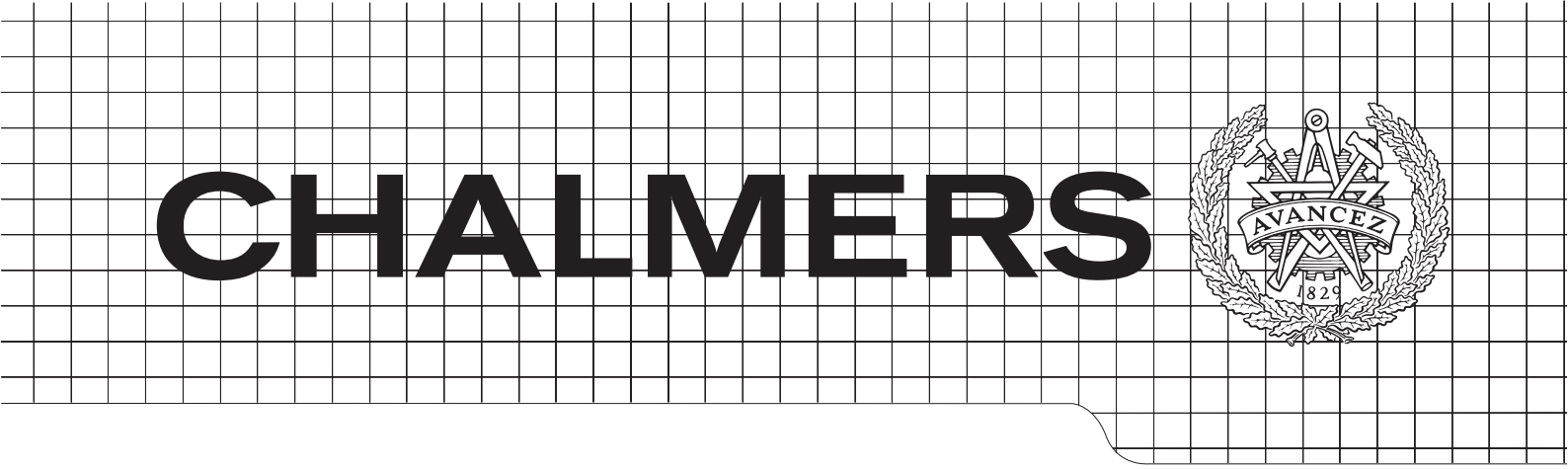

Design and implementation of a fault-tolerant drive-by-wire system

Master of Science Thesis in Embedded Electronics System Design

Alexander Altby

Davor Majdandzic

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2014

Alexander Altby, Davor Majdandzic.

Examiner: Johan Karlsson.

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Sweden
Telephone + 46(0)31-772 1000

Department of Computer Science and Engineering
Gothenburg, Sweden June 2014

Abstract

This thesis presents the design and implementation of a prototype for a drive-by-wire system in road vehicles. The prototype extends an existing (non-fault-tolerant) prototype with fault tolerance by implementing distributed brake system and dual modular redundancy for a central control unit. The steering is made redundant by utilizing the distributed brakes and using the braking capability on either side of the car. This will cause the car to turn in corresponding direction, i.e., steer-by-brakes. A hardware monitor is designed and implemented in the redundant central control units. The purpose of the hardware monitor is to restart the control unit in case of a failure.

The non–fault-tolerant prototype is being used as a reference design when analysing the reliability and safety of the fault tolerant design. An analysis is made to verify the lowest failure rate that the design must tolerate in order to meet a target reliability of 99.999% after 10 years. The thesis follows the guidelines of the standard for functional safety in road vehicles, ISO 26262.

Acknowledgements

Special thanks to Behrooz Sangchoolie, PhD student in the fault tolerance area at Chalmers University, for the technical support, helping with the report, giving continuous feedback and providing his technical expertise in the fault tolerant area.

Also, thanks to Johan Karlsson, professor in dependable Real-Time Systems at Chalmers University, for assisting with his technical expertise in the fault tolerant area and being the examiner for this thesis.

We would also like to thanks the attendants of the reading group regarding the ISO 26262; Behrooz Sangchoolie, Pierre Kleberger, Fatemeh Ayatolahi and Aljoscha Lautenbach. The learning and insight of your technical expertise has raised our interest in the area of fault tolerance. You have been an influence and large part of the outcome of this thesis.

Last but not least, thanks to David Rydén at Sigma technology for accepting the proposed thesis and providing the industrial expertise in the automotive area. Also, thanks to Alexandra Angerd at Sigma technology for giving us assistance and introduction of the previous developed system.

Alexander Altby & Davor Majdandzic, Gothenburg June 2014.

ABBREVIATIONS

ADC	Analog to Digital Converter
ABS	Anti-lock Brake System
ASIC	Application-Specific Integrated Circuit
ASIL	Automotive Safety Integrity Level
CAN	Controller Area Network
CCU	Central Control Unit
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
GPIO	General-Purpose Input/Output
E/E	Electrical/Electronic
ECU	Electrical Control Unit
EMI	ElectroMagnetic Interference
ESP	Electronic Stability Program
FFA	Functional Failure Analysis
HARA	Hazard Analysis and Risk Assessment
HW	Hardware
JTAG	Join Test Action Group
LED	Light Emitting Diode
LCD	Liquid Crystal Display
ODE	Ordinary Differential Equation
SIL	Safety Integrity Level
SPI	Serial Peripheral Interface
SW	Software
SWIFI	Software Implemented Fault Injection

Contents

List of Figures

List of Tables

1 Introduction

One of the next big things in vehicle industry is self-driving cars or autonomous cars. An autonomous car require that actuators that control the motion of the vehicle, can be interacted with electronically. Therefore a *drive-by-wire* system is needed. A *drive-by-wire* system replaces the mechanical systems in a traditional vehicle by using electrical/electronic (E/E) systems to perform fundamental vehicle functions.

The drive-by-wire system includes steer-by-wire, brake-by-wire and throttle-by-wire. The "by-wire" expression means that the information, from the sensor to the actuator of the different systems, is transferred electronically through wires and not by traditional hydraulic systems or mechanically through struts or shafts.

The advantage of using drive-by-wire rather than mechanical systems is that reduction of cost, moving parts and weight can be achieved. Since the steering rack can be removed, the car's shock impact, in case of a collision, can be improved. Using an electrical based system will also increase the information flow and ease up the interconnect between different components in the car, facilitating the use of safety functions such as; ABS (anti-lock brake system), ESP (electronic stability programme), etc.

Electromagnetic interference (EMI) [1] and ionizing radiation [2][3] are two examples of error sources that may introduce failure to an integrated circuit in an E/E system. The systems that constitute the functionality of the drive-by-wire system must be able to handle the failures in a predictable way, since losing the control of a car in high velocity may notoriously end in lethal outcome. Therefore, the drive-by-wire system needs to be as fault tolerant as reasonably possible.

Safety can be achieved by introducing safety mechanisms in vital components of the drive-by-wire system. The safety mechanisms will have to target faults that can put the drive-by-wire system in a state which may have lethal outcome for the driver or other road users. A standard for functional safety in road vehicles, ISO 26262 [4], was released November 2011 to provide the vehicle industry with guidelines on how to developing safe-critical applications in a vehicle.

This thesis deals with the development of a fault-tolerant architectural design for a drive-by-wire system. By following the guidelines of the ISO 26262 standard, this thesis targets possible fail sources and how to deal with them in the drive-by-wire system. A safety analysis is made to ensure that the reliability and safety of the designed system is improved.

An implementation of a prototype for a fault-tolerant design is also made to demonstrate the feasibility of the proposed architecture. The prototype was implemented by extending an existing non-fault-tolerant prototype developed in the thesis made by Angerd and Johansson2. The design of the non-fault-tolerant prototype is used as a reference when comparing the reliability and safety improvements in the system where fault tolerant is designed.

This thesis makes the following contributions:

- Proposition of a fault-tolerant architectural design. The design consists of a distributed brake system with steer-by-brake functionality and redundancy in vital units.
- Reliability analysis showing how a required reliability can be achieved for the fault-tolerant design. The analysis proposes a Markov model for the design and compares it to a non-fault-tolerant design. Further, a safety analysis is proposed showing the steady-state safety of the fault-tolerant design.
- Realization of a prototype to show the feasibility of the proposed architecture. The prototype shows the key components contributing to the fault tolerance in the architecture.

The key components are the hardware monitor, the fault detection mechanisms and the failure avoidance mechanisms.

The content of this thesis is outlined as follows: Section 2 explains the non-fault-tolerant drive-by-wire system made by earlier thesis students. Section 3 describes the background and theory of the different techniques used in this thesis. Section 4 is the concept phase in which the items of the drive-by-wire system are defined. Certain risks and hazards is also identified in section 4. The methods which is to be applied to the items, in order to prevent the risks and hazards from occurring, is presented in section 5. Section 6 shows that the safety strategies applied to the fault-tolerant design increases the reliability and safety compared to the reference design. Section 7 presents how the safety mechanisms and methods is implemented. A timing analysis is represented in section 8 in order to show how the implemented methods and mechanisms effects the execution time of the software. In section 9, the safety mechanisms are tested and verified. Finally a discussion and conclusion is made in section 10 and 11.

2 Existing prototype

The fault-tolerant drive-by-wire prototype is based on an earlier master's thesis project made by Angerd and Johansson [5]. The existing prototype consists of a central control unit (CCU), ECUs for the steering, brake and throttle systems and sensors for the driver input. An overview of the non-fault-tolerant design is shown in Figure 2.1.

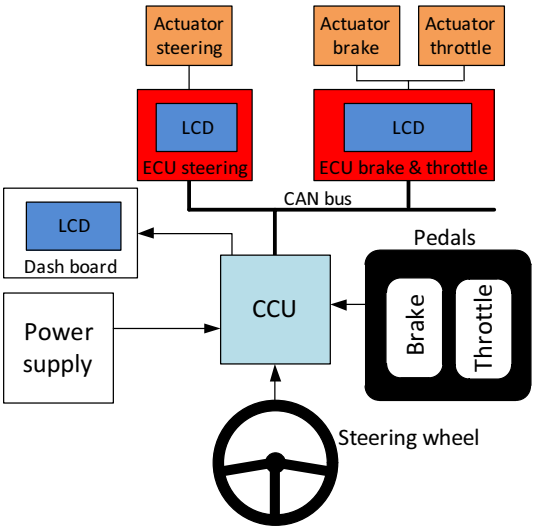

Figure 2.1: System overview of the non-fault-tolerant drive-by-wire system. The steering wheel and pedals for brake and throttle, consists of the system's sensors.

A block diagram of the reference drive-by-wire system is shown in Figure 2.2. The three sensors for throttle, brake and the steering are connected to the central control unit (CCU). The CCU is connected to two electronic control units (ECUs) via a controller area network (CAN) bus, one ECU for throttle and brake and another one for steering. The two ECUs are connected to the actuators controlling the rear-wheel drive and steering.

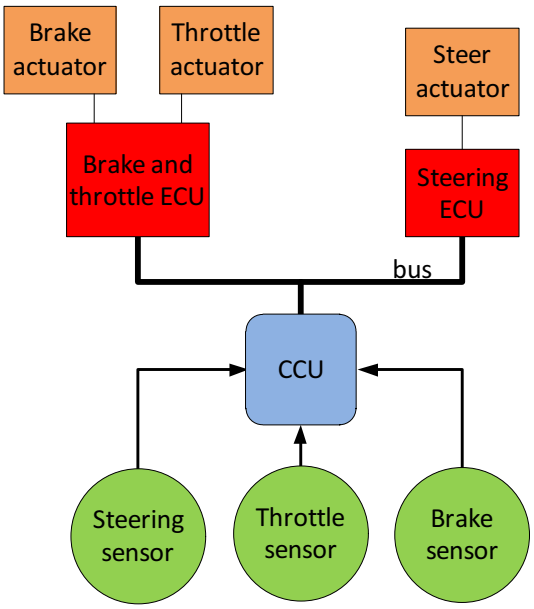

Figure 2.2: Block diagram of the non-fault-tolerant drive-by-wire system made by previous thesis students.

The CCU is implemented on a Texas Instruments Hercules development board, TMS570LS3137 [6]. It uses an operating system called FreeRTOS [7] which is an open-source real-time kernel. The sensors for brake and throttle are connected to two AD converters on the development board of the CCU. The steering wheel is connected to two general-purpose–I/O pins on the development board. The development board also has a CAN interface which enables for CAN communication from the CCU to the ECUs.

The ECU for brake/throttle is implemented on an 8-bit Atmega [8]. Since the Atmega does not have a CAN interface, a separate SPI to CAN circuit is used. A servo and an LED array is used to represent the brake value. An LED array and an electric motor connected through a H-bridge to the CPU represents the throttle value. An LCD display is also connected to the CPU which displays the current brake and throttle values. A similar ECU used for brake/throttle is also used for steering without the LED arrays, H-bridge and electric motor [5].

To do a fair comparison between the fault-tolerant design and the design of the existing prototype, a reference drive-by-wire system is defined. The non-fault-tolerant system, hereinafter referred to as reference system, is shown in Figure 2.3

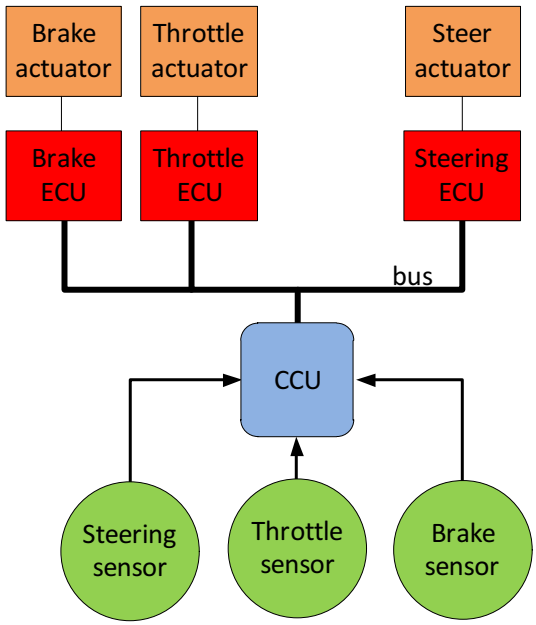

Figure 2.3: Block diagram of the reference drive-by-wire system with no fault tolerance.

Compared to the previous system, shown in Figure 2.2, the reference system has separated brake and throttle ECU.

3 Technical background

This sections explains the concept of fault tolerance. Common attributes that are used to measure a system's ability to avoid failures, are stated and explained. The concept of a hazardous event and how a system can propagate to such a state is further explained. Different techniques of fault injections in order to test the robustness of a system, are also explained. Definition of, and difference between, single-point fault and multiple-point faults are described. Different techniques in order to increase safety, such as; voting, redundancy and fault avoidance and forecasting are elaborated. Overview of local, central and distributed architecture designs are also elaborated on and explained in this section.

3.1 Dependability

Dependability is the system's ability to avoid failures. Dependability is described using four different attributes according to [9][10]. The following four attributes are:

- *Availability* - Readiness for usage. The probability that the system will work as intended at any given time.
- *Reliability* - Continuity of service without failure. The probability that the system will work as intended under a specific time.
- *Safety* - The probability of avoiding a catastrophic event.
- *Maintainability* - Ability to handle repairs, modifications and updates.

In a drive-by-wire system all of these can be of importance. Since a poor *reliability* and *safety* in the drive-by-wire system can harm the driver, these are the highlighted attributes in this thesis.

3.2 Faults, errors and hazards

A hazard is the occurrence of an event that puts people in risk of danger [9]. Example of a hazard is loosing a wheel of the vehicle when driving. This event puts the driver, pedestrians or other road users at risk of getting injured. A hazard occurs when a *fault* propagates to an *error* that is not covered by safety mechanisms in a system.

A *fault* is either a random event (e.g. a bit-flip or frozen memory bit) or a systematic event (e.g. bug in a code) in a system or subsystem. The duration of faults can be categorized into three groups; *transient faults*, *intermittent faults* and *permanent faults* [9]. *Transient faults* are faults that appear and disappears under a short period of time. Example of a transient fault is a temporary random bit-flip in the memory. *Intermittent faults* are faults that appears sporadically. The source of an intermittent fault is therefore usually hard to discover. Intermittent faults can be caused by cold soldering or loose connectors. *Permanent faults* are the kind of faults that when they occur, they remain indefinitely [9]. Electromigration [11] and design faults are two examples of permanent fault sources.

An *error* is when the fault propagates and changes the output of the system in such way that it behaves in an unwanted manner [12]. An example could be when a bit flip occurs in a memory cell that stores the current steering value. If there is no safety mechanism to detect that fault, the fault will result in an error when the steering value is sent to the electronic control unit (ECU) for steering.

A fault can either be latent or active [12]. When latent, the fault does not directly alter the systems behaviour in such way that it causes an error. A latent fault may however, in combination with another fault, propagate to an error. See multiple-point faults, section 3.6. When a fault is active, the system is altered from its correct behaviour and creates an error. If the system can not handle the error, i.e., an un-covered fault, system failure will occur. The system failure can either propagate to a fail-safe state or a critical failure which is interpreted as a hazard. In some cases the fail-safe state can be achieved by turning off the system. When a system failure occurs in a subsystem, the failure may propagate to higher system levels and cause faults. This may result in failure of the whole system. [12]

Figure 3.1 shows a flowchart of how a system is exposed to a fault, how the fault can propagate to an error and how the result of this propagation can result in a system failure. The system failures; fail safe and critical failure, are further elaborated in section 3.3.

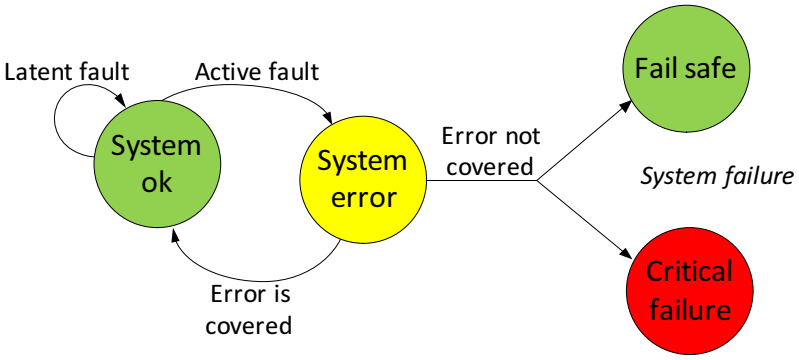

Figure 3.1: Flowchart showing how a fault propagates to an error and leads to a failure in a system.

3.3 Degree of operation

In fault tolerance there exists four types of degrees explaining a system's current ability of operation after one fault [13]. These degrees are:

- Fail operational - System is still operational after a fault.
- Degraded operational - System is operational but at degraded functionality after a fault. Degraded operation refers to an acceptable operation that alters from the correct system behaviour. An example of degraded operational is loosing control of one brake actuator in a car's brake system. Since three out of four brake actuators still are functional, the brake system is at degraded mode.
- Fail safe - System at safe operation after a fault. Safe operation is when the system goes to a safe operational state in order to prevent critical failure. Safe operation may for example be fail silent. Fail silent is when the system does not send output data in case of failure, which is in some cases a better alternative than sending faulty data.
- Critical failure - System at critical operation after a fault. A critical operation is a fault that propagates through the system and results in a failure. The failure makes the system differ from its correct behaviour so that it may result in a catastrophic event. An example of a critical failure is loosing the steering functionality in a vehicle.

3.4 Integrity level

Nuclear plants, airplanes, vehicles and toasters are all systems where safe operation is of importance. The safety clearly involves the risk of potential hazardous events, usage of the system and the severity if a hazard occurs. Since different requirements are needed for different safety systems the concept of integrity level arises. Safety integrity level (SIL) is based on the probability of a system to perform its intended functions within a period of time [9].

The SIL determines the required failure rate of the given system. The international standard concerning E/E and programmable electronic devices are known as IEC 1508 [14][9]. The failure rates for the continues mode of operation, according to the standard, is stated in table 3.1.

Table 3.1: Required failure rate according to the Safety Integrity Level (SIL) according to the standard IEC 1508 [9].

SIL	Failure per year
4	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-2}$ to $< 10^{-1}$

With the arising of the standard ISO 26262 [4], safety integrity level for automotive vehicles has been assessed. The Automotive Safety Integrity Level (ASIL) is assessed from the hazardous outcome related to the failure of a system. Each hazardous event is assessed from the outcome based on severity of the injuries, amount of time the system is exposed to the possibility of the hazardous event of occurring, and the controllability that the driver can act to prevent any injuries. The hazardous event is then assigned an ASIL according to the ISO 26262 standard. The ASILs are classified A, B, C, D and QM. ASIL D is classified for the most severe hazards and A for least severe. Quality Managment (QM) is set to a system where there is no safety required. The ASIL is further explained in section 6 of this thesis.

3.5 High Reliability

This section elaborates the term of high reliability of a system that is used in this thesis. A system's failure rate is considered to be improbable if it has a failure rate lower than 10^{-9} failures/hour [13][9].

This thesis suggests a high reliability to be the same as the lowest reasonable failure rate for a simplex system. By calculating the system reliability for one year using an exponentially distributed model for a simplex system, a high reliability of 99.999% (five-nines) is reached. This is considered to be a reasonable value for a high reliability system.

3.6 Single-point and multiple-point faults

A *single-point fault* is the kind of fault that violates the safety goal of the system according to part 1 of ISO 26262 [4]. Therefore it is of importance to cover as many single-point faults as possible with safety mechanisms in order to prevent them from propagating into a failure. An example of a single-point fault in Figure 2.2 is if the communication bus gets disconnected. If the CCU cannot communicate with the ECUs, this will cause the whole system to fail.

A *multiple-point fault* is when the occurrence of multiple independent faults may violate the safety goal, according to part 1 of ISO 26262 [4]. The safety goal is violated when the system enters the state of critical failure. When there exists a safety mechanism that is designed to prevent a certain failure to violate the safety goal, this certain failure will be classified as; detected, perceived or latent. If the failure mode is detected by the safety mechanism it is classified as detected. If the failure is not detected by the safety mechanism but perceived by the user, it is classified as perceived. If the failure is neither detected or perceived, it is to be classified as latent. A latent fault does not directly cause the system to fail. However, in combination with the failure of another subsystem, the latent fault may become active and cause the whole system to fail, i.e., a multiple-point fault. It is stated in ISO 26262 that only two independent faults are to be considered for multiple-point faults unless they are shown to be of relevance.

3.7 Techniques for fault tolerance

To make a system more safe and reliable, safety mechanisms have to be added for the system to be able to tolerate certain faults, and prevent the system from propagating to a critical failure. This section describes different techniques for increasing the reliability of a system. Fault tolerance can be implemented in both hardware (HW) and software (SW). HW implemented fault tolerance can be made by utilizing *duplex-modular redundancy* or a *lock-step CPU* to detect a fault [9]. The combination of *triple-modular redundancy* and a *hardware voter* can be used in order to prevent a failure. Fault tolerance implemented in SW can be established by implementing *temporal redundancy*, *information redundancy*, *voting* and/or *forecasting* [9].

Temporal redundancy is when information is processed multiple times, in different aspects of time, and use voting to ensure that the system has not been introduced to any effects causing incorrectness. This adds protection against transient faults such as bit flips in a register etc.

Information redundancy is to store several copies to ensure that the information is not being altered. A voting algorithm, e.g. majority voting, is used to decide which value is the correct one.

The use of *duplex-modular redundancy* in HW means to provide a backup system to increase safety. The backup system can either work in hot-stand-by (backup system working in parallel), warm-stand-by (backup system getting check-points) and cold-stand-by (backup system starts when primary unit fails). Using *triple-module redundancy* is to provide two backup systems. If one of the systems should fail, the other two can still work as a duplex-modular redundant system. By adding a redundant backup system, the overall safety is increased. However, by introducing more components, the fail rate is also increased. An analysis has to be made to ensure that the fail rate does not increase more than the overall safety of the system. In that case, there is no need to add a redundant system in the first place.

A *lock-step CPU* is a fault-detection technique used to increase the safety of a system. Lock step uses several identical CPUs and compares the outcome of each CPU. If the outcome of one CPU differs from the other CPUs, this CPU is to be treated as faulty. If the outcome differs in a lock-step system with only two CPUs, it is still possible to say that a failure is detected but not which of the two CPUs that is incorrect. This technique can be used to prevent a fault from propagating to a failure. For example, if a fault occurs in one of the CPUs such as a bit flip in a memory cell or register, fault detection is made by the lock-step technique. Since the fault is detected, the system's safety mechanism can prevent system failure.

Voting in software corresponds to reading information multiple times or from different sources (providing the same information). The information is then analyzed using a voting algorithm to ensure that the correct, or most probable, value is returned. When using majority voting to

retrieve the correct value of an input, fault detection can be made if the majority of the values differ from each other. Fail prevention can be made only if a majority of the values are the same. For example, if two out of three values are the same, fail prevention is established.

In addition to the triple-modular redundant system, an external *hardware voter* can be used. The output from the three modules can be compared and the vote can occur instantly. A hardware voter adds to the system cost and complexity which may increase the total fail rate of the system.

Forecasting refers to predicting an output when input information differs. This can especially be good when checking the correctness of the value for fault detection. Another way can be to predict a correct value to avoid critical failures. This could be better than potentially sending bad information which can propagate to catastrophic events in the whole system.

3.8 Fault injection

Fault injection is a method for testing and verifying the robustness of a system, and to verify the effectiveness of the fault tolerant mechanisms. By introducing faults such as bit flips in registers or memory, one can analyse whether the system detects the fault or not. If a fault is detected, the system should be able to either prevent it from propagating or continue to a safe state. There exist mainly four types of fault injection techniques to test and verify the robustness of systems. These are *software-implemented fault injection*, *physical-implemented fault injection*, *radiation-implemented* and *simulation-based fault injection* [15].

In *physical-implemented fault injection*, two common techniques are pin-level fault injection and test port-based fault injection. Pin-level fault injection can be done by using a probe to force different signals to high or low on circuit level. In test port-based fault injection, test access ports are used to interact with the integrated circuit to inject bit flips into memory and registers.

Software-implemented fault injection can either be made on runtime or pre runtime. On pre runtime the instructions or predefined values are altered. Runtime software-implemented fault injections demands additional software which in its turn injects faults to the system.

Radiation-implemented fault injection can be done by exposing the system to electromagnetic interference or heavy-ion radiation.

Simulation-based fault injection is technique for making a model of the original system and testing it in a virtual environment. For example, when testing a power plant it is not wise to put the reactor in a potentially critical state when testing it. Instead simulation-based testing is used, where the system's environment is simulated.

3.9 Architecture designs

This section explains different architectures for a drive-by-wire system described in "On distributed control-by-wire system for critical applications" by Roger Johansson [13]. The three architectures explained are the local, centralized and distributed architectures.

3.9.1 Local

A local architecture is shown in Figure 3.2. It is the first generation of architectures and is simple for understanding, designing and implementing. Each subsystem contains its own sensor and electronic control unit (ECU) which is connected to the actuator. The disadvantages with

the local architecture is that sensors may need to be duplicated when information is needed in different controls, which is generally costly and not practical. From a safety perspective, the usage of a local architecture in a drive-by-wire system is unsafe since every subsystem is a single-point of failure. This means that if either one of the components shown in Figure 3.2 does fail, the drive-by-wire system as a whole fails.

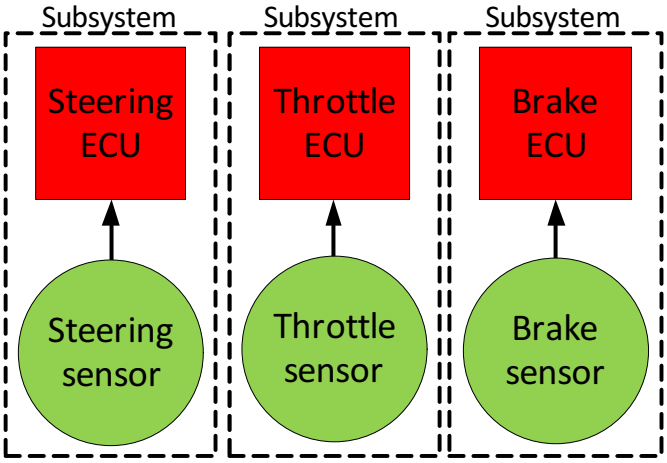

Figure 3.2: Local architecture: Each subsystem consists of one sensor and one electronic control unit (ECU).

3.9.2 Centralized

A centralized system is based on one common control unit that gathers all the sensor information and distributes it to the actuators. It is the second generation of architectures and is suitable for systems that have a natural safe state (e.g. the system automatically shuts down when a failure is detected). The advantage of the centralized architecture is that information from all sensors is available for processing and distribution to the actuators. When sensors from different sources are available for a control unit, their values can be combined to form a more general picture of the vehicles condition. For example, if the user apply both brake and throttle, there is no reason for the actuators to act accordingly since this can damage the vehicle. A centralized architecture for a drive-by-wire system is shown in Figure 3.3. From a safety perspective it is not recommended for safety-critical applications, since the architecture consists of several single-point of failures (e.g. control unit and communication bus). With the usage of additional software and hardware, it may be suitable only if it can be redundant enough so that safety criteria is met.

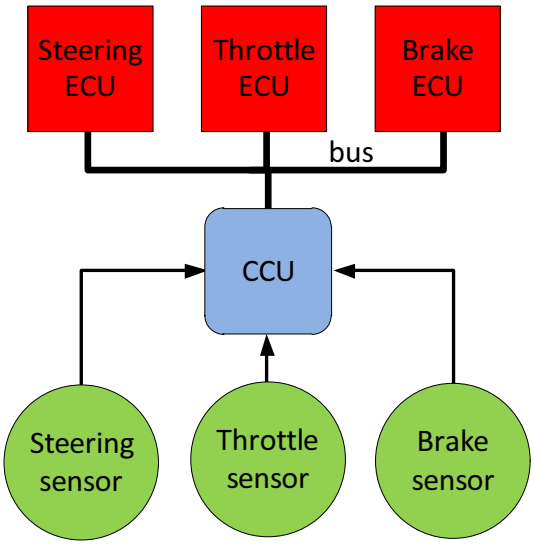

Figure 3.3: Centralized architecture: The system concists of one central control unit (CCU) for all the sensors and electronic control units (ECUs).

3.9.3 Distributed

The distributed architecture is the third generation of architectures and is shown in Figure 3.4. It is the most highly recommended architecture for safe-critical systems according to "On distributed control-by-wire system for critical applications" by Roger Johansson [13].

Compared to the centralized architecture, the distributed architecture divides functions into sub functions. This arises several redundancies and increases safety for a vehicle. One example is to have distribution in the brake system. When distributing the brakes into two electronic control units (ECUs), one ECU controlling the brake actuators on the front right wheel and back left wheel, brake system 1, and the other ECU controlling the brake actuators on the front left wheel and back right wheel, brake system 2. If brake system 1 fails, the vehicle still has degraded functionality in the brakes. When distributing brakes to all four wheels, degraded steering can be obtained when main steering system has failed. This gives additional redundancy to the drive-by-wire system in both the brakes and steering. The concept of steer-by-brakes is further elaborated in section 4.4.2.

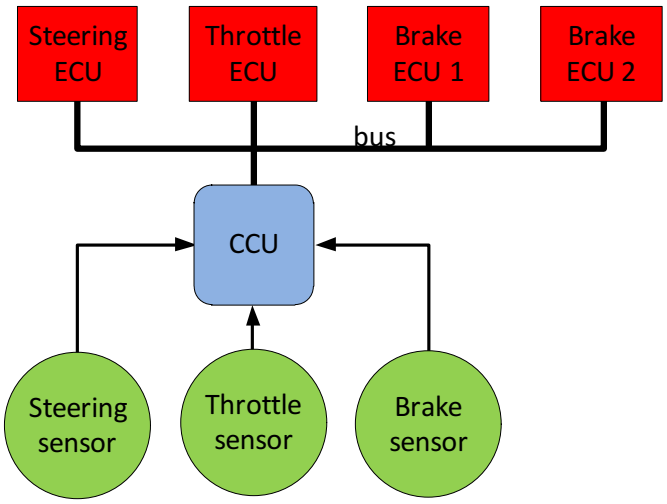

Figure 3.4: Distributed architecture: The brake ECU is distributed to control the brakes on each side of the vehicle.

4 Concept phase

Before implementing a system, this thesis follows the guidelines of the concept phase described in part 3 of the ISO 26262 standard [4]. Figure 4.1 shows the overview of the safety life cycle of the ISO 26262 standard. The dashed marked phases are not included in this report. This section deals with the concept phase of the safety life cycle.

First, an item definition is made where the reference system is described at functional level. Then the initiation of the safety life cycle is analysed, to determine if a new development should be initiated or a modification of the reference system will be made. A hazard analysis and risk assessment is then made of the reference system to determine the hazardous events. Each hazardous event is then classified to an ASIL according to the ISO 26262. Lastly, a functional safety concept is made to extract the safety goal of the developed system.

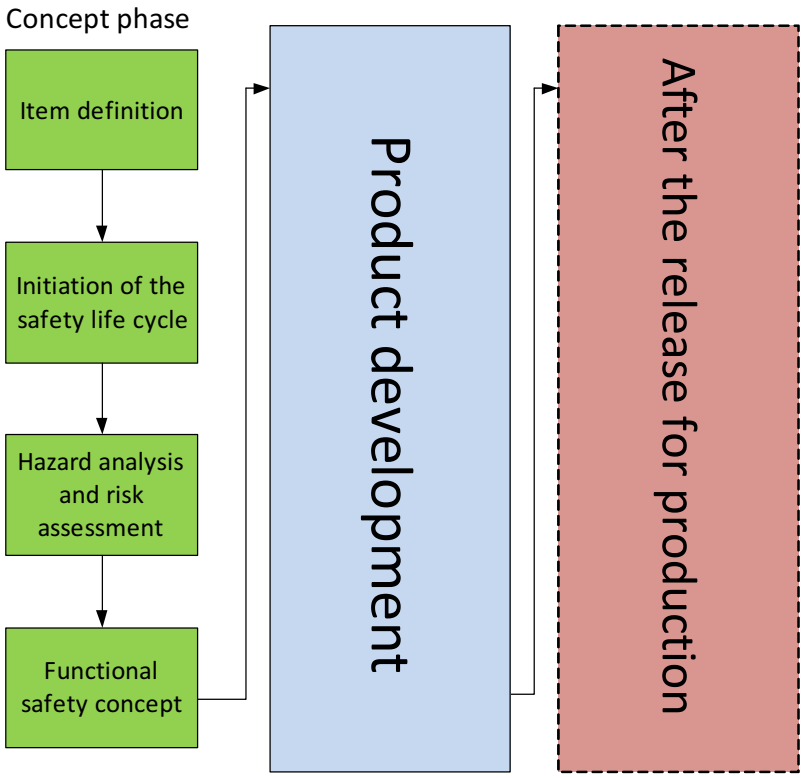

Figure 4.1: Overview of the safety life cycle for the concept phase. Phases marked with a dashed frame are not included in the thesis.

4.1 Item definition

An item is one or several systems that implement a function at vehicle level. An element refers to electrical and electronic (E/E) components of the system or other technologies[1]. An item is built of one or several elements. The items that constitutes the drive-by-wire system is: steer-by-wire, brake-by-wire and throttle-by-wire. These items and their functions are further described in Table 4.1. The functionality of the drive-by-wire system is considered failed if either of these items fail.

[1] other technologies refers to technologies that are out of the scope of the ISO 26262 standard, such as hydraulic or mechanical technologies [4].

Table 4.1: The items of the drive-by-wire system and description of their functionality.

Item	Function	Description of the intended function
Steer-by-wire	Steer control	Control the steering of the vehicle
Throttle-by-wire	Speed control	Increase the speed of the vehicle
Brake-by-wire	Speed control	Decrease the speed of the vehicle

4.1.1 Functional concept

This section provides information about the functional and non-functional requirements[2] of the different items. The purpose of the functionality is explained with a use case and the different operating modes are stated for the functions. The interaction between the elements of the items shown in Table 4.1 are explained, including the elements based on other technologies.

To understand the user's interaction with the system, a use case is developed. Figure 4.2 shows how the user interacts with the different functions of the drive-by-wire system. Since the user needs to be able to control the steering angle and speed regulation of the vehicle, the main functions: steer control and speed control, are defined. The steer control needs to be able to steer the vehicle right and left using the front wheels. The speed control needs to be able to increase and decrease the speed of the vehicle. For speed interaction the vehicle consists of a throttle and a brake pedal and for steering-angle interaction the vehicle has a steering wheel.

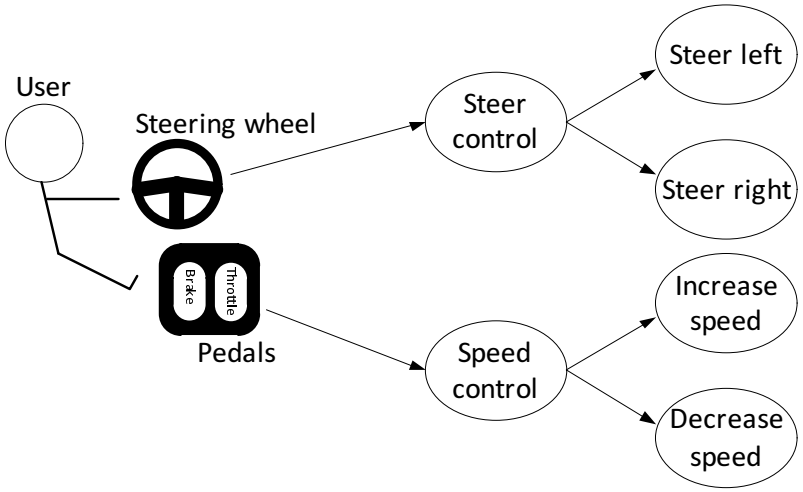

Figure 4.2: Use case of the items showing the different functions and how the user interacts with the functions.

The drive-by-wire system have to consist of the following elements, to be able provide the functionality described in Figure 4.2.

- Central control unit (CCU).
- Brake and throttle electronic control unit (ECU).
- Steer ECU.
- Steering wheel and pedals including sensors.
- Communication bus.

[2]non-functional requirement refers to criteria used to judge the operation of an item, rather than the behaviour [4].

- Brake, throttle and steering actuators.

The main purpose of the CCU is to convert the output values of the steering sensor, braking sensor and the throttle sensor into recognizable values and transfer them using the communication bus. Each ECU fetches the data from the communication bus of its interest. The ECU then outputs the value to its corresponding actuator which controls the functionality intended. The interaction between the different elements are shown in Figure 4.3.

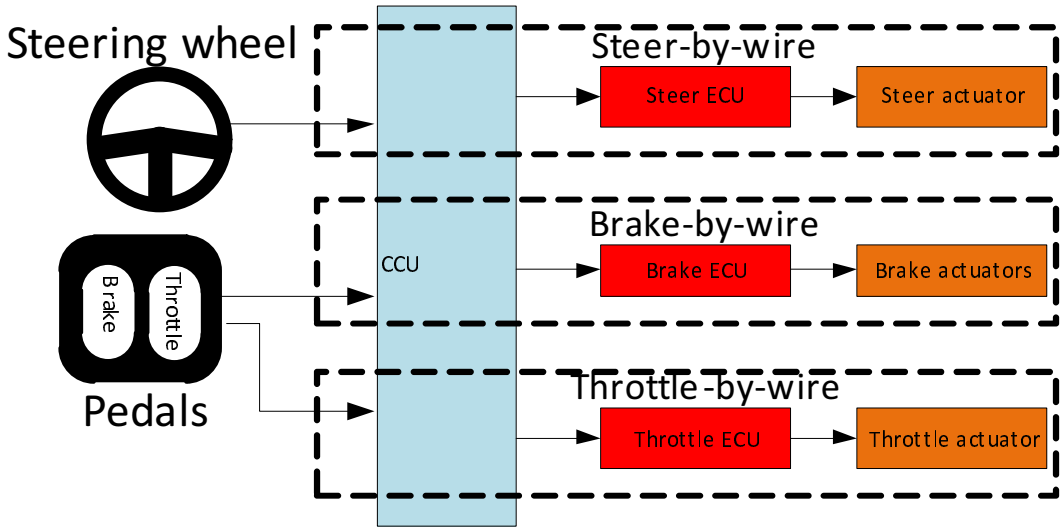

Figure 4.3: The interaction between the elements of the items; Steer-by-wire, Brake-by-wire and Throttle-by-wire.

The functional concept is only bounded to the functions of the drive-by-wire system. It should be mentioned that the prototype of the drive-by-wire system does not include vital elements such as power supply of the vehicle, real-sized actuators and a full sized engine or reasonable sensors for steering wheel and pedals. Considerations regarding these elements are to be made in real implementation of the drive-by-wire system.

4.1.2 Usage in different environments

An analysis regarding how the environmental factors contributes to the occurrence of hazardous events are to be established. In order to do so, different environments have to be specified and evaluated regarding the probability of exposure. Explanation of the environments the drive-by-wire system is considered to be frequently exposed to is shown in Table 4.2. The frequency is described in the terms often or seldom according to part 3 of the ISO 26262 standard [4]. Often refers to situation occurs during almost every drive on average. Seldom refers to the situation occurs a few times a year for the great majority of drivers. The impact refers to the impact the environment has on the drive-by-wire system. Low refers to no or very small impact and high refers to significant impact on the system. The impact on the drive-by-wire system is high only in manoeuvre situations. Manoeuvre situations is therefore the considered environment when evaluating the risk in section 4.2.

Table 4.2: Explanation of the environments the drive-by-wire system is considered to be exposed to.

Environment	Examples of situation	Frequency	Impact
Road layout	Highway, secondary road, country road.	Often	Low
	Highway exit ramp, intersection	Seldom	Low
Road surface	Dry or wet road, asphalt.	Often	Low
	Packed snow and ice, slippery leafs on road, gravel.	Seldom	Low
Manoeuvre situations	Accelerating, decelerating, steering vehicle	Often	High
	Driving in reverse, parking.	Seldom	High

4.2 Hazard analysis and risk assessment

Based on the item definition, a hazard analysis and risk assessment (HARA) is made and safety goals are established. The HARA determines the potential hazards of an item and safety goals can be determined. First, the classification of the impact factors; *severity*, probability of *exposure* and *controllability* are elaborated for the item. From these factors an ASIL of the hazardous events can be determined. Then a safety goal can be assigned for the item according to ISO 26262 [4].

Severity is the classification of the injuries caused by the hazardous event on the driver, other road users or pedestrians. Severity has the classifications S0-S3 where class S0 refers to no injuries and S3 refers to life-threatening injuries caused by the hazardous event. Table 4.3 explains how different classifications are interpreted according to the ISO 26262 standard [4].

Table 4.3: Description of the injury impact for the different severity classification S0-S3.

Classification	S0	S1	S2	S3
Description	None	Light	Severe	Fatal

Exposure is the probability of exposure for the system or subsystem, i.e has a low exposure that can cause the hazardous event. For example if a system is rarely used, i.e has a low exposure, the probability of the hazardous event occurring in that system is low. Exposure has the classifications E0-E4, where E0 is a system that has an incredibly low exposure and E4 a high probability of exposure. E0 could for example be a system that is never used (e.g a system considered to be used for future use), the exposure of that system is then defined as incredible. An exposure of E4 means that the system is used almost every drive on average, e.g systems for steering, throttle or brake. Table 4.4 shows how the different classifications are interpreted according to the ISO 26262 standard [4].

Table 4.4: Description of the probability of exposure of the systems that can cause the hazardous event. The exposure is classified in E0-E4 and determination is based on operational situation of the target.

Classification	E0	E1	E2	E3	E4
Description	Incredible	Once a year	Few times a year	Once a month	Every drive

Classification of the *controllability* is the plausibility that the driver, or other person at risk, can gain sufficient control to avoid the hazardous event from happening. Controllability is classified as C0-C3 according to Table 4.5. A controllability classification of C0 is for example when the system causing the potential hazardous event goes into a safe state, e.g some driver assistance

systems can for instance just be turned off. A controllability classification of C3 is when driver cannot avoid the hazardous event from happening due to the system failure, e.g the brake system stops working. Assumptions of the controllability are decided according to ISO 26262 part 3 [4]. It is stated that the driver is considered to be in appropriate condition (e.g not influenced by drugs or tiredness), has proper driver training, normal physical health and reaction and complying with legal regulations. Controllability also include the avoidance of pedestrians' (or other road users') actions to avoid the hazardous events.

Table 4.5: Description of the controllability of the drivers or other persons at risk to avoid the hazardous event. The controllability is classified in C0-C3.

Classification	C0	C1	C2	C3
Description	Controllable in general	Simply	Normally	Difficult or uncontrollable

4.2.1 Hazard identification and classification

This section identifies the hazardous events that can occur in case the item fails. To decide the ASIL of different hazards that may occur in the functional level of a system, a function and failure analysis (FFA) is made.

The FFA is shown in Table 4.6. The basic functions of the item with different class of failures, i.e., *omission*, *commission* and *stuck-at value*. *Omission* means that the function does not return a value, i.e., fail silent. *Commission* means that the function returns random values, sometimes referred to as babbling idiot. *Stuck-at value* means that the functions returns a constant value. For each function a worst case environment and worst case failures are determine and the different classifications from the hazardous events are set. The worst case environments are either slow driving in dense traffic area or high speed traffic such as a highway. The classification are: severity (S), probability of exposure (E) and controllability (C). From the classifications the ASIL are extracted using table 4.7.

Table 4.6: Hazard identification and ASIL determination based on the class of failures of the worst case environments and failures.

Function	Class of failure	Worst case environment/failure	Classification	ASIL
Accelerate	Omission	Railroad crossing / No acceleration	S3,E1,C2	QM
	Omission	Highway / No acceleration	S1,E4,C1	QM
	Commission	City (traffic light) / Sudden acceleration	S3,E4,C3	D
	Stuck at value	City / Constant acceleration	S3,E4,C3	D
Brake	Omission	City / Loss of brake	S3,E4,C3	D
	Commission	Highway / Sudden brake	S3,E4,C3	D
	Stuck at value	Highway /Constant retardation	S2,E4,C2	B
Steering	Omission	Highway / Loss of steering	S3,E4,C3	D
	Commission	Highway /Sudden steering angle	S3,E4,C3	D
	Stuck at value	Highway / constant steering angle	S3,E4,C3	D

Table 4.7: ASIL determination according to part 3 of the ISO 26262 standard [4].

		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

4.2.2 Safety goal

For each hazardous event with corresponding ASIL, a safety goal is defined in this section. The hazardous events with their corresponding ASIL is shown in table 4.6. If information to the actuators cannot be provided correctly, the FFA presents which default message should be sent in case of system failure for the different functions. Accelerate has least severity when in omission. The least severity of brake and steering is when stuck at value. Brake function should be able to override the throttle functions. When accelerate has a commission or stuck-at-value failure, the driver should be able to brake to avoid injury.

When the functionality is in degraded mode, i.e., one wheel unit has failed or steering has failed, the driver should be notified to stop the vehicle. This will sufficiently lower the risk of another failure occurring compared to having the driver driving around in a vehicle with degraded functionality. The fault tolerant time interval should be sufficient time for the driver to slow down and pull to the side of the road. The fault tolerant time interval is the time in between the occurrence of a fault and a possible hazard [4].

The safety goal is to ensure that the system will not fail so that it causes a hazard. Therefore these safety goals have been established for the item:

- The brake should be able to override throttle function.
- The brake *functionality shall not fail.*
- Driver should be warned when at degraded functionality.
- The throttle system needs to be fail silent in case of failure.
- The steer system needs to be stuck at value in case of failure, enabling steer-by-brakes functionality.

When stated *functionality shall not fail*, i.e., the system needs to fulfill the the highest possible reliability in these cases.

4.3 Functional safety concept

After the safety goal is determined for each item, detailed safety requirements are derived to achieve the safety goals. The safety requirements inherit the ASIL corresponding to the safety goal. Safety goals can be achieved by combining different functions to put the driver in a safe state. For example a failure that causes full throttle can be overridden by brakes if brakes is given higher priority than the throttle. From the safety goals explained in section 4.2.2 and the ASIL from Table 4.6 in section 4.2.1, the safety goal can be achieved by:

- Using a safe state (e.g switching off the system causing the hazardous event in case of failure).
- Consider the fault tolerant time interval (e.g a warning lamp stating to pull over to the side of the road). Reducing the exposure time in a critical event - where a hazard may or may not occur.
- Including maximum values for unwanted changes in system. For example speed limit in degraded mode.

The safety requirements are determined for the different systems of the functions. The different systems are then connected to the different elements explained in Figure 4.4. The safety requirements that are supposed to be implemented in E/E are:

- Brake-overriding-throttle function in the central control unit (CCU).
- Distributed brake system in the CCU and distributed brake ECUs.
- Redundancies in both hardware and software for the CCU.
- Steer-by-brakes compatibility for the brake ECU.
- Setting steering at forward position in degraded mode to enable steer-by-brakes.
- Notification to the driver, in case of failure, to inform that the vehicle is in degraded mode.

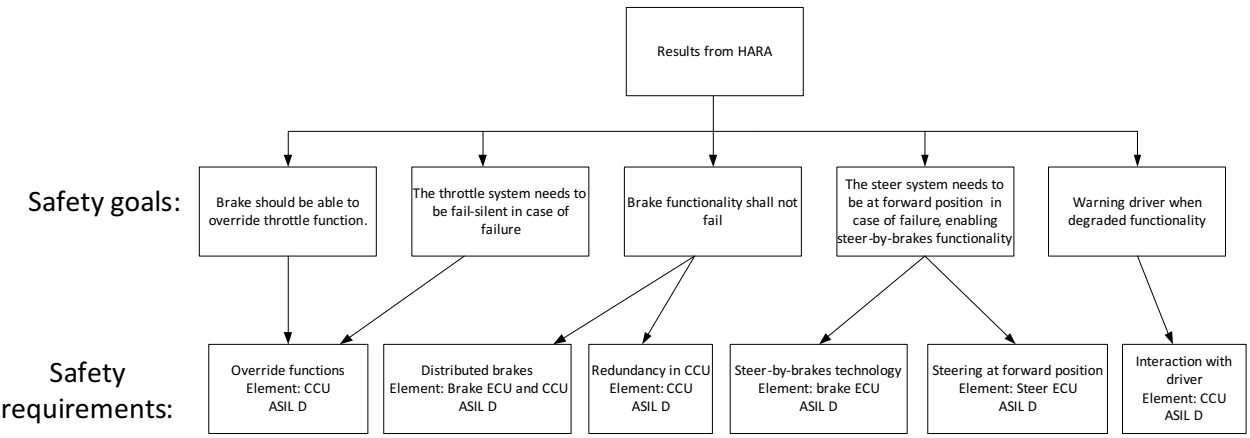

Figure 4.4: Safety requirements with their corresponding element, determined from the safety goals.

The design decisions drawn from the safety goal is explained in section 5. The functional safety is further elaborated in section 6.

4.4 Modifications to the reference design

The reference drive-by-wire system with no fault-tolerance, explained in section 2, is modified for developing a system that is suitable for fault-tolerance. This section explains the modifications done to the reference drive-by-wire system and the impacts of these modifications.

4.4.1 Distributed brake system

The reference system only contains one brake ECU. When implementing a distributed architecture the system can sustain a higher reliability since the distributed brakes are independent of each other. Therefore a modification has been made to use a distributed brake system for the drive-by-wire system. The distributed brake system will use two ECUs to control two diagonal actuators each.

4.4.2 Steer-by-brakes

When distributing the brake system, a technology for emergency steering can be implemented. A steer-by-brake [16][17] functionality will enable the driver to, in a very limited way, turn the car left and right using the steering wheel and the distributed brakes. This will be a critical backup system if the steering system suffers from a critical failure.

The distribution of the brake ECU into two ECUs that controls two actuators, makes it possible to control the amount of brake impact on each side of the vehicle separately. By applying brake force to both wheels on one of the sides of the car, there will appear a difference in rotational speed between the both sides. This difference in rotational speed will generate a torque around the center of the car, making the car slightly turn towards the side on which the brake force are being applied. In case of a failure in the ECU responsible for controlling the steer actuator, it is still possible to stop the car at the side of the road by utilizing the distribution of the brakes. This assumes that both brake ECUs are functional.

Another way of utilizing the distribution of the wheels to maintain steering capability, in case of a failure in steer ECU, is to apply brake force on one of the front wheels. According to [18], this will generate a yaw moment on the front axis which will enable to control the steering angle of the front wheels. This does however assume feedback of the steering angle. It does also require that both brake ECUs and all brake actuators are functional.

4.4.3 Impact analysis

When modifying the reference system with a distributed brake system and steer-by-brake functionality, changes are made in the functionality of the system. The two modifications are important to increase the safety of the drive-by-wire system. Figure 4.5 shows the modifications to the use case (compare with Figure 4.2) and Figure 4.6 shows the modification to the interaction diagram (compare with Figure 4.3). The steer control is now able to control the vehicle via traditional steering (i.e controlling the angle of the front wheels) and with distribution of the brakes (i.e steer-by-brakes).

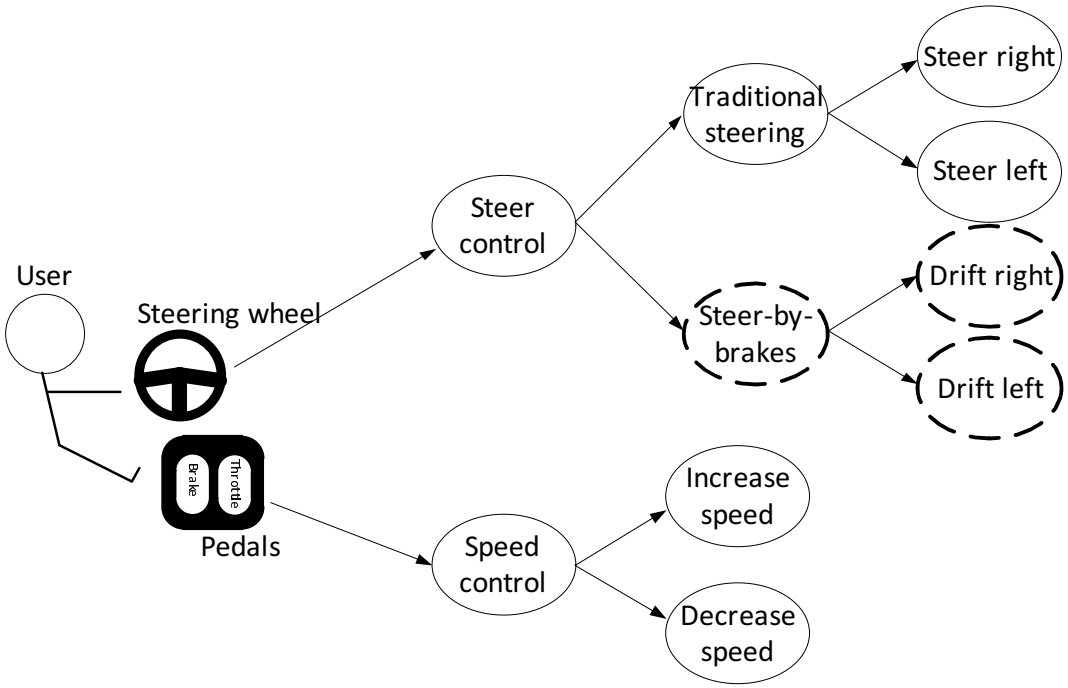

Figure 4.5: Use case of the modified items explaining how the user interact with the modified system. The dashed lines indicates that the steering functionality is a backup mode, only used if the primary steering fails.

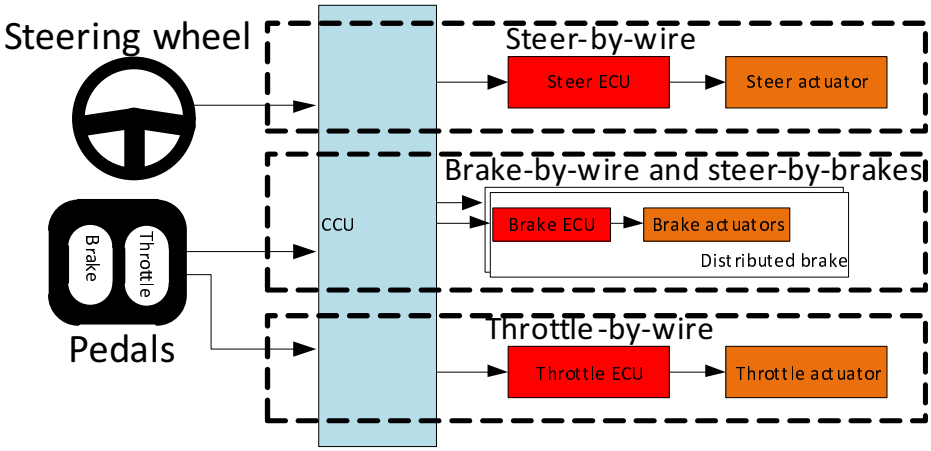

Figure 4.6: The modified items' interaction with the different subsystems and its corresponding elements.

The modifications enables the steering and brake subsystem to be operational in degraded mode. The three different modes are explained below for the distributed brake system regarding brake functionality.

- Full functionality - Both distributed brakes are operational.
- Degraded functionality - One of the two distributed brakes are functional.
- System failure - Both distributed brakes have failed.

The three different modes for the steering functionality are explained below:

- Full functionality - Traditional steering unit is operational.

- Degraded functionality - Both distributed brakes are operational.
- System failure - Primary unit and at least one of the distributed brakes has failed, steering capability can no longer be ensured.

The two operations complement each other. If the brakes are at degraded functionality, the system can warn the driver of degraded functionality of the vehicle. If the steering fails the system can warn the driver for the degraded mode.

5 Product development

This section explains the redundant strategies made for the development of the central control units (CCUs). It follows the product development phase of the ISO 26262, part 4, 5 and 6 [4]. The safety life cycle of the product development is shown in Figure 5.1. The development follows a top-down approach, starting from development at system level and branching down to the implementation at hardware and software level.

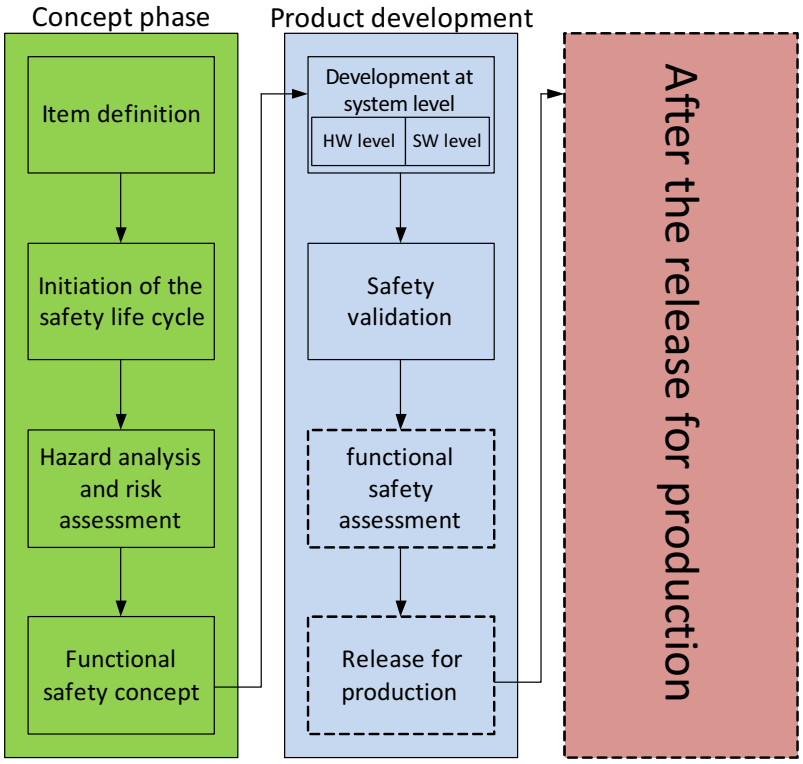

Figure 5.1: Overview of the safety life cycle for the product development. Phases and subphases marked with a dashed frame are not included in the thesis.

5.1 System design

The reference system is using a centralized architecture, where the central control unit (CCU) controls the ECUs for brake, throttle and steering. The architecture design for the reference design is shown in Figure 5.2. Notice that the brake ECU controls all the actuators (marked with A in the figure). Since the architecture of the reference design is not an efficient way to introduce redundancy, a distributed architecture is developed in this thesis, explained in section 4. A system overview of the distributed architecture design is shown in Figure 5.3. The architecture uses separated brake ECUs to control one actuator on each side of the vehicle. Brake ECU 1 control the actuators A1, and brake ECU 2 control the actuators A2 (seen in Figure 5.3).

The weakest link of the architecture is the central control unit (CCU) since it has the highest complexity and is therefore considered to have the highest fail rate in the system. The sensors and CAN bus is out of range for this thesis and is therefore considered ideal in the sense that they are immune to faults.

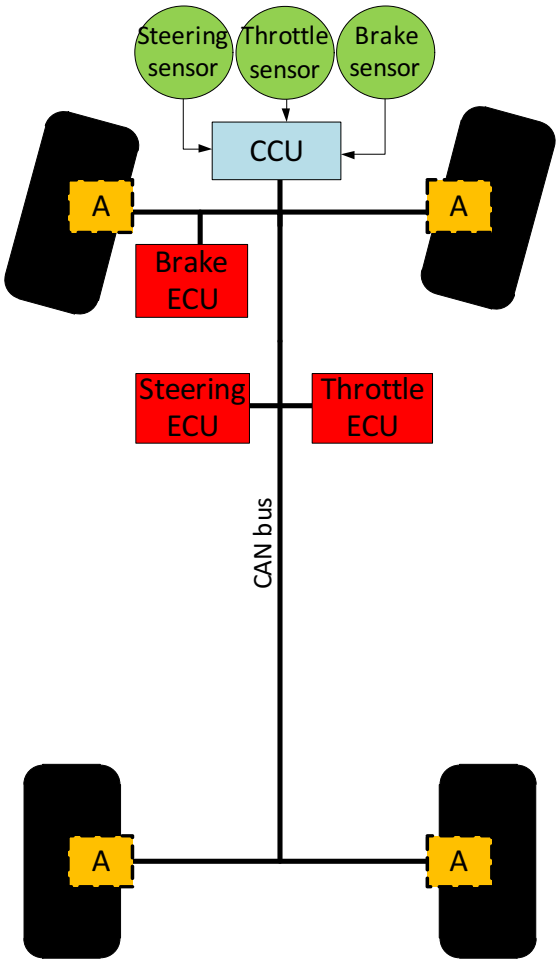

Figure 5.2: Centralized architecture for the drive-by-wire used in the reference design.

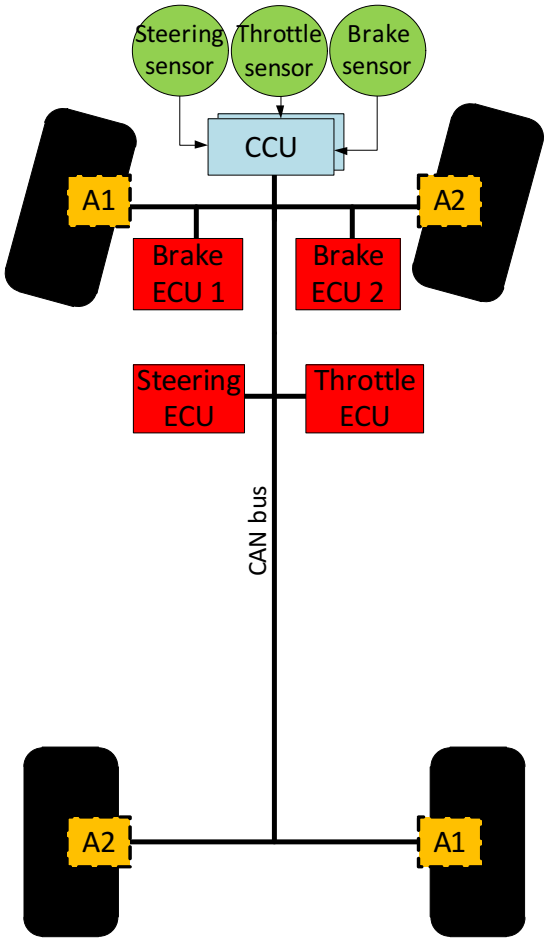

Figure 5.3: Distributed architecture used in the fault-tolerant design. Brake ECU 1 controls the actuators A1 and Brake ECU 2 controls actuators A2.

5.2 Hardware design

This section gives an overview of the hardware (HW) design of the product development. The hardware design deals with the implementation of hardware parts to increase the reliability of the fault-tolerant design.

5.2.1 Duplex-modular redundancy for central control unit

The hardware needs to be designed so that it has a failover system to ensure safety if the CCU has a permanent error that results in a failure. The system will run in duplex mode where there will be two CCUs having the same safe-critical functions for the drive-by-wire system. One of the CCUs will act as main (or primary) CCU while the other CCU will run as backup.

The two CCUs can either work in hot-standby, warm-standby or cold-standby. Since the processed sensor values for the ECUs are time critical, and resource utilization of the CPU is not an issue in the referenced design [5], decision is made to use hot-standby failover in the design. The backup CCU will work in hot-standby mode to take over the functions of the primary CCU if the primary CCU fails.

5.2.2 Error detection

The development boards chosen for implementation of the CCUs already have error detection by the implementation of a lock-step CPU. The lock-step CPU can detect both permanent hardware faults and temporary faults occurring in the design which the software cannot handle. The lock-step technique can be used to prevent a fault from propagating to a failure. Lock-step and error detection is elaborated in section 3.7.

5.2.3 Reset handling

When a temporary error is detected in a central control unit (CCU), it will have to be restarted in order to guarantee further safe execution. If the primary CCU has a permanent error resulting in a failure, the secondary CCU will have to take over. The reset handling refers to the the design of how the active CCU will be restarted if an error is detected. Active CCU can be referred to both primary CCU and secondary CCU, depending on which CCU that is currently executing.

This section presents two designs for restarting a CCU; *distributed-reset design*, shown in Figure 5.4 and *self-reset design*, shown in Figure 5.5. The CCUs have a monitoring function (hereinafter referred to as monitor) to detect when an error occurs in the CCU. If an error is detected, the monitor will restart the erroneous CCU using a power switch located on the CCU. If the CCU has a permanent failure, i.e., the error still appears after a restart, the secondary CCU will take over and the primary CCU will be shut off.

In the *distributed reset design*, shown in Figure 5.4, the monitor is implemented in software (SW) of the other CCU. The SW implemented monitors enables the CCUs to reset each other when an error occurs by controlling the reset signal of its neighbours. This can be implemented in two ways. Either each and every CCU gets the status of the error pin and the reset signal of all of the other CCUs. Or each CCU gets the status of the error pin and the reset signal of the two closest CCUs, shown in Figure 5.4. If this design is to be used in a system with several CCUs, this demands a huge number of interconnects to enable reset handling between all CCUs.

In the *self-reset design* shown in Figure 5.5, each CCU will have an external hardware (HW) monitor. This increases the cost and design complexity of the system when making a safe implementation of the HW monitor. The self-reset design is suggested for systems where more than two CCUs holds the safe-critical functions. The benefits of using this design is that it is easy to further introduce more CCUs that runs the safe-critical functions.

Even though the distributed-reset design seems like the cheapest alternative for this project where only two CCUs is to be used, the self-reset design is better from a safety perspective. Since the monitor have the functionality to permanently shut down the CCU, it might occur that a failure in the distributed self-reset monitor implemented in one of the CCUs causes the other CCU to permanently shut down. This will result in a system failure since there will be no functional CCU. However, if the same error occur in the case of the self-reset design, the primary CCU will not be affected if the monitor of the secondary CCU forces it to shut down, since they are independent of each other.

The monitor elaborated in this section is not included in the CCU and will be implemented in this thesis. The implementation of the self-reset monitor is further explained in section 7.

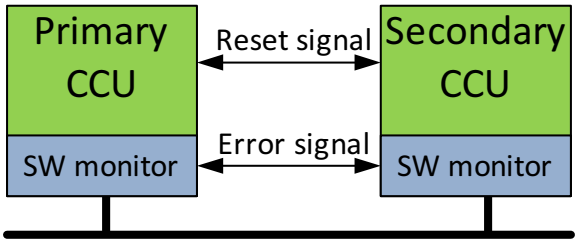

Figure 5.4: Distributed-reset design for the central control unit (CCU) where the monitor is implemented in software (SW).

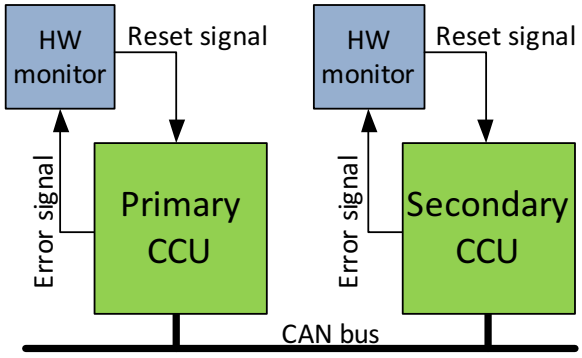

Figure 5.5: Self-reset design for the central control unit (CCU) using a hardware (HW) monitor for resets.

5.3 Software design

The next step is to design the software (SW) to ensure redundancy and safety in the CCU. The SW design will be implemented in the CCU to handle the specific drive-by-wire function to enable safety for the driver. The system needs to include both redundancy in time (temporal redundancy) and information redundancy. Table 5.1 presents different methods for error detection at software level [4]. The recommendation for each method is graded dependently of the current ASIL of the item it is implemented in. "o" means that the method has no recommendation either against or for the usage for the current ASIL. "+" means that the method is recommended for the current ASIL. "++" means that the method is highly recommended for the current ASIL. The ISO 26262 standard recommends that one or several of the methods in Table 5.1 are used when developing the software design.

Table 5.1: Mechanisms for error detection at software level. [4]

Method	A	B	C	D
Range checks of input and output data	++	++	++	++
Plausibility check	+	+	+	++
Detection of data errors	+	+	+	+
External monitoring facility	o	+	+	++
Control flow monitoring	o	+	++	++
Diverse software design	o	o	+	++

The methods for *range checks of input and output data* and *plausibility check* were chosen to be part of the software design. An overview of the software design is shown in Figure 5.6.

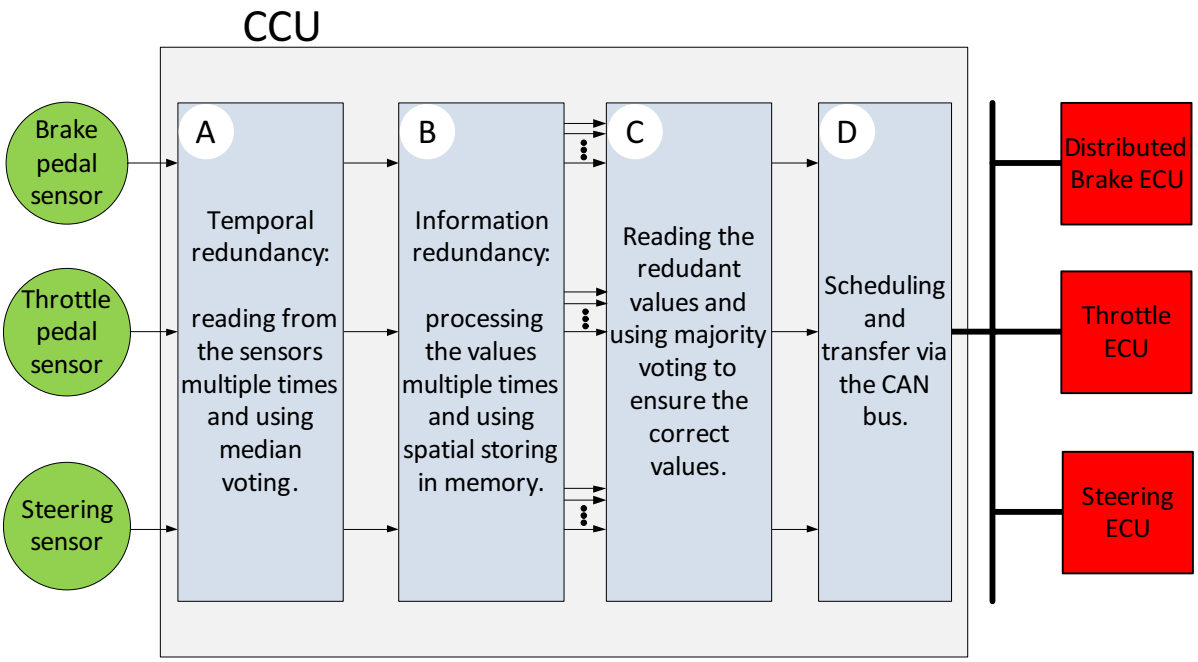

Figure 5.6: Software design of the safety mechanisms in the central control unit.

In block A the data is read from the sensors. A *range check* will be made for each reading to ensure that the value is in range. By reading the values several times, temporal redundancy is accomplished. By separating each read with a short time gap it is possible to detect whether the data is continuous or not. For example if the sensor cable is broken, the data will most likely be affected by high frequency noise. If the data varies to much in between each read, it will be seen as faulty. A *plausibility check* is made by comparing the values with each other using a median voter. If the measured values differ too much from each other and no median value can be decided, the system will restart or hand over to a backup system.

Block B processes the information from the correct sensor value provided by the reference block. It takes the sensor value, processes it several times and stores it in multiple places to ensure information redundancy. If a bit flip, caused by e.g. ionizing particles, occurs in one of the memory cells where the sensor values are stored, it will most likely affect other memory cells which is located physically close to it. Therefore it is a good idea to physically separate the memory location of each sensor value, to minimize the event of one bit flip altering the data of several sensor values.

In block C the multiple values are retrieved from the memory. The multiple values stored in the memory are read and majority voting is made to ensure their correctness. If majority of the values from the memory are the same, the majority voter will ensure fault prevention by providing the correct value. If majority voting cannot be done due to majority of the values are different, the majority voter will ensure fault detection to prevent a system failure.

Block D handles transfer of the values to the ECUs. A *range check* of the output values is made before sending them to the ECUs, to ensure that the values are in acceptable limit.

When adding the possibility to restart a CCU, considerations regarding state dependent information have to be taken into account. The decision of using a duplex CCU adds the possibility of retaining such information after a restart since the other CCU will hold the correct values. If primary CCU have to be restarted, secondary CCU will continue execution and update eventual state dependent information. In the startup phase of the CCU, there will have to be some functionality responsible of synchronizing such state dependent information.

In case of a system startup, both CCUs shall assume default values. However, some safety mechanism needs to be implemented to ensure that no CCU assumes default value when the system is actually running.

6 Safety validation

The safety validation is a part of the ISO 26262, shown in Figure 5.1 in section 5 of this thesis. This section aims to verify the reliability and safety of the developed design explained in section 5 of this thesis. The developed design is hereinafter known as the *fault-tolerant system*.

Section 6.1 gives an overview of the reference design and the fault-tolerant design in a fault tree diagram. Each system is then analysed to show the single point of failures in the two systems.

In section 6.2, a comparison between the reliability of the two systems is made. An analysis is made to show how the reliability of the two systems changes in time. The systems are modelled using Markov chains. The section also gives a detailed analysis of the reliability for the fault-tolerant system.

Section 6.3 makes an analysis of the safety for the fault-tolerant system. The steady-state safety is calculated for each subsystem. Safety refers to the system's ability to prevent failures that can lead to a catastrophic event [19]. The steady-state safety is the probability that a fail-safe state is reached, i.e., catastrophic failure is prevented.

6.1 Overview

The fault tree for the reference system is shown in Figure 6.1. The fault tree is elaborated from the centralized architecture, shown in Figure 5.2. The fault tree shows all the single point of failures in the reference system. Engine and power failures are marked with a yellow triangle which indicates that there are underlying fault trees that needs to be further investigated. This is however, not included in this thesis.

The fault-tolerant system is shown in Figure 6.2. The fault tree is elaborated from the distributed architecture, shown in Figure 5.3. The red triangle marked with steering/brake failure indicates that there is an underlying subsystem that may cause a failure. When introducing the functionality of steer-by-brakes, failure of the steering system becomes dependent of the brake system. For example if one of the brake ECU fails, steering functionality can not be in degraded mode. Therefore, failure of the steering ECU and one brake ECU will put the drive-by-wire system in critical failure. Since this occurrence can not be represented by the use of a fault tree, it will be further analysed using a Markov model in section 6.2. The red triangle of brake and steer will be referred to as brake-and-steer system.

In addition to the steering and brake redundancy, the fault-tolerant design uses duplex CCUs which adds to the fault tolerance of the system. For example, if one of the CCUs would fail, the system would still be functional as shown in the fault tree in Figure 6.2. Improvements have also been made to ensure safety of the brake-and-steer system.

The subsystems regarding the engine, power and sensors failure, shown in figures 6.1 and 6.2, are not further analysed in this thesis. They are mentioned to make the reader aware of other critical parts of the drive-by-wire system.

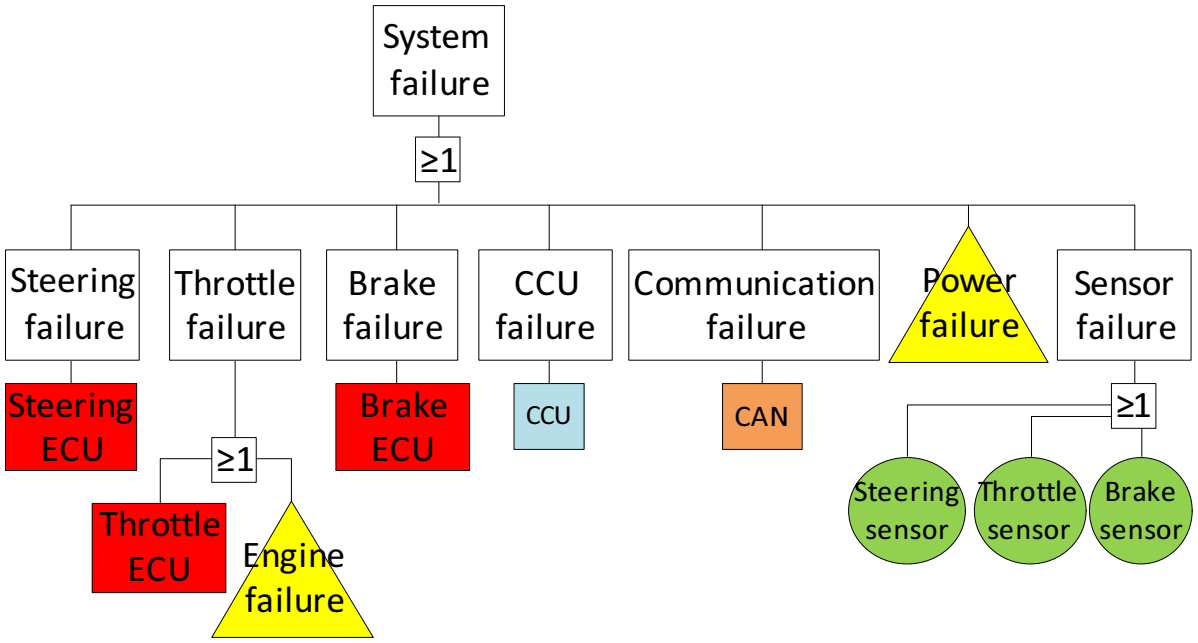

Figure 6.1: Fault tree of the reference system. The yellow triangles indicate that there are underlying fault trees.

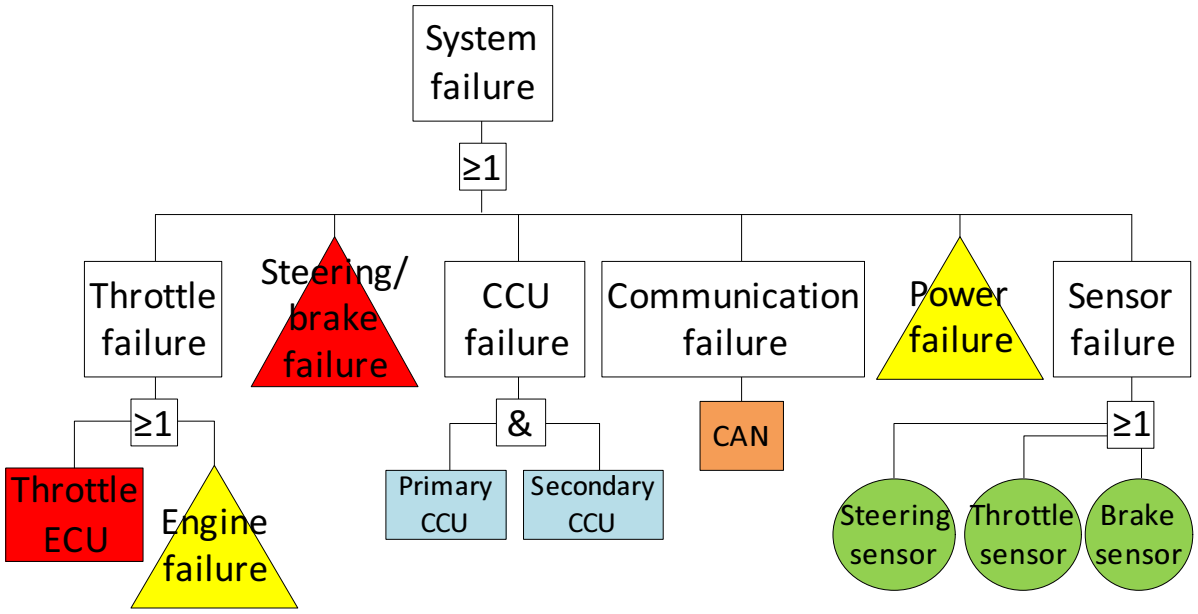

Figure 6.2: Fault tree for the fault-tolerant system. The yellow triangles indicate that there are underlying fault trees. The steering/brake failure is analysed in section 6.2.

6.2 System reliability

Reliability is the system's ability to perform its intended function on demand without failure [19]. Since the reference system does not have any degraded state, both systems are considered to be functional until failed.

This section gives a comparison between the reliability of the reference system and the fault-tolerant system and shows how the reliability decreases over time for both systems. The com-

parison is made for the central control unit (CCU) and the brake-and-steer system. Since the CCU and brake-and-steer system are independent of each other, a model can be extracted using exponentially-distributed continuous-time Markov modelling [9]. A Q-matrix is extracted from the Markov model and the ordinary differential equations (ODE) are set. The ODE are then solved using Laplace transforms and MATLAB for finding the solution of the ODE under a given time.

The failure rates of the systems are estimated from the Safety Integrity Level (SIL) of the standard IEC 1508 (see section 3.4) and the quantitative analysis from the ISO 26262 standard [4] (see section 3.4). The lifespan of a vehicle is estimated to be 10 years according to studies made by [20] and [21].

6.2.1 Central control unit

The design of the CCU in the reference system uses a simplex design. The Markov model is shown in Figure 6.3. A failure of the CCU will result in a system failure.

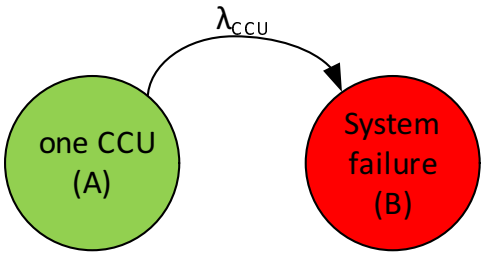

Figure 6.3: Markov model of the central control unit (CCU) for the reference system. λ_{CCU} is the failure rate for the CCU.

The Q-matrix extracted from the Markov model is shown in Equation 1a. The extracted ODE from the Q-matrix is shown in Equation 1b. Equation 1c shows the Laplace transform to the s-domain of the ODE. The expression is then transformed back to the time domain using inverse Laplace transform and the probability of being in state A according to Figure 6.3, is expressed. The calculated reliability of the reference system's CCU is shown in Equation 1d.

$$Q = \begin{bmatrix} -\lambda_{CCU} & \lambda_{CCU} \\ 0 & 0 \end{bmatrix} \tag{1a}$$

$$P'_A(t) = -\lambda_{CCU} P_A(t) \tag{1b}$$

$$\mathcal{L} \Rightarrow P_A(s) = \frac{1}{s + \lambda_{CCU}} \Rightarrow \mathcal{L}^{-1} \Rightarrow P_A(t) = e^{-\lambda_{CCU} t} \tag{1c}$$

$$R_{CCU} = P_A(t) \Leftrightarrow R_{CCU} = e^{-\lambda_{CCU} t} \tag{1d}$$

The current CCU design uses a duplex system, and the Markov model is shown in Figure 6.4. In each state, there is a number combination. The left number shows how many working CCUs are in the state, the middle number shows how many CCUs are under repair (under restart) and the rightmost number shows how many CCUs are in a non-reparable state (permanently damaged).

The initial state is state A shown in Figure 6.4. In state A both CCUs are functional. If a transient or intermittent fault occurs and is covered by the system, transition to state B will occur. State B indicates that there is one working CCU and one under repair. The CCU under repair is restarted by the hardware monitor and a transition back to A will occur in the repair ratio, μ_r. The repair ratio is equal to the $(restart\_time)^{-1}$ of the CCU, the restart time is estimated to be approximately 1 ms. The system covers both transient and intermittent faults. Since the repair of a CCU uses power-off restart, an intermittent failure will continuously restart the system until the fault disappears. The restart of the system is further elaborated in section 7.

If another failure occurs to the working CCU under the time when the first CCU is restarting, transition from state B to state D will occur. Since both CCUs have failed, the whole system is considered as failed. Both CCUs can also be exposed to a non-covered fault. This means that the system will transit from state A to state D, i.e system failure. A non-covered fault is not detected by the system and will therefore not be restarted by its hardware monitor. The hardware monitor can only restart the CCU when a failure is detected. If a fault that alters the system from its correct behaviour occurs, transition to state D will occur.

The CCUs can also be exposed to a permanent fault, i.e fault that is not repairable, e.g a broken transistor in the CCU or fault in the hardware monitor. If the permanent fault is covered, transition from state A to C will occur. In state C one CCU is working and the other one is detected as permanently damaged and shut of by the hardware monitor. In this state the system has no repair-rate since a restart of the CCU can not repair the permanent damage. If another fault occurs to the working CCU, transition to from state C to D occurs.

In Figure 6.4, λ_t is the transient failure rate and λ_p is the permanent failure rate for one CCU. The repair rate is referenced as μ_r for one CCU, c_t is the coverage for transient and intermittent faults and c_p is the coverage for permanent failures.

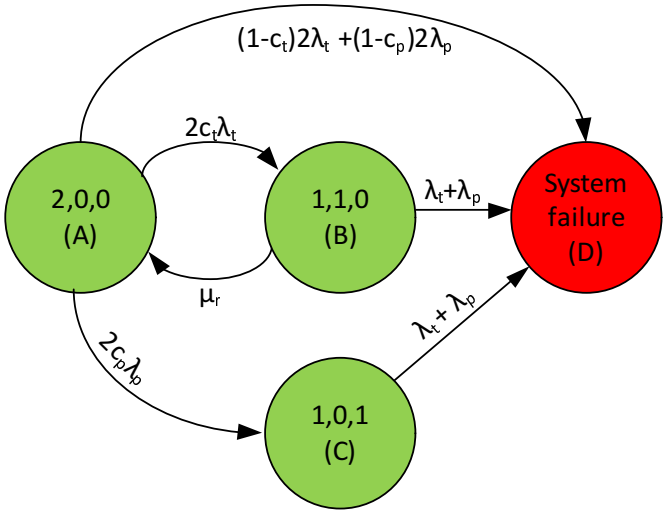

Figure 6.4: Markov model of the central control units (CCU) for the fault-tolerant system.

Equation 2a shows the Q-matrix extracted from the Markov model for the CCU of the fault-tolerant system (shown in Figure 6.4). Equation 2b shows the ODEs extracted from the Q-matrix. Since the Markov model suffers from stiffness between state A and B due to the high repair rate compared to the low failure rate of the transient failures (including intermittent faults) [22]. Solution in MATLAB based on the the numerical differentiation formulas [23] is used to solve

the ODEs shown in Equation 2b and calculate the reliability shown in Equation 2c.

$$Q = \begin{bmatrix} -2(\lambda_t + \lambda_p) & 2c_t\lambda_t & 2c_p\lambda_p & (1-c_t)2\lambda_t + (1-c_p)2\lambda_p \\ \mu_r & -(\mu_r + \lambda_t + \lambda_p) & 0 & \lambda_t + \lambda_p \\ 0 & 0 & -(\lambda_t + \lambda_p) & \lambda_t + \lambda_p \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2a}$$

$$\begin{aligned} P'_A(t) &= -2(\lambda_t + \lambda_p)P_A(t) + \mu_r P(B) \\ P'_B(t) &= 2c_t\lambda_t P_A(t) - (\mu_r + \lambda_t + \lambda_p)P_B(t) \\ P'_C(t) &= 2c_p\lambda_p P_A(t) - (\lambda_t + \lambda_p)P_C(t) \end{aligned} \tag{2b}$$

$$R_{CCU} = P_A(t) + P_B(t) + P_C(t) \tag{2c}$$

Since the CCU is a E/E system, the failure rate is estimated from the SIL according to the IEC 1508 standard [9][14] (see section 3.4 in this thesis). Table 6.1 shows example values for failure rates for one CCU with the different ASIL levels. The failure rates are extracted from Table 3.1 and approximated to failures per hour. The estimated failure rates are for transient and intermittent failures.

Table 6.1: Example values for transient failure rates for different ASIL.

SIL (IEC 1508)	Failures per year		ASIL (ISO 26262)	Failures per hour
4	$\geq 10^{-5}$ to $< 10^{-4}$		D	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 10^{-4}$ to $< 10^{-3}$		C	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-3}$ to $< 10^{-2}$		B	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-2}$ to $< 10^{-1}$		A	$\geq 10^{-6}$ to $< 10^{-5}$
			QM	None required

Table 6.1 shows a possible source for derivation of random hardware failure, stated in the ISO 26262 standard, part 5 [4]. It shows that the values from Table 6.1 follows a good estimation related to the ISO 26262.

Table 6.2: Example values for random hardware failures values according to the ISO 26262 [4].

ASIL (ISO 26262)	Failures per hour
D	$< 10^{-8}$
C	$< 10^{-7}$
B	$< 10^{-7}$

The factor between permanent and transient failures (including intermittent failures) are estimated from the quantitative analysis treated in part 10 of the ISO 26262 standard [4]. The permanent failure rate is estimated to be 10 times lower than the failure rate for the transient failure rate. For example, if the transient failure rate is estimated to $< 10^{-8}$, the permanent failure rate is $< 10^{-9}$. This is considered for both the previous and fault-tolerant system for the CCU. The coverage factor are assumed to be the same for both permanent and transient failures for the CCU.

The reliability extracted from Equation 2c, for the fault-tolerant system using duplex CCUs, is compared to the reliability from Equation 1d, for the reference system using a simplex CCU.

Figure 6.5 shows the comparison between the CCU for the reference system (dashed lines) and the fault-tolerant system (continuous lines). The different failure rates are measured in failures per hour and the coverage is considered ideal. The comparison shows that the reliability of the fault-tolerant system is always higher than the reference system in the vehicle lifespan.

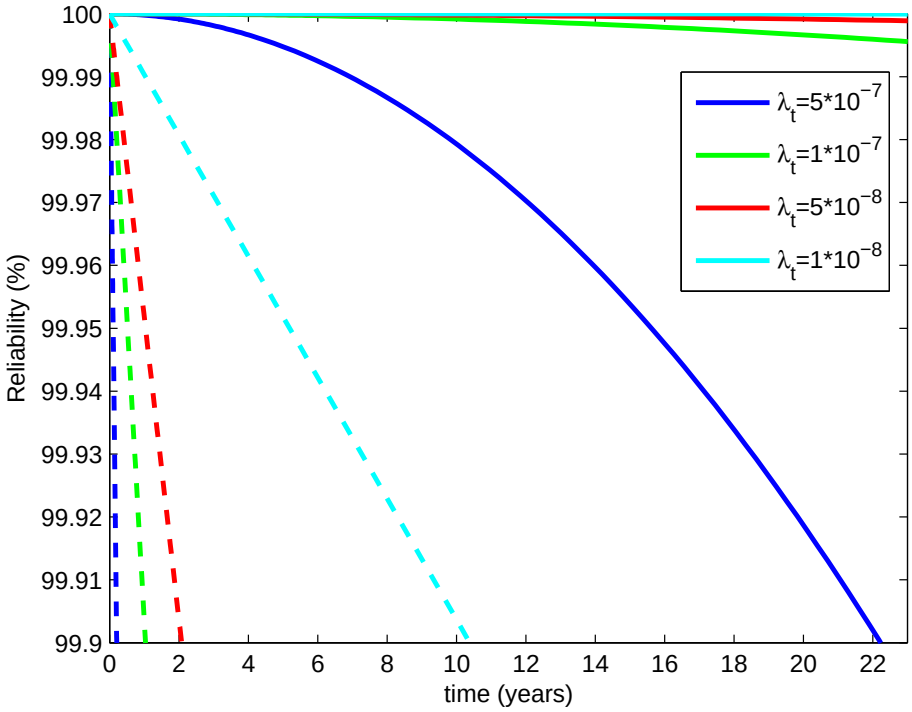

Figure 6.5: Reliability for the central control unit of the reference system (dashed line) and fault-tolerant system (continuous line). The different failure rates are measured in failures per hour and the coverage is considered ideal.

Figure 6.6 shows the reliability of the fault-tolerant system in the vehicle lifespan using an ideal coverage. The figure shows that failure rates lower than or equal to $\lambda_t = 10^{-7}$ failures per hour meets a high reliability of five-nines (i.e 99.999%) in 10 years.

To see how the coverage of the system effects the reliability, further analysis has been made where the coverage factor is decreased. Figure 6.7 shows the reliability of the systems using a coverage factor of 99%. Failure rates lower than or equal to $\lambda_t = 5 * 10^{-9}$ failures per hour fulfils the reliability of five-nines in 10 years.

Figure 6.8 shows the reliability of the systems using a coverage factor of 98%. The only tolerable failure rate that meets the five-nine reliability in ten years is $\lambda_t = 10^{-9}$ failures per hour.

Figure 6.9 shows the reliability of the system using a coverage factor of 95%. By further lowering the coverage to 95%, the systems reaches its limit using a failure rate of $\lambda_t = 10^{-9}$ failures per hour. Lowering the coverage even further the five-nine reliability cannot be achieved for 10 years lifespan of a vehicle.

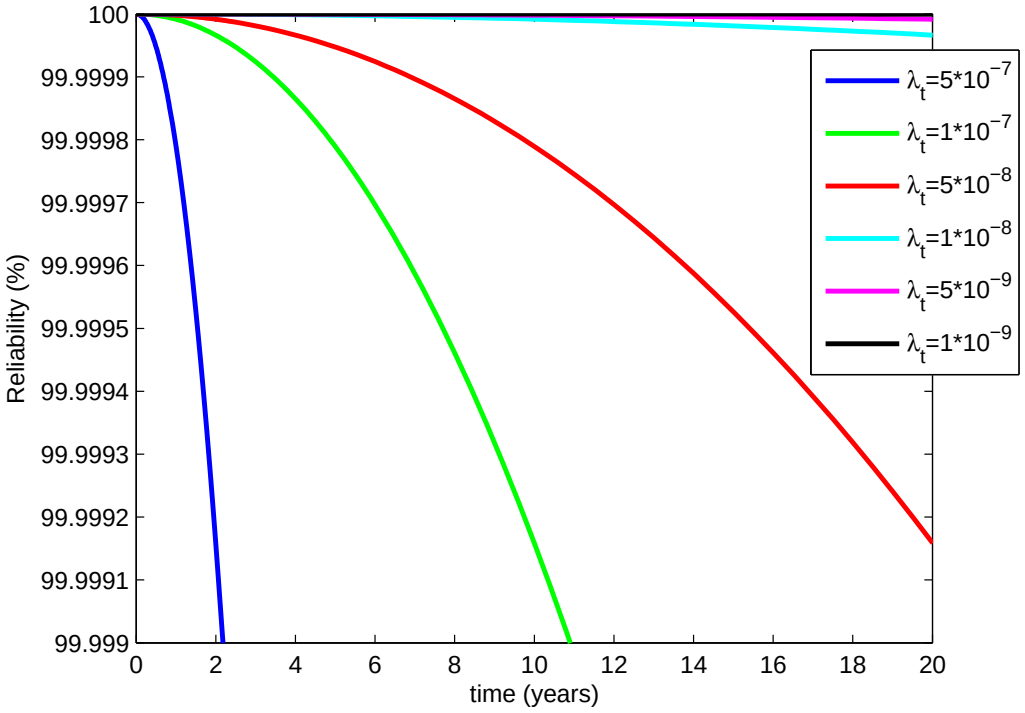

Figure 6.6: Reliability for the fault-tolerant system's central control unit using ideal coverage. The different failure rates is measured in failures per hour.

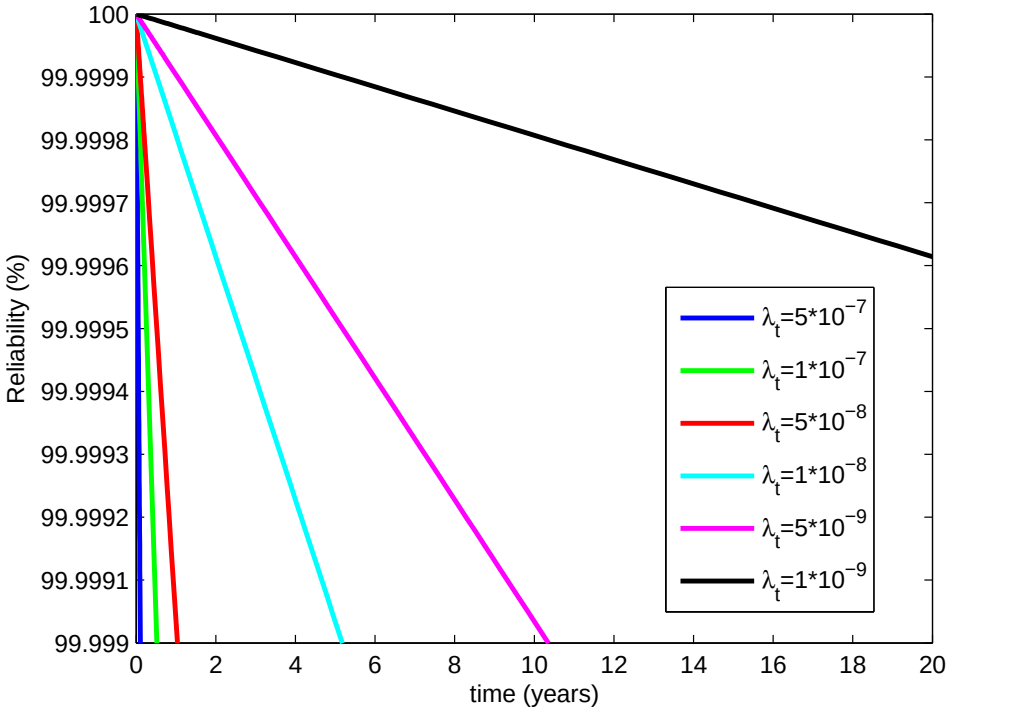

Figure 6.7: Reliability for the fault-tolerant system's central control unit using a coverage of 99%. The different failure rates are measured in failure per hour.

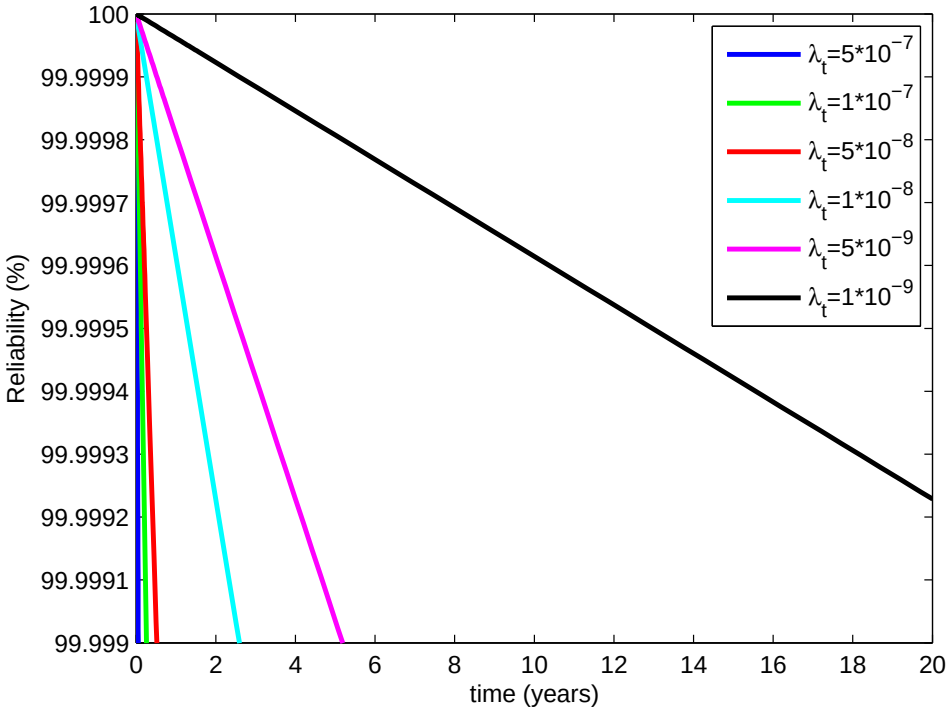

Figure 6.8: Reliability for the fault-tolerant system's central control unit using a coverage of 98%. The different failure rates are measured in failure per hour.

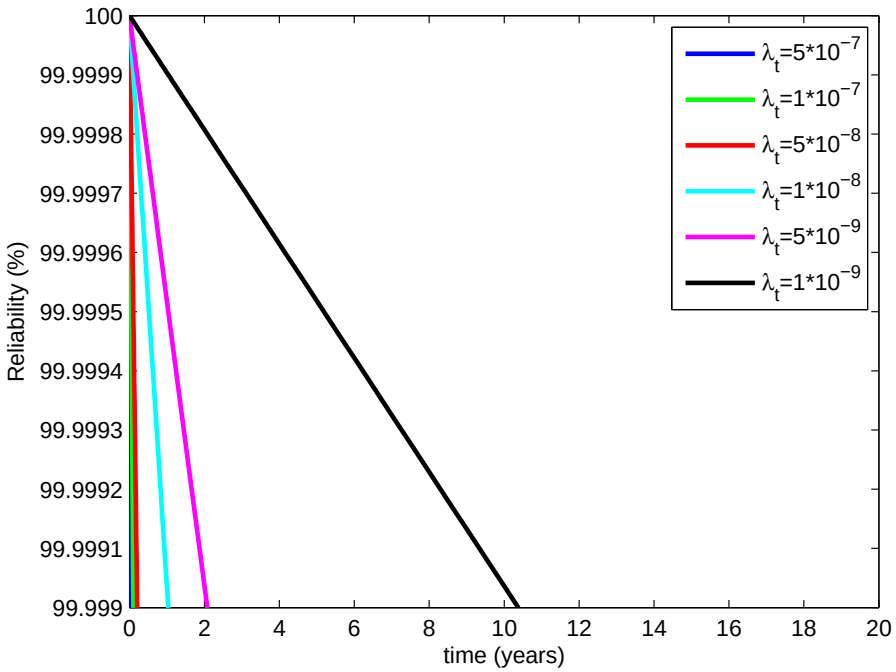

Figure 6.9: Reliability for the fault-tolerant system's central control unit using a coverage of 95%. The different failure rates are measured in failure per hour.

6.2.2 Brake-and-steer system

The reference system has two independent systems for the brake and steering ECU, seen in the fault tree in Figure 6.1. The Markov model for the reference system is shown in Figure 6.10. System initially start from state A. A failure in the brake system will result in a brake failure (state B), where as a failure in the steer system will result in a steering failure (state C).

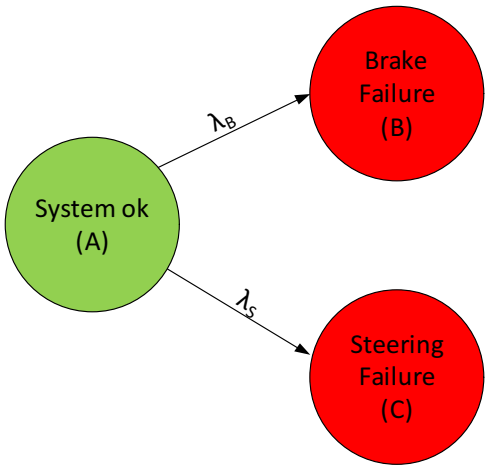

Figure 6.10: Markov model of the brake-and-steer system for the reference design. λ_B is the failure rate for one brake ECU and λ_S for the primary steering unit.

The Q-matrix of the Markov model in Figure 6.10 is represented in Equation 3a. The ODE extracted from the Q-matrix is shown in Equation 3b. The calculation of the Laplace transform from the ODE, inverse-Laplace transform and reliability of being in state A is represented in Equation 3c. The reliability of the previous brake system and steer system is represented in Equation 3d.

$$Q = \begin{bmatrix} -(\lambda_B + \lambda_S) & \lambda_B & \lambda_S) \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3a}$$

$$P'_A(t) = -(\lambda_B + \lambda_S)P_A(t) \tag{3b}$$

$$\mathcal{L} \Rightarrow P_A(s) = \frac{1}{s + (\lambda_B + \lambda_S)} \Rightarrow \mathcal{L}^{-1} \Rightarrow P_A(t) = e^{-(\lambda_B + \lambda_S)t} \tag{3c}$$

$$R_{BS} = P_A(t) \Leftrightarrow R_{BS} = e^{-(\lambda_B + \lambda_S)t} \tag{3d}$$

The fault-tolerant system for brake and steer uses a distributed brake architecture which increases safety for both steering and brakes. The brake system is still operational after failure of one distributed brake ECU. The steer system also has additional redundancy since the steer-by-brakes technology works as a backup if the steer ECU has failed using the distributed brakes (both brake ECUs needs to be functional).

The Markov model of the fault-tolerant brake-and-steer system is shown in Figure 6.11. Initially the system starts in state A, meaning that both brake and steering are at full functionality. The failure rate for the brakes are represented as λ_B and the coverage for the brake system is c_B.

Since there are two brakes ECUs in the vehicle, the transition from state A to B is two times the failure rate for the brakes. The failure rate for the steer system is represented λ_s and the coverage c_s. When a failure of one brake ECU occurs, a transition from state A to B occurs. The system is then in the degraded functionality for brakes. This means that the system is still operational, but the brakes works in degraded mode. If another brake fails when in state B, the system can no longer provide brake functionality for the vehicle. Therefore, transition from state B to D occurs, i.e the brake system have failed. The functionality of the steer-by-brakes is dependent on both brake ECUs being functional. If not, the functionality of steer-by-brakes cannot be guaranteed. Therefore, if a steering failure occurs when in state B, transition from state B to E will occur. State E implies that the steering has failed. When the vehicle is at full functionality (state A), and suffers from a steering failure, transition from state A to C will occur. State C implies that the steering is operational using the steer-by-brakes functionality for steering. If one brake ECU fails, transition from state C to E will take place, i.e steering failure. The system may also be exposed to non-covered faults. This is showed in Figure 6.11 where a transition goes from state A to a failure state (D or E). A non-covered failure is for example a non-detected fault in the brake or steering ECU that propagates to a system failure.

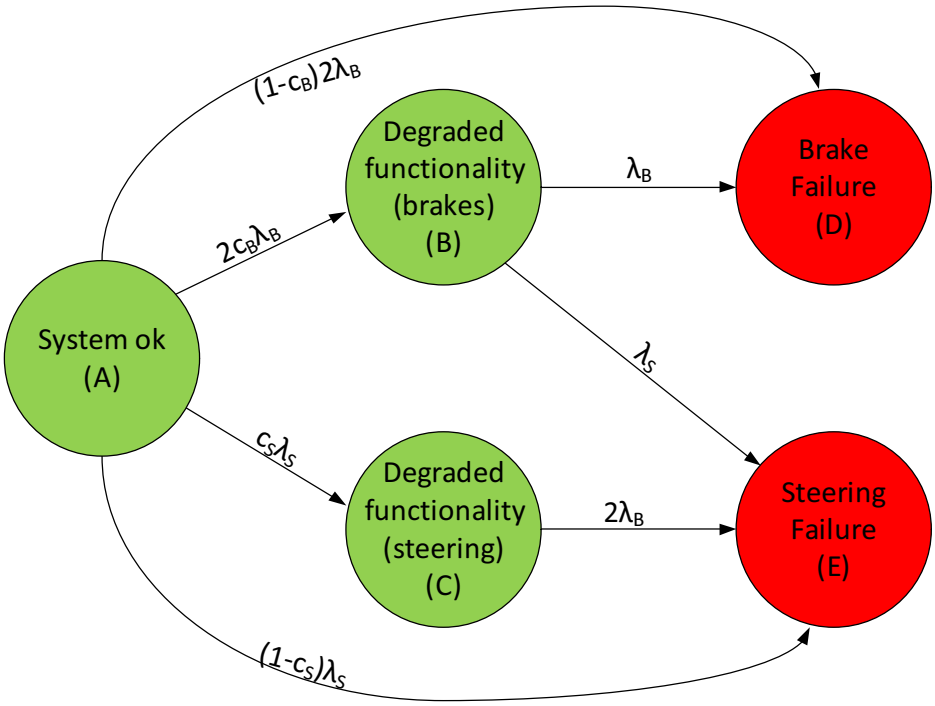

Figure 6.11: Markov model of the brake-and-steer system for the fault-tolerant design.

The Q-matrix extracted from the Markov model is shown in Equation 4a. The extracted ODE from the Q-matrix is shown in Equation 4b. Equation 4c shows the Laplace transform to the s-domain of the ODEs, the inverse-Laplace transform and calculated probability of being in state A, B and C according to Figure 6.11. Equation 4d shows the reliability for the brake-and-steer system at functional mode (both full and degraded functionality).

$$Q = \begin{bmatrix} -(2\lambda_B + \lambda_S) & 2c_B\lambda_B & c_S\lambda_S & (1-c_B)2\lambda_B & (1-c_S)\lambda_S \\ 0 & -(\lambda_B + \lambda_S) & 0 & \lambda_B & \lambda_S \\ 0 & 0 & -2\lambda_B & 0 & 2\lambda_B \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4a}$$

$$\begin{aligned} P'_A(t) &= -(2\lambda_B + \lambda_S)P_A(t) \\ P'_B(t) &= 2c_B\lambda_B P_A(t) - (\lambda_B + \lambda_S)P_B(t) \\ P'_C(t) &= c_S\lambda_S P_A(t) - 2\lambda_B P_B(t) \end{aligned} \tag{4b}$$

$$\mathcal{L} \Rightarrow \begin{bmatrix} P_A(s) = \frac{1}{s+2\lambda_B+\lambda_S} \\ P_B(s) = \frac{2c_B\lambda_B P_A(s)}{s+(\lambda_B+\lambda_S)} \\ P_C(s) = \frac{c_S\lambda_S P_A(s) - 2\lambda_B P_B(s)}{s} \end{bmatrix} \mathcal{L}^{-1} \Rightarrow \begin{bmatrix} P_A(t) = e^{-(2\lambda_B+\lambda_S)t} \\ P_B(t) = 2c_B(e^{-(\lambda_B+\lambda_S)t} - e^{-(2\lambda_B+\lambda_S)t}) \\ P_C(t) = c_S(e^{-2\lambda_B t} - e^{-(2\lambda_B+\lambda_S)t}) \end{bmatrix} \tag{4c}$$

$$\begin{aligned} &R_{BS} = P_A(t) + P_B(t) + P_C(t) \Leftrightarrow \\ &R_{BS} = e^{-(2\lambda_B+\lambda_S)t} + 2c_B(e^{-(\lambda_B+\lambda_S)t} - e^{-(2\lambda_B+\lambda_S)t}) + c_S(e^{-2\lambda_B t} - e^{-(2\lambda_B+\lambda_S)t}) \end{aligned} \tag{4d}$$

The example values for the failure rates are presented in Table 6.1 and Table 6.2 for the different ASILs. The failure rates are considered to be the same for the brake and steering ECUs and are measured in failures per hour.

The reliability of brake-and-steer system, shown in Equation 4c, is compared to the reliability of previous brake-and-steer system, shown in Equation 3c. The fault-tolerant system is considered to be functional as long as control of steering and brakes can be achieved. This means that the degraded-mode operation is included as an operational functionality in the reliability calculations.

Figure 6.12 shows the reliability for the brake-and-steer system for both the reference system (dashed lines) and for the fault-tolerant system (continuous lines). The failure rate for steering and brake is considered to be the same and is represented as λ_{BS}, i.e $\lambda_{BS} = \lambda_B = \lambda_S$. The comparison shows that the reliability of the fault-tolerant system is higher than the reference system in the vehicle lifespan. Notice that failure rate of $\lambda_{BS} = 3 * 10^{-8}$ failures per hour for the reference system reaches higher reliability than the fault-tolerant system when having a failure rate of $\lambda_{BS} = 5 * 10^{-7}$ failure per hour after approximately 11 years. This is interesting from a design perspective, where it could more efficient to use the reference design with a lower failure rate, instead of a fault-tolerant design with a higher failure rate, in cases where the reliability can be 99.5% or lower.

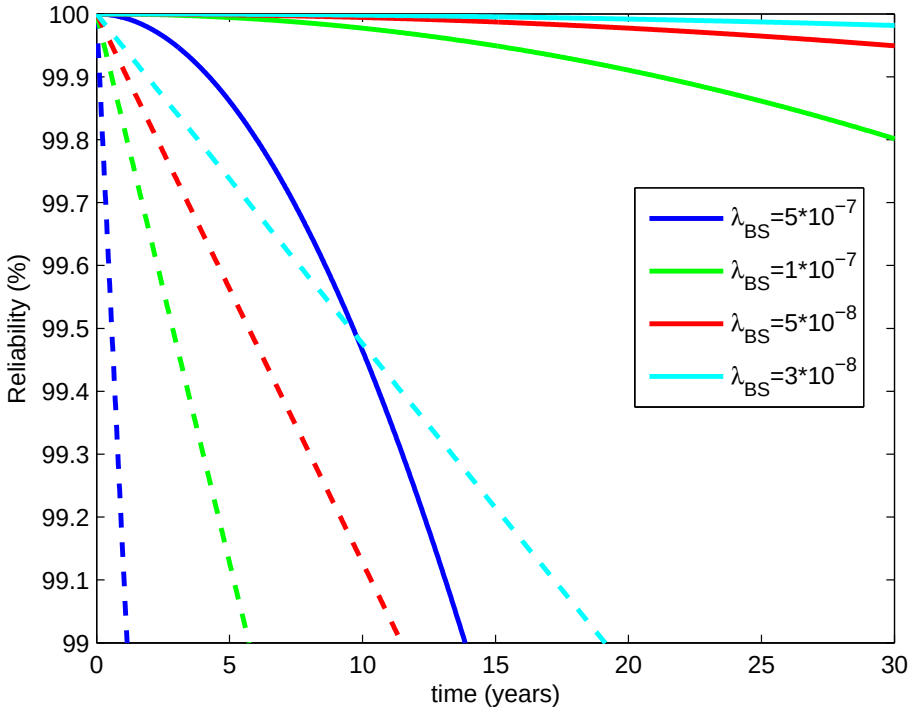

Figure 6.12: Comparison in reliability for the brake-and-steer system of the reference (dashed line) and fault-tolerant system (continuous line). The different failure rates are measured in failures per hour and the coverage is considered ideal.

Further analysis of the intersection points of the two systems were made. Figure 6.13 shows the intersection points for different failure rates. The dashed lines represents the reference system and the continuous lines represents the fault-tolerant system. Notice how the reliability is improved for the fault-tolerant system compared to the reference system with the lowering of the failure rate.

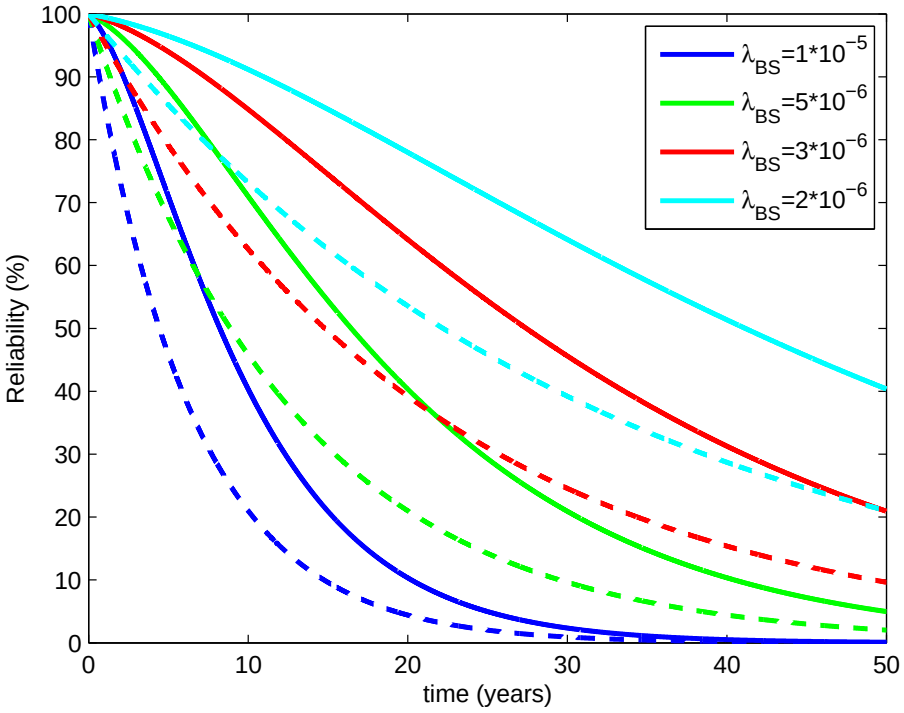

Figure 6.13: Intersection points for the reference system (dashed lines) and fault-tolerant system (continuous line).

Figure 6.14 shows the reliability of the fault-tolerant system for different failure rates. The failure rates are measured in failures per hour. A failure rate lower than $\lambda_t = 1 * 10^{-8}$ failures per hour meets the high reliability of five-nines (99.999%) for the vehicle lifespan of 10 years.

Figure 6.15 shows the reliability of the fault-tolerant system with a coverage of 99%. Notice that a failure rate of $1 * 10^{-9}$ failures per hour is needed to meet the five-nine reliability for 10 years.

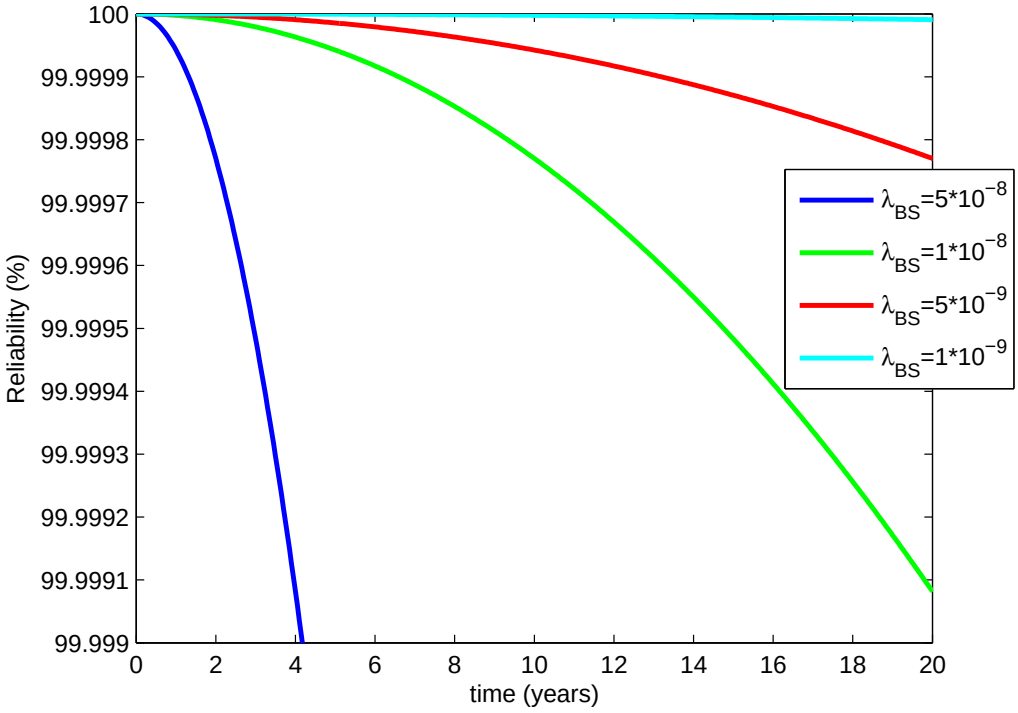

Figure 6.14: Reliability of the brake-and-steer system for the fault-tolerant design using ideal coverage. The different failure rates are measured in failures/hour.

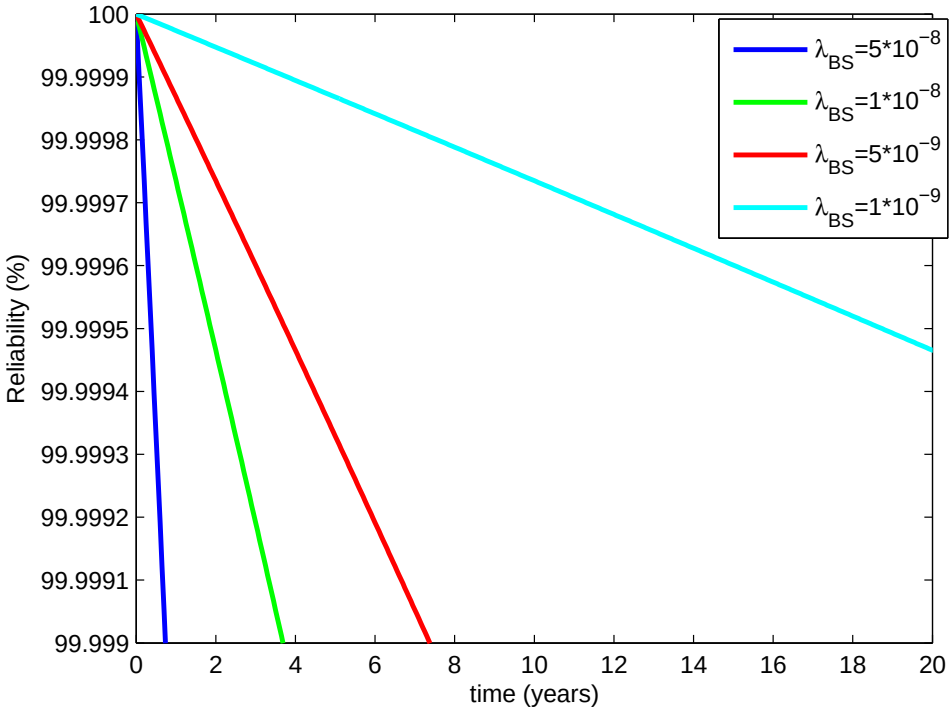

Figure 6.15: Reliability of the brake-and-steer system for the fault-tolerant design using a coverage of 99%. The different failure rates are measured in failures/hour.

By further lowering the coverage factor, the system cannot meet the five-nines in reliability for a reasonable failure rate. Figure 6.16 shows the reliability of the fault-tolerant system when using a coverage factor of 97%. The brake-and-steer system meets the five-nine reliability after ten years, when using a failure rate of $1 * 10^{-9}$ failures per hour.

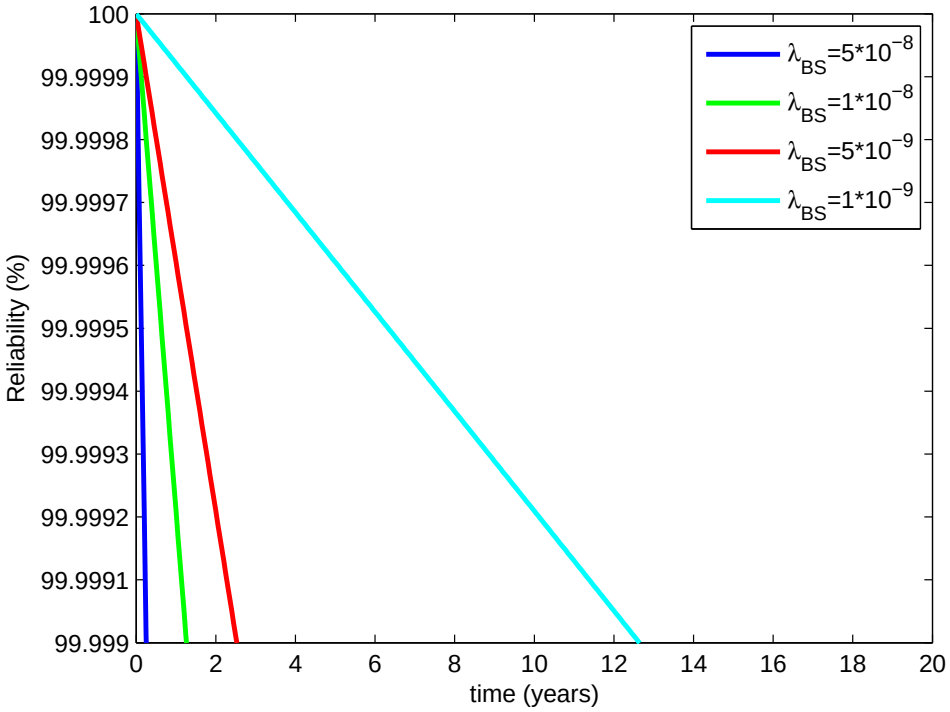

Figure 6.16: Reliability of the brake-and-steer system for the fault-tolerant design using a coverage of 97%. The different failure rates are measured in failures per hour.

6.3 System safety

This section analyses the safety of the drive-by-wire system. The fail-safe state of the system is considered to be when the vehicle has stopped at the side of the road. Analysis of the safety focuses on the steady-state safety of the system [9]. The steady-state safety is the probability that the system reaches a fail-safe state compared to a critical-failure state over a continues period of time.

Figure 6.17 shows the Markov model for the safety of the throttle subsystem. State A refers to that the subsystem is operational. The failure rate for the throttle system is λ_T and the coverage for the throttle system is c. If a covered failure occurs, transition from state A to state FS occurs. This implies that the throttle is shut off to avoid a hazardous event. If a non-covered fault occurs, the system is forced into a critical failure, transition from state A to state CF occurs. The critical failure could for example be stuck-at maximum value.

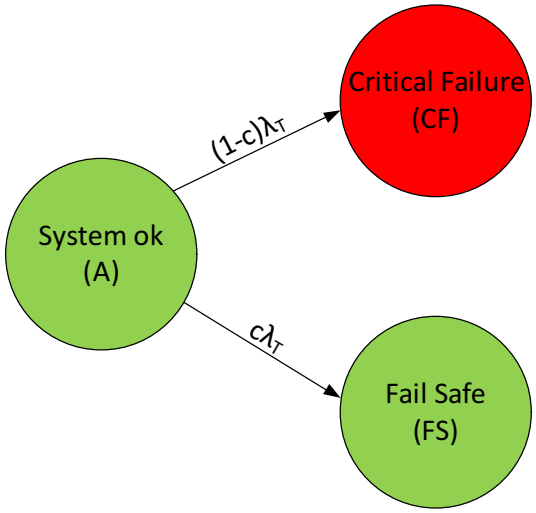

Figure 6.17: Markov model with safe state and critical state for the throttle subsystem. λ_T is the failure rate for the throttle ECU and c is the coverage.

The steady-state safety is calculated in equation 5. The equations provides the failure rate for the throttle ECU (λ_T) and coverage factor for the throttle ECU (c).

$$S_{Throttle} = P_{A \to B} = c\lambda_T \tag{5}$$

Figure 6.18 shows the steady-state–safety model for the CCU system. The failure rate for the transient and intermittent failure rate is λ_t and for the permanent failure rate λ_p. The coverage factor for the transient and intermittent failures are c_t and for the permanent failures c_p. The *slow-down time* for the vehicle is $(\mu_{sd})^{-1}$. The slow-down time is a safety mechanism to ensure the safety of the driver. The safety mechanism slows down the speed of the vehicle to lower the severity of the hazardous event or prevents it from happening (driver stops the vehicle).

A more detailed description of the state transitions are explained in section 6.2. In state A all CCUs are working. When one CCU is exposed to a permanent failure, transition from state A to C occurs. To ensure that the system goes into a safe failure state, the system will limit the speed of the vehicle to ensure that the driver can pull to the side of the road. This lowers the probability of the severity of the hazardous event and increasing the safety of the vehicle. The slow-down time of the vehicle is $(\mu_{sd})^{-1}$, i.e., the time it takes for the current speed to be lowered to a reasonable/non-hazardous speed. One example could be that the vehicle is on the high-way and having a driving speed of 70 km/h. When reaching state C, the speed is forced (either by decreasing throttle, engine brake or slight brake power) down to 20 km/h. The time it takes for the vehicle to lower the speed from 70 km/h to 20 km/h is the slow-down time. This gives the driver time to pull to the side of the road. If a transient or intermittent failure occurs, the CCU will restart itself, the time for the restart is μ_r^{-1}.

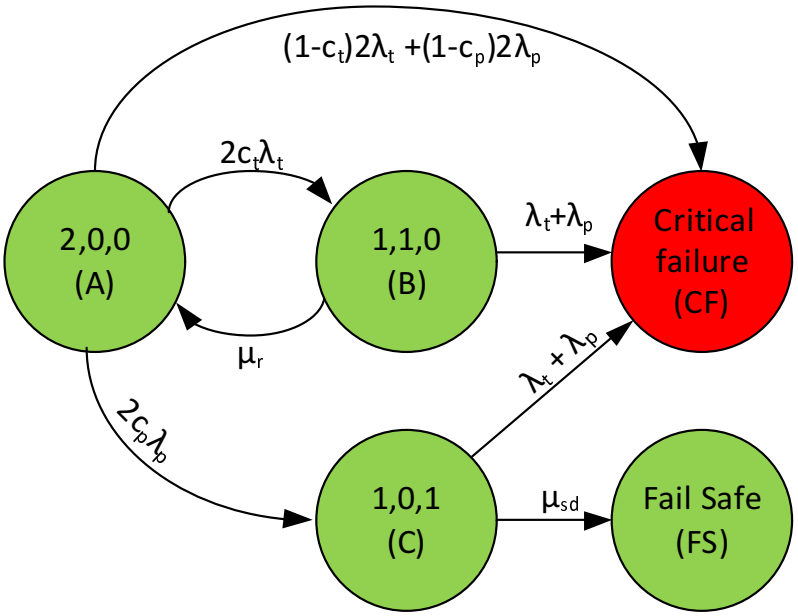

Figure 6.18: Markov model with the safe state and critical state for the CCU system.

The equation for the steady-state safety of the CCU system is provided in Equations 6a and 6b. The failure rate for the steer and brake ECUs is represented as λ_s and λ_B and coverage for steer and brake ECUs is represented as c_s and c_B. The n is the number of restarts the CCU does. For example, if there are 10 covered transient failures where a restart of the CCU occurs and then a covered permanent failure occurs, the system will transit back and forth between state A and B 10 times, and then transition from state from A to C will occur.

$$S_{CCU} = (P_{A\to C} * P_{C\to FS}) + (P_{A\to B} * P_{B\to A})^n * (P_{A\to C} * P_{C\to FS}) \tag{6a}$$

$$S_{CCU} = (P_{A\to C} * P_{C\to FS})(1 + (P_{A\to B} * P_{B\to A})^n) = \frac{2c_p\lambda_p}{(2(\lambda_t+\lambda_p)} * \frac{\mu_{sd}}{(\mu_{sd}+\lambda_t+\lambda_p)} \tag{6b}$$

The Markov model suffers from stiffness between state A and B due to the fact that $\mu_r >> 2c_t\lambda_t$ (see Figure 6.18). Since the restart time of the system is small and a low failure rate for the CCU is needed. This implies that for a high number of transient failures, the general equation shown in 6a can be simplified to equation 7.

$$S_{CCU} = (P_{A\to C} * P_{C\to FS}) = \frac{2c_p\lambda_p}{(2\lambda_t+2\lambda_p)} * \frac{\mu_{sd}}{(\mu_{sd}+\lambda_t+\lambda_p)} \tag{7}$$

Figure 6.19 shows the Markov model for the brake-and-steer system. A more detailed description of the state transitions for the brake-and-steer system is explained in section 6.2. When the system is in degraded functionality (state B or C), the driver will be warned to pull the vehicle to the side of the road. The pull-over time is estimated to be μ_B when in degraded mode for brakes and μ_S when in degraded mode for steering.

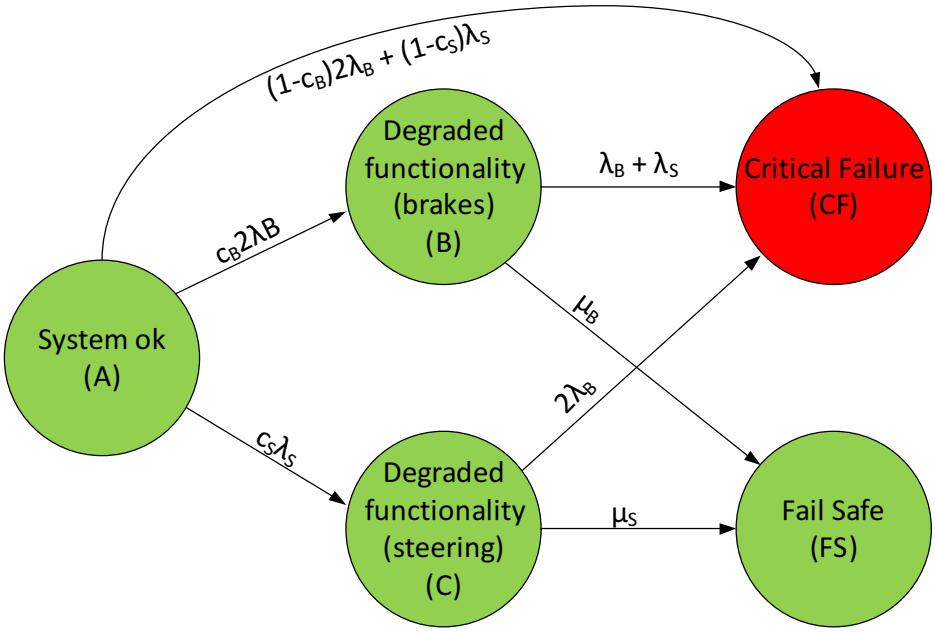

Figure 6.19: Markov model with safe state and critical state for the brake-and-steer system. The time it takes for the driver to stop is estimated with μ_s in case of a steering failure and μ_B in case of a brake failure.

The equation for the stead-state safety of the brake-and-steer system is provided in equations 8a and 8b. The equations provides the failure rate for the steering and brake ECUs (λ_s and λ_B) and coverage for steering and brake ECUs (c_s and c_B).

$$S_{BS} = P_{A\to B} * P_{B\to FS} + P_{A\to C} * P_{C\to FS} \tag{8a}$$

$$S_{BS} = \frac{c_s 2\lambda_s}{(2\lambda_B+\lambda_S)} * \frac{\mu_B}{(\lambda_B+\lambda_s+\mu_B)} + \frac{c_S\lambda_S}{(2\lambda_B+\lambda_S)} * \frac{\mu_s}{(\mu_s+2\lambda_B)} \tag{8b}$$

7 Implementation

This section explains how the proposed methods in section 5 can be implemented in a fault-tolerant prototype. The implementations for the prototype are divided into software and hardware.

Figure 7.1 shows an overview of the fault-tolerant prototype. The implementations have been implemented on the existing prototype explained in section 2. An overview of the prototype can be seen figure 2.1. In the existing prototype, major changes has been made for the brake-and-throttle ECU and the central control unit (CCU). The throttle and brake functionality has been divided into two separated ECUs for the fault-tolerant prototype. An implementation of a brake ECU has been made to show the functionality of the distributed brake system and steer-by-brakes functionality. For the CCU the fault-tolerant prototype now operates in duplex mode. Further, safety mechanisms has been implemented in software and hardware to prevent, detect and recover from failures in the CCU.

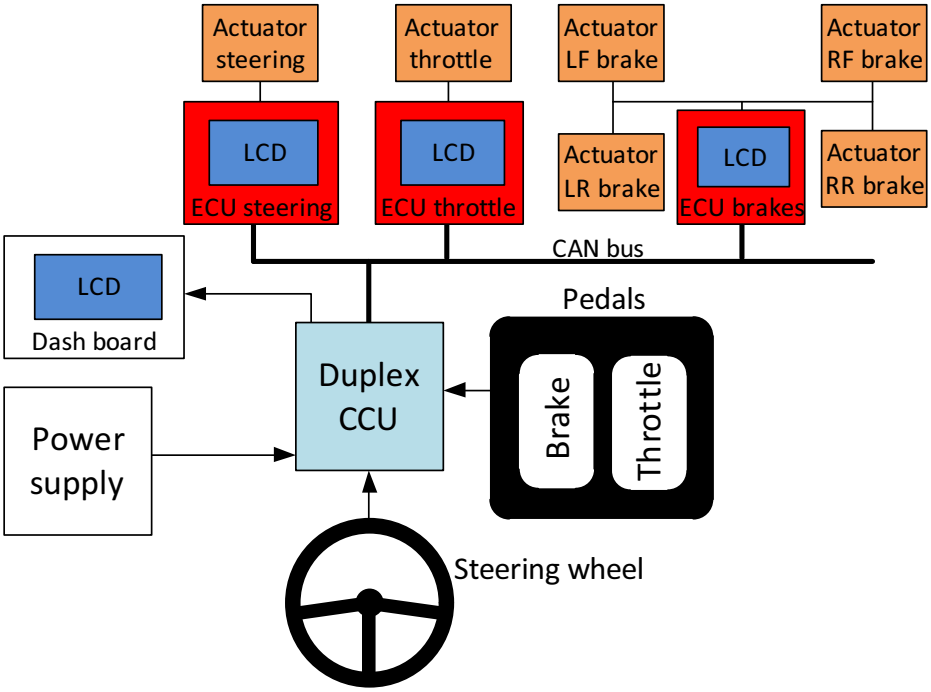

Figure 7.1: Overview of the fault-tolerant prototype. The central control unit (CCU) runs in duplex modular redundancy.

7.1 Software

In this section, the tasks in the CCU are described. Both new tasks and modifications made to tasks in the existing prototype are mentioned. Two voting algorithms have been implemented and are further described in section *Voting*. The implementation of how a CCU handles loss of steering value when restarting is mentioned in *Recapture of steering value*. How a CCU detects when the other CCU has failed is described in *Determination of a failed CCU*.

7.1.1 Tasks

In the fault-tolerant prototype there have been major changes in all safe-critical tasks. The safe-critical tasks that are implemented in the CCUs are; *throttle task*, *brake task*, *steer task*,

CAN_ RX task, *CAN_ TX task* and *watchdog task*. Figure 7.2 shows a flow chart of the tasks in the fault-tolerant prototype. The task with rounded corners, i.e., LCD_task, is only executed in the primary module and not implemented in the backup module. The tasks with dashed frame are tasks that are added to the fault-tolerant prototype.

The scheduling priority of the tasks has also been altered to better suite the priority of the tasks. For example, the task responsible for displaying data on the LCD should not have the same priority as the task responsible for measuring the brake value.

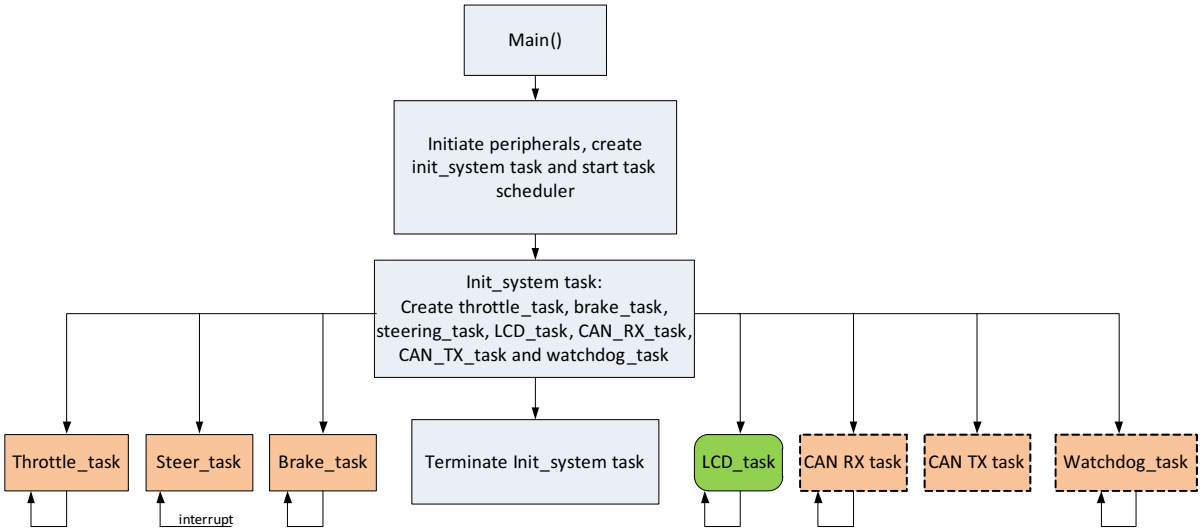

Figure 7.2: Flow chart of the tasks in the fault-tolerant prototype. Added tasks are noted with a dashed box. Rounded box is task that are not safe critical.

The implementation of throttle and brake tasks (i.e *throttle task* and *brake task*) are quite similar to each other. Figure 7.3 shows an overview of the throttle/brake task. Both the throttle and brake task read from one of the ADCs on the development board. They read the values five times to ensure temporal redundancy of the sensors. The time in between each read is separated with a small time gap of one millisecond. The values are then voted using median voting (see section 7.1.2) to receive a correct value. The value is then processed three times and stored in three different memory spaces. Both tasks do also update a watchdog variable to notify the watchdog task that they are still alive and working properly.

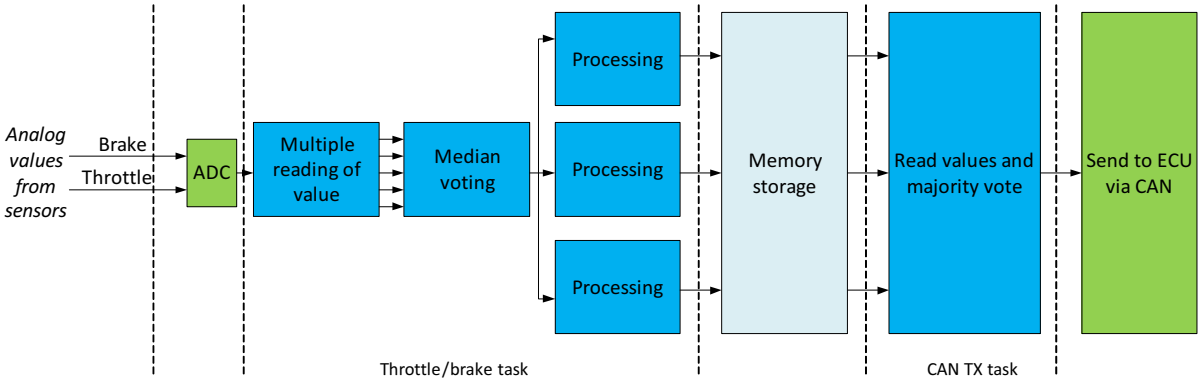

Figure 7.3: Overview of the functionality for throttle-by-wire and brake-by-wire.

Watchdog task executes periodically every 60 ms. The main responsibility of the watchdog task is to monitor the watchdog variables of every task. In every execution of the watchdog task,

the watchdog variables are being checked and reset to 0. If any of the variables are still 1 this means that the task that corresponds to that variable have not executed properly and missed its deadline. For example, if the watchdog variable of throttle task is equal to 1, it means that it have not been set to 0 in throttle task. This is a sign that throttle task have not been executed properly. Another responsibility is to monitor an internal error variable. If the implemented median voter or majority voter cannot determine a correct value, the internal error variable will be set high. If any of the watchdog variables have not been restored, or the internal error variable is set, it is to be seen as a failure and the error pin is set high.

In the existing prototype, the task responsible for sending messages on CAN was called CAN task. CAN task have now been renamed to *CAN_ TX task*. The CAN_TX task uses majority voting to fetch the processed sensor values for throttle, brake and steering to ensure that data was not corrupted by bit flips. The correct value is sent via the CAN bus to corresponding ECU. The CAN task is also responsible for sending an alive package on the CAN bus which notifies the other CCU that it is working properly.

The reference implementation of the CCU did not implement the ability to receive messages on CAN. This was basically because the CCU was the only node on the bus that did send information on CAN, so there was no need for the CCU to receive messages on CAN. The fault-tolerant design does however include two CCUs which needs to communicate with each other. This requires a way for both CCUs to receive messages on CAN. A new task, *CAN_ RX task*, was added to the CCU in order to handle the incoming CAN messages. When a new CAN message is received by the CAN peripheral of the CPU, it will generate an interrupt that activates the interrupt handler. The interrupt handler will in its turn wake up the CAN_RX task in case that the received message was of any interest of the CCU. The CAN_RX task will then process the received data and eventually respond with some information, depending on the message id of the received message.

Each CCU implements CAN communication by defining a number of message boxes. Each message box can either be in receive mode or transmit mode, and is defined for a certain CAN message id. The CAN message ids for each kind of message is presented in Table 7.1. The message boxes is internally defined in each unit, and do not have to be the same among the units in a system. However, CAN message ids are globally defined, which means that the CAN message id for a message in one unit is the same for another in the same system.

Table 7.1: Id for the different CAN messages.

ID	Message
0x1	Throttle value
0x2	Brake values
0x3	Steer request
0x4	Alive signal
0x12C	Steering value

No changes has been made to the *steering task* since all the safety mechanism concerning the steering is made in CAN_TX and CAN_RX. These safety mechanisms include recapturing of the steering value and steer-by-brakes.

7.1.2 Voting

Two voting algorithms have been developed in order to detect or prevent faults. A *majority voter* have been developed to be able to specify a majority in a set of values. A *median voter* have

been developed to verify the plausibility of a value.

The algorithm used for the *majority voting* is MJRTY by Boyer & Moore [24] which is a linear-time majority voter. The majority voter is used by the CAN_TX task to determine if there has been any bit flips since the values was processed and stored. If a majority cannot be decided, a local error flag is risen to inform the watchdog task. Otherwise the correct value is returned.

The *median voter* determines the median value of a number of values in a list. If the values differ too much, the median voter will notify the watchdog task by raising the error flag. The median voter takes a list of values and the number of values in the list as argument. It compares the first value with the other values in the list by calculating the difference between the first value and next value, one by one. If the difference between the first value and another value is below a pre-defined constant, a counter is increased. The pre-defined constant sets the sensitivity of the median voter. When all values have been compared with the first value, the counter is being investigated. If the counter is greater than the original number of values divided by two, the first value of the list is the median value. If the counter is less than the original number of values divided by two, the median voting will be called with the same list but shifted once to the left and the number of values decreased by one. If the number of values is equal to one and no median value have been established, the error flag will be risen.

7.1.3 Recapture of steering value

If one of the CCUs encounter an error and gets restarted by the hardware monitor, it loses all temporary data that is stored in variables on ram memory, including sensor values. Most of the data is independent of current state and can easily be fetched after a restart. This applies to the sensor values of throttle and brake. Steering value however, is a little more complicated. Since the CCU counts the number of interrupts fetched from the steering sensor in order to keep track of the steering value, the CCU cannot continue after a restart since it has no information regarding the angle of the wheels. To retain the wheel angle, the system utilizes the fact that the backup CCU runs hot standby and therefore holds the correct value of the steering value. A request will be sent on CAN in the startup sequence of the CCU, in order to retrieve the current steering value from the other CCUs. If both CCUs starts up at the same time, none of them will get a response and both will, *after 3x2 ms*, assume that the steering sensor is in home position.

The timeout of 2 ms is based on the time it takes for one CCU to send a request and get the response. In the meantime, the other CCU have to get the request, process the data and send the response, which is illustrated in Figure 7.4. If the first request or response for some reason does not arrive, the CCU tries to send a request two times more before it assumes that the steering sensor is in home position, hence *after 3x2 ms*. The time in between a request and a response can vary a bit from time to time, depending on current load on CAN bus and the time it takes for the CCU to catch the interrupt from the CAN peripheral. Measurements were made to estimate the time it takes in between a request and a response. The result showed that the time varies from 0.2 ms to 1 ms. Therefore 1 ms plus an overhead of 1 ms was chosen as the timeout.

To make sure that steering value is recaptured as early as possible, recapturing is made before the OS spawns the tasks that constitutes the drive-by-wire functionality of the CCU, including the CAN_RX task. This means that the interrupt handler must check if the CAN_RX task is spawned before it can redirect any incoming CAN messages. Since this is not the case when the CCU waits for the response of the steer value, it has to frequently check for incoming CAN messages, i.e., polling.

The case when both CCUs startup at the same time is illustrated in Figure 7.5.

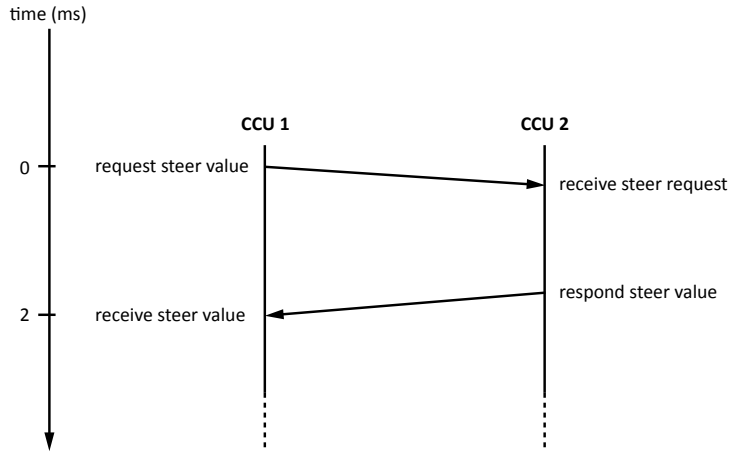

Figure 7.4: Messages on CAN bus to recapture steering value.

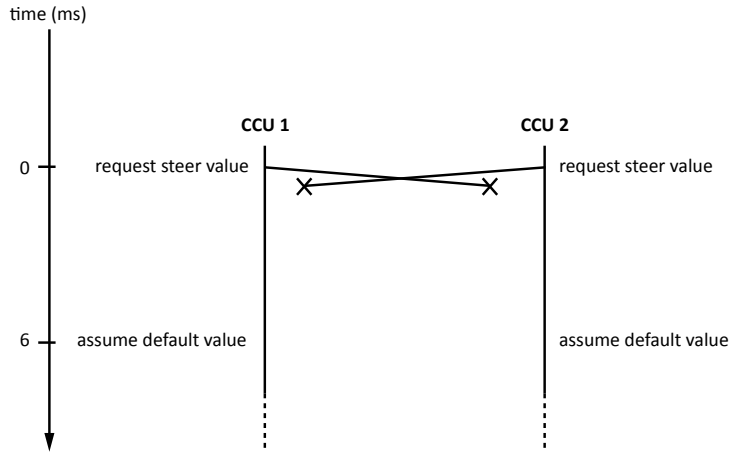

Figure 7.5: Messages on CAN bus for recapturing the steering value when both CCUs startup at the same time.

7.1.4 Determination of a failed central control unit

When primary central control unit (CCU) encounter a failure, it will either get restarted or shut down by its hardware monitor. When the primary CCU is either restarting or shut down, it will temporarily or permanently stop sending data on the CAN bus and the backup CCU can determine that it is time to take over. Though, when backup CCU encounters an error, there is no way for primary CCU to get this information since the backup CCU is standby and does not send any information on CAN. To resolve this problem, it was decided to let backup CCU send an alive signal periodically on CAN. If primary CCU does not get the alive signal within a certain time window, it will assume that the backup CCU is either shut down or temporarily restarting. With this information, both CCUs will have knowledge whether the other CCU is present or not. The active CCU can then prevent itself from restarting when the other CCU is broken, even though a failure is detected. This feature is implemented to prevent both CCUs from restarting at the same time. If there would be some point in time when no CCU is active,

information regarding current steering angle will be lost, which would lead to a critical failure of the drive-by-wire system.

7.2 Hardware

This section describes the implementations made in hardware. It involves the duplex CCU, the implementation of the Hardware monitor and the Distributed-brake ECU. The platforms used for main CCU and backup CCU is described in Development platforms.

Figure 7.6 shows how the duplex system will look like. Both CCUs are connected to a hardware monitor, which monitors the error pin of corresponding CCU continuously. When an error is detected, the hardware monitor will restart the CCU. Sensor values will be distributed to both CCUs and the ECUs will get values from the active CCU sending the values via the CAN bus.

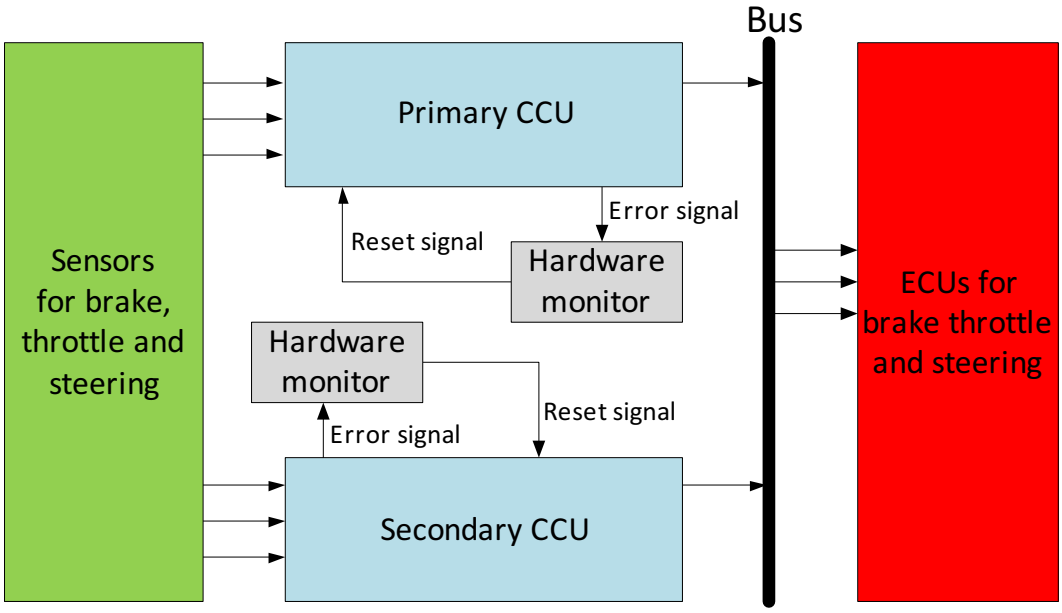

Figure 7.6: Hardware implementation of the fault-tolerant system showing the main central control unit (CCU) and the backup CCU running in hot-standby mode.

7.2.1 Primary and backup central control unit

A picture of the primary central control unit (CCU) can be seen in Figure 7.7. On the left hand side of the lid, there are four toggle switches. These switches are used for software implemented fault injections to force the CCU to set the error pin after various times. First switch sets the error pin every 100 ms, second switch corresponds to 60 ms and third switch corresponds to 20 ms. Fourth switch is reserved for future use. The error pin is connected to the hardware monitor which is mounted beneath the lid. The three diodes visible on the right hand side of the lid shows the current status of the hardware monitor. On the top-right corner of the lid, there is a switch which enables the functionality to turn the hardware monitor on or off. On the right hand side of the CCU (not visible in the picture), there is a connector to connect the CCU to the CAN bus, the sensors and communication to a display.

A picture of the backup CCU can be seen in figure 7.8. The connector in the middle is corresponding to the connector of the primary CCU. The three diodes and the toggle switch at the

top do also correspond with the functionality of diodes and the toggle switch at the top right corner of the primary CCU.

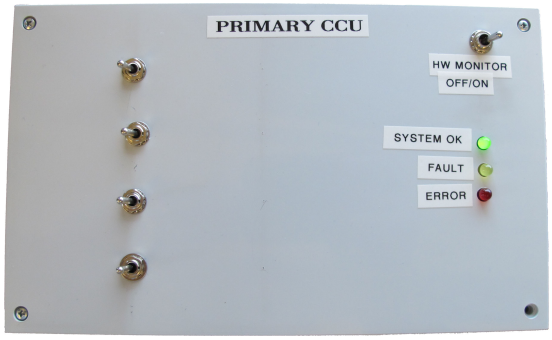

Figure 7.7: Picture of the main/primary central control unit.

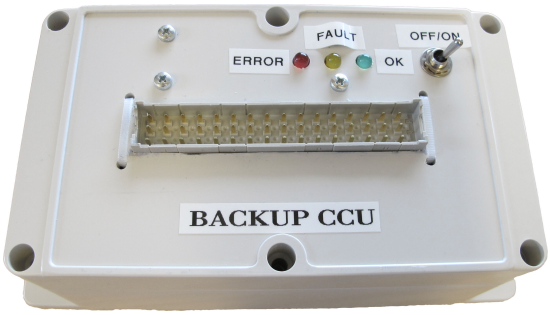

Figure 7.8: Picture of the implemented backup/secondary central control unit.

7.2.2 Hardware monitor

The hardware (HW) monitor, which can be seen in Figure 7.9, is a separate circuit board responsible for restarting the CCU if a failure is detected. The functionality is implemented in a Xilinx XC9572XL CPLD [25]. The CPLD runs at a clock speed of 10 MHz provided by an oscillator. A linear voltage regulator converts the voltage from 5 V to 3.3 V which is required by the CPLD and the oscillator. Three diodes, green, yellow and red, is used to present the current status of the hardware monitor. There is also a JTAG interface to enable reconfiguration and debugging of the hardware in the CPLD. The signal ERROR (further referred to as nError) is the signal going from the error pin on the CCU to the HW monitor, which indicates when an error has occured. The RESET (further referred to as nReset) is the signal going from the HW monitor and used to reset the CCU when an error has occurred.

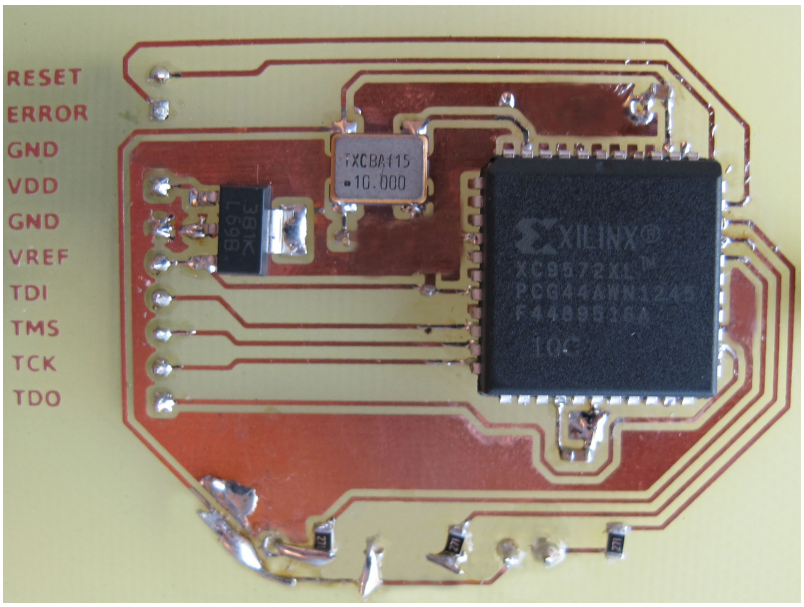

Figure 7.9: Picture of the hardware monitor.

The state diagram for the implementation of the HW monitor used to monitor the primary and secondary CCU is shown in Figure 7.10. When no error is detected by the CCU the HW monitor

is in state S0, which is the initial state. When an error is detected and nError signal goes low, the transition to state S1 occurs. State S1 has two main functions. The first one is to prevent a restart of the CCU if a glitch has occurred, showed in Figure 7.11, but also to ignore the initiation of the nError (occurs after system reset) of the CCU when a reset has occurred, shown in Figure 7.12. The delay is 7 ms, which is the length of the initiation of the nError of the CCU after reset.

After the delay in S1, the internal signal "Delay ok" goes high and transition to state S2 occurs. State S2 checks if the error was a glitch or if the reset initiation has been made of the nError signal. If one of these has occurred, the nError signal will be 1 after the delay and transition to S0 will occur. However, if the nError signal is still 0 after the delay, transition to state S3 occurs. At state S3 a system restart will be initiated to the CCU by the HW monitor. When nError is set to 1 again, the CCU has successfully restarted and transition from state S3 to state S1 will occur.

Another feature of the HW monitor is the internal max time signal, referred to as MT in Figure 7.10. The signal MT keeps track of a counter in the CPLD which, when reaching a maximum value, will be set to 1 and transition to state "Lock" will occur. When in lock state, the HW monitor will keep the reset signal at 0, which will shut down the CCU and preventing it from corrupting the rest of system. Example of the state transitions for reaching the lock state is shown in Figure 7.13.

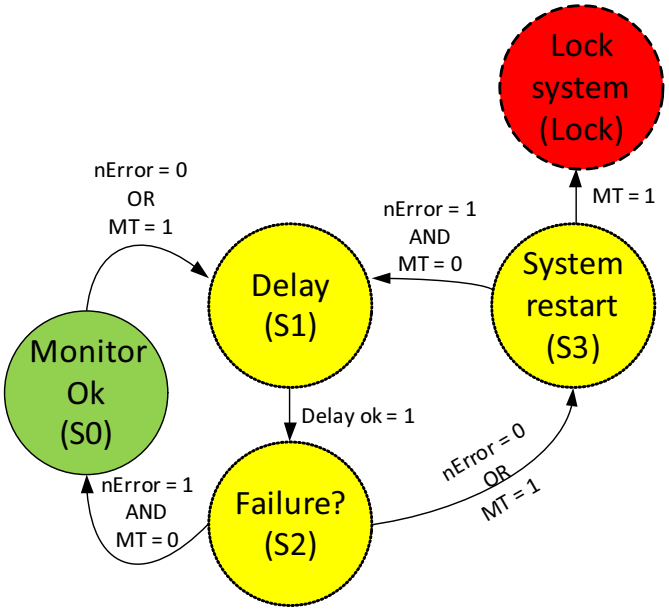

Figure 7.10: State diagram of the hardware monitor.

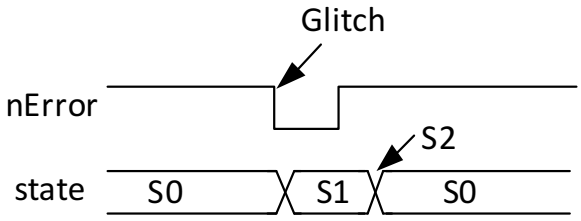

Figure 7.11: State transition of the hardware monitor when a glitch occurs on the nError signal.

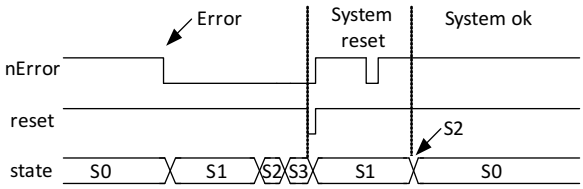

Figure 7.12: State transition of the hardware monitor when a transient fault occurs.

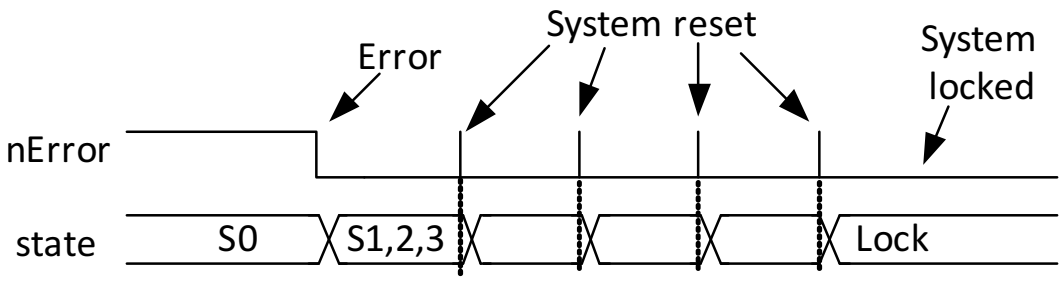

Figure 7.13: State transition of the hardware monitor when a permanent fault occurs and the state is forced into the lock state.

The algorithm only increases the counter in the yellow states in Figure 7.10 (state S1,S2 and S3). When the HW monitor is not detecting an error (state S0), the counter counts down instead. This gives a balance to what is assumed as a permanent HW failure, intermittent failure and transient failure. Behavioural simulation of the hardware functionality is shown in Figures 7.14, 7.15, 7.16 and 7.17. The y-axis in the figures shows the count number of the counter and the red top line indicates the maximum tolerable value for the counter which sets the MT signal to 1.

The system reaches a lock down when the time between start and failure is under 100 ms, shown in Figures 7.14, 7.15 and 7.16. A time over 100 ms, the system reaches relaxation, seen in Figure 7.17. The highest time between start and failure is designed for the system to be able to send at least one value to the ECUs before restarting. This time was estimated to be around 80 ms after start-up.

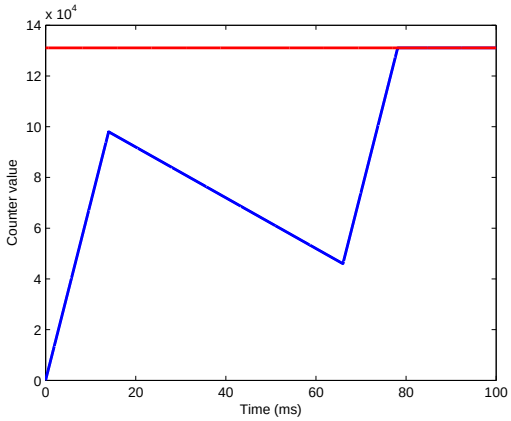

Figure 7.14: Function of the hardware monitor when time between failures in the central control unit is 40 ms. Number of resets before locking the CCU is 1.

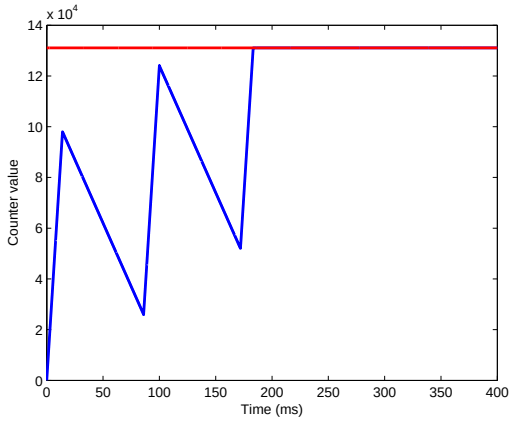

Figure 7.15: Function of the hardware monitor when time between failures in the central control unit is 60 ms. Number of resets before locking the CCU is 2.

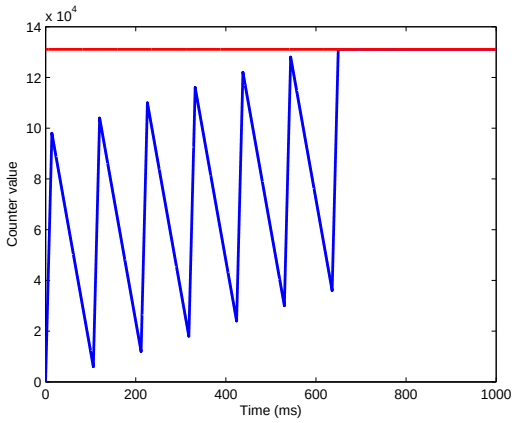

Figure 7.16: Function of the hardware monitor when time between failures in the central control unit is 80 ms. Number of resets before locking the CCU is 6.

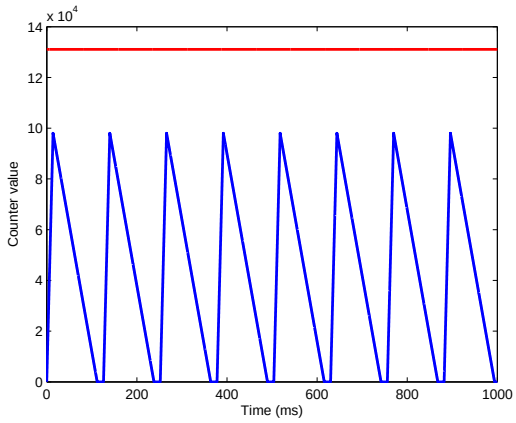

Figure 7.17: Function of the hardware monitor when time between failures in the central control unit is 100 ms. The CCU relaxes after each reset and is therefore not locked.

7.2.3 Development platforms

The platform that is used by the main CCU is the Texas Instruments Hercules development board TMS570LS3137 [6] on which the reference system was implemented [5]. The backup system is implemented on a Texas Instruments Hercules Launchpad TMS570LS04 [26]. The main reason for using the launchpad board is that it is compatible with the reference implementation, it is recommended for the ISO 26262 standard and has all the requirements that is needed for this thesis (supports CAN protocol, lock-step etc).

7.2.4 Electronic control unit for the distributed-brake

The distributed brake electronic control unit (ECU) is implemented to simulate the steer-by-brakes and distributed brake system for the drive-by-wire system. Figure 7.18 shows a picture of the implemented distributed brake ECU. The ECU has four servos controlled by pulse width modulation (PWM) from a Texas Instruments Hercules Launchpad TMS570LS04. When the steer-by-brakes functionality of the distributed brake ECU is activated in the CCU, the system distributes the brake power in accordance to the current steering angle. This simulates the vehicles control of using the brakes to turn.

Figure 7.18: Picture showing the prototype of the brake ECU that simulates the functionality of the distributed brakes and steer-by-brake.

8 Timing analysis

Since implementation of fault tolerance decreases a systems timing performance, this section gives a analysis of the *run time* of each task in the CCU of the existing prototype and the fault-tolerant prototype. The *run time* in this section is defined as the time from start to finish of a task. The run time was measured by setting an I/O-pin, on the development board, high when a task starts executing and setting it low when the same task stops executing. Each task, except in Figure 8.2, executed exclusively to ensure that the presence of other tasks did not affect the results.

The scheduler in FreeRTOS uses static priorities to decide which task to execute [7]. Every task is then given a short time slot in which it may run. When two or more tasks with same priority are competing for execution, round robin algorithm is used decide which one to choose.

8.1 Existing prototype

The brake and throttle tasks are similar in their functionality. Both of them reads the sensor value from the ADC and uses functions to convert the thottle/brake value to readable value for the ECUs. The task responsible for steering activates when the user changes the position of the steering wheel. The steering sensor then sends two pulse trains on two I/O pins on the development board. The difference in phase between the two pulse trains tells whether the steering wheel is moved to the left or to the right. The processed sensor values are read in the CAN task and handled for sending via CAN to the ECUs [5].

The LCD task reads the throttle, brake and steering data and presents the results onto a display.

The steering task activates on interrupt from the steering sensor, i.e., the steering wheel. As seen in Figure 8.1, the run time for the steering task is 0.3 μs.

Task for throttle, brake and CAN executes periodically every 10 ms, which can be seen in Figure 8.3. As seen in Figure 8.2, the run time for throttle and brake tasks is 8.5 μs and CAN task is 3 μs.

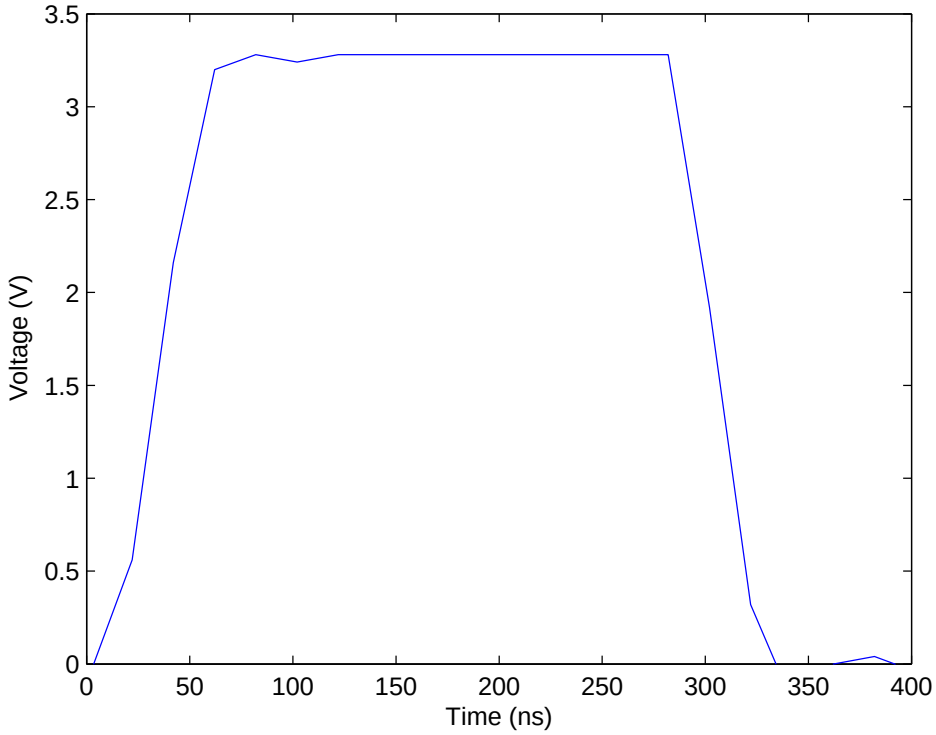

Figure 8.1: Run time for the steering task in the existing prototype.

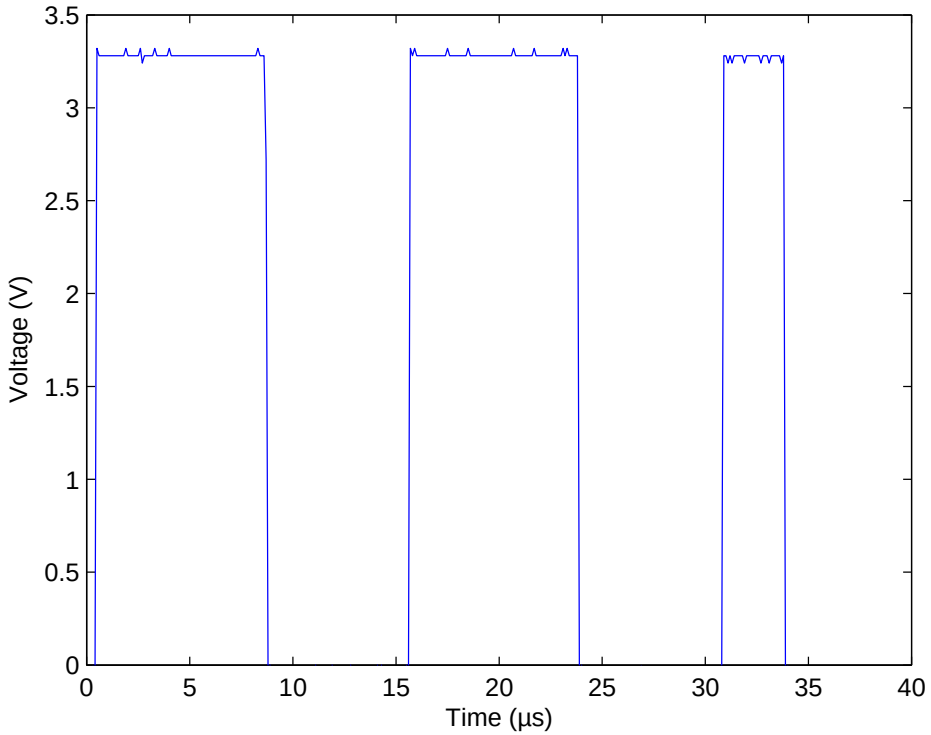

Figure 8.2: Run time for the throttle task(leftmost), brake task(middle) and CAN task(rightmost), in the existing prototype.

The LCD task is responsible for writing current sensor information to the LCD display. Since this is more time consuming, the LCD task only has to execute once every 250 ms. As seen in Figure 8.3, the run time for LCD task is approximately 52 ms.

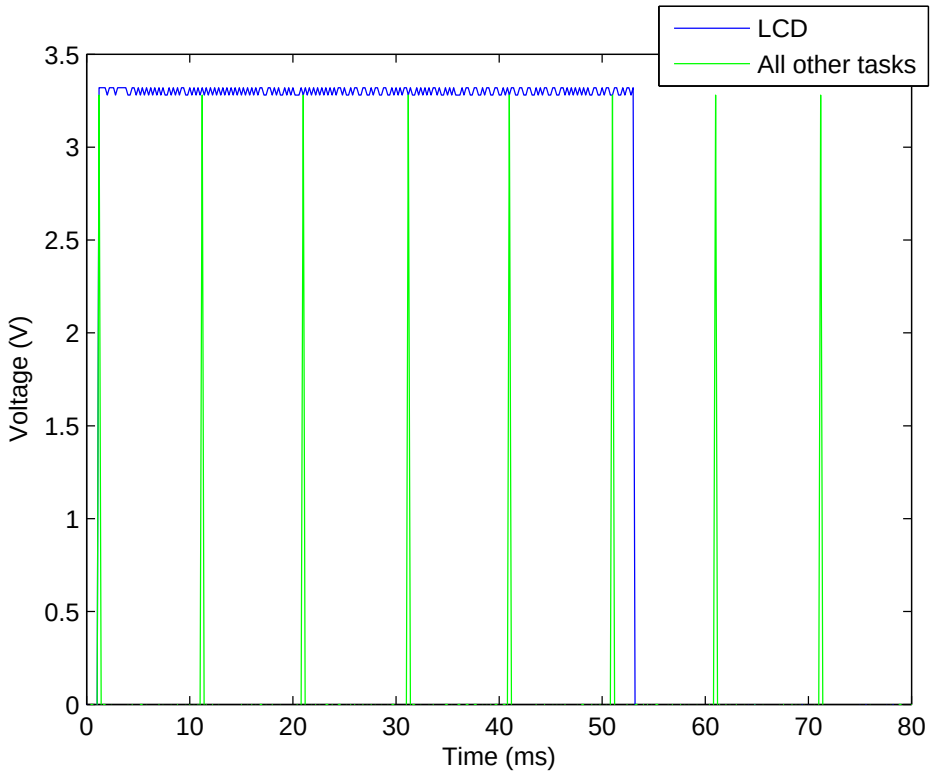

Figure 8.3: Overview of the run times for all periodic tasks in the existing prototype, i.e., throttle task, brake task and CAN task referred to as other tasks and the LCD task.

There is also some time in between each task since the operating system has to store and load the stack for each task. The overhead time for switching from one task to another is approximately 7 μs. An overview of the timing for the different tasks are shown in table 8.1.

Table 8.1: An overview of the run time for the tasks in the existing prototype.

Task name	Priority	Measured run time
Steering_task	Interrupt	0.3 μs
Brake_task	1	8.5 μs
Throttle_task	1	8.5 μs
CAN_task	1	3 μs
LCD_task	1	52 ms

8.2 Fault-tolerant prototype

This section provides the timing analysis of the tasks in the fault-tolerant prototype. The fault-tolerant prototype has divided the CAN task into a receiver task (interrupt based) and transmitter task (CAN_TX and CAN_RX). The watchdog task is also implemented to detect any faults in the system. The different tasks are explained in detail in section 7. The steering task is not altered and no further analysis is made. The run time for the steering task can be seen in Figure 8.1.

Figure 8.4 shows the run time for the brake and throttle tasks of the fault-tolerant prototype. The brake and throttle task reads and processes the sensor value five times to ensure temporal redundancy between the sensor reads. Between each read, the task is preempted, enabling other tasks to be executed in the meantime.

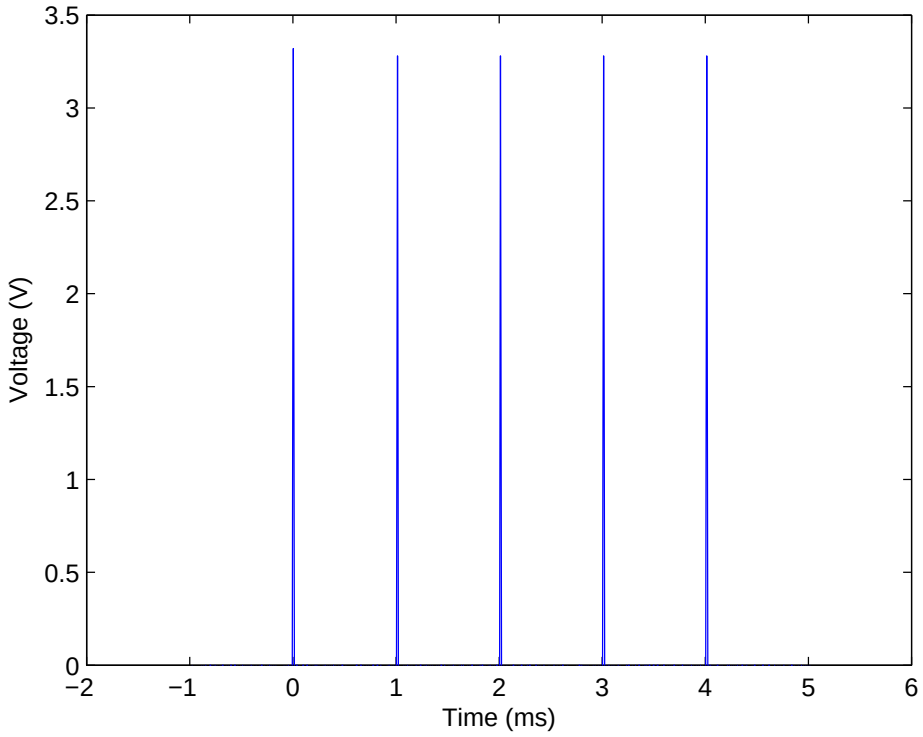

Figure 8.4: Overview of the run time for the Brake (or Thottle) task in the fault-tolerant prototype. The sensor is reading the values 5 times for the median voter with a delay of 1 ms between each read.

Figure 8.5 shows the run time of one sensor value reading for the throttle and brake task. Both tasks are similar in structure and therefore the run time is practically the same. The time between the two tasks is overhead time within the FREERTOS kernel.

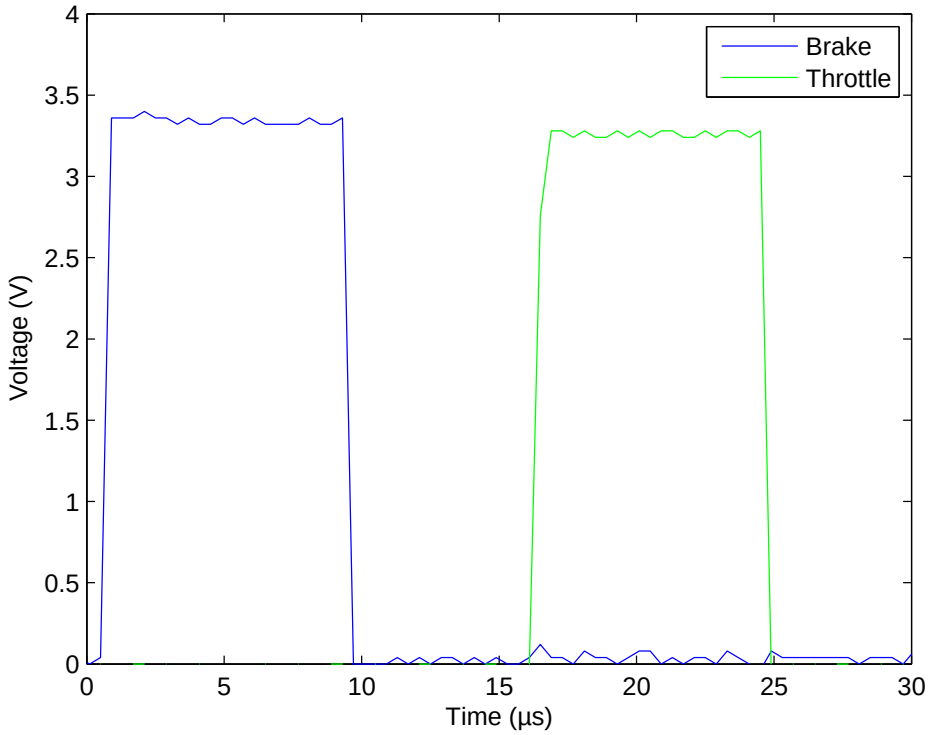

Figure 8.5: Detailed run time for one sensor reading for the brake and throttle tasks in the fault-tolerant prototype.

The CAN task has been divided into two tasks in the fault-tolerant prototype, *CAN_RX* and *CAN_TX*. The CAN_RX task is executed when a valid CAN message is received. An interrupt based function activates when CAN messages are being received and checks if the CAN message is of interest. The run time of the CAN_RX task can be seen in Figure 8.6. The CAN_TX task is executed periodically and sends values for brake, throttle and steering to the different ECUs via the CAN bus. Figure 8.7 shows the run time for the CAN_TX task. The run time for CAN_TX differs somewhat depending on if the primary steering or steer-by-brakes functionality is active. From the figure it can be seen that the run time is 1 μs longer for when the steer-by-brakes functionality is active compared to the primary steering.

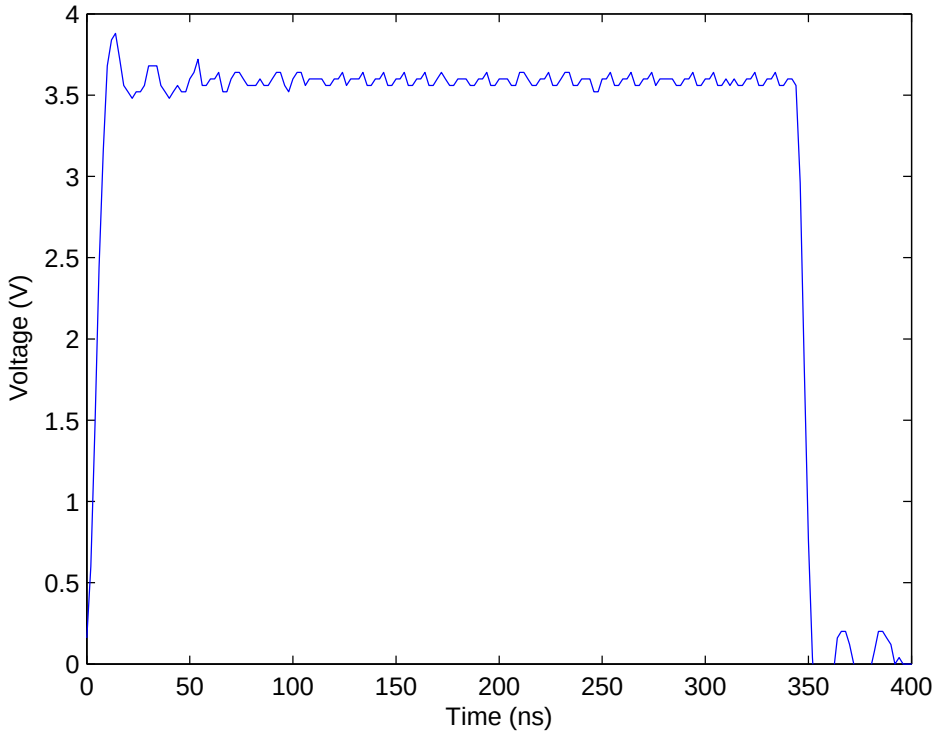

Figure 8.6: Run time for the CAN_RX (CAN receive) task in the fault-tolerant prototype.

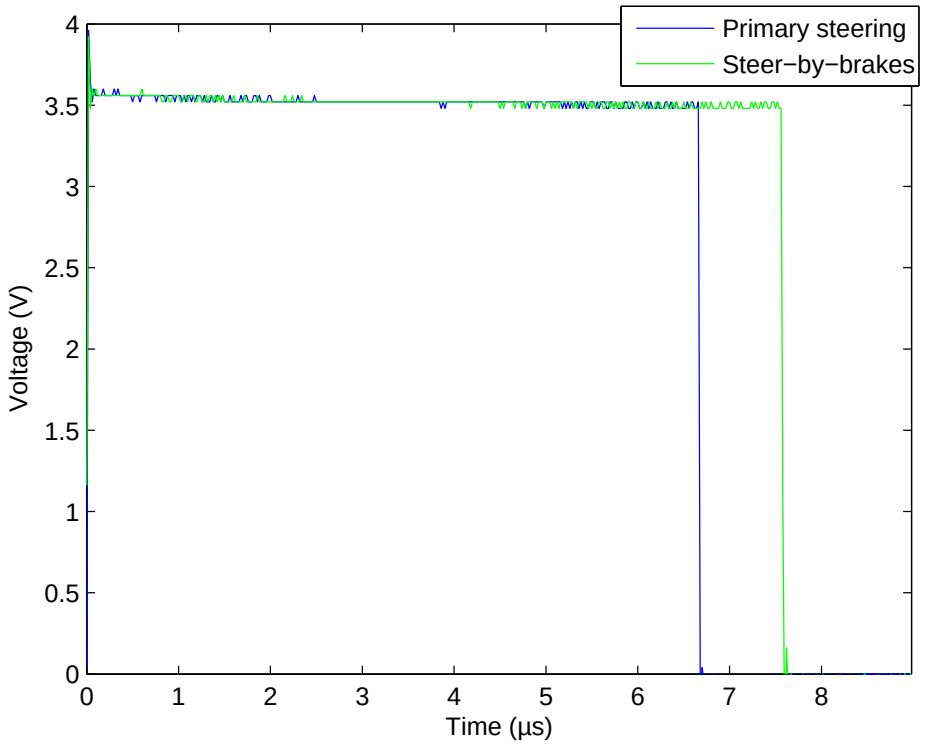

Figure 8.7: Run time for the CAN_TX (CAN transmit) task in the fault-tolerant prototype, with the primary steering and with the primary steering failed and steer-by-brakes active.

The run time for the watchdog task can be seen in Figure 8.8. The run time for the watchdog is 0.7 μs.

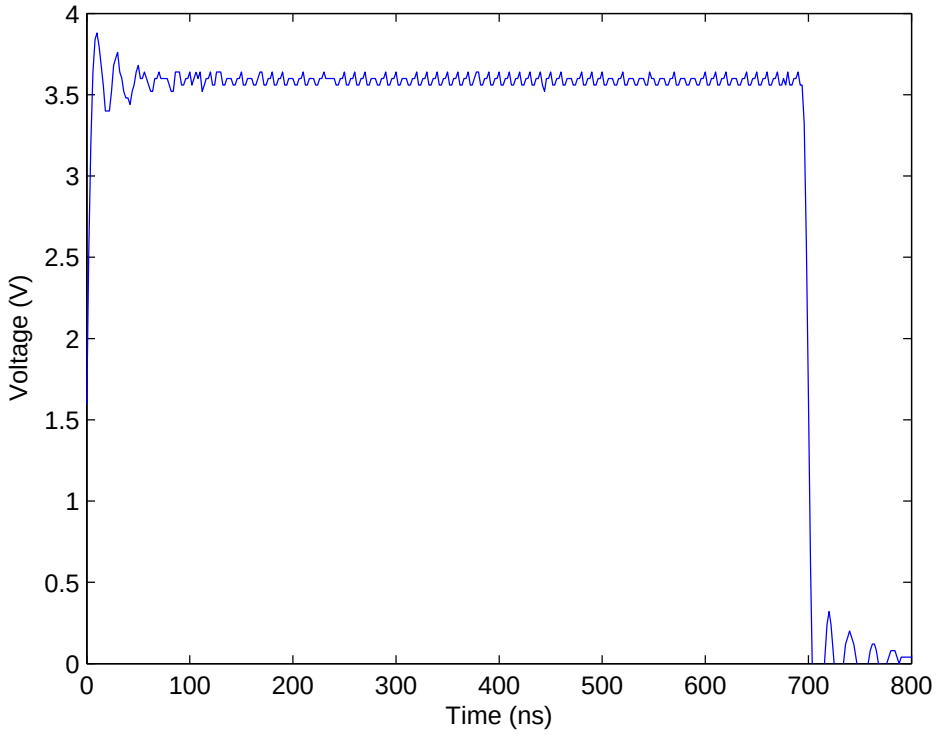

Figure 8.8: Run time for the watchdog task in the fault-tolerant prototype.

The run time of all tasks including the LCD task (only executed on the main CCU) is shown in Figure 8.9. Notice that all tasks are periodic and the LCD task finishes before its intended deadline of 250 ms. The figure shows that all tasks meet their deadlines and the system is stable. The LCD task is finished executing around 52 ms and the other tasks continue their execution at a rate of 10 ms. The LCD task is executing again at time 250 ms. It should be mentioned that the LCD task is not utilizing the CPU for 52 ms. The LCD_task consists mainly of waiting time when communicating with the LCD display [5]. Other tasks can execute in the meantime as the LCD_task waits for response from the display.

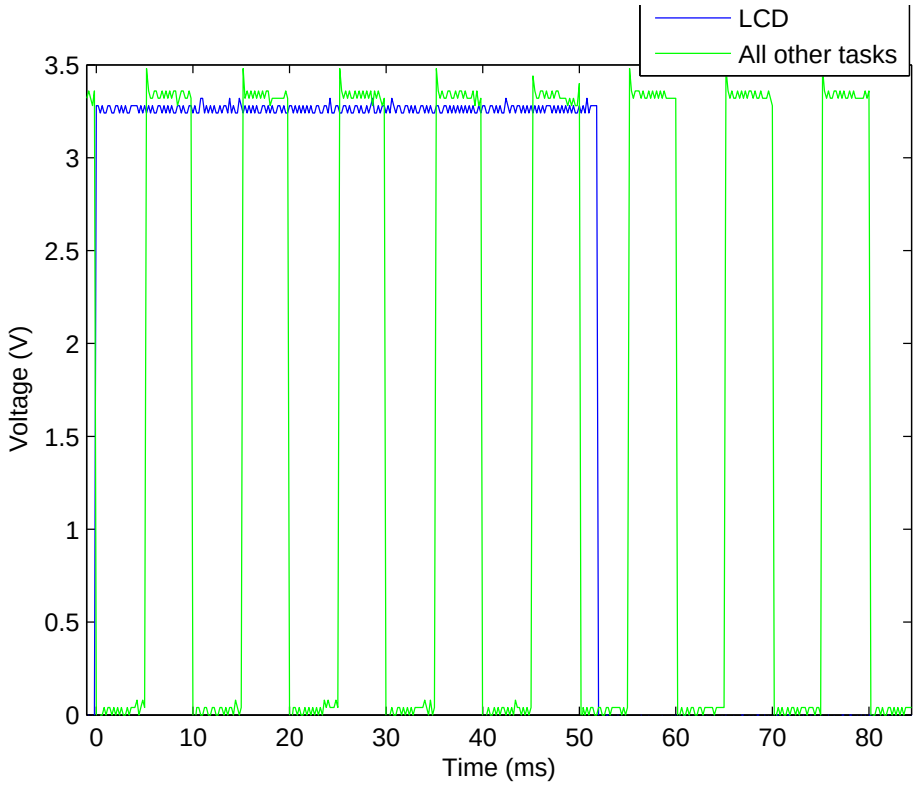

Figure 8.9: Run time for all tasks, in the fault-tolerant prototype. Blue line indicates the LCD task, green line indicates all other tasks.

Table 8.2 shows an overview of the run time for the different tasks. The task holding the longest run time is the LCD task, followed by the brake and throttle task and last the CAN_TX task. The other task are shown to be negligibly small. It should be noted that neither the LCD task, throttle task or the brake task are fully occupying the CPU load when executing. Large parts of the tasks are only wait statements ordering the task to wait a number of milliseconds. For example, the brake and throttle tasks processes the read value and then waits for 1 ms before reading again. Other tasks can run during the time it is waiting for the next value read.

Table 8.2: Overview of the run time for the tasks in the fault-tolerant prototype.

Task name	Priority	Measured run time
Steering_task	Interrupt	0.3 μs
Brake_task	1	4 ms
Throttle_task	1	4 ms
CAN_RX	Interrupt	0.35 μs
CAN_TX	1	6.8 μs to 7.8 μs
Watchdog_task	1	0.7 μs
LCD_task	2	52 ms

9 Test and verification

This section explains the methods used to test and verify the functional safety mechanisms of the fault-tolerant prototype. The tests includes Software Implemented Fault Injections (SWIFI) and pin-level fault injection. Figure 9.1 shows where the faults were injected. The fault injections were the following:

1. Corrupting the sensors value for brake and throttle. By disconnecting the brake and throttle sensors, the ADC will no longer be provided with stable signals. The input to the ADC will therefore be affected by high-frequency noise from rest of the electronics. The ADC will still output a signal, though with a frequently varying voltage level. Since the multiple reading of values is separated by 1 ms, the discontinuities in the sensor value will be detected by the median voter.

2. Corrupting sensitive memory banks. The processed data are stored in multiple memory locations. By manually, in software, changing the values of one of these, the functionality of the majority voter is tested. Since two correct values out of three still makes a majority, no error is raised. By changing two out of three values, the majority voter can not establish a majority and it will notify the watchdog task that an error have been encountered.

3. SWIFI of the read sensor values. By changing the sensor values of throttle and brake, during the time they are being read, the functionality of median voter can be tested and verified. To enable this test, software have to be added to the function that reads the sensor values for throttle and brake. By introducing faults so that 3 out of 5 sensor reads differ more than 300 between each other, the median voter should detect this fault and notify the watchdog task.

4. Fault injection at system level. To enable steer-by-brakes, there has to be some way to detect that the steering ECU or the steering actuator have failed. This functionality is supposed to be implemented in the steering ECU itself. Since it is not covered by the scope of this thesis to implement functional safety for the ECUs, there is a button connected to both CCUs. When this button is pushed down, the CCUs will assume that steering ECU have detected an error. The active CCU will then enable steer-by-brakes.

   Fault injection to test the duplex redundancy is made by disconnecting one of the CCUs from the CAN bus. This will be detected by the other CCU which will take over execution. This can be accomplished since both CCUs are monitoring the CAN bus. For example, if main CCU is disconnected from the CAN bus, it can not send any data on the CAN bus. The watchdog task of the backup CCU will notice this when the timer, corresponding to latest received message, of the primary CCU runs out. The backup CCU will then assume that main CCU has malfunctioned and take over the execution.

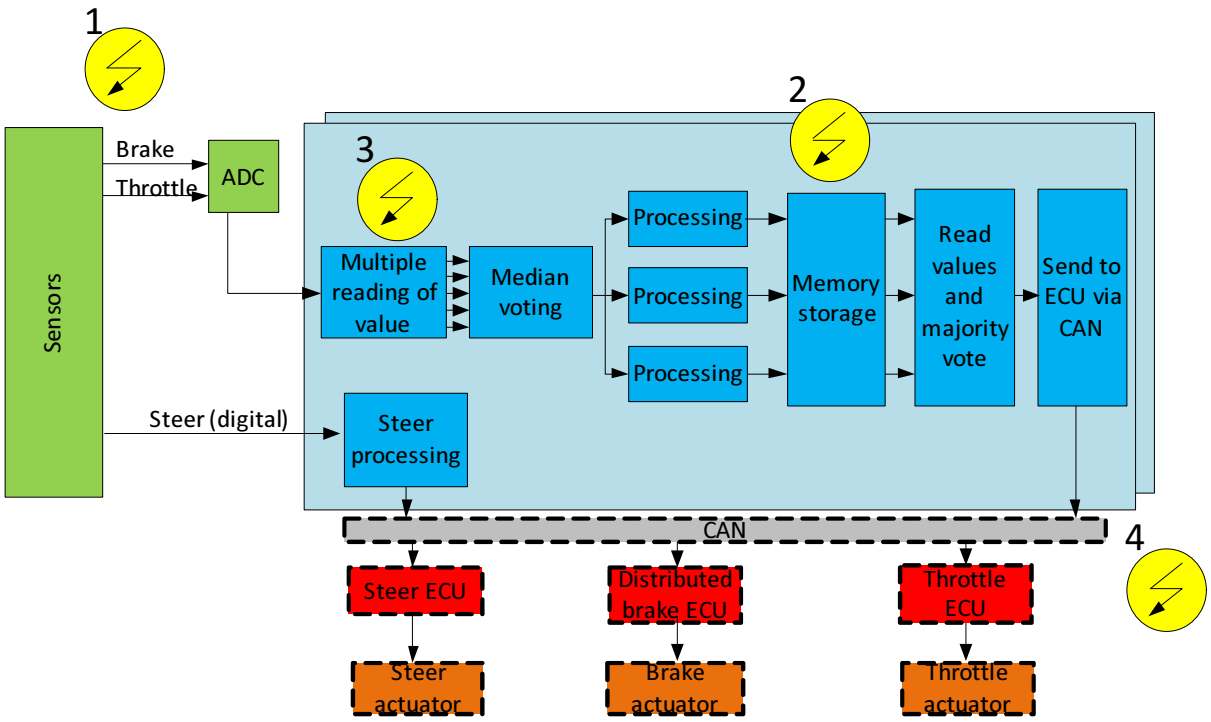

Figure 9.1: Overview of the functional testing of the fault-tolerant prototype. The fault injections are marked with a yellow circle with a number.

9.1 Software implemented fault injections

Fault injections are emulated in the software code and at the input of the system. Code for testing is made to ensure the verification of units in the system.

9.1.1 Testing the functionality of hardware monitor

By connecting three different buttons to three general purpose input/output (GPIO) pins on the main CCU representing 100 ms, 60 ms and 20 ms of fail rate respectively, one could analyse the functionality of the hardware monitor. A new task was created in order to simulate this behaviour. The task was named error_task, and when the button representing 100 ms fail rate is held down, the error_task will activate the error pin after 100 ms. Since 100 ms is above the accepted limit of the hardware monitor this will result in a continuous restart loop until the button is released. When the button of 60 ms is held down the hardware monitor will restart the CCU a couple of times until the hardware monitors counter reaches maximum value and locks the CCU. When the button of 20 ms is held down the hardware monitor will restart the CCU once and then lock the CCU. An important assumption is that the error_task starts executing as soon as possible. If for example, the error_task starts executing 10 ms after restart, this would add to the time period of the fail rate. These tests confirms that the functionality of the hardware monitor works as intended.

9.1.2 Testing the functionality of majority voter

Majority voter protects the values of throttle and brake from being altered during the time after they have been processed and before they are sent to the ECUs. By changing one of the values,

which are being voted, during this time slot one can see that the system remains unchanged. If two values are changed, no majority can be established and the watchdog task will be informed to activate the error pin. As an effect of this the system will be restarted by the hardware monitor.

9.1.3 Testing the functionality of median voter

The functionality of the median voter was tested and verified by changing the sensor values of throttle and/or brake in the meantime as they were read. The changes made to the reading of the sensor values involved increasing or decreasing the values. If 3, or more, out of 5 readings differ with more than 300, the median voter will notify the watchdog task that an error has occurred. If 2, or less, out of 5 readings differ with more than 300, a fault is detected but can be prevented from propagating to an error (right value is provided). This fault injection was implemented by changing the software of the function responsible for reading the sensor values for brake and throttle.

9.2 Pin fault injection

By disconnecting the sensor for throttle and brake, the input to the CCU will get affected by high frequency noise. This will result in that the values of the sensors will look discontinues over time.

By disconnecting one of the CCUs from the CAN bus, it is possible to test how the other CCU responds. If the backup CCU gets disconnected, the main CCU acts as normal, but prevents the watchdog task from reporting any errors to the hardware monitor. If the main CCU gets disconnected, the backup CCU takes over execution and prevents the watchdog task from reporting any errors to the hardware monitor. The prevention of reporting errors to the hardware monitor is active when there is only one functioning CCU left in the system. This is due to that the steering value will be lost if the last functioning CCU gets restarted.

10 Discussion

In this thesis, we have suggested an architecture for a fault-tolerant drive-by-wire design and compared it to a non-fault-tolerant reference design. The architecture for the fault-tolerant design implements a distributed brake system with steer-by-brakes functionality (brake-and-steer system) and dual modular redundancy for a central control unit (CCU).

We can clearly see that by introducing dual modular redundancy for the CCU, the reliability increases significantly. With the failure rate for the CCU being $5 * 10^{-}7$ failures per hour, a reliability of approximately 99.99% lies within the lifespan of a car for the fault-tolerant design whilst the non–fault-tolerant design does not achieve a reliability of 99.9% for the same failure rate after one year. This observation is represented in Figure 6.5.

Reliability for the brake-and-steer system achieves a 99% reliability after approximately 14 years with a failure rate of $5 * 10^{-7}$ failures per hour. The reference design achieves a reliability of 99% after approximately 1 year with the same failure rate, as can be seen in Figure 6.12.

We can conclude that the suggested fault-tolerant design achieves a significantly higher reliability compared to the non-fault-tolerant reference design. The fault-tolerant design achieves a reliability of 99.99% within the lifespan of a car, concidering a failure rate of $5 * 10^{-}7$ failures per hour.

An analysis was made to show the intersection points of the steer and brake system for the reference design and fault-tolerant design. The analysis is shown in Figure 6.13. When lowering the failure rate, the fault-tolerant design gets a larger reliability improvement compared to the reference design. This is important from a designer perspective, where the reliability increase for the fault-tolerant design compared to the reference design is shown to improve for smaller failure rates.

A sensitivity analysis was established with respect to the coverage factor for the fault-tolerant design. The analysis shows that the duplex CCU needs to fulfill a coverage factor of at least 95% and a failure rate of 10^{-9} failures per hour to meet the target reliability of five-nines (99.999%) in a vehicle lifespan of 10 years. For the brake-and-steer system a coverage factor of 97% is needed to reach the target reliability in the vehicle lifespan for the same failure rate.

Since the reference design cannot guarantee a predicted behaviour in case of failure, there was no mechanism to guarantee safety. The fault-tolerant design has safety mechanisms to ensure transition to a safe state if the system suffers from a failure. In section 6.3, we describe how safety can be implemented in the fault-tolerant design according to the ISO 26262. Figure 6.17 shows how the safety of the throttle system in the fault-tolerant design was developed to ensure safety. We suggest that when the CCU system or brake-and-steer system is in degraded mode, a warning lamp should be lit to notify the driver. The driver is then assumed to stop the vehicle or drive to the closest workshop. Figure 6.18 and 6.19 shows the probability of the CCU and brake-and-steer system, respectivley, to ensure safety. The calculation of the safety can either be estimated using two random variables for transition from the degraded state to the critical-failure state and fail-safe state. One can also estimate the maximum slow-down time for the driver or the system to put the vehicle to a stop.

The purpose of making the prototype of the fault-tolerant design was to demonstrate how a system can be implemented when considering fault tolerance by following the guidelines of the ISO 26262. The contributions covered in this thesis, to make the previous prototype more tolerant to faults, is the implementation of a hardware monitor, distributed brake system with steer-by-brakes functionality and safety mechanisms in the CCU to handle fault detection for sensor values.

Two different designs for the hardware monitor is suggested; distributed-reset design and self-reset design. The one which was implemented in the prototype was the self-reset design. One important aspect which advocates the self-reset design is that the occurrence of a non-covered fault will not affect the execution of the other CCU.

We do consider the hardware monitor as ideal. This means that the failure rate is not taken into account when making the analysis for safety and reliability of the whole drive-by-wire system. The design of the hardware monitor in this thesis gives a good layout of how a monitor can be designed to detect different kind of faults and prevent them for further causing damage to the system.

The distributed brake system was implemented to demonstrate the functionality of the steer-by-brakes and distributed brake system. The system receives the brake values of all four brakes from the CCU, were the CCU makes the decisions regarding how each brake will be affected by the steer-by-brakes functionality.

11 Conclusion and future work

Design of a fault-tolerant drive-by-wire system was developed and compared to a non-fault tolerant design to show the significant increase in reliability and safety when fault tolerance is introduced. This thesis proposes a cost effective fault-tolerant design consisting of a distributed brake system with steer-by-brakes functionality and dual modular redundancy for the central control unit.

We have also provided contributions to an existing non-fault tolerant prototype. Considering the design choices of the fault-tolerant design, we have implemented a hardware monitor, distributed brake system with steer-by-brake functionality and safety mechanisms in the central control unit. We have further shown how testing of the functional correctness for both prototypes was performed.

Development of the system architecture was influenced by the consideration that the system would be implemented in a more realistic future design. With this consideration, the fault-tolerant prototype has a good layout for further development, such as implementation of real actuators and sensors for a full-sized vehicle. For future development it would also be of interest to develop real ECUs for throttle, brake and steering to be used for the CCU in the fault-tolerant prototype.

All communication works through one CAN bus. Since the CAN bus is a single point of failure in the system, we suggest that this part should be made redundant. One solution for this is to have a redundant CAN bus working as a duplex system. Failure in one of the CAN buses will still provide communication between the different subsystems. Another solution is to use a star-net between the modules in the system. When a failing CAN-line is detected (due to cable brake or babbling idiot phenomena), the CAN communication can be redirected to take another path in the network of the ECUs and CCUs. The second solution is also more cost and weight effective.

References

[1] N. Estep, J. Petrosky, J. McClory, Y. Kim, and A. Terzuoli, "Electromagnetic interference and ionizing radiation effects on cmos devices," *Plasma Science, IEEE Transactions on*, vol. 40, no. 6, pp. 1495–1501, June 2012.

[2] T. D. o. E. E. Ma and P. S. N. L. Dressendorfer, *Ionizing radiation effects in MOS devices and circuits*, Jan 1989. [Online]. Available: http://www.osti.gov/scitech/servlets/purl/6945296

[3] E. Snow, A. Grove, and D. Fitzgerald, "Effects of ionizing radiation on oxidized silicon surfaces and planar devices," *Proceedings of the IEEE*, vol. 55, no. 7, pp. 1168–1185, July 1967.

[4] "Road vehicles - functional safety (iso 26262)," Geneva, Switzerland, 2011.

[5] A. Angerd and A. Johansson, "Design and implementation of a central control unit in an automotive drive-by-wire system."

[6] Tms570ls31x hercules development kit @ONLINE. Texas Instruments. Accessed: 2014-02-01. [Online]. Available: http://www.ti.com/tool/tmds570ls31hdk

[7] Freertos - a free rtos for small real time embedded systems @ONLINE. Accessed: 2014-03-11. [Online]. Available: http://thetoolchain.com/datasheets/FreeRTOS_manual.pdf

[8] 8-bit atmel microcontroller with 128kbytes in-system programmable flash @ONLINE. Atmel. Accessed: 2014-02-01. [Online]. Available: http://www.atmel.com/Images/doc2467.pdf

[9] N. R. Storey, *Safety Critical Computer Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996.

[10] A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[11] A. W. Strong and I. X. (e-book collection), *Reliability wearout mechanisms in advanced CMOS technologies*. Piscataway, NJ; Hoboken, NJ: Wiley-IEEE Press, 2009. [Online]. Available: www.summon.com

[12] P. Johannessen, "On the design of electrical architectures for safety-critical automotive systems," Ph.D. dissertation, Chalmers University uf technology, department of Computer science and Engineering, 2004.

[13] R. Johansson, "On distributed control-by-wire system for critical applications," Ph.D. dissertation, Chalmers University of technology, department of Computer science and Engineering, 2005.

[14] J. Brazendale, "Iec 1508: Functional safety: Safety-related systems," in *Software Engineering Standards Symposium, 1995. (ISESS'95) 'Experience and Practice', Proceedings., Second IEEE International*, Aug 1995, pp. 8–17.

[15] D. Skarin, "On fault injection-based assessment of safety-critical systems," Ph.D. dissertation, Chalmers University uf technology, department of Computer science and Engineering, 2010.

[16] T. Pilutti, "Vehicle steering intervention through differential braking effects on cmos devices," *Journal of dynamic systems, measurement, and control*, vol. 120, no. 3, pp. 314–, 1998.

[17] ——, "Vehicle lateral control and yaw stability control through differential braking," *Industrial Electronics, IEEE International Symposium*, vol. 1, pp. 384 – 389, July 2006.

[18] J. Demerly, "Method and system for providing secondary vehicle directional control through braking," Oct. 2004, uS Patent 6,808,041. [Online]. Available: https://www.google.com/patents/US6808041

[19] T. A. Peter Alan Lee, *Fault Tolerance - Principles and Practice*, 1990, vol. 3.

[20] "Average vehicle age," 2010, statistics. [Online]. Available: http://www.acea.be/statistics/tag/category/average-vehicle-age

[21] "Polk finds average age of light vehicles continues to rise," August 2013. [Online]. Available: https://www.polk.com/company/news/polk_finds_average_age_of_light_vehicles_continues_to_rise

[22] A. Bobbio and K. Trivedi, "An aggregation technique for the transient analysis of stiff markov chains," *Computers, IEEE Transactions on*, vol. C-35, no. 9, pp. 803–814, Sept 1986.

[23] Ordinary differential equation @ONLINE. Accessed: 2014-05-12. [Online]. Available: http://www.mathworks.se/help/matlab/ordinary-differential-equations.html

[24] R. S. Boyer and J. S. Moore, "Mjrty: A fast majority vote algorithm," in *Computational Logic - Essays in Honor of Alan Robinson*, 1991, pp. 105–118.

[25] Xc9572xl high performance cpld @ONLINE. XILINX. Accessed: 2014-05-26. [Online]. Available: http://www.ti.com/tool/launchxl-tms57004

[26] Hercules tms570ls04x/03x launchpad evaluation kit @ONLINE. Texas Instruments. Accessed: 2014-02-01. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds057.pdf