; /*Bits exchanged at Read
Time*/
            EIM.EICHEN.B.EICH18EN |= 1;              /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                  /*Enable Module*/
        #endif
        /* Check if there if ERM_CH19 is wanted*/
        #if ERM_CH19 !=ERM_CHANNEL_NOT_INIT
            EIM.EICHD18_WORD1.B.B0_3DATA_MASK |= 1111; /*Bits exchanged at Read
Time*/
            EIM.EICHEN.B.EICH18EN |= 1;              /*Enable Channel to induce errors*/
            EIM.EIMCR.B.GEIEN |= 1;                  /*Enable Module*/
        #endif
}
```

```
void Err_InitERM(void)
{
    /* Check if there if ERM_CH0 is enabled - SRAM0 Error Injection*/
          #if ERM_CH0 !=ERM_CHANNEL_NOT_INIT
              ERM.CR0.B.ENCIE0=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
              ERM.CR0.B.ESCIE0=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
              ERM.CR0.B.ENCIE1=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
              ERM.CR0.B.ESCIE1=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
          #endif

          /* Check if there if ERM_CH1 is enabled - SRAM1 Error Injection*/
          #if ERM_CH1 !=ERM_CHANNEL_NOT_INIT
              ERM.CR0.B.ENCIE2=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
              ERM.CR0.B.ESCIE2=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
          #endif

          /* Check if there if ERM_CH2 is enabled - SRAM2 Error Injection*/
          #if ERM_CH2 !=ERM_CHANNEL_NOT_INIT
              ERM.CR0.B.ENCIE3=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
              ERM.CR0.B.ESCIE3=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
          #endif
          /* Check if there if ERM_CH3 is enabled*/
          #if ERM_CH3 !=ERM_CHANNEL_NOT_INIT
              ERM.CR2.B.ENCIE16=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
              ERM.CR2.B.ESCIE16=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
```

```
        #endif
        /* Check if there if ERM_CH4 is enabled*/
        #if ERM_CH4 !=ERM_CHANNEL_NOT_INIT
            ERM.CR0.B.ENCIE4=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR0.B.ESCIE4=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
        #endif
        /* Check if there if ERM_CH5 is enabled*/
        #if ERM_CH5 !=ERM_CHANNEL_NOT_INIT
            ERM.CR0.B.ENCIE5=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR0.B.ESCIE5=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
        #endif
        /* Check if there if ERM_CH6 is enabled*/
        #if ERM_CH6 !=ERM_CHANNEL_NOT_INIT
            ERM.CR0.B.ENCIE6=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR0.B.ESCIE6=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
        #endif
        /* Check if there if ERM_CH07 is enabled*/
        #if ERM_CH7 !=ERM_CHANNEL_NOT_INIT
            ERM.CR0.B.ENCIE7=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR0.B.ESCIE7=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
        #endif
        /* Check if there if ERM_CH8 is enabled*/
        #if ERM_CH8 !=ERM_CHANNEL_NOT_INIT
            ERM.CR1.B.ENCIE8=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR1.B.ESCIE8=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
        #endif
        /* Check if there if ERM_CH9 is enabled*/
```

```
        #if ERM_CH9 !=ERM_CHANNEL_NOT_INIT
            ERM.CR1.B.ENCIE9=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR1.B.ESCIE9=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
        #endif
        /* Check if there if ERM_CH10 is enabled*/
        #if ERM_CH10 !=ERM_CHANNEL_NOT_INIT
            ERM.CR1.B.ENCIE10=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR1.B.ESCIE10=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
        #endif
        /* Check if there if ERM_CH11 is enabled*/
        #if ERM_CH11 !=ERM_CHANNEL_NOT_INIT
            ERM.CR1.B.ENCIE11=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR1.B.ESCIE11=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
        #endif
        /* Check if there if ERM_CH12 is enabled*/
        #if ERM_CH12 !=ERM_CHANNEL_NOT_INIT
            ERM.CR2.B.ENCIE12=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR2.B.ESCIE12=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
        #endif
        /* Check if there if ERM_CH13 is enabled*/
        #if ERM_CH13 !=ERM_CHANNEL_NOT_INIT
            ERM.CR1.B.ENCIE13=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR1.B.ESCIE13=1; /*Trigger an interrupt if channel 0 has more than one bit
exchanged*/
        #endif
        /* Check if there if ERM_CH14 is enabled*/
        #if ERM_CH14 !=ERM_CHANNEL_NOT_INIT
            ERM.CR1.B.ENCIE14=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
```

```
            ERM.CR1.B.ESCIE14=1; /*Trigger an interrupt if channel 0 has more than one bit exchanged*/
        #endif
        /* Check if there if ERM_CH15 is enabled*/
        #if ERM_CH15 !=ERM_CHANNEL_NOT_INIT
            ERM.CR1.B.ENCIE15=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR1.B.ESCIE15=1; /*Trigger an interrupt if channel 0 has more than one bit exchanged*/
        #endif
        /* Check if there if ERM_CH16 is enabled*/
        #if ERM_CH16 !=ERM_CHANNEL_NOT_INIT
            ERM.CR1.B.ENCIE15=1; /*Trigger an interrupt if channel 0 has one bit exchanged*/
            ERM.CR1.B.ESCIE15=1; /*Trigger an interrupt if channel 0 has more than one bit exchanged*/
        #endif

        /*TODO: Investigate how to inject errors in flash*/
        /*Trigger interrupt if some error is found on the different blocks of FLASH memory**/
        ERM.CR2.B.ENCIE17=1;
        ERM.CR2.B.ESCIE17=1;

        ERM.CR2.B.ENCIE18=1;
        ERM.CR2.B.ESCIE18=1;

        ERM.CR2.B.ENCIE19=1;
        ERM.CR2.B.ESCIE19=1;

        ERM.CR2.B.ENCIE20=1;
        ERM.CR2.B.ESCIE20=1;

    Enable_Interrupt(ERM0_IRQn);
    Enable_Interrupt(ERM1_IRQn);
```

```
}




void FCCU_clear_faults(void)
{
    /* 1. Write the proper key into the FCCU_NCFK register */
    //Non-critical fault key = AB34_98FEh
    FCCU.NCFK.R = 0xAB3498FE;
    /* 2. Clear the status (flag) bit NCFSx => the opcode OP12 is automatically
    /* Read all NCFS registers to clear all faults.*/
    /* For details which faults can be cleared see Table 7-36. FCCU Non-Critical Faults Mapping
in RM */
    FCCU.NCF_S[0].R = 0xFFFFFFFF; // read FCCU.NCF_S0 register
    /* Verify if state change was successful */
    while (FCCU.CTRL.B.OPS != 0x3); //Operation status successful
    /* NCFS_1 register clear */
    FCCU.NCFK.R = 0xAB3498FE; //Non-critical fault key = AB34_98FEh
    FCCU.NCF_S[1].R = 0xFFFFFFFF; // clear FCCU.NCF_S1 register
    /* Verify if state change was successful */
    while (FCCU.CTRL.B.OPS != 0x3); //Operation status successful
    /* NCFS_2 register clear */
    FCCU.NCFK.R = 0xAB3498FE; //Non-critical fault key = AB34_98FEh
    FCCU.NCF_S[2].R = 0xFFFFFFFF; // clear FCCU.NCF_S2 register
    /* Verify if state change was sucessful */
    while (FCCU.CTRL.B.OPS != 0x3); //Operation status succesfull
}//FCCU_clear_faults




void Err_InitFCCU(void)
```

```
{
        FCCU_clear_faults();
        FCCU.CFG_TO.B.TO = 7;             /*Timeout for Congig state*//*Later review this
value*/
        __asm__("svc #0x00"); // System Call to enable Supervisor Mode
        /* Unlock configuration */
         FCCU.TRANS_LOCK.R = 0xBC;
        /* provide Config state key */
         FCCU.CTRLK.R = 0x913756AF;          //key for OP1
         /* enter config state - OP1 */
         FCCU.CTRL.R = 0x1;                //set OP1 - set up FCCU into the CONFIG mode
         /* wait for successful state transition */
         while (FCCU.CTRL.B.OPS != 0x3); //operation status successful

         /*****************************************************************
          *
          *             Lockstep error channel Configuration
          *         Functional RESET/ISR
          * ***************************************************************/

         /*Configure NCF channel*/
         FCCU.NCF_CFG[0].B.NCFC1 = 1;/*Lockstep channel enable*/
         /* 1-Enable Functional Reset// 0-Enable ISR*/
         FCCU.NCFS_CFG[0].B.NCFSC1 = 0;
         /*Enable Timeout signal for any channel*/
         FCCU.NCF_TO.B.TO = 5;                                  /*Review this value*/
         /*Enable alarm-state*/
         FCCU.NCF_TOE[0].B.NCFTOE1 = 1;
         /*Enable Alarm state reaction*/
         FCCU.IRQ_ALARM_EN[0].B.IRQEN1 = 1;
         /*Enable Channel*/
         FCCU.NCF_E[0].B.NCFE1 = 1;
```

```
/*******************************************************************************************/
        /* set up the NORMAL mode of FCCU */
        FCCU.CTRLK.R = 0x825A132B;          //key for OP2
        FCCU.CTRL.R = 0x2;                  //set the OP2 - set up FCCU into the NORMAL mode
        while (FCCU.CTRL.B.OPS != 0x3); //operational status successful

        Enable_Interrupt(FCCU0_IRQn);
        Enable_Interrupt(FCCU1_IRQn);

}
```

Snippet 1: Err_Cfg.c

References

[1] B. Chen, “practices and Challenges for Achieving Functional Safety of Modern Automotive SoCs,” 2019.

[2] C. Tiang, C. Zong, L. He e X. W. Y. Don, *Fault tolerant control method for steer-by-wire system,* Changchun: International Conference on Mechatronics and Automation, 2009.

[3] Cédric Wilwert, N. Navet, Y.-Q. Song e F. Simonot-Lion, *Design of automotive X-by-Wire systems,* 2007.

[4] B. M., “Fault-Tolerant Platforms for Automotive Safety-Critical Applications,” 2003.

[5] Z. Hu e F. Zhang, “Reliability Analysis of Redundant Steering-by-Wire System,” em *Sixth International Conference on Intelligent Systems Design and Engineering Applications (ISDEA)*, Guiyang, 2015.

[6] B. Euram, “X-by-wire – safety related fault tolerant systems in vehicles, final report,” 1998.

[7] Koshal, “DifferenceBetween.com,” 15 June 2011. [Online]. Available: https://www.differencebetween.com/difference-between-safety-and-vs-security/. [Acedido em 16 December 2019].

[8] ISO, *ISO 26262: Road vehicles - Functional Safety,* 2018.

[9] Nelson e V. P., “Fault-Tolerant Computing: Fundamental Concepts,” IEEE, Auburn, 1990.

[10] M. Hitt, *Fault-tolerant avionics.,* 2006.

[11] Reilly e B. Michael, Programming Embedded Systems in C and C++, O Reilly, 1999.

[12] T. Noergaard, Embedded Systems Architecture - A Comprehensive Guide for Engineers and Programmers, 2005.

[13] A. Schnellbach, *Fail-operational automotive systems,* Graz, 2016.

[14] “ISO 26262-1 Road vehicles — Functional safety — Part 1: Vocabulary,” 2018.

[15] “ISO 26262-2 Road vehicles — Functional safety — Part 2: Management of Functional Safety,” 2018.

[16] “ISO 26262-3 Road vehicles — Functional safety — Part 3: Concept phase,” 2018.

[17] “ISO 26262-4 Road vehicles — Functional safety — Part 4: Product development at the system level,” 2018.

[18] “ISO 26262-6 Road vehicles — Functional safety — Part 6: Product development at the software level,” 2018.

[19] “ISO 26262-9 Road vehicles — Functional safety — Part 9: Automotive safety integrity level (ASIL)-oriented and safety-oriented analyses,” 2018.

[20] A. Avizienis, J. .. Laprie, B. Randell e C. Landwehr, “Basic Concepts and taxonomy of dependable and secure computing,” em *IEEE Transactions on Dependable and Secure Computing*, 2004.

[21] E. Dubrova, Fault-tolerant design, Springer, 2013.

[22] L. Montecchi, P. Lollini e A. Bondavalli, “Towards a MDE Transformation Workflow for Dependability Analysis,” em *16th IEEE International Conference on Engineering of Complex Computer Systems*, Las Vegas, 2011.

[23] Institute of Electrical and Electronics Engineers, “IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.,” New York, 1990.

[24] H. Kopetz, “Real-Time Systems: Design Principles for Distributed Embedded Applications,” em *Kluwer Academic Publishers*, 1997].

[25] C. Rodrigues, I. Marques, S. Pinto, T. Gomes e A. Tavares, “Towards a Heterogeneous Fault-Tolerance Architecture based on Arm and RISC-V Processors,” 2019.

[26] I. Umer, “Design methods and practises for fault prevention and management in spacecraft,” em *Tech. Rep. 20060022566, NASA* , 2005.

[27] N. Kanekawa, E. Ibe, T. Suga e Y. Uematsu, Fault-Tolerant System Technology. In: Dependability in Electronic Systems, New York: Springer, 2011.

[28] H. Quinn, Z. Baker, T. Fairbanks, J. L. Tripp e G. Duran, “Software Resilience and the Effectiveness of Software Mitigation in Microcontrollers,” em *IEEE TRANSACTIONS ON NUCLEAR SCIENCE*, 2015.

[29] B. Randell, “System structure for software fault tolerance,” em *International Conference on Reliable Software*, 1975.

[30] J. Sloan, R. Kumar e G. Bronevetsky, “Algorithmic Approaches to Low Overhead FaultDetection for Sparse Linear Algebra”.

[31] J. Laprie, "Dependable computing and fault tolerance: concepts and terminology," em *Proceedings of 15th International Symposium on Fault-Tolerant Computing (FTSC-15)*.

[32] J. Von Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliablecomponents," em *Shannon, C., McCarthy, J. (eds.) Automata Studies*, Princeton , 1956.

[33] A. vižienis, "Fault-tolerant systems," em *IEEE Trans. Comput.25(12), 1304–1312*, 1976.

[34] MarFtin, "Spaceradiation.eu," 2 February 2020. [Online]. Available: https://spaceradiation.eu/redundancy-and-its-types-in-computer-systems/. [Acedido em 9 March 2020].

[35] M. Yang, G. Hua, Y. Feng e J. Gong, Fault-Tolerance Techniques for Spacecraft Control Computers, Wiley, 2017.

[36] J. Abella e C. H. and, "Live: Timely error detection in light-lockstep safety critical systems," em *In 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014.

[37] R. D. Kral, J. S. Chong e A. L. Schreiber, "Implementation of a Loosely-Coupled Lockstep Approach in the Xilinx Zynq-7000 All Programmable SoC™ for High Consequence Applications," Sandia National Laboratories, Albuquerque.

[38] NXP, *S32K2TV Datasheet,* Abstatt: NXP Semiconductors, 2019.

[39] A. Becker, "KalmanFilter.NET," 2018. [Online]. Available: https://www.kalmanfilter.net/default.aspx//#aboutAuthor. [Acedido em 13 May 2020].

[40] Vector, "Vector," CAN interfaces, [Online]. Available: https://www.vector.com/int/en/products/products-a-z/software/canoe/. [Acedido em 19 08 2020].

[41] A. M. Lister, Fundamentals of operating systems, 2013.

[42] M. Vuori, "Agile development of safety-critical software," em *Tampere University of Technology*, 2011..

[43] T. Instruments, *ISO1042-Q1 Automotive Isolated CAN Transceiver With 70-V Bus Fault Protection and Flexible Data Rate,* 2020.

[44] NXP, *TJF1052i Galvanically isolated high-speed CAN transceiver,* 2016.

[45] D. J. Barrenscheen, "On-Board Communication via CAN without Transceiver," SIEMENS, 1996.

[46] T. Instruments, *ISO772x-Q1 High-Speed, Robust EMC, Reinforced Dual-Channel Digital Isolators,* 2020.

[47] S. Ray e e. al, “Extensibility in automotive security: Current practice and challenges: Invited,” em *Proc. 54th Ann. Design Autom. Conf.*, New York, 2017.

[48] C. Tian, C. Zong, L. He, X. Wang e Y. Dong, “Fault tolerant control method for steer-by-wire system,” em *International Conference on Mechatronics and Automation*, Changchun, 2009.

[49] DeFranco, P. A. Laplante e J. F., “Software engineering of safety-critical systems: Themes from practitioners,” em *IEEE Transactions on Reliability*, 2017.

[50] G. S. Rodrigues, F. Rosa, A. B. D. Oliveira, F. Lima, L. Ost e R. Reis, “Analyzing the Impact of Fault-Tolerance Methods in ARM Processors under Soft Errors Running Linux and Parallelization APIs,” em *IEEE Transactions on Nuclear Science*, 2017.

[51] A. Goleman, D. Boyatzis e R. Mckee, “Operating Systems,” em *Operating Systems*, 2019.

[52] J. Hatcliff, A. Wassyng, T. Kelly, C. Comar e P. Jones, “Certifiably safe software-dependent systems: challenges and directions,” em *Proceedings of the on Future of Software Engineering*, 2014.