FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Validation Middleware for Mixed Criticality Networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Studiums

Technische Informatik

eingereicht von

Thomas Mair
Matrikelnummer 0425444

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: O.Univ.Prof. Dr.phil. Hermann Kopetz
Mitwirkung: Univ.Ass. Dipl.-Ing. Armin Wasicek

Wien, March 22, 2011

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Thomas Mair
Friedhofweg 6
4800 Attnang-Puchheim

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

....

(Ort, Datum, Unterschrift)

Abstract

This thesis investigates on the secure sharing of information in a mixed-criticality system. Our approach to fulfil these safety and security requirements in such a system is to establish a set of strict rules for the communication between the tasks of the different integrity levels. These rules implement an integrity model and have to guarantee that the information flow between the criticality levels happens appropriately. In many applications it also might be required to use a piece of unreliable information from a low criticality level in a higher one. Communication in this direction is considered as illegal in many integrity models. Therefore a special mechanism is needed to upgrade the integrity of the data. This thesis introduces a mechanism called "Validation Middleware" which upgrades the reliability of data from diverse redundant sources. To achieve this goal, inexact voting techniques are realised which produce a trusted output and define a criteria for determining correct and incorrect from data which is not necessarily identical. These mechanisms were developed and studied in context of the Time-Triggered System-On-Chip architecture. This architecture provides a spatial and temporal firewall between each task by partitioning the system into single autonomous subsystems called the micro components. These subsystems are connected through a deterministic Network-on-Chip which uses so-called encapsulated communication channels to prevent the messages from interfering with each other. These encapsulated communication channels transport messages at a predefined point in time from a single source to one or more destinations. We tested the fault tolerance mechanisms inside the "Validation Middleware" by creating an application from the automotive context which compasses ABS and odometer subsystems.

Our results show that the deterministic behaviour of the Network-on-Chip and the temporal and spatial partitioning of the encapsulated communication channels, combined with the use of Totel's integrity policies, provides a suitable environment for the use in a mixed-criticality application. In addition, we point out the existence of a middleware in the Time-Triggered System-on-Chip architecture which enables upstream communication flows.

Kurzfassung

Diese Arbeit behandelt die sichere Verteilung von Informationen in einem System mit verschiedenen Kritikalitätsstufen. Eine Möglichkeit, diese Security- und Safetyanforderungen zu erfüllen, ist die Einführung von strengen Regeln für die Kommunikation zwischen den einzelnen Anwendungen in den verschiedenen Stufen. Diese Regeln formen eine sogenannte “integrity policy”. In vielen Anwendungsfällen ist es erforderlich unzuverlässige Daten einer niedrigen Stufe in einer höheren verwendet zu können. Ein Informationsfluss in dieser Richtung ist bei Verwendung einiger bekannter “integrity policies“ nicht zulässig. Um dies zu ermöglichen, wird ein spezieller Mechanismus, genannt “Validation Middleware” benutzt, welcher die Zuverlässigkeit von Daten aufwertet, indem diese von diversen redundanter Datenquellen bezogen werden. Damit dies garantiert werden kann, wurden mehrere Voting-Algorithmen realisiert, die ein Kriterium für die Gültigkeit der Daten erzeugen. Die Eingangsdaten müssen nicht unbedingt gleich sein und können voneinander abweichen. Diese Algorithmen wurden unter Verwendung der “Time-Triggered System-on-Chip” Architektur entwickelt und evaluiert. Diese Architektur erzeugt eine räumliche und zeitliche Firewall zwischen den einzelnen Teilsystemen des System-on-Chip, indem sie das System in einzelne autonome Subsysteme, sogenannte “micro components”, partitioniert. Diese Subsysteme sind über ein deterministisches Network-on-Chip verbunden, welches sogenannte “encapsulated communication channels” benutzt, um die Beeinträchtigung der Kanäle untereinander zu verhindern. Diese “encapsulated communication channels” senden Nachrichten zu einem vordefinierten Zeitpunkt von einem Sender an einen oder mehrere Empfänger. In dieser Arbeit werden die fehlertoleranten Mechanismen der “Validation Middleware” anhand eines Fallbeispiels getestet. Dieses Beispiel beschäftigt sich mit den Subsystemen ABS und Kilometerzähler eines Automobils.

Unsere Resultate zeigen, dass das deterministische Verhalten des Network-on-Chip und die räumliche und zeitliche Partitionierung der “encapsulated communication channels”, kombiniert mit dem Benutzen von “Totel’s integrity policy”, eine geeignete Umgebung für eine Anwendung mit verschiedenen Kritikalitätsstufen erzeugt. Zusätzlich zeigen wir die Existenz einer Middleware im Time-Triggered System-on-Chip, welches einen aufwärts erfolgenden Informationsfluss ermöglicht.

Acknowledgements

First of all a very big Thanks to the advisor of this thesis, Univ. Prof. Dr. Hermann Kopetz, who has enabled this work. Many Thanks to Dipl.-Ing Armin Wasicek for his patience, all the answered questions and the many useful hints. Thanks to Dipl.-Ing Roland Kammerer for answering many questions about the hardware and also to Ing. Leo Mayerhofer for his technical aid during my work.

Also I would like to thank my parents for giving me financial and personal support during my whole lifetime and especially during the time of my studying. Last but not least a big thanks to many of my study colleagues whose discussions and group works made a lot of course work easier and more fun.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation

During the last years the complexity in the semiconductor industry has increased dramatically. A modern x86 processor contains of billions of transistors and the number is still increasing. The “end“ of Moore’s law has been predicted and in order to continue it, one solution is to build multi-core processors. The resulting concurrency by using parallel computing increases complexity. But this is not the only field where this increase of complexity starts to be a problem. In embedded systems and especially in automotive and avionics equipment (leveraged by software) a tremendous increase can be observed. For example, a modern luxury car has up to 80 embedded microcontrollers installed, which control most of the processes inside the car. Also about 30% of the overall costs of a modern car are produced by electronics, a number which is likely to increase with future applications like steer-by-wire. [BCWW07]

With more embedded microcontrollers the network degree rises and corresponding to it the need in improving security and safety becomes particularly important. One key in achieving this, at reasonable cost, is to keep the system as simple as possible. On the contrast to, the steady increase of functions in modern products, this is exactly the opposite of what happens nowadays. It is nearly impossible to understand or predict the behaviour of a System-on-Chip (SoC) which contains of more than a billion transistors. [Kop08] A key principle to achieve simplicity in system design is a higher level of abstraction and the partitioning of the system into single functions. It is not the embedded system itself that has to be simple, but it’s model. In these models, irrelevant detail information needs to be omitted to produce an easy understandable description of the system for a specific purpose. In the automotive area a modern approach to reduce complexity is to put the different car functions on one single chip. Contrary to the traditional setup, where these components are deployed separately on the car. Although they are on the same silicon die the functions are still completely autonomous and concurrent. All these separate elements communicate through a NoC. This way of designing such a system is already used by a number of important SoC architectures. [BM06]

In this thesis we will use a special implementation of a SoC called the Time-Triggered System-On-Chip (TTSoC) [Pau08] [ES07] which provides a component based design approach to reduce complexity and to enable composability. The TTSoC realizes this by decoupling the communication infrastructure from the computational units. It also provides determinism with respect to the communication interface, by using a sparse global timebase (see section 2.1) and a priori defined communication schedule.

By putting all these functions on one single chip, it is needed in many situations of system design, that these tasks can have different criticality levels. Consequently, it must be ensured that a fault in a task with low criticality never influences one with a higher level. [TBDP00] For example, a design fault in the audio system should never influence the brakes. One way of guaranteeing this, is to build critical parts of the software with the same confidence as the non critical ones. But not only does this increase complexity, it also makes the system very expensive. Another way to fulfil these safety and security requirements, are strict rules for the communication between the tasks on the SoC. Some models like Bell-LaPadula, Biba or Clark and Wilson implement these communication rules and are the groundwork for systems with multiple layers of security. [BL75] [Bib77] [CW87] These rules are also called *integrity policies* and have to guarantee a proper way of communication in the security domain on the SoC.
By constantly following the guidelines of these rules and by using the properties of the TTSoC with a deterministic communication interface and temporal and spatial partitioning, the concept is suitable for the use with mixed criticality applications.
In addition to this it is possible to show, that the modular design of the TTSoC architecture implements the requirements of the "MILS Seperation Kernel" by design. [WESK10] The Multiple Independent Layers of Security (MILS) architecture is an industry-ready approach to facilitate temporal and spatial partitioning, which is based on concepts introduced by Rushby. [Rus81]

1.2 Problem Statement

This thesis investigates on the secure sharing of information in a mixed-criticality system. As the communication in such a system has to follow strict rules, a communication model needs to be chosen which checks the integrity in an one-way information flow.

In a paper published at the *University of Toulouse* in 2009 [LDPA09] (see section 3.1), this is implemented by using Totel's integrity model as a basic concept for communication in a mixed-criticality application. There the take-off and maintenance procedures for new aircraft generations are analysed, in which communication between the low secure off-board computer and the high secure on-board computer is carried out. As this is considered to be illegal because of the integrity rules of Totel's model [TBDP00] (see section 2.2) the reliability of the data has to be increased by using a so-called Validation Object (VaO). This VaO needs diverse redundant input data which is produced by using untrusted Commercial off the Shelf (COTS) hardware such as *Windows* or *Linux* for the execution of the safe user program. The resulting output of both executed versions of the program is compared by using a decision algorithm inside the VaO.

One of the main problems of this work is that a determinism of both operating systems has to be assumed. This is not provided in reality, because both operating systems use at least different preempting scheduling algorithms. Because of this, it is likely to happen that the comparison of the data may fail. This thesis investigates on how this determinism issue can be solved and in addition focuses on the realisation of different decision algorithms to establish a consensus between the redundant subsystems.

1.3 Contribution

Mixed criticality is the concept of allowing applications at different levels of criticality to interact and co-exist on the same computational platform. In this thesis, Totel's model [TBDP00] is used as an integrity policy combined with the TTSoC as a basic architecture. Because of this, the message exchange in the spatial and temporal domain can be guaranteed by the basic attributes of the TTSoC. Therefore the security attributes of Totel's model like the integrity kernel and the operating system micro-kernel can be replaced. The TTSoC implementation provides process isolation mechanisms by design and the realization in hardware hardens the system against many kinds of security attacks. [Hub08] Additionally, the NoC used in this architecture establishes encapsulated communication channels, which transport messages at a predefined point in time from a single source to one or more destinations. Because of the established global time base and a stable initial state [Kop97] a *deterministic* behaviour of the components inside the TTSoC is guaranteed. This determinism facilitates validation and enables voting, which is needed in order to perform checks over the output of redundant sources.

The major contribution of this thesis is the design of a so-called Validation Middleware (VaM) which allows information flows from a low- to a high security level. Communication in this direction is considered as illegal, because of the integrity rules of Totel's model (see section 2.2) But this kind of upward information flow is in fact needed in many situations of system design.

To upgrade the integrity of the upstream information flow the data has to be based on redundant and possibly *diverse* sources. [AC77] The most common and intuitive way of doing this is a majority voting algorithm, but in some situations these inputs need to be compared and determined to be correct, even if the data deviates from each other. This raises the need for an inexact voter, which defines a criteria for determining correct and incorrect inputs from data which is not necessary identical. In order to do that, a method has to be found that can be used in as many fields of interests as possible and is easily adjustable to different application scenarios.
This thesis analyses anomaly detection algorithms in order to detect system failures or possible fraud in an application where dependability is important. All the different approaches are compared in simplicity, runtime and efficiency by using a simulation environment, with context to the automotive area.

This work combines the deterministic behaviour of the Time-Triggered Network-On-Chip (TTNoC) with Totel's integrity policy. It shows that this combination provides a suitable environment for the use in a mixed criticality application. In addition the possible existence of a middleware in the TTSoC architecture is pointed out, in order to enable upstream communication flows. Therefore a flexible and reuseable approach has to be found which is applicable in different user applications.

1.4 Structure of this Thesis

Chapter 2 gives the background that leads to the implementation of fault tolerant mechanisms of the VaM, by using the TTSoC as a basic platform. First it explains in detail, how the fundamental attributes of the TTSoC like strict partitioning into reusable subsystems and the communication through deterministic encapsulated communication channels fit into the needed security concepts. Then some basic integrity models are given like the Bell-LaPadula, Biba, Clark and Wilson or Totel's model. These models establish rules for up- and downgrading information flows in a system which is based on integrity levels. Next the basic concept of the MILS architecture is mentioned which was developed to make the certification of high integrity systems easier. Then the fundamentals of N-version programming are given, whose concept provides criteria for the creation of redundant and diverse inputs. Finally the basics about anomaly detection are discussed, which are used to provide fault tolerant mechanisms in order to update information integrity.

Chapter 3 examines the related work that has been done in this field. First it mentions a work of the *University of Toulouse* which tries to improve the take-off and maintenance procedure of an aircraft, by connecting unsafe COTS hardware onto a secure airplane. Next some NVP experiments of the *University of California* are given and finally a work of the Technical University of Vienna is mentioned where the TTSoC architecture is compared to the MILS architecture.

The following chapter 4 is dedicated to the VaM and specially to the system model that has lead to it. The focus also lies on the integrity policy modifications needed for a correct information flow. Finally the VaM is discussed and the anomaly detection algorithms which are used for it's implementation are explained.

The automotive case study, in order to test the different algorithms is covered in the following chapter 5. It explains the basic structure of the system and the interaction of the different components with each other. Also all stand alone programs like the TORCS car simulation or the fault injection software are described. In order to investigate the security mechanisms of the environment an attack model for the secure subsystems is created.

The results of these simulations are given in the following chapter 6 where the inexact voting algorithms inside the VaM are tested. These tests are carried out by using different fault injection scenarios during runtime.

The last chapter 7 draws conclusions based on these insights. In this context it justifies design choices and outlines their optimality. It also gives an outlook of future work which can be done in this field.

CHAPTER 2

Basic Concepts

This chapter explains the basic concepts that have been used in this thesis and also focuses on technologies, which were used as groundwork for implementing the VaM. First it explains the fundamental ideas about the Time-Triggered Architecture (TTA) and the resulting realization of the Time-Triggered System-On-Chip (TTSoC), which is used as the basic architecture in this thesis. In the following section it explains the concept of the traditional integrity policies like Bell-LaPadula, Biba, Clark and Wilson and Totel's model, which are needed as a groundwork in this thesis. Then the chapter defers to the fundamentals and types of anomaly detect which are needed for the implementation of the VaM. Finally, the basics of fault injection are explained which are needed to test the behaviour of the used algorithms.

2.1 The Time-Triggered Architecture and the Time-Triggered System-on-Chip

The Time-Triggered Architecture (TTA) provides an infrastructure for designing distributed embedded real-time systems. [KB03] The main characteristic is the exact notion of time and the decomposition of the system into clusters and nodes. The main problem in a distributed system is the infinite precision of time, because this fact makes it impossible to order events and create a deterministic behaviour. Kopetz introduced the *sparse timebase* [Kop92] in order to achieve that. This model partitions time into an infinite sequence of durations of activity and silence, as it is depicted in figure 2.1.

With this notion of time it is possible to order events and generate a deterministic behaviour, which enables the creation of a global timebase, where all nodes have access. This timebase is now divided into prior specified periodic rounds. This procedure is called the Time Division Multiple Access (TDMA) schedule and creates a so called timeslot for every single message. With this pre-defined TDMA schedule it is possible to guarantee a worst case delay and therefore a system with a hard real-time behaviour. With this model a collision free network commu-

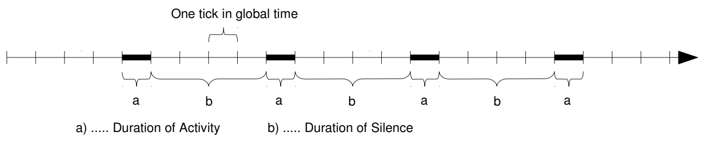

Figure 2.1: The sparse timebase used in the TTA (Kopetz 1992)

nication can be established. In the following section a few systems are described, which use this kind of network communication.

TTA implementations like TTP/A, TTP/C and TTE

TTP/A and TTP/C are two implementations of protocols which have been created at the *Institute of Computer Engineering* at the *Technical University of Vienna*. [EI03] They are using the scheme of the Time-Triggered Architecture (TTA) to establish a collision free bus communication. TTP/A aims on the sector of low cost microcontrollers, for integration of sensors and actuators into a network. It is also standardized under the smart transducer interface specification and offers a number of advantages in technology, cost and complexity management.
On the other instance, TTP/C focuses on highly dependable realtime systems, implements a replicated bus system and a guardian for preventing bubbling idiot failures. It also consists of a membership service to inform the application, if an error in the communication system has occurred. Also a fault tolerant global time base, which is not relying on a central time server is provided.

An other implementation of the *Technical University of Vienna*, which uses the TTA, is the Time-Triggered Ethernet (TTE). [KAGS05] This communication system provides real-time and non-real-time traffic on a single communication architecture by using the IEEE Ethernet standard. Two different traffic categories are provided:

- The standard Ethernet traffic which is event triggered and conform to the widely used IEEE standard.
- The time-triggered traffic which is temporally guaranteed. It provides a deterministic end- to end communication which is scalable and therefore facilitates certification. It provides a global timebase and eliminates error propagation through strong fault isolation mechanisms.

The Time-Triggered System-On-Chip

The TTSoC is a novel architecture for SoC which manages the high complexity of such a system by partitioning it into single components. [Hub08] [ES07] Each component needs a strict interface design which facilitates a pre-verification of each subsystem and makes it reusable. This is very important, because in order to control the increasing complexity in system design a good system model with clear defined interfaces is needed. In the TTSoC every component is completely autonomous and only connected over the trusted TTNoC with other components. This means that there are no hidden channels, which interact with each other. Because of the Time-Triggered (TT) implementation of the NoC it is possible to guarantee a deterministic behavior of the communication system. Determinism is an important prerequisite, because it consequently enables the prediction of future behaviour of the system. It is also much easier to abstract a model from deterministic systems, than from non-deterministic and also the reproduction of test cases is only possible with guaranteed deterministic behaviour. This determinism is one of the main goals of the TTSoC architecture.

The TTSoC architecture is built around a deterministic NoC which interconnects multiple IP-Cores called the micro components with each other [OPHES06] (see figure 2.2). Each micro component is a self-contained computational unit, that interacts with the system through an interface which is connected to the NoC. This interface is called the Trusted Interface Sub-System (TISS) and is located with the user application inside the host. This TISS provides the basic core functions for the application and acts as a guardian for the NoC to prevent an application specific fault from distributing throughout the network. It always accesses the NoC at a priory known point in time by using a TDMA scheme. The order of the messages sent over the network is specified in a list called the Message Descriptor List (MEDL). Because of the interface design of the TISS the host application cannot modify the order of the messages. Additionally, a module called Trusted Network Authority (TNA), administers the routing of the NoC and ensures that only safe configurations are used.
Together the TISS and the TNA form a so called Fault Containment Region (FCR) which is a set of components, which are certified and prevent faults in the application layer from propagating throughout the system. [LH94] Also each host is considered to be in each own FCR.
In addition there is also one micro component called the Resource Management Authority (RMA), which allocates resources needed for the other micro components to access the NoC. This resource management is checked by the TNA against predefined constraints, like message conflicts or availability of resources. If the schedule is valid, the TNA routes the message through the NoC. This NoC provides the connection between the micro components through periodic or sporadic message passing. The basic concept behind this message passing are the encapsulated communication channels. (see section 2.1) They transport the messages at a previously known point in time from the sender to at least one receiver. (Broadcast and multicast is supported but limited as explained in [Pau08] chapter 6.2) The network itself is compound of standalone fragment switches, the micro components and bidirectional channels which connect all this components. Making this channels bidirectional gives an additional freedom in routing. The switches themselves are completely unaware about the encapsulated communication channels and just locally know how to switch the messages. The complete routing information from the sending TISS to the receiving is managed by the TNA.

The green components in figure 2.2 form the Trusted Sub-System (TSS). They have to be implemented as simple as possible in order to facilitate certification. Their implementation is also considered to be free of design faults and has to be certified, at the same level as the most critical micro component in the system.

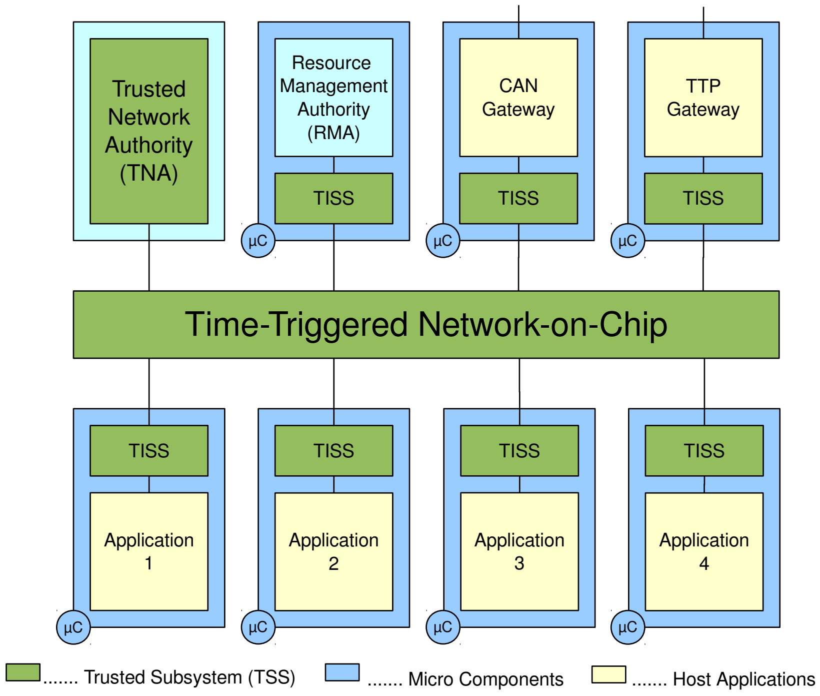
Trusted Network Authority (TNA)
Resource Management Authority (RMA)
CAN Gateway
TTP Gateway
TISS
μC
Time-Triggered Network-on-Chip
Application 1
Application 2
Application 3
Application 4
.... Trusted Subsystem (TSS)
.... Micro Components
.... Host Applications

Figure 2.2: Basic layout and exemplary configuration of a TTSoC

Encapsulated Communication Channels

The encapsulated communication channels are unidirectional data channels, which transport messages at a predefined point in time. They prevent the different micro components from interfering with each other, with respect to the temporal and spacial domain. Modification in the value domain is not covered by the encapsulation. For example, delaying or overwriting a message is not possible throughout these communication channels.
The TISS, which is located between the host application and the TTNoC, establishes the encapsulated communication channel to the TISS of an other application. This channel is created exclusively at a priori known point in time by using the TDMA scheme. As already mentioned earlier, the TISS is part of the TSS and therefore considered to be free of faults. This ensures, that the application in the user space is not able to interfere directly with the NoC. The endpoints inside the TISS are called ports. Every micro component can have multiple input and output ports, but can only send once at a time.

The Time Format

The NoC also establishes a global time base by implementing a clock synchronisation mechanism in the TSS. [Pau08] The timebase is available for all components in the TSS and is based on the physical second. This is similar to the implementation used in the Global Positioning System (GPS) and provides a granularity of about 230 pico-seconds and a length of about 136 years. The binary time format is 64 bit long, where the upper 32 bit represent the full seconds and the lower 32 bit the fractions of one second. (see figure 2.3)

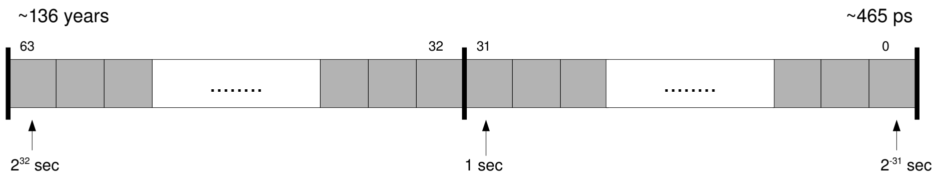

Figure 2.3: Time format of the TTSoC

From this time format a periodic control system is derived, which defines periods and phases of the pulses in the pulsed data stream. These pulses are sent through the encapsulated communication channels and are the actual messages which are are aligned according to the time format. In order to define a period in the pulsed data stream, a special bit is chosen, called the period bit. As previously mentioned the time format is organized in a power of two. This leads to periods of 1 second, $\frac{1}{2}$ second, $\frac{1}{4}$ second and so on. This restriction gives rise to a significant reduction of complexity in the implementation of the time format, but has the drawback that a specific timing is not possible. For example, later in the thesis a period of 20 milliseconds is required. This is not possible due to the realization of the timebase and therefore a period of 15 milliseconds is used.
As the alignment of pulses to periods are not exact, pulses can collide. As a consequence the basic period is subdivided by pulse phases. A phase defines a temporal offset of a pulse with respect to the start of a period.

2.2 Integrity Policies

This integrity policies provide dependability through a set of integrity levels *I*. The different tasks of a safety or security critical application are vertically subdivided into their corresponding integrity levels with a partial order relation ($\leq$). The content of these integrity levels is not generally definable and needs to have a different specification under each application.

Classic Integrity Models

The following integrity models are composed of tasks that consist of subjects S (the active elements like users or processes) and objects O (the passive elements like information containers). Both security models contain of the following functions and relations [BL75]:

- The function *il:* $S \cup O \rightarrow I$ which associates the integrity level to each subject S or object O
- *obs:* a relation on $S \times O$ where a subject $s \in S$ is able to observe an object $o \in O$. (reading)
- *mod:* a relation on $S \times O$ where a subject $s \in S$ is able to modify an object $o \in O$. (writing)
- *inv:* a relation on $S \times S$ where a subject $s_1 \in S$ is able to invoke an other subject $s_2 \in S$ (executing)

Bell-LaPadula

The Bell-Lapadula model [BL75] arose because of the need for a mathematical framework and model for a secure computer system. It is based on the work of Lampson [Lam69] who initially introduced a representation of security problems through a formal medium of expression. Lampson first used the concept of object, subject and access matrix. A lot of other research, mainly emerged out of the military sector, has been done in this field before the Bell-LaPadula model was created at *MITRE corporation* in 1974. [Bel74] [Wei69] This model is a safety model with hierarchical ordered structures, which protects the confidentiality of information by using in-system rules and a dynamic access matrix model. It should not be possible to read information from a higher security level or to write information into a lower one. This model basically controls the information flow and consists of the following three rules:

- No-Read-Up: A subject should not read an object in a higher security level
  or formal: $\forall (s,o) \in S \times O, obs(s,o) \Rightarrow il(o) \leq il(s)$
- No-Write-Down or *-property: A subject in a higher security level should not write into an object from a lower level
  or formal: $\forall (s,o) \in S \times O, mod(s,o) \Rightarrow il(s) \leq il(o)$
- Discretionary Security Property: It enables to control the access of a subject to an object, by using a free definable *access-control matrix*

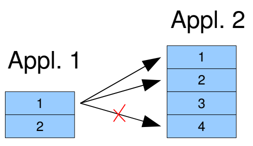

Figure 2.4: Illustration of a Bell-LaPadula writing operation

Inside the access control matrix, there are definable access rights. They consist of the following universal rights, like read-only, append, execute, edit and edit-read. The current access set is defined through the triple subject, object and access matrix. The subject owns access rights (read, edit, ...) to the object. For example, the tuple (Person, Document, {*read*}) means, that the subject "Person" is allowed to *read* the Object "Document". One of the main problems of the Bell-LaPadula model is the limited expressiveness. For example, a subject is writing into an object with a higher security level, which is an allowed operation. Because of the in-system rules, it is not possible for the subject to verify the operation by reading the written data.

The Bell-LaPadula Model was mainly developed for military use, because division into security levels fits perfectly into this area. Since 1976 this model is standard in the TCSEC (Orange Book) of the american military. [Gal87]

Biba

Improving the shortcomings of the Bell-LaPadula model, the Biba model [Bib77] was introduced at the *MITRE corporation* in 1977. This model is a multi security level policy and is mainly used in the field of data security. It is the inversion of the Bell-LaPadula rules and checks the integrity of a one-way information flow by using the following rules:

- No-Read-Down: A subject should not read an object in the lower security level
  or formal: $\forall(s,o) \in S \times O, obs(s,o) \Rightarrow il(s) \leq il(o)$
- No-Write-Up: A subject in a lower security level should not write into an object from a higher level
  or formal: $\forall(s,o) \in S \times O, mod(s,o) \Rightarrow il(o) \leq il(s)$
- Invocation: A subject should not invoke an other subject in a higher level
  or formal: $\forall(s_1,s_2) \in S \times S, inv(s_1,s_2) \Rightarrow il(s_2) \leq il(s_1)$

Because of this rules the information flow is limited and can just be carried out from a high integrity object into a lower one. Therefore it is possible, that an object in a low level can have data which is more trustable than it's own integrity level. Even it this object never modifies this data, the integrity level has to be decreased to the level of the receiving object. This is a main disadvantage of the Biba model, because the level of integrity can only decrease but never increase.
An other disadvantage in this model is that it is not providing confidentiality. To solve this problem it should be combined with an other model or by using the Bell-LaPadula model.
The Biba model is used in computer file systems for managing read and write access to files.

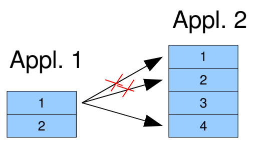

Figure 2.5: Illustration of a Biba writing operation

Clark and Wilson policy

This integrity model by Clark and Wilson, [CW87] [TBDP00] focuses on the conservation of data integrity. The main idea behind this policy is, that each secure data item can only be modified by a certified application. This certification guarantees, that the integrity of the data is kept intact. In generally the model distinguishes between Unconstrained Data Items (UDI) and Constrained Data Items (CDI) which stands for data items which are certified or not.
The model also defines two additional notations. The well-formed transactions or so called Transformation Procedures (TPs) for maintaining the integrity of the data and the separation of duties or Integrity Verification Procedures (IVPs) for providing the ability to verify the data.

The Clark and Wilson policy focuses on the integrity of the data modifications and not on information flow control like the Biba model. This model uses the following rules:

- Data of an integrity level can remain in this level when it is only modified by a procedure of at least the same level.

- Information flow from a low to a high level only is possible, if the data itself has a high integrity level.

Totel's Integrity Model

Many of the traditional implementations of such models like the previously mentioned Biba or Clark and Wilson models implement the required in-system rules in a mathematical framework and prevent upstream communication.

A more looser integrity model is Totel's model [TBDP00] [TBDB$^+$01], which is closely related to the Biba model. Contrary to Biba, where active entities called subjects access passive object entities, Totel's model has only one kind of entities called objects. These objects provide services that can be requested by a client. Each of them is classified within a particular integrity level, which indicates how it can be trusted and corresponding to it how high the requirements for dependability are. If an object creates a message, this message inherits the integrity level from it's creator. A reception of a message enables former defined integrity rules to check if this message is valid or not.

The main aim of the integrity model is to prevent in system error propagation from a low security level to a higher one, because this would automatically downgrade the security level of the receiving object. Therefore Totel's model introduces a special kind of object called the *Validation Object*. This object has to be seen as an **exception for the integrity policy rules**, which are defined in the following section. The purpose of this object is to provide fault-tolerance mechanisms to prevent the downgrade of the integrity of the received data. This provides reliable information from data which is possibly defective and adds an additional freedom in system design, because it enables object upward communication.

Totel's model uses three different kinds of objects which are all assigned to one or more security levels. It distinguishes between Single Level Object (SLO)s, Multi Level Object (MLO)s and the already mentioned Validation Objects. SLOs are assigned to a single integrity level corresponding to the confidence they are providing.
MLOs are objects which can be assigned to more than one integrity level and provide a solution, if a service is needed in different integrity levels and is validated for the highest one. Therefore in Totel's model it is possible to use this kind of objects in all lower security levels, which allows a reuse of services.

Integrity Rules

In order to define the rules used in Totel's model, we need a set of objects O and a set in integrity levels I, where the function *il:* $O \rightarrow I$ associates an integrity level $il(o)$ to every object $o \in O$. All elements in I have a partial order relation. ($\leq$)

The only rule a SLO has to follow is, that it cannot *access* information from a lower integrity level and it cannot *accept* information from a lower level. Consequently an object can be read by a lower or an equal integrity level or can write to a lower or an equal integrity level. The rules for SLO to SLO access are:

1. invoke reading
   $\forall (o_1, o_2) \in O \times O, \;\; o_1 \; read \; o_2 \Rightarrow il(o_1) \leq il(o_2)$
   o_1 can read from o_2 only if o_1 has at most the same integrity level as o_2

2. invoke writing
   $\forall (o_1, o_2) \in O \times O, \;\; o_1 \; write \; o_2 \Rightarrow il(o_1) \geq il(o_2)$
   o_1 can provide information to an object o_2 only if o_1 has at least the same level as o_2

3. invoke read-writing
   $\forall (o_1, o_2) \in O \times O, \;\; o_1 \; readwrite \; o_2 \Rightarrow il(o_1) = il(o_2)$
   this is a logical *and* between the read and write rule and stands for an entire information exchange

Like an SLO, the MLOs are also assigned to an integrity level, which in this case is called it's intrinsic level or highest level. To provide the same rules for MLOs like they are given for SLOs, a few more rules are needed. It is required to define MLO to SLO, SLO to MLO and MLO to MLO access. The exact definitions of these rules are not worked out in this thesis and can be found in the literature. [TBDB+01] The basic structure of Totel's model is sketched in figure 2.6 by realizing an exemplary instantiation, which uses the above defined rules.

In this figure, SLO1 in security level 1, reads from the object SLO3 which is in level 3. This is allowed by rule 1, because the integrity level of the reading object is less or equal to the read object.
Rule number 2 is illustrated by SLO4 writing to SLO3, because therefore the integrity level of the writing object has to be higher or at least the same level as the written object.
The third rule is shown in all read and write operations, which are conducted by not leaving the

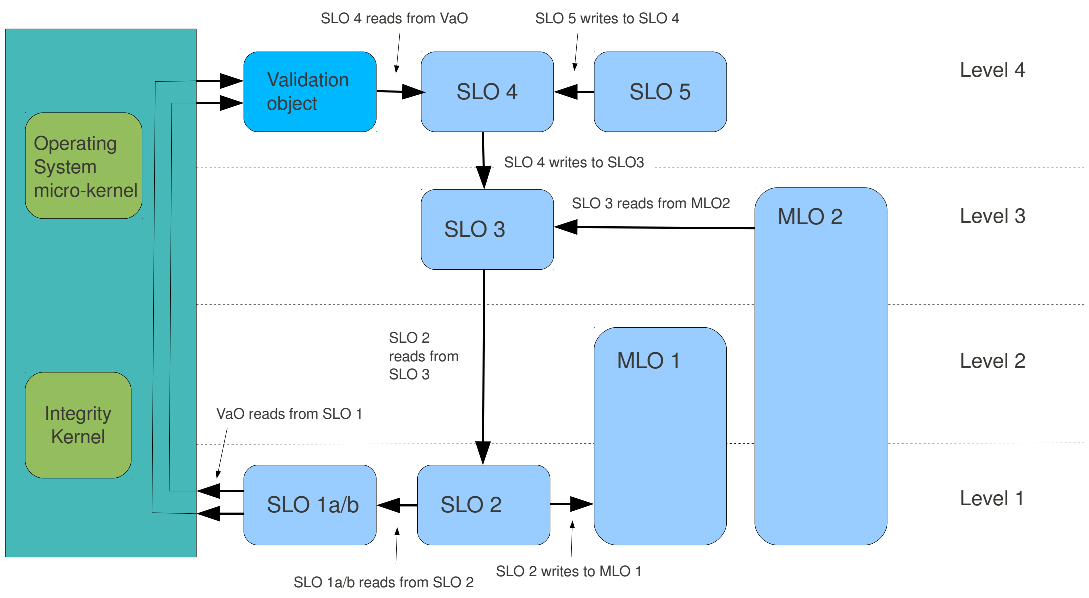

Figure 2.6: Integrity architecture of Totel's model sketched by an exemplary instantiation

integrity level, like it is done by SLO1 reading from SLO2.
The already mentioned exception to the integrity rules, which the "Validation Object" is using to provide upstream communication is given by SLO4 reading SLO1. This is a forbidden read operation which violates rule 1. To enable this, the "Validation Object" is interconnected and provides fault tolerant mechanisms to guarantee valid results in the value domain. Valid message exchange in the spacial and temporal domain are guaranteed by using the integrity kernel and the operating system micro-kernel. To be able to provide the fault tolerance mechanisms, SLO1 needs to be implemented in at least two independent ways, which is given in figure 2.6 by replicating SLO1 into SLO1a and SLO1b.

The illustration of the MLOs in figure 2.6, is given to provide completeness of the model and is not closer specified in this thesis.

An example assignment of the integrity levels is *FIPS 140-2*, which is based on *the U.S. government computer security standard.* [EBB01] *FIPS 140-2* defines four security levels, simply named "Level 1" to "Level 4" and are defined as following:

Level 1 This level provides the lowest level of security in our system and can be compared with using a normal COTS hard- or software. For example, a PC with a Windows or Linux operating system, which just provides standard security mechanisms.

Level 2 Security level 2 adds the feature, that a possible break-in always needs to leave some evidence behind.

Level 3 In addition to the evidence from level 2, which indicates a break-in, level 3 is attempting to prevent the intruder from gaining sensitive data.

Level 4 This level provides the highest amount of security mechanisms in the system. A possible fraud or break-in into the system always needs to be detected and prevented. Therefore corresponding security mechanisms have to be provided.

2.3 The Multiple Independent Layers of Security Architecture

The Multiple Independent Layers of Security (MILS) architecture [AFOTH06] was developed to make the certification of high integrity systems easier. It is based on the concepts of separation by Rushby [Rus81] and it is a requirement for the ARING 653 [Pri07] standard. Mainly it focuses on partitioning of the system, in a way that the single components can't interfere with each other. Through this partitioning a hierarchy of security services is produced, where each level uses the security service of a lower level to provide new security functionality that can be used by higher levels. The basic structure of this architecture is given in figure 2.7. It compounds of a shared single processor and the following three layers:

a) Partitioner Layer: This layer contains the separation kernel, which is a small piece of certified code. It acts as a partial and temporal firewall and provides the four fundamental security functions which are needed for a robust partitioning of an application:

1) Data Isolation: The memory address spaces of a component has to be completely autonomous from other partitions. Changing the state of a component should never influence the execution of an other component and also all others should not influence the current executing component.

2) Information Flow: This requirement modifies the Data Isolation requirement, because the components are allowed to communicate with each other. This communication has to happen in form of authorized communication channels. The architecture has to ensure that these channels are the only way how the partitions communicate with each other. (no hidden channels)

3) Sanitization: It has to be ensured that the processor never leaks classified data to unclassified processes while executing. This can be realized by cleaning all shared resources before an other component uses them.

4) Damage Limitation: If a fault happens at an unclassified component it should never distribute throughout the network. As address spaces are separate and the communication is managed through authorized communication channels, this attribute is provided through the other security functions.

b) Middleware Layer: This layer provides application specific dependent functions, like drivers, real-time data distribution services, resource allocation or object oriented inter-partition communication. It is also responsible to ensure end-to-end communication through labeling, filtering and maintaining information flow controls. The middleware layer can facilitate a communication channel between two applications with different security levels by using filter techniques.

c) Application Layer: This layer contains of the main application and has to be assigned to the desired security level.

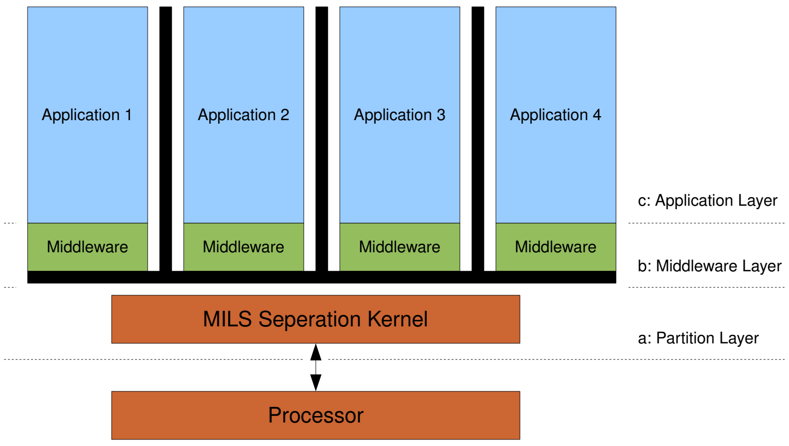

Figure 2.7: The MILS architecture [Rus81]

The MILS architecture is a design approach for high dependable systems that manages the high complexity through modular design and layer enforcement through integrity policies. It is a similar approach like the one used in this thesis but in contrast to the TTSoC which is a multi processor system, the MILS architecture uses a single processor. Details about the comparison of the two architecture is given in the work from Wasicek in section 3.4.

2.4 N-Version Programming

There are two main groups of software fault tolerance approaches, which are called single version software and multi version software. [Avi95] The single version approach consist of a single implementation of a system which includes the fault tolerant mechanisms inside the design. In contrast, multi version software stands for the autonomous development of two or more versions of a system. The idea behind this approach is that single versions of a program which are developed under strict independent circumstances will also fail independent from the other versions. This effect is called *design diversity*.
The two main approaches for multi version software are called Recovery Blocks (RB) and N-version programming (NVP) and are defined as following.

Recovery Block – Model

This technique was established by *Brian Randell* in 1975, as a first approach for multi version software. [Ran75] It uses two or more versions of a program, where one version is operating as a primary version and all others are in standby. If the output of this primary version fails at the acceptance test, the system falls back to a recovery state and the operation is recalculated by the alternate version of the program. The system recovery is considered completed, if the acceptance test succeeds. In order to provide the already mentioned recovery point, a memory is needed which provides a starting point for the alternate versions.

N-Version Programming – Model

The concept of NVP, which was started at the *University of California* by *A. Avizienis* in 1977 is based on the independent creation of $N \geq 2$ redundant software programs called versions. [AC77] [Avi95] The "*Independent generation of programs*" which was stated by Avizienis means that the N-versions of the program need to be created by N-groups or single persons which do not interact with each other during the software development process. Whenever it is possible, different algorithms, programming languages or even operating systems should be used. All N-versions of the program are running in parallel and simultaneously produce output data which needs to be compared by using a selection- or voting algorithm. This so called acceptance test is the main difference to the RB model. Details about the process of the N-version software (NVS) is depicted in figure 2.8. As N different versions of a software are needed, the development effort is N times higher than in a single version approach. However, because of the division of the system into subsystems the complexity is not increasing.

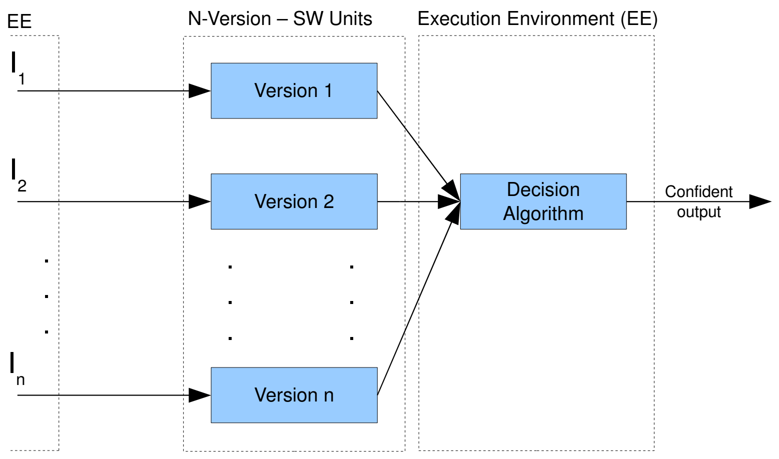

Figure 2.8: The N-version programming model [AC77]

As providing N different versions of a software is not a fully fault tolerance system, an Execution Environment (EE) is needed. Therefore every version of the software needs a so-called *fault tolerance feature* which allows each version to be a member of the NVS. Also the EE needs to provide an environment for executing these versions. It needs to minimise the probability that an undetected fault inside one version is influencing all other versions. In the EE the selection algorithm is also implemented, which guarantees a decision between the output values of the versions. For systems with $N = 2$ versions, the word comparison is used and for systems with $N > 2$ it is called voting. The different voting techniques and especially the inexact voters are discussed in section 4.

2.5 Anomaly Detection – Fundamentals

Anomaly detection tries to find patterns in data that do not conform to the expected behaviour of the whole data set. It can be used in a wide range of fields. For example, in fraud detection for credit cards, intrusion detection in computer systems, fault detection in safety critical systems and a lot of other areas. [BK09]

Figure 2.9 illustrates a brief example in a two dimensional data set, where the data has two normal regions N_1 and N_2. Most observations are situated in this area. Points which are sufficiently far away from this region, like points O_1, O_2 and O_3 are considered to be *anomalous*.

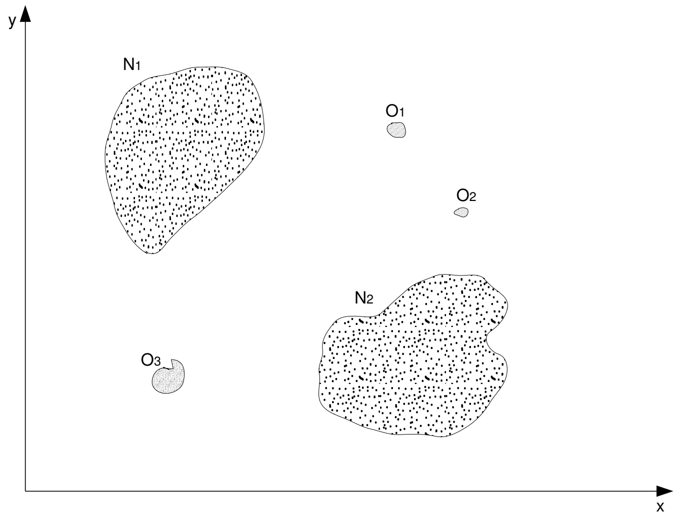

Figure 2.9: A simple example of anomalies in a 2-dimensional data set

At an abstract point of view an anomaly is a defined pattern that does not conform to normal behaviour. The easiest approach to detect such an anomaly is to find all objects which have a normal behaviour and mark all objects which are not in this normal region as anomalies. This sounds like an easy solution but there are several factors which make this approach very challenging:

- The definition of a normal behaviour which includes every possible occurrence is not always easy to find. It can happen that the boundary between normal and abnormal is very tight.
- In many situations even the normal behaviour is evolving over time and therefore also the definition of an anomaly has to change.
- Many procedures to find anomalies need training data as an input to determine if a data set is anomalous. Getting this training data often is a major issue.
- It can be difficult to distinguish between noise and the anomalies themselves. [TCL90]

The main aspect of an anomaly detection algorithm is the nature of the data, which can be described by using a set of attributes. For example, uses the data always the same dimension, are the characteristics always the same and in what field is it used? The data can also be *discrete*, *continuous* or *categorical*. An other important aspect of the different types of anomaly detection is the nature of the desired anomaly. They can be classified into the following three categories:

Point Anomalies: If a single data instance can be detected as an anomaly by analysing the attributes of all the data other data instances. Then this anomaly can be classified as a point anomaly, which is the easiest form of anomaly detection.

Contextual Anomalies: If data is declared as an anomaly, with respect to the specific context in which it occurs, than this data is declared to be a contextual anomaly. The notion of context is defined by the surroundings and environment in which the data is used and has to has to be defined application specific. It can be part of the problem statement itself. [SWJR07]

Collective Anomalies: This happens if a collection of data is declared to be anomalous in respect to the entire data set. All the single data instances of an anomaly are not declared as anomalies by themself, but there occurrence in the collection is anomalous.

Point anomalies can occur in many data sets, but collective anomalies can only occur in data instances where the data is related to each other. A point anomaly and a collective anomaly, can also be a contextual anomaly, when they are considered in respect to the context. Thus also a point or collective anomaly can be transformed to a contextual anomaly by adding the application specific context.

An other important aspect of the different types of anomaly detection is the nature of the desired anomaly. In this thesis we will focus on the most popular and also easiest form of anomaly, called the point anomaly. This anomaly can be a single data instance and can be classified as an anomaly by looking at all the other data instances in the set.

To indicate if a sample is in the set of normal or anomalous, exact labelling of the data is important. Covering all occurrences of anomalies is generally a difficult approach and is often not realisable. It is sometimes easier to label all normal occurrences in the data set. Therefore defining exact rules for labelling the data is often done manually by a human expert. Based on the labels available, anomaly detection techniques can work with one of this three modes:

Supervised Anomaly Detection: This technique assumes the availability of training data, where the provided data has labeled instances for anomalous and normal data. In operation, every new data instance is compared to the training data set to determine, which class it belongs to.

Semi-Supervised Anomaly Detection: Compared to the supervised technique, the training data of this approach just has labels for the normal data. Since this technique does not require labels for the anomaly class, it is usable in a wider range of applications.

Unsupervised Anomaly Detection: Techniques that work in unsupervised mode, do not require training data at all and therefore are most widely applicable. This technique is based on the assumption that normal data is much more frequent than anomalous data. If this is not given, they produce a hight false suspect rate.

There are a lot of different approaches to conduct anomaly detection. In this thesis the focus lies on three techniques, which are called *Nearest Neighbour Bases*, *Statistical* and *Clustering Based* anomaly detection. (see section 4.3) Others, which are not covered in this thesis are *Classification Bases*, *Information Theoretic* or *Spectral Based* anomaly detection.

2.6 Fault Injection Techniques

This techniques are important to analyse the reliability of computer systems and there are many different approaches in this area. [HTI97] In order to understand and identify potential failures, *fault injection* techniques are used for studying the dependability of a system. This is not only done during the design phase of the system, also already working prototypes are tested with this methods. Most important thing is to firstly understand the basic structure and behaviour of the system, it's criticality level and the in built fault tolerant functions. Then the tools for injecting a fault into the system can be chosen.

For every phase of a project, the ways for injecting a fault into the system are different. For example, in an early state of system design, simulation-based fault injection is used as a cheap evaluation for the dependability of a system. This mechanisms are useful to evaluate the effectiveness of a fault-tolerant mechanism, but it needs the availability of accurate input parameters which is difficult to provide.
On the other hand, *prototype-based fault injection* allows to evaluate the system without any assumptions about it's design. This faults can either be injected at the hardware- or the software level. The injection of a fault can provide information about the origin of the failure, but it is suitable for studying emulated faults only. It never can provide dependability measurements like availability or reliability.
Instead of injection faults, it is also possible to directly measure data by using *measurement-based analysis*. [IT96] This data often provides information about error and failure character-

istics but is limited to detected errors. Additionally the data needs to be collected over a long time, because failures and errors tend to occur infrequently.

Injection Methods

Choosing the method of fault injections depends on the nature of fault, which should be detected and the effort required to create them. The two main methods are hard- and software based fault injections. On the one hand, if a data corruption fault should be produced, a software fault injector [VM97] is better suited. In the recent years this kind of techniques became more attractive, because they can be used at target applications and operating systems and are compared to expensive hardware relatively cheap. If the target is an application, the fault injector can be inserted in the application itself or as a layer between the operating system. This makes this approach very flexible, but limited by not be able to access the hardware level. Software fault injectors can be categorised on when the actual fault is injected: during compile time or runtime. The easiest runtime faults are timeouts, which inject a fault after a pre- defined duration. Other faults are traps, which are triggered by certain events inside of the application or code-insertion faults, which are inserted before a specified instruction. Some projects located in the software based fault injection are:

Ferrari: The Fault and Error Automatic Real-Time Injection (Ferrari) project, uses software traps to inject CPU, memory and bus faults. [KKA92] When the trap is triggered, it injects faults at the desired location, by manipulating a memory location or changing a register. The fault can be either transient or permanent.

Doctor: Integrated Software Fault Injection Environment (Doctor) allows the injection of CPU faults, memory faults and network communication faults. [HSR95] Doctor uses timeout, traps and code-modification to trigger the fault injection.

On the other hand, if a stuck at fault needs to be produced, a hardware fault injection [ELOGV$^+$11] is preferable. This kind of fault injection, uses addition hardware to introduce faults into the target system. The injector can either have contact to the actual target system or without contact. Generally this methods are suited for low level faults. Some tools which have been developed in this area are:

MARS: The Maintainable Real-Time System (MARS) architecture is a distributed fault tolerant architecture. [KFG$^+$92] It uses electromagnetic fields to induct contactless fault injection: Two charged plates are mounted around a circuit board, which causes a fault injection. Small wires that act as antennas are used to inject the fault at the desired pin.

Messaline: This tool is developed in Toulouse, France, and can inject stuck-at, complex logic faults and others. [ACL89] It uses active probes and sockets to conduct pin-level fault injection.

FIST: The Fault Injection System for Study of Transient Fault Effect (FIST) project realises both contact and contactless fault injection methods, to inject transient faults into a circuit.

[GKT89] FIST can inject faults directly into a chip, which can't be done with pin-level fault injection. It can produce random faults inside the chip by using heavy-ion radiation, which causes single- or multiple bit flips.

In this thesis we focus on *simulation-based fault injection*, where we inject faults into a simulation environment which is explained in detail in section 5.5. The faults injected into this system are software based.

CHAPTER 3

Related Work

This chapter outlines some related work which has been done in this field. It starts by looking at a work which comes out of the avionics area, in which an airplane maintenance procedure is optimised by using COTS hard- and software. In order to upgrade the dependability of the output from these systems, fault tolerance mechanisms are used similar as it is done in this thesis. Next some NVP approaches are given which where carried out at the *University of California* where also the groundwork for this fault tolerance approach was done. Finally a work of the *Technical University Vienna* is mentioned, where the TTSoC architecture is compared with the well known MILS architecture and because of this it is concluded, that the TTSoC can be used in a mixed criticality application.

3.1 Connecting Commercial Computers to Avionics Systems

This work, published at the *University of Toulouse*, analyses the take-off and maintenance procedures for new aircraft generations, in which communication between on- and off-board computers is carried out. [LDPA09]

Take-off Procedure

First the take-off procedure of a modern airplane is optimised. In the original process, a set of flight parameters from the airport and the plane need to be processed by the pilot. With this data the pilot can calculate the take-off profile and then has to enter the data manually into the aircraft management system. In order to speed this procedure up, this approach uses a laptop, which is directly connected to the aircraft management system. This laptop calculates the profile and enters the data. Basically this approach considers, that a save application which is running on an unsafe operating system, sends information to a critical on board computer. For this an information flow from a low criticality level to a higher one is needed. This approach uses four criticality levels, which have been identified as: Flight Management (FM), Aircraft Operation and Maintenance (A/C OM), Aircraft Information System (A/C IS) and Open World (OW). The

updated information flow in this procedure can be organised as depicted in figure 3.1.
Task 1 is the take-off profile calculation task, which is considered to be unsafe, due to the low integrity level of the operating system. *Task 2* stands for the information directly provided by the airline, which is considered to be OW too and exchanges information with *task 1*. *Task 3* represents the airport data, which is sending information to the laptop in *task 1* and is considered to underlie a stricter policy, because this is a standard procedure for airports. Task 4 stands for the application, which directly exchanges information with the aircraft management system. The communication from task 1 to task 4 is a critical upstream communication, which is strictly forbidden in classic integrity models like mentioned in section 2.2. Therefore a mechanism is needed to upgrade the reliability of the data, which is explained in the following section.

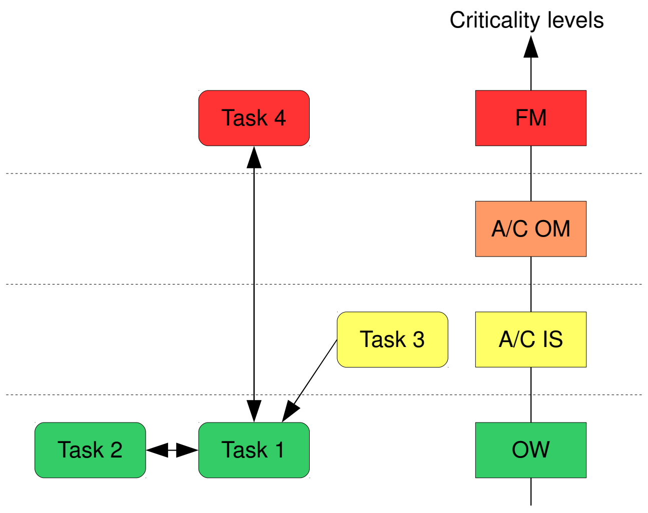

Figure 3.1: The new information flow for the take-off procedure [LDPA09]

Maintenance Procedure

Secondly a similar strategy is used to optimise the maintenance procedure, which is very cost causative and therefore needs a fast handling. In the traditional procedure the maintenance operator gets faulty behaviour of the plane from the on-board log. With this information he tries to fix the problem by using electronic- or paper manuals and by interacting with the maintenance terminal. This procedure can be optimised in a way, that the maintenance operator directly accesses the plane by using his maintenance laptop, where he is able to see the on-board logs and is guided by the electronic manuals. This optimised procedure also induces upward information flows from the maintenance laptop in the OW domain to the domains with the higher criticality levels.

In both case studies an information flow from a low integrity level to a higher one is needed, because the maintenance or calculation laptop uses untrusted COTS hardware such as *Windows* or *Linux*. This kind of operating systems are not offering sufficient protection for the application and therefore special mechanisms are needed, which are discussed in the following section.

Providing Upstream Communication

In multi-level security systems, mechanisms are needed to control the information flow and check the integrity of the data that is sent from one level to the other. Therefore an integrity model is used, which provides rules for up- and downgrading information flows. In order to resolve these security issues, this work uses the model called Totel's multilevel integrity policy, which is already described in detail in section 2.2. [TBDP00] [TBDB$^+$01] Combining now the information flow of figure 3.1 with the security attributes of Totel's model, the message exchange has to be reorganised as depicted in figure 3.2.

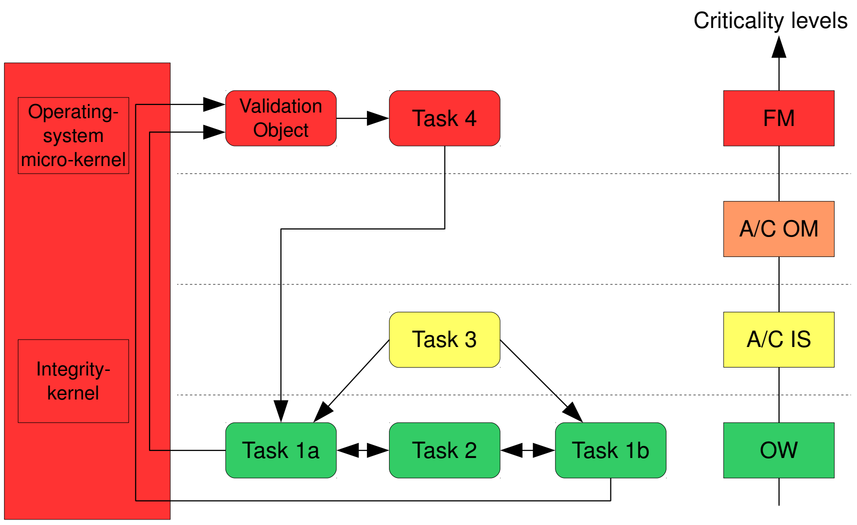

Figure 3.2: The new information flow for the take-off procedure by using Totel's integrity model [LDPA09]

The main change in this figure is the communication from Task 1 to Task 4, because a bidirectional information flow from a higher security level to a lower one is needed. Task 4 is allowed to write Task 1 and also Task 1 is allowed to read Task 4 but all other communication activities between these two tasks are forbidden. As this is needed by the take-off procedure, a "Validation Object" has to be interconnected, which upgrades the reliability of the data in the value domain. This upgrading is done by implementing a voter in the VaO and therefore redundant input data is needed. To create this, task 1 is split up into at least two autonomous tasks called task 1a and task 1b. These newly created tasks use different execution platforms for the take-off calculations. One is using *Windows* as an operating system and the other one is using *Linux*. To be able to execute both operating systems on one laptop simultaneously, a virtualization is needed which has to be considered to be free of faults. The redundant output of both tasks is compared inside the VaO and because of this redundancy check it can be used in the higher integrity level.

Determinism Issues

One of the main problems with this approach is that a determinism of both operating systems has to be assumed. As this is not provided in reality, because both operating systems use at least different pre-empting scheduling algorithms, it is likely to happen, that the comparison of the data may fail because of a deviation in the temporal domain.
This determinism problem is the major difference to the approach used in this thesis. Because of the use of the TTSoC and it's deterministic communication service, it is guaranteed that two redundant values are not permuted or have a deviation between each other.

3.2 N-version Programming – Experiments

This section covers a few carried out examples of the NVP approach which are described in detail in section 2.4 and were introduced by Avizienis in 1979. [AC77] This work is mainly focused on software problem statements, but it is not limited to it. For example, in the avionics project from the *Boeing 737-300* [WYF83] and also the Airbus airliners, the NVP approach is used. [Tra88] The following approaches were carried out to test the characteristics and properties of N-version programming.

The 3-version RATE Program Experiment

The Region Approximation and Temperature Estimation (RATE) experiment was carried out at a graduation seminar course at the *University of California Los Angeles* in 1977. There the experiment was done in form of a programming assignment, were the students where asked to form teams of two people. Totally there were 16 teams which created 16 programs where the best seven were chosen for further investigation. These seven programs got grouped into 12 combinations and tested by using 32 test cases. This gave them a combination of 384 test cases, where 290 contained no bad version, 71 one bad version, 18 two bad versions and 5 three bad versions.
The 290 test cases generated acceptable results, the 71 cases with one bad version produced 59 acceptable results and the other experiments all produced unacceptable results. Based on this data, two main difficulties in NVP were observed:

- Sometimes a version of the program produced an error that caused the operating system to continue the execution and thus also the other two may correct versions failed. Because of this the 3-version program was not able to continue beyond this point.
- In some cases the output of a program was incorrect or even missing. If two versions of the program had a missing output at the same time, it could happen, that this wrong output was the same and therefore the whole result was considered to be wrongly valid.

The main result of this study was, that the environment where the experiment was executed was poorly suited for the use as an NVP platform. A solution to this problem is the *DEDIX 87* project which is explained in the following:

DEDIX 87 – A Supervisory System for Design Diversity Experiments

This project was started in 1985 at the University of California and was created to provide a research platform for the investigation of design diversity in order to create fault tolerant systems [AGK$^+$85] [ALS$^+$87]. The purpose of DEDUS 87 is to supervise the execution of N diverse versions of a software as a fault tolerant unit. It also provides a transparent interface for each version, the user and the surrounding system so that each component doesn't need to be aware of the other components and can run completely autonomous. A few attributes of the DEDIX system are:

- DEDIX is a distributed system which can be executed at different sites in order to benefit from parallel execution and to survive a crash of the minority of instances.
- It is guaranteed that an application can run with any number of different versions and also special requirements for the software are not needed.
- A reliable decision algorithm is available that creates consensus from all available versions. The algorithms are able to deal with different formats and the user is able to switch the used algorithm.
- It also provides recovery mechanisms for disagreed wrong versions and also removes failed versions where the recovery attempt failed or is not available.
- The DEDIX system runs on the distributed Locus environment and is portable for all UNIX systems.

DEDIX combined with the N-user programs can be seen as a fault tolerant multi-version system. For this, the DEDIX platform is the supervisor which guards the execution of the software versions. The environment itself can be executed on a single computer and also in a network which protects against most hardware faults. In a network the communication is realised over standard ethernet, where every single host has it's own instance of the DEDIX software and a diverse version of the application.

3.3 A Time-Triggered System-on-Chip – Prototype

A working prototype of the Time-Triggered System-On-Chip (see section 2.1) was created at the *Vienna University of Technology* by Christian Paukovits in 2008. [Pau08] This prototype was originally realised on an Altera[1] Field Programmable Gate Array (FPGA) and was successfully ported to the following models:

[1]Altera Corporation, `http://www.altera.com`

- Altera Cyclon II^{TM} Series (Device: EP2C35)
- Altera Stratix II^{TM} Series (Device: EP2S60)
- Altera Stratix III^{TM} Series (Device: EP3SL150)

The implementation on FPGAs was chosen because they are far more inexpensive than using ASIC technology and do not demand for infrastructure like a vacuum-clean environment. The main drawback of FPGAs is the 3-5 times slower execution time and the 35 times more size needed on the chip. [KR07]

The current implementation is dimensioned in such a way, that it is unlikely that a target application will ever exhaust the available number of ports or the size of the memory inside the TISS. Therefore this implementation is usable in various dimensions of target applications.

In this approach a special prototype hardware was used for each FPGA family. The MPSoC Development Kit manufactured by *TTTech*[2] was taken as a primary platform for the prototype. By using this development kit, the whole design was split up on several FPGAs. The reason for this was that at this time no FPGA with enough size was available. Therefore this development kit emulates a SoC.

In order to implement the system on a single FPGA, it was recreated by using the NIOS II^{TM} Development Kit. This hardware is a COTS product which enables designing and prototyping of a wide range of embedded applications. In order to use the system on more advanced FPGAs, it was also implemented by using the Altera Stratix III^{TM} Development Kit. With this large scale FPGAs, this implementation came close to the scale of the multi FPGA implementation of the MPSoC Development Kit.

The Altera Stratix III^{TM} Development Kit was used in this thesis to create the simulation environment described in detail in chapter 5.

3.4 The Time-Triggered System-On-Chip in Mixed-Criticality Applications

This work was created at the *Vienna University of Technology*, where also the SoC implementation called the Time-Triggered System-On-Chip (TTSoC) was created. [WESK10] In this work it is shown, that the TTSoC architecture provides a set of key features, that act as a spatial and temporal firewall between the micro components, which is closely related to the well known MILS architecture. [TBDP00] The MILS architecture is based on the concept of separating the system into single autonomous subsystems, which access to a single shared processor is managed by a trusted partitioner layer. (see section 2.3) This layer provides the four fundamental security functions which are needed for the partial and temporal separation of the applications.

[2]TTTech Computer AG, `http://www.tttech.com`

The TTSoC provides these four functions by design and goes even further, by also separating the processor through implementing each component as an autonomous IP-Core. These IP-Cores are connected through a NoC which uses the concept of the TTA. (see section 2.1) By using this approach, a deterministic behaviour of the communication channels can be guaranteed which facilitates verification and certification. The TTSoC implements the core requirements of the separation kernel from the MILS architecture by design. The statements of these properties are listed as following:

1) Data Isolation: This means that every application needs to have an own memory address space, which is exclusively accessed by the application. This attribute is provided in the TTSoC through the separation of the system in subsystems called the micro components. There is no hidden channel between every single micro component and the only interaction with the NoC is guarded through the TISS. As the amount of on-chip memory is limited, there can be a lack of memory in huge applications. Therefore an off-chip memory is needed which has to be managed by an Memory Management Unit (MMU). This MMU again has to guarantee that there are no shared resources between the micro components.

2) Information Flow: This attribute is provided by the encapsulated communication channels of the TTSoC. These channels provide strict rules for communication which are supervised by a trusted component called TNA. The TNA acts as an operator and connects the port of the sending micro component with the receiving port, by driving network switches. It is guaranteed by design, that the application running on the micro component is not able to interfere with the route controlled by the TNA. To transmit a message, the application layer simply writes the information into a memory location and maps this memory to the corresponding output port. The interface of the application layer is that restricted, that it cannot influence when or how often a message is transmitted through the network. Therefore the application layer cannot interfere with the TTNoC and the information flow is guaranteed by design.

3) Sanitization: Because of the fact that every micro component is completely autonomous, sanitization is not a problem. Every micro component is realized as a single IP-Core and therefore no resource is shared with each other.

4) Damage Limitation: This is one of the main features of the TTSoC architecture. Due to the partitioning of the system into FCRs, faults cannot distribute throughout the network and are therefore self-contained.

The TTSoC architecture establishes a spatial and temporal firewall between each micro component and as a consequence it guarantees, that a fault in one component cannot distribute throughout the NoC. In addition it facilitates certification, by using a deterministic communication channel on the NoC and by partitioning the system into single FCRs. [PK09]

Middleware Layer

In the TTSoC architecture it is possible to extend the security mechanisms by adding additional security functions into the middleware layer. Distinguishing from the MILS architecture, this middleware layer is located inside the application section of the micro component. As the micro components are single FCRs and guarded by the TISS, a fault cannot distribute throughout the middleware into other micro components. The middleware needs to be verified at the same integrity level as the application, which is running on the component. As it is probable, that this application is running on a high integrity level, the fault tolerant mechanism in this middleware needs to be easy to verify. In this thesis we focus on the implementation of such a middleware on a multi-processor system. Therefore we realize different mechanisms which implementation is discussed in chapter 4.

CHAPTER 4

Validation Middleware

This chapter explains how the previously discussed security policies can be used in combination with the TTSoC architecture by implementing a Validation Middleware (VaM) to upgrade the reliability of upstream information flows. First the system model is mentioned, which generally describes the kind of systems the solution is applicable. Next some possible realisations for the mechanisms inside the VaM are provided. All approaches used in this thesis implement the basic concept of anomaly detection, whose fundamentals are given in section 2.5. These algorithms are implemented in a simulation environment created in chapter 5 and the results of these simulations are evaluated in chapter 6.

4.1 System Model

In this thesis we implement a multi integrity level system by using the TTSoC architecture as a basic platform. (see section 2.1) In order to establish a secure environment, a segmentation into single autonomous subsystems is needed. Between these subsystems a spatial and temporal separation has to be created in order to reduce complexity of the whole system. In the TTSoC this is guaranteed by the notion of micro components, which can be seen as a basic unit of abstraction in order to create autonomous reusable subsystems. Each micro component is interconnected through a *deterministic* and predictable TTNoC, which uses the communication technique called the *pulsed data streams* that is based on the TDMA scheme. This SoC architecture prevents any unintended interference between the micro components, which enables the integration of mixed-criticality subsystems.

Each micro component on the SoC gets assigned to it's desired integrity level and as every subsystem exclusively holds one partition, each component is completely autonomous. With these different integrity levels the communication between them has to be supervised by an additional integrity policy. Otherwise it would be possible for a component in a low level to interact with a higher one and maybe induce a fault or a fraud into the secure component. Therefore Totel's multilevel integrity model is used as a basic concept for message exchange between the different integrity levels. This model creates the following three rules which are described in detail in section 2.2. As this integrity policy is thought to be used in a single processor system, it has to be slightly modified for the use combined with the TTSoC architecture.

One major change are the integrity checks at reception, which check the whole information flow produced between the components and are carried out by the integrity kernel. Because of the use of the TTSoC architecture, these integrity checks are not needed, as messages can only be received at a-priori known points in time. These reception instants are defined statically and are performed through messages in encapsulated communication channels. Therefore the integrity rules have to be applied at design time and the strict obedience of these rules is guaranteed by the TNA.

In the adjusted model it is also necessary to neglect the definition of MLOs, because they are not fitting in the system design. Because of the encapsulation property, which is guaranteed in the TTSoC architecture, one object just can be in one integrity level. Therefore the distinction from SLO and MLO is not necessary and all entities are simply called objects "OBJ". In Totel's model the *Validation Object* is a special kind of object, which is able to upgrade the information integrity by using fault tolerance mechanisms. In our adjusted model this object is simply dragged into the secure object itself in form of a middleware. The most important property of the VaM is that it has to be validated at the same integrity level as the application in the micro component itself. Therefore it is desirable to keep this middleware simple and reusable.

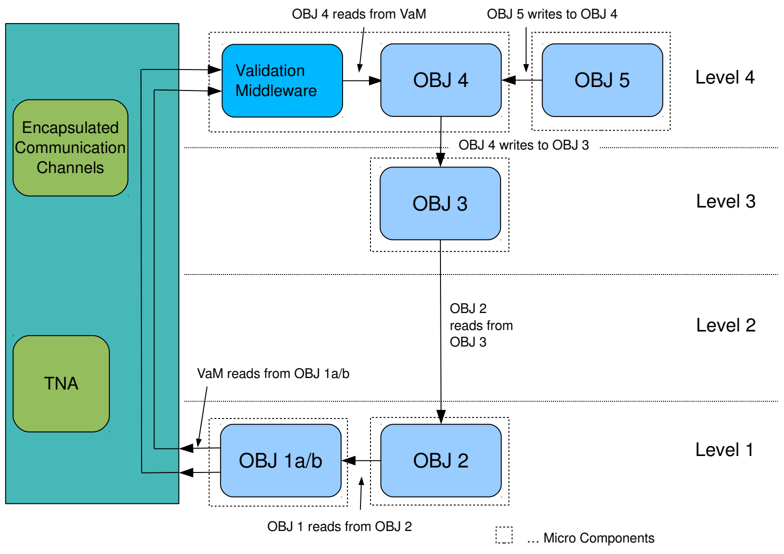

Figure 4.1: An exemplary instantiation of the modified Totel's model

Figure 4.1 illustrates the changes made to Totel's model by using it with the TTSoC architecture as a spatial and temporal firewall between the objects.

The figure shows the execution of rule one of Totel's model by OBJ1 reading OBJ3, rule two by OBJ4 writing OBJ3 and rule three by every communication process which is not changing the integrity level. The upstream information flow is given by OBJ4 reading OBJ1, which is not conform to the rules of Totel's model and therefore needs an upgrade of the dependability of the information flow. In order to create this dependability upgrade, OBJ1 needs to be replicated in *at least two* independent objects called OBJ1a and OBJ1b. The objects introduced in Totel's model are already grouped in micro components, which indicates how this exemplary instantiation can be implemented on the SoC. Figure 4.2 shows the example from figure 4.1 implemented by using the TTSoC architecture.

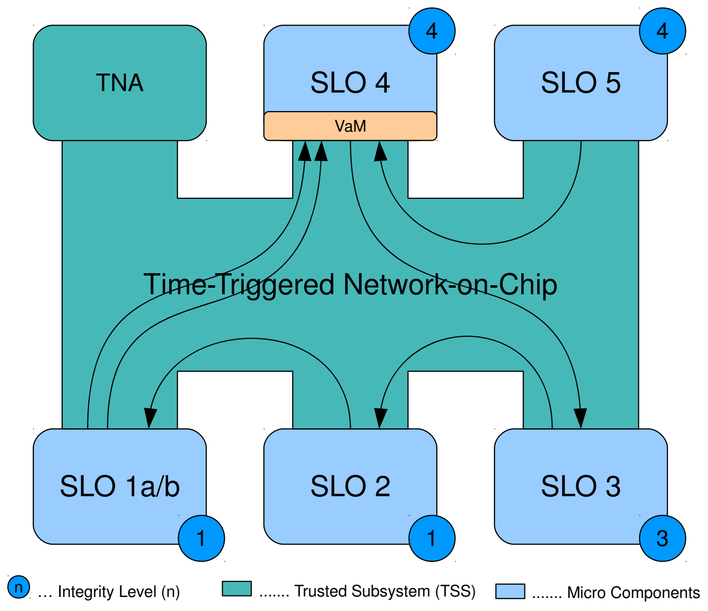
TNA
SLO 4
4
VaM
SLO 5
4
Time-Triggered Network-on-Chip
SLO 1a/b
1
SLO 2
1
SLO 3
3
n ... Integrity Level (n)
.... Trusted Subsystem (TSS)
.... Micro Components

Figure 4.2: The exemplary instantiation implemented on the TTSoC

4.2 Design of the Validation Middleware

The VaM is used to upgrade the dependability of the data in an upstream information flow. To be able to do that, the unclassified input data needs to provide the information from redundant sources. The basic information flow is given in figure 4.3, which provides N different and potentially diverse inputs.

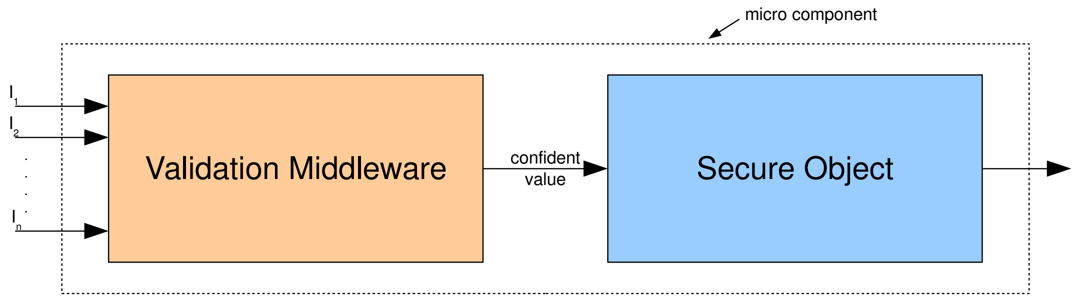

Figure 4.3: The “Validation Middleware“ with redundant Inputs (I_{1-n})

These redundant input channels are provided by *diverse* replicas and need to fulfil a set of key requirements in order to be completely independent from each other. This approach is called NVP and was introduced by *Avizienis* in 1977. [AC77] [Avi95] There he defines the notion of N-version programming, which creates the needed fault tolerance requirements by using two or more versions of a piece of soft- or hardware. (see section 2.4) The main goal of NVP is to eliminate similar errors, which can be introduced by having relations between the different programs called versions. This independence of each version is called *design diversity*. The main purpose of such required diversity is to eliminate the commonalities between the different efforts because they have the potential to cause related faults. Therefore the probability for significantly diverse approaches has to be increased. This is realised by using different algorithms, programming languages, tools and environments in order to archive complete independence between each version. The output of each version needs to be compared by a output selection algorithm or a voting algorithm. This is exactly the problem which arises by realising the VaM and is discussed in the following sections.

The most common and intuitive way of comparing the multiple inputs of the VaM is a majority voting algorithm, but this brings some difficulties. [LCE89] In some situations, the inputs need to be compared and determined to be correct, even if the data deviates from each other. This raises the need for an inexact voter, which defines a criteria for determining correct and incorrect inputs from data which is not necessary identical. The needed requirements these algorithms need to fulfil are given in the following section.

Algorithm Requirements

The comparison of the outputs and the finding of a final result is carried out by selection- or voting algorithms. These algorithms should be able to detect errors and prevent bad values from propagating throughout the system into the secure area of the application. In addition they should be able to define an error condition or a prepared sequence of valid outputs, when they are not able to find an output which provides the needed high confidence. Therefore the used approach always needs to find the result which has the highest probability for correctness.

The algorithms are realised in the application layer of the micro component as a middleware.

If this application is at the highest security level (Level 4), the middleware layer also has to be verified at this high level. In order to keep the effort for this verification as low as possible the used technique should be simple but still effective. Therefore all algorithms should run in an online approach, possibly without knowing anything from the previous run. These kinds of algorithms without a memory facilitate verification because the number of possible outcomes of a set of input data is always the same.

The focus lies on approaches which are feasible to use in many different problem statements and are able to detect faults, intrusions or even both. The needed sources of data (e.g. sensors) often induce noise or even miss a whole value, and therefore should provide filter mechanisms by preprocessing the input data. As memory often is limited, resource constrains should be considered which is why, a light-weight approach has to be found. As the data is collected in a distributed fashion, communication channels are needed which facilitate a deterministic behaviour and prevent from flooding or overwriting of messages.

One possible approach to realise such an algorithm is the use of anomaly detection which is discussed in the following section.

4.3 Anomaly Detection – Algorithms

Anomaly detection can be used as a fault tolerant approach to upgrade the information integrity. In this paper three different techniques of anomaly detection algorithms are discussed and compared to each other. These three techniques are *Nearest Neighbour Bases*, *Statistical* and *Clustering Based* anomaly detection. [BK09] As there are many different approaches in each category, the best suited has to be found.

Most approaches described in this section, try to find possible outlier values and establish a set of valid values, which have to form a majority. If a majority of valid values is found, an average value is calculated, which is trustable and can be used in a secure environment.
The main goal of the algorithms is to define the criteria how such a majority can be created. In this thesis a majority is formed of values, which are close to each other in the value domain. This sounds like an easy achievable problem, but it is quite hard to define where the border between valid and invalid is set. The algorithms in the following sections try to define such a border in different ways and are summarized in table 4.1.

	Time Complexity	**Space Complexity**
k^{th} **Nearest Neighbour with Delta-Value**	$O(n^2)$	$O(n)$
Probabilistic Boxplot Method	$O(n\ log(n))$	$O(n)$
Histogram Method	$O(n)$	$O(n)$
Single-Linkage Clustering	$O(n^2)$	$O(n^2)$

Table 4.1: VaM algorithm overview

The format of the input data is a one dimensional data set of *point* data instances from the same data type. The way how an anomaly value is calculated in data sets with more dimensions and different data types is discussed by *Tan P.-N* in 2005 [PNMV05]. As already mentioned, anomalies can be of different types and it has to be decided in advance, which kind of anomalies should be possible to detect. In this case we focus on point anomalies, because also single data instances can be considered as anomalous. This is the simplest and also most used kind of anomaly.

k^{th} Nearest Neighbour with Delta-Value

This algorithm calculates an anomaly score of each data instance, by counting the number of nearest neighbours (k) that are not more than a distance d apart from the given data instance. [SPP+06] [KNT00] This method is like calculating the global density of every data instance. The basic calculation algorithm of the anomaly score is given in figure 4.4, where $d(i, j)$ stands for the distance d from data sample i to sample j.

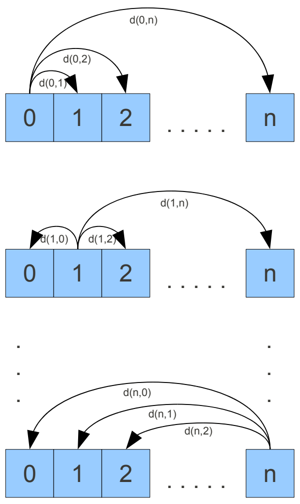

Figure 4.4: Distance calculation of the k^{th} nearest neighbour algorithm

With an anomaly score for every data instance, it needs to be defined from which score the data is anomalous or not. In this thesis the algorithm is used with a small data set and therefore the majority of all values is used. So if the anomaly score is higher or equal to the majority of data instances, the value is clearly of normal behaviour. If enough data samples with a dense neighbourhood are found, the whole round of samples is declared to be valid. Otherwise the data of the whole round is discarded. An unresolved round is the main problem of this algorithm because it can happen that this is not an option in some applications. One possible solution for this problem would be to just use the largest set of valid values even this set is not a majority of values. Unfortunately an other problem arises with this way of calculation because when there are two largest sets with the same size, one of them has to be chosen without knowing which set is the one with the real result.

Defining the distance d is the main challenge of this algorithm and is application dependet. The only way to find an effective value is to use test data, where all possible occurrences of anomalies are marked. Then the distance d needs to be adjusted, until most occurrences of anomalies are properly detected.

Time Complexity

As every single value needs to find a majority and therefore needs a distance value to all other data instances, a runtime of $O(n^2)$ is needed to get a full distance matrix. With this matrix a majority can be built and an end result can be found.

Probabilistic Boxplot Method

This algorithm produces a boxplot diagram by using the input data samples. [MTL78] [BK09] This kind of diagram is normally used for the graphical illustration of statistical data but it can also used as one of the easiest statistical techniques to detect anomalies. An exemplary boxplot diagram is given in figure 4.5. The blue box in the middle of the picture consists of 50% of the data samples and is delimited by the lower and upper quantiles. The lines to the left and the right of the box are called the lower and upper whiskers. All data samples, which are outside of these whiskers are depicted as anomalous. The $*$-anomaly is called a mild outsider, because it is located between the 1.5 and $3 * IQR$ boarder. The o-anomaly is called an extreme outsider because it is outside the $3 * IQR$ range.

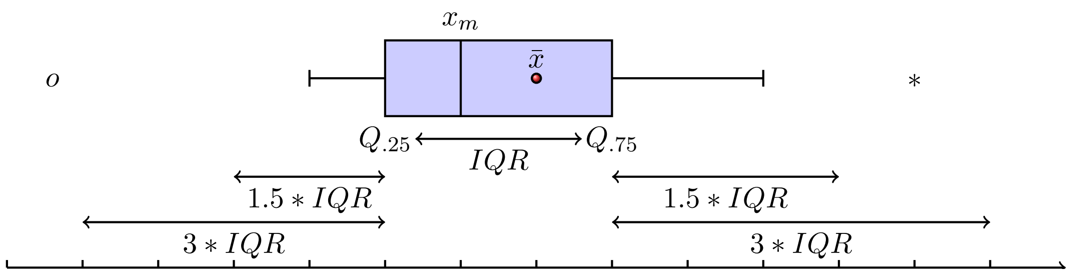

Figure 4.5: An exemplary boxplot

A boxplot is fully defined by five values. The mean, median, 25^{th} percentile, 75^{th} percentile and the interquartile range. The input data is a set of values, which need to be sorted from the lowest to the highest value. The progression is labelled x_1, x_2, ... , x_n where n is the highest value. If the number of data samples is odd, the median (x_m) is calculated as following:

$$x_m = x_{\frac{n+1}{2}} \tag{4.1}$$

If the number of samples is even:

$$x_m = \frac{x_{\frac{n}{2}} + x_{\frac{n+2}{2}}}{2} \tag{4.2}$$

If $\frac{1}{4}(n+1)$ is integer, quantile $Q_{.25}$ is calculated as following:

$$Q_{.25} = x_{\frac{1}{4}(n+1)} \tag{4.3}$$

If it is not integer:

$$Q_{.25} = (x_{integer(\frac{1}{4}(n+1))} + x_{integer(\frac{1}{4}(n+1))+1}) * decimal(\frac{1}{4}(n+1)) \tag{4.4}$$

If $\frac{3}{4}(n+1)$ is integer, quantile $Q_{.75}$ is calculated as following:

$$Q_{.75} = x_{\frac{3}{4}(n+1)} \tag{4.5}$$

If it is not integer:

$$Q_{.75} = (x_{integer(\frac{3}{4}(n+1))} + x_{integer(\frac{3}{4}(n+1))+1}) * decimal(\frac{3}{4}(n+1)) \tag{4.6}$$

Finally the interquartile range can be calculated by $IQR = Q_{.75} - Q_{.25}$. This is the most significant factor for marking a data sample as an anomaly, because if a data sample is outside the range of $x_m \pm (IQR * 1.5)$ this sample is marked as an anomaly. The factor of 1.5 is not a fixed quantity and was originally defined by John W. Tukey. [Tuk77] If an application needs higher or lower accuracy this value can be changed.

Time Complexity

The calculation of the five needed values for a boxplot, can be carried out in $O(n)$. However, before this can be done the values need to be sorted, which needs at least a runtime of $O(n\ log(n))$ if the fast *quicksort*, *heapsort* or *mergesort* algorithms are used. Therefore the runtime of the boxplot method is $O(n\ log(n))$.

Histogram Method

The histogram based anomaly detection algorithm is one of the simplest non-parametric statistical technique used in this area. A histogram is a graphical illustration of a frequency distribution. It is based on the classification of data into bins, which width can be fixed or variable. The size of the bins represents the relative frequency of the data inside the box. An exemplary histogram is pictured in figure 4.6.

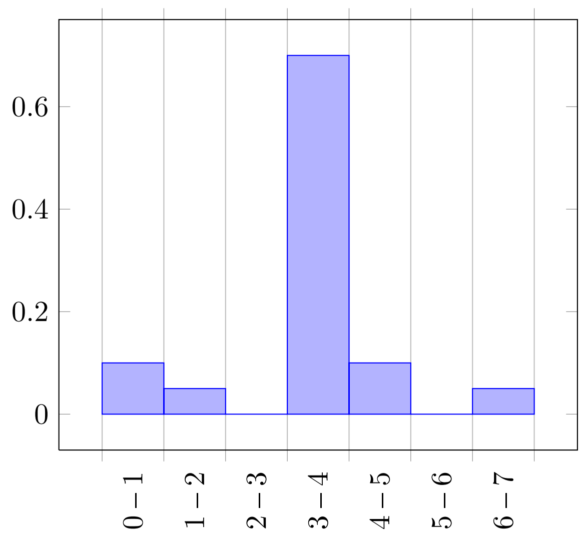

Figure 4.6: An exemplary histogram

The algorithm basically consists of two steps. The first step involves building a histogram based on the input data. In the second step, each data sample gets assigned to a bin of the histogram. After this is done, the bin with a majority of the values forms the end result of the algorithm.

Time Complexity

This algorithm is relatively simple, because every data sample just needs to get assigned to it's box. The size and number of boxes can be created in advance and therefore the runtime of this algorithm is denoted to $O(n)$.

Single-Linkage Clustering

As every other algorithm of this kind, clustering tries to find a structure in a collection of unlabelled data. A cluster is a set of objects, which are similar to each other. This technique is generally an unsupervised approach and can be grouped into three different categories as written by *A. Banerjee* in 2009: [BK09]

1. *"Normal data instances belong to a cluster in the data, while anomalies do not belong to a clustering"*
2. *"Normal data instances lie close to their closest cluster centroid, while anomalies are far away from their closest cluster centroid"*
3. *"Normal data instances belong to large and dense clusters, while anomalies either belong to small or sparse clusters"*

Single-Linkage clustering is part of the third category and declares instances belonging to clusters whose size is below a certain threshold value as anomalous. It belongs to a method of cluster analysing called hierarchical clustering, which tries to build a hierarchy of clusters. It generally works as a bottom up approach, where all data instances start in its own cluster and pairs of clusters are merged and moved up the hierarchy. To be able to decide which cluster should be merged, the distances between the data instances are calculated. Mathematically the distance between the data instances is described by the expression: [Mat00]

$$D(R,S) = \min_{r \in R, s \in S} d(r,s) \tag{4.7}$$

In Equation 4.7, R and S are sets, which consist of elements (r and s) called clusters and $d(r,s)$ denotes the distance between the two elements r and s. Given a set of N elements, all of these elements are at first located in an own cluster. The calculated distances between all of the elements are written into a $N * N$ matrix, which is the basic process of hierarchical clustering. [Joh67] The clusterings get sequence numbers assigned where $L(k)$ is the level of the k^{th} clustering. The following algorithm shows the single steps of Johnson's Single-Link clustering algorithm.

The algorithm merges the clusters until a specified limit (MERGE_MAXIMUM) is reached or the majority of values are in one cluster. This limit is a key factor, because the algorithm has to stop merging the clusters at some point. If the algorithm stops at a point where a majority of values is always in one cluster than it produces always an output. For example if there are five different values available and the algorithm always stops when there are two clusters remaining, there is always a cluster with three elements. As three is always a majority of five, a result is always found, even if the values have a huge deviation from each other.

Algorithm Example

To illustrate Johnson's algorithm, an example distance matrix is used to go through the first merge process. At the beginning $L = 0$ for all clusters. At first the lowest distance value needs to be found in the matrix. This value is marked red in table 4.2. Then cluster C and F get merged into a new cluster, whose distance values are always the lowest distances from the old clusters.

	A	**B**	**C**	**D**	**E**	**F**
A	0	662	877	255	412	996
B	662	0	295	468	268	400
C	877	295	0	754	564	138
D	255	468	754	0	219	869
E	412	268	564	219	0	669
F	996	400	138	869	669	0

Table 4.2: Init distance matrix of the Johnson's algorithm

Algorithm 1 Johnson's Single-Linkage Clustering Algorithm

```
upon $\langle singlelinkINIT \rangle$ do
    $L(0) = 0$
    $m = 0$
    for all $i, j$ such that $0 \leq i, j \leq N$ do
        add $d[(i), (j)]$ to $D$
    end for
end upon

upon $\langle singlelinkMERGE, [D] \rangle$ do
    for all $i, j$ such that $0 \leq i, j \leq N$ do
        $d[(r), (s)] = \min d[(i), (j)] \in D$            //Find least dissimilar pair of clusters
    end for
    $m = m + 1$
    $L(m) = d[(r), (s)]$
    delete $r, s$ from $D$            //Delete rows and cols from clusters r,s
    add $k$ to $D$ with $d[(k), (r, s)] = \min d[(r), (s)], d[(s), (r)]$
end upon

upon $\langle singlelinkCLUSTER, [D] \rangle$ do
    repeat
        $singlelinkMERGE(D)$
    until $(m \geq MERGE\_MAXIMUM)$
end upon
```

After this is done, value m gets incremented and the level of $L(C, F) = 138$. The new distance matrix looks like in table 4.3. Then the second merge round of the algorithm can begin and again the lowest element is searched.

	A	**B**	**C,F**	**D**	**E**
A	0	662	877	255	412
B	662	0	295	468	268
C,F	877	295	0	754	564
D	255	468	754	0	219
E	412	268	564	219	0

Table 4.3: The matrix after one merge

Then the algorithm continues, by merging cluster D and E into a new cluster and by setting the level of the cluster to $L(D, E) = 219$.

	A	**B**	**C,F**	**D,E**
A	0	662	877	255
B	662	0	295	468
C,F	877	295	0	564
D,E	255	468	564	0

Table 4.4: The matrix after the second merge

The merging of the matrix continues until all data samples are in one cluster or the merge limit is reached.

Time Complexity

The time complexity of Single-Linkage clustering is $O(n^2)$. This is because the calculation of the distance matrix already takes $O(n^2)$ calculation steps and therefore is very time consuming. The smallest element in the matrix can be found during the creation of the matrix. Merging and deleting clusters can be done in $O(n)$ and updating the next best merge in each step also in $O(n)$. [JMF99]

CHAPTER 5

Automotive Case Study

This section discusses a case study used to test the different algorithms explained in section 4.3, by implementing an exemplary application with context to the automotive area. For this application we took two inner car functions, which have mixed criticality requirements for safety and security.

Odometer Subsystem

The first subsystem is an odometer layer which computes and stores the current mileage. In order to get the distance covered by the car, the snapshot of the current speed needs to be taken and multiplied with the elapsed time of the last snapshot. In order to get this elapsed time, the global timebase of the TTSoC can be used. This time is multiplied it with the speed of the car and added to the final result of the odometer. This resulting speed value needs to be stored during the whole lifetime of the car because the resale value depends on it.

ABS Subsystem

The second subsystem is an Anti-lock Braking System (ABS), which simple purpose is to prevent wheel lock-up during heavy braking. [BDN+04] This system is introduced to provide a safer driveability of the car and to prevent the abrasion of the wheels. In practice a module called Electronic Control Unit (ECU) detects the wheel lock-up as a sharp increase in wheel deceleration. There are several different implementations of ABS. In our case a four channel, four sensor system is used, which has a sensor on each wheel and separate valves for braking pressure to each wheel. In this ABS implementation, the wheel deceleration is not used to get the brake force value. Therefore a reference speed value is measured at the engine, which makes it possible to calculate a slip value for each wheel. If this value is below a fixed ABS slip limit, the subsystem computes the appropriate brake force, for the desired wheel. The detailed implementation of this ABS subsystem is given in the following algorithm.

Algorithm 2 The implementation of the ABS subsystem

```
1: upon $\langle calcABS, [speed\_w1, speed\_w2, speed\_w3, speed\_w4, speed\_eng, brkfc] \rangle$ do
2:   if $speed\_eng \leq ABS\_MINSPEED$ then
3:     return $brkfc$
4:   end if
5:   $slip[0] = speed\_w1/speed\_eng$
6:   $slip[1] = speed\_w2/speed\_eng$
7:   $slip[2] = speed\_w3/speed\_eng$
8:   $slip[3] = speed\_w4/speed\_eng$
9:   for all $i$ such that $0 \leq i \leq 4$ do
10:    if $slip[i] \leq ABS\_SLIP$ then
11:      $abs\_brkfc[i] = brkfc * slip[i]$
12:    end if
13:  end for
14:  return $abs\_brkfc$
15: end upon
```

Subsystem Requirements

The traditional setup of a car is to deploy the ABS and odometer tasks separately on it. This means that the low secure subsystems are physically separated from the high secure subsystems and therefore the setup is straightforward.
In order to reduce complexity and to establish a reusable environment, modern realisations tend to an integrated setup. For this, the ABS and the odometer are implemented on a SoC as a mixed-criticality application. In this integrated setup, it is often required that data needs to be sent from the low secure tasks to the high secure applications. Therefore integrated setup requires a mechanism to upgrade the reliability of the data before it is used in the high secure subsystem. In order to do that a Validation Middleware (VaM) is used to upgrade the reliability of the data.

In this application each subsystem has its own safety and security requirements. For example, a failure of the ABS system might have severe consequences on the safety side, whereas the information processed within this subsystem does not constitute an important asset, therefore the security requirements are low. (If someone wants to crash a car maliciously, there are easier ways like tampering the brakes to achieve this.) Contrarily, the odometer represents a major asset, because a cars resale value depends on the mileage shown by this device. No car has crashed because of an erroneous odometer. Therefore its safety requirements are low. In order to provide the high requirements for security in the odometer, it is located in the highest security level (Level 4). As this subsystem needs untrusted speed data from a sensor as an input, arrangements need to be made in order to upgrade the reliability of the input data. To achieve this, the odometer receives five redundant speed values from the four wheels and the engine. By using these five untrusted values, the VaM inside the micro component of the odometer upgrades the dependability of the data and calculates a trusted speed value. This speed value is now used in the odometer layer to compute and store the current mileage.

5.1 Simulation Environment Structure

In order to establish a simulation environment as realistic as possible, a state of the art implementation of the TTSoC is used. As already described in section 3.4 the TTSoC provides a well suited environment for implementing a mixed-criticality application. The basic structure of the previously mentioned exemplary application is given in figure 5.1.

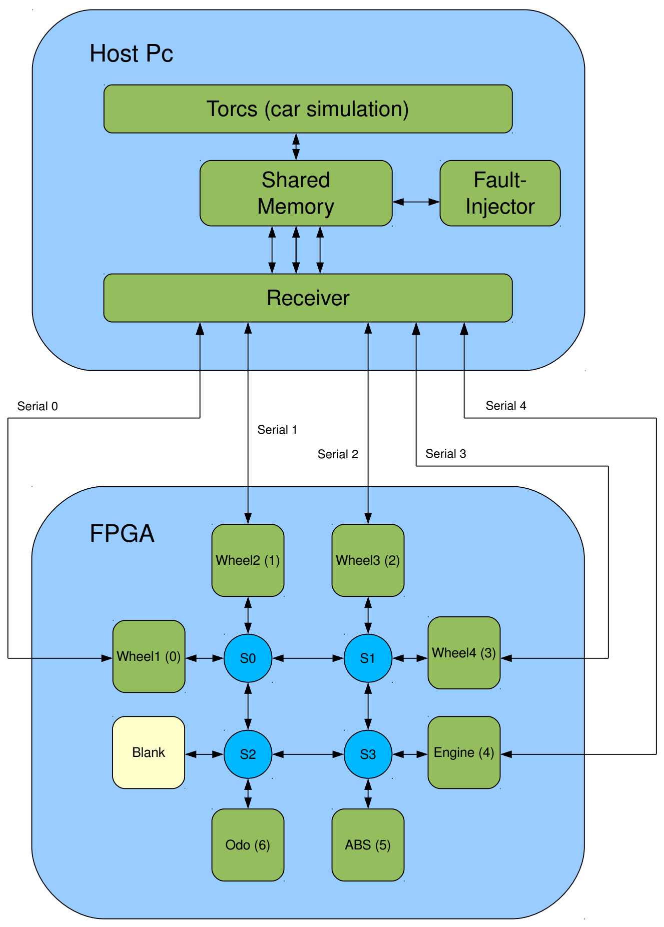

Figure 5.1: The complete simulation environment

Basically this system is divided into two parts. The car simulation which is running on a normal host PC and the ABS and odometer subsystems, which are implemented on the external hardware. To establish a relatively simple connection from the host PC to the hardware, these two parts are connected through five autonomous serial-interfaces.

On the host PC a repeater program is running which reads from the serial devices, checks the data integrity and stores the data into a shared memory. This memory is read by a car simulation named The Open Racing Car Simulator (TORCS), which provides a realistic physics environment and a very good interface to program robots. A robot is a small plugin which makes it possible to steer the car by a user program.

5.2 Basic Layout of the TTSoC in the Simulation Environment

As already described in detail in chapter 2.1 a TT implementation of a SoC is used to provide the required security mechanisms needed in the simulation environment. As a basic platform, the Altera Stratix III^{TM} Development Kit was used, which provides enough resources for implementing all cores on one single FPGA. (see section 3.3) There are seven autonomous IP-Cores, which are connected through the TTNoC. The fundamental layout of this SoC is shown in figure 5.2.

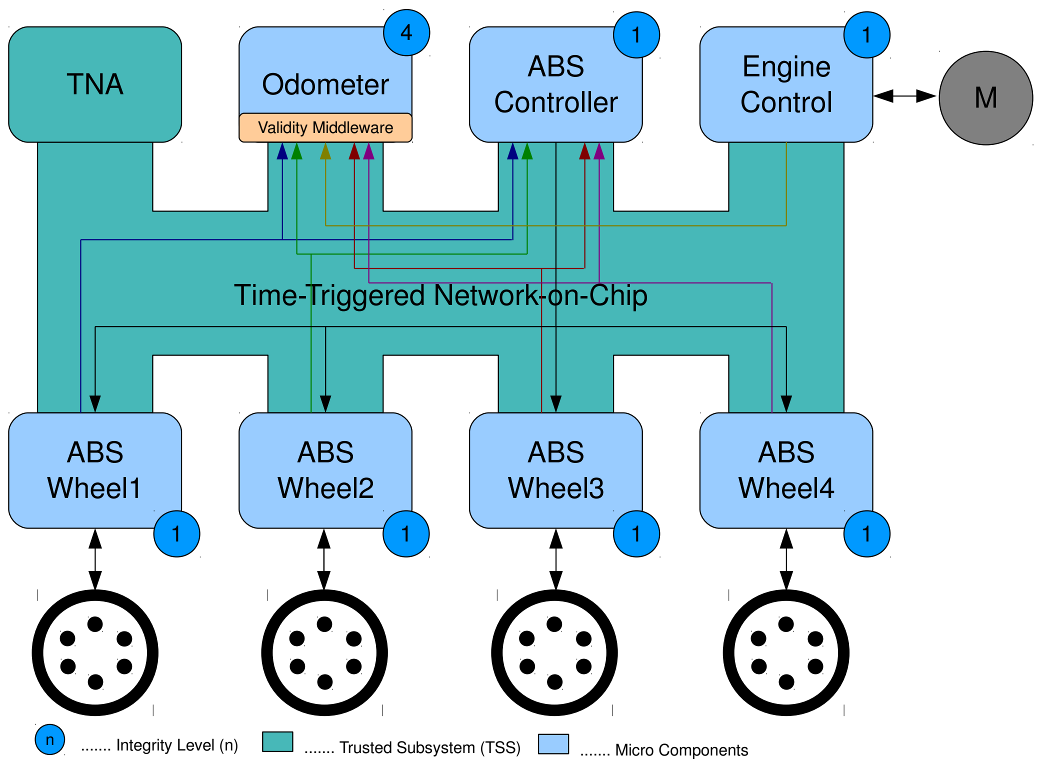

Figure 5.2: The layout of the TTSoC

Basically there are four wheel components which receive speed and brake force values from the sensors on their corresponding wheels. In the simulation environment these sensors are represented by the car simulation running on the host PC. Additionally there is also an engine node which also receives a speed value from the car simulation. The last two cores at the TTSoC contain the secure odometer task and the safe ABS task. The communication between these tasks has to be organised by obeying the integrity policy described in detail in section 2.2.

The NoC itself is realised through standalone fragment switches (see section 2.1) where every switch has four bidirectional lanes. In this simulation environment four switches are connected in a way that enough possibilities for routing are enabled. In figure 5.1 these four switches are named $S_{[0-3]}$.

Message Scheduling

To exchange messages between the processors, the communication is organized in rounds. One round is divided into different Timeslot (TS) in which one communication or calculation task can be carried out. In the VaM application the structure of the communication between the tasks is mainly a send, calculate, send back for the unclassified ABS and a send, calculate for the secure odometer. Therefore the odometer node is just receiving data. A detailed description of the message scheduling is given in table 5.1. TS_6 and TS_{11} are empty to reserve enough time for odometer, ABS calculation and PC to serial communication.

Receiving / **Sending**	**Wheel1**	**Wheel2**	**Wheel3**	**Wheel4**	**Engine**	**Odom.**	**ABS**
Wheel1						TS_1	TS_2
Wheel2						TS_2	TS_1
Wheel3						TS_3	TS_4
Wheel4						TS_4	TS_3
Engine						TS_5	
Odom.	TS_7	TS_8	TS_9	TS_{10}			
ABS							

Table 5.1: Scheduled messages in one round (TS_x stands for Timeslot number x)

PC to Hardware Connection

The host PC is connected with the hardware through five stand alone serial-interfaces. Four of them are directly connected to the IP-Cores of the wheels and the last one is connected to the engine core. These interfaces for communication are used as an easy and fast way of PC to hardware communication and are good enough for the use in this simulation environment. This is, because every computation has a dedicated line and communication is unidirectional. Normally these data connections are provided by a CAN bus, like it is used in nearly all modern cars. For a detailed description of the exact layout of these serial interfaces see appendix A.1.

5.3 Odometer Attack Model

As the odometer subsystem is located in the highest level of the integrity model, the security requirements for this subsystem are high. In order to analyse possible attacks, an *attack tree model* is created. This model was introduced by *Amenaza Technologies Limited*[1] in 2003 [IM03] and is a graphical representation of different ways how an attack on a secure system can look like. It consists of nodes which are connected to each other. Generally there are three kinds of nodes: The AND nodes, OR nodes and the sub-tasks. The first one is called the *root* node and is at the top of the tree. It represents the overall target of the attacker. All nodes under a particular node are called *children* and conversely the node above is called the *parent* node.

On the one hand, if all children sub-tasks need to be fulfilled in order to realise the goal of the parent task, the parent is called an AND node. On the other hand, if the attack is successful because one or more of the children tasks are fulfilled, then the parent task is called an OR node.

The segmentation of the different tasks into small sub-tasks can be continued until the desired level of detail is reached. The tasks which are considered to be at this limit are called *atomic*. An attack tree model for the high secure odometer subsystem is given in figure 5.3.

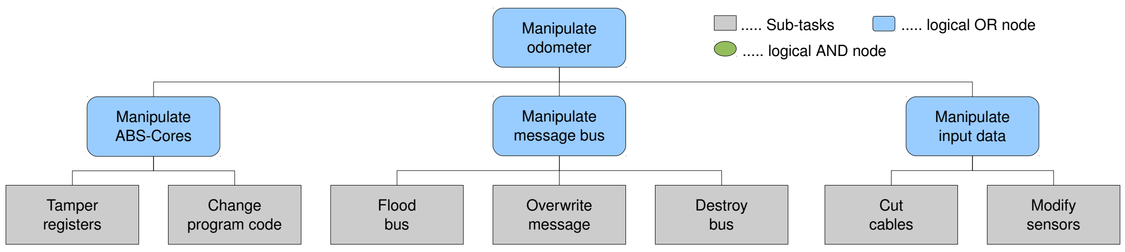

Figure 5.3: Attack model of the odometer subsystem

This figure points out that there are three main parts of the system where attacks at the odometer subsystem can be carried out:

1. The first possible attack is the direct manipulation of the IP-Cores on the TTSoC. Therefore the easiest way to do this is to overwrite the registers where the speed values of the sensors are stored. A second more costlier attempt is to manipulate the whole program running on the ABS cores of the wheels by simply downloading a modified version or just parts of it. The prevention of both varieties of attacks are not covered in this thesis. As the odometer core receives data from five different sources, at least three subsystems need to be manipulated in order to produce a wrong result.

2. The second possible target is to attack the communication interface between the subsystems of the TTSoC. This can be carried out by flooding the communication system with

[1]Amenaza Technologies Limited, `http://www.amenaza.com`

messages in order to block the whole communication or to periodically overwrite messages to induce wrong speed information.
As the communication channels between the single IP-Cores on the TTSoC are carried out by using encapsulated communication channels, both possible attacks are prevented by design. This is given because these encapsulated channels act as a spatial and temporal firewall between the single subsystems. Damaging the complete NoC in order to prevent a message exchange between the subsystems would cause a malfunction on the odometer subsystem and is also not covered in this thesis.

3. The third scenario is the manipulation of the input data itself. This data is provided by measuring the current speed of the car from five redundant sources by using sensors. The easiest way to manipulate these sensors is to simply cut the cables which connect the sensors with the communication system. An other possible attack is to modify the measurement of the installed sensor. This modification depends on the kind of sensor which is used for the conversion. In the optimal situation there are different kind of sensors on each data source which increases diversity between the speed value and also aggravates manipulation.
   This kind of scenario is not completely preventable, but because of the fact that the input data of the odometer is dependent on five redundant sources at least three sources need to be modified.

5.4 PC-Receiver

The repeater is a simple program, which has to be started as the first application of the simulation environment. It simply opens the five serial-interfaces and waits for incoming data. After a so called welcome byte has been received, the program excesses a shared memory and reads the current speed values of the four wheels and the engine. It reads the speed values out of the memory and sends them back over the serial interfaces. The data structure of the shared memory is organized as following:

- Five speed values, four of them come from the sensors of the wheels and the remaining one comes from a sensor directly from the engine.
- One brake force value which contains the current brake force from the brake pedal. This value is either one or zero, for braking on or off.
- Four brake force values received from the hardware. These values come directly from the ABS subsystem and are conducted into each wheel.
- The status word of the simulation environment. It contains important status information for the ABS and odometer cores. The complete structure of this word is pictured in figure 5.4.

RS Bit When this bit is set zero, there currently is no race in progress and therefore the odometer and ABS tasks are not running.

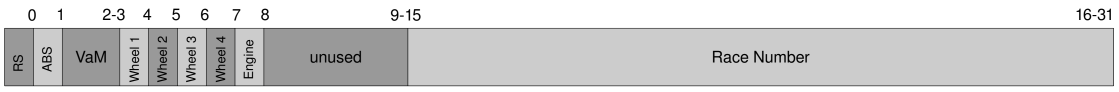

Figure 5.4: The structure of the status word

ABS Bit When this bit is written one, the ABS task is enabled. Otherwise the car is braking normally and the wheels can block during heavy braking.

VaM Bits These two bits are for selecting the different algorithms used in the VaM. If both bits are written zero, the k^{th} *Nearest Neighbour with Delta-Value* is used. If the first bit is one, the *Probabilistic Boxplot* method is used. If the second bit is one, *Histogram Method* is used and finally if both bits are written one, *Single-Linkage Clustering* is enabled.

Wheel Bits The following four bits are to enable and disable the wheels. These bits are for the fault injector to simulate wheel damage.

Engine Bit This bit is to enable and disable the engine.

Race Number The remaining bits are for the current race number. With these remaining 16 bits, this field provides race numbers in the range from zero to 65535.

5.5 Software Based – Fault Injector

As described in section 2.6 *software fault injectors* are widely used, because they do not require additional hardware and can be inserted directly into the application. Software based fault injection is normally used in an application, where a working prototype is already existant. In this thesis a car simulation is used to produce the required input data, which is thought of being close to reality. To produce completely trustable data, the sensors of a real car need to be taken.
The fault injector used in this simulation environment injects faults during runtime by indicating a fault by a trap or an exception. This trap can be specified by the user and has to be injected during an ongoing race. The program itself is standalone and explicitly owns the rights to modify the shared memory at runtime. The user interface of the program is completely text based and provides a few features which are listed as following:

- To simulate a fault at the wheels, the wheel sensor can be disabled, which simply freezes the value at the last value. By rerunning this option the wheel sensor is enabled again.
- It also provides a feature to switch through the VaM algorithms while a race is in progress.
- In order to simulate a possible security attack, the current speed of a sensor can be set at a fixed level. By rerunning this option the sensor is enabled again.
- The ABS can be enabled and disabled.

- In order to reproduce a fault injection scenario with different algorithms, it is possible to define a fault injection schedule. This schedule has to be defined before the race is started by entering the name of the faulty sensor, the start- and end time and the modified speed value. If a race is started, the previously defined faults get injected automatically and the scenario can be reproduced as often as desired. The number of fault injection slots is set to four and can be changed by modifying an in-program variable.

5.6 TORCS – Robot

The Open Racing Car Simulator (TORCS)[2] is a highly portable multi platform car simulation. It provides a realistic physics environment and therefore is well suited to be used as a research platform. In addition it includes an easy to use interface for programming robots. A robot is a program, that drives the car independent from user input. It gets executed from TORCS and receives information about the current position on the track and the status of the car from a structure provided by the car simulation. By using this information, the robot can compute how fast the car can go, in order to stay on track. The robot returns the data to TORCS and the next simulation step can be performed.

In this thesis the main focus is not on programming the best robot, it is more on providing a well enough driving robot, that stays on the track and drives fast enough for the ABS to kick in. To facilitate this, one of many available robot tutorials were used to program the basic steering and braking functions of the robot. In this thesis the "berniw" robot[3] is used. This basic robot is simply extended by a function which accesses the shared memory and reads and writes the speed and brake force values. This enables an easy communication between the car simulation and the external hardware.

The update frequency of the car simulation is appointed at 22 milliseconds. Therefore it produces new speed and brake force values in each period, which has to be read with the same frequency on the TTSoC. As the time format on the TTSoC is realized as it is described in section 2.1, it is not possible to adjust the timebase to exactly 20 milliseconds. Therefore the time in the TSS of the TTSoC is set to 15 milliseconds. This is the next slowest possible value available in the global timebase of the TTSoC. It provides enough accuracy for the car simulation to produce smooth braking at the wheels. The driving car steared by the “flow” robot is given in figure 5.5.

[2]Available at: `http://torcs.sourceforge.net/`
[3]Manual at: `http://www.berniw.org/`

Figure 5.5: The car simulation TORCS with a car steared by the “flow” robot

CHAPTER 6

Analysis of the Algorithms

This section compares the different implementations of the VaM, by using the simulation environment introduced in section 5. This simulation provides four different ways how the fault tolerance mechanisms in the VaM can look like. The realisation of the algorithms are explained in detail in section 4.3 and are called:

- k^{th} Nearest Neighbour with Delta-Value
- Probabilistic Boxplot Method
- Histogram Method
- Single-Linkage Clustering

In order to test the efficiency of the algorithms, they are used during races carried out by a car simulation called TORCS, which is explained in section 5.6. The VaM implemented on the odometer core gets four speed values from sensors on the wheels and one value from the engine sensor. These speed values are sampled with a frequency of 15 milliseconds by the wheel and engine nodes of the TTSoC. This is because the TORCS car simulation is producing speed values with a frequency of 22 milliseconds and as no value can be neglected the TTSoC needs to run faster. Depending on the track, the car that is driving on, and the acceleration speed it is possible that the wheels can spin or lock-up. This is also measured by the sensor on the wheel and therefore these values can deviate from each other. This variance is particularly huge if the car is driving at full speed and then hits the brakes with maximum force. The speed value of all five sensors during a race is given in figure 6.1. As the car has a rear wheel drive, it stands out that especially the front wheels start to lock-up during heavy braking. (Wheel one and two in the figure) This locking is reduced by the ABS but due to the high speed of the car it can't be prevented. As the wheel sensors tend to produce wrong information, the engine sensor produces a speed value which is more trustable. Therefore the value from the engine is more diverse to the the wheel sensors than the wheel sensors to each other. In order to take this into account, the engine value can be weighted more than the wheel sensors by introducing an emphasis of the values.

The deviation of the sensor values is the main testing criteria of the anomaly detection algorithms used in the VaM. The algorithms need to find a majority of speed values with a small variance and eliminate possibly wrong or inaccurate values. The optimal behaviour of an algorithm in this exemplary application is that it always finds a manipulated speed value, which tries to increase or decrease the current value of a sensor and therefore changes the current mileage. Secondly it should neglect values from wheels which are currently locked up or spinning, because the speed values from these wheels are of course not the true speed of the entire car.

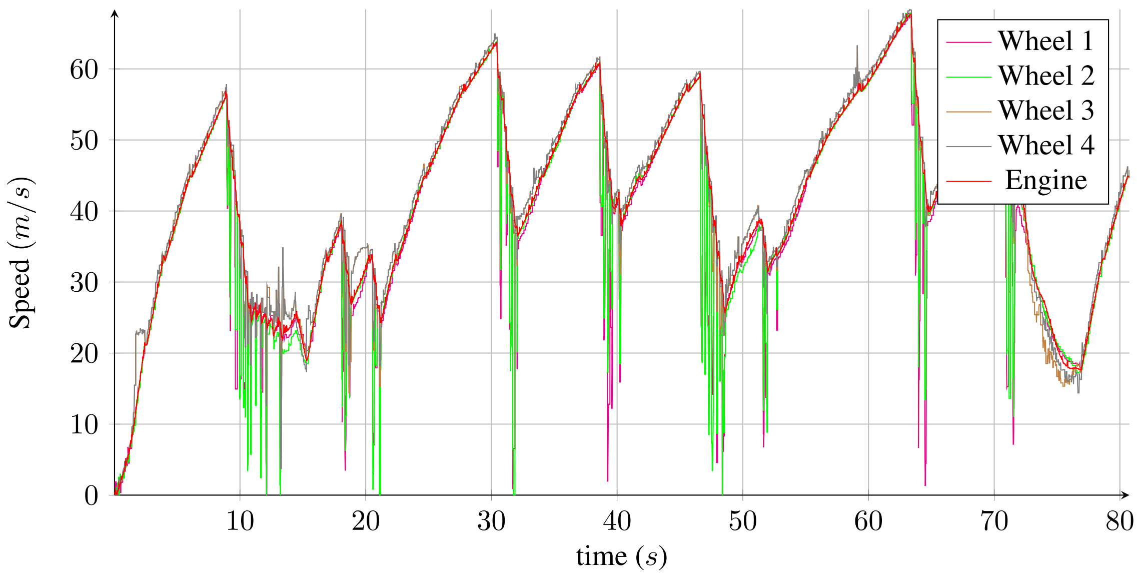

Figure 6.1: All five speed values during one lap of a race

All algorithms produce a set of boolean values as an output. These values illustrate, if the corresponding speed data sample is in the majority and therefore is used in the calculation. For all five speed values one corresponding valid value is created. The last boolean value shows, if the result as a whole forms a majority and is valid to be used in the ongoing calculation.

In the following sections a closer look at the behaviour of the different algorithms is taken. Therefore they are tested with different parameters and under different circumstances. The most important criteria is, that the algorithms are as application-independent as possible, to make this approach reusable in many different fields and that an injected fault is always detected.

6.1 k^{th} Nearest Neighbour with Delta-Value

This algorithm calculates a distance matrix between all values of the data set and includes all values which are inside a given threshold into the majority. A detailed description of this algorithm is given in section 4.3. If this algorithm is used in the same race as depicted before in figure 6.1 a valid value for all five speed values is generated. This value is either *true* or *false* depending on, if the value is used in the calculation or not. The result of a normal run without any faults injected is given in figure 6.2 and depicts the boolean valid values over time.

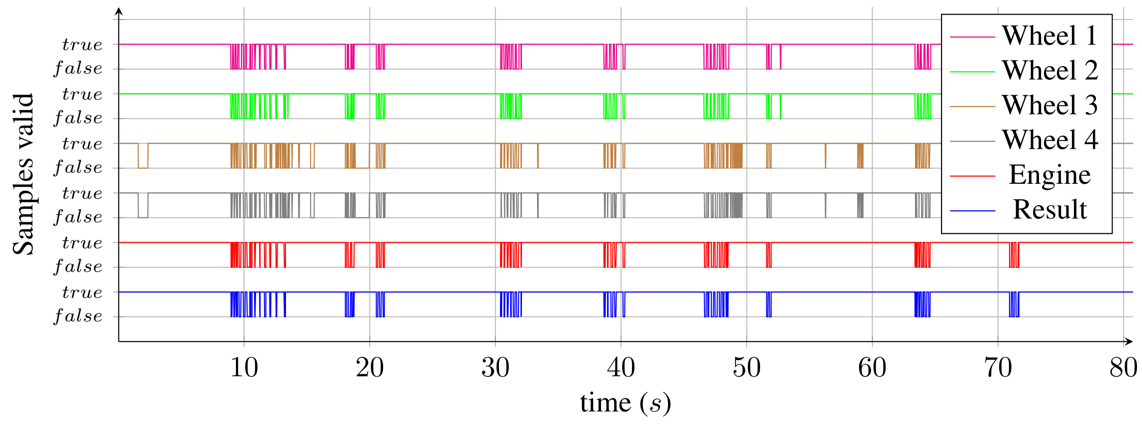

Figure 6.2: Result of the k^{th} Nearest Neighbour – Algorithm (no faults injected)

The blue line on the bottom is representing the complete result of the algorithm during one race. It is easy to see that in some cases the algorithm is not able to produce a valid output because a majority of valid speed values can't be found. This happens mainly during a heavy breaking maneuver and as it is depicted in table 6.1, it happens in 7.5% of all test cases. When this happens, the old speed value is reused what is the best and safest approximation for this situation.

	Percent	Matched Samples	All Samples
Wheel1 samples valid	90.70%	4448	4904
Wheel2 samples valid	90.95%	4460	4904
Wheel3 samples valid	87.34%	4283	4904
Wheel4 samples valid	90.48%	4437	4904
Engine samples valid	93.27%	4574	4904
Result valid	**93.58%**	**4589**	**4904**

Table 6.1: Result of the k^{th} Nearest Neighbour – Algorithm (no faults injected)

If the engine sensor is considered to be valid, the information provided by it is much more predictive because this sensor is not influenced by blocking like the wheel sensors are. Therefore it is three times more weighted than the other sensors and the result gets very dependant on it. If a fault now is injected at a sensor which leads to a deviation outside the threshold, the k^{th} nearest neighbour algorithm marks this value as invalid and neglects it. A race with four injected faults on wheel1 (time: 10s to 20s), wheel2 (time: 25s to 40s), wheel4 (time: 30s to 40s) and the engine (time: 50s to 65s) is carried out. The injected faulty speed values have been chosen significantly different from the original value. If the faulty speed would be chosen as an average value, it would be possible, that the car coincidentally drives in the range of the fault, and therefore an invalid speed value is considered to be valid.

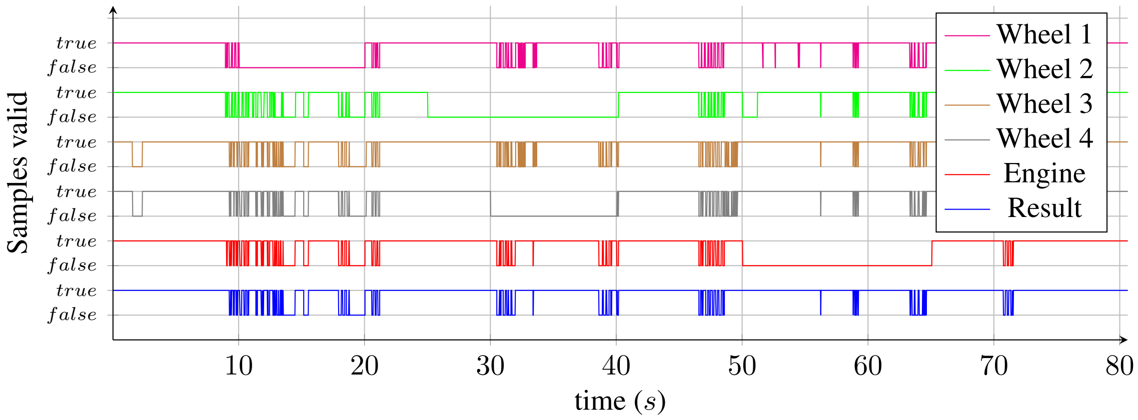

Figure 6.3: Result of the k^{th} Nearest Neighbour – Algorithm (faults injected on W1, W2, W4 and Eng.)

As it is depicted in figure 6.8 and also in table 6.4, all injected faults got properly detected in every sample. In order to refine and approve the overall sensitivity of the algorithm, it needs to be adjusted to the occurring situation which is done in the following refinement section.

	Percent	Matched Samples	All Samples
Wheel1 samples valid	79.09%	3854	4873
Wheel2 samples valid	68.68%	3347	4873
Wheel3 samples valid	84.40%	4113	4873
Wheel4 samples valid	76.67%	3736	4873
Engine samples valid	70.82%	3451	4873
Result valid	**88.49%**	**4312**	**4873**
Wheel1 fault wrong det.	0.00%	0	603
Wheel2 fault wrong det.	0.00%	0	897
Wheel3 fault wrong det.	0.00%	0	0
Wheel4 fault wrong det.	0.00%	0	598
Engine fault wrong det.	0.00%	0	908

Table 6.2: Result of the k^{th} Nearest Neighbour – Algorithm (faults injected on W1, W2, W4 and Eng.)

Figure 6.4 shows the resulting speed value (blue line) of this race with the five old speed values transparent in background. It is obvious that the found result value is quite close to the real speed of the car and a definite improvement of reliability of the data is produced. The threshold value of these two experiments was set to $3m/s$. Changing this value is discussed in the following section. From second 18 to second 20 of the race a duration is pictured where no majority was found. In this case the value of the last conversion is reused which is the best solution for this situation.

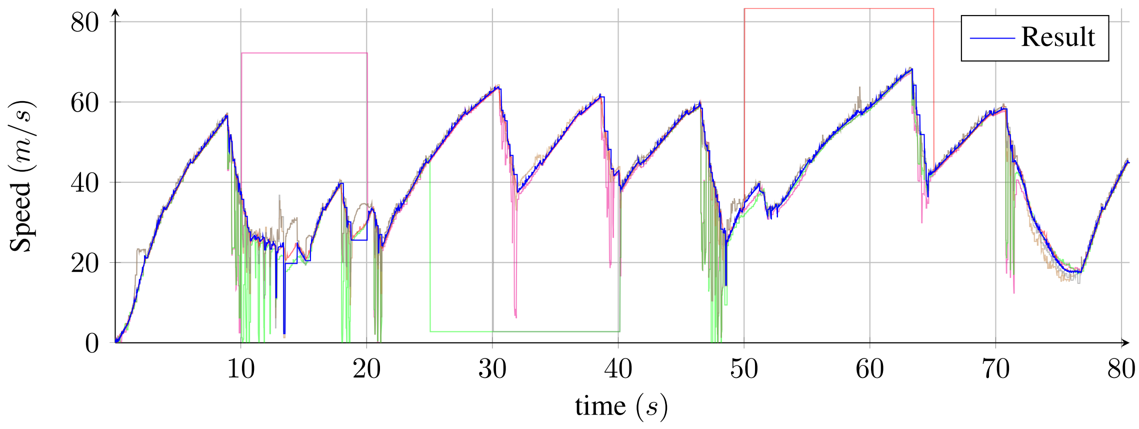

Figure 6.4: Resulting speed value of the k^{th} Nearest Neighbour – Algorithm (faults injected on W1, W2, W4 and Eng.)

Algorithm Refinement

This algorithm is mainly dependent on the threshold value which has to be adjusted corresponding to the car's highest speed, the condition of the wheels and most of it the speeding and braking behaviour of the driving person.

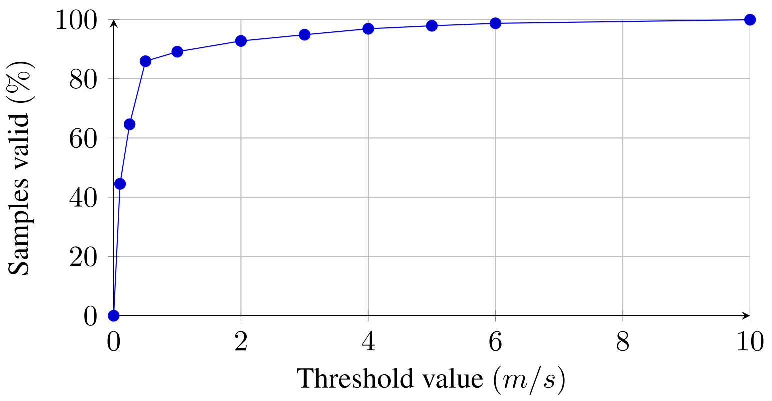

Figure 6.5: Samples valid for the corresponding threshold value

Figure 6.5 shows a number of experiments carried out with different threshold values. In this race no fault was injected and so all disturbances come from normal car behaviour. This figure points out, that a threshold value of $2m/s$ would still be feasible to use in an ongoing race.

The algorithm limits in detecting a fault scenario are when three injected faults form a majority. This can happen if three or more values deviate from the normal behaviour and are erroneously the same. Otherwise all injected faults are at least detected and marked as not usable.

6.2 Probabilistic Boxplot Method

This algorithm prints a boxplot diagram for each set of data samples and declares all values which are outside of the whiskers as anomalous. Details about this algorithm are given in section 4.3. At the first experiments of this algorithm all values even the injected faults were detected as valid. The main problem of this was the big range of the IQR and the huge multiplication factor of 1.5. If these parameters are used as it is explained in detail in section 4.3 the system is very inaccurate. By doing some experiments, a suiteable multiplication factor of 0.9 was found. The results are given in figure 6.8 and table 6.3.

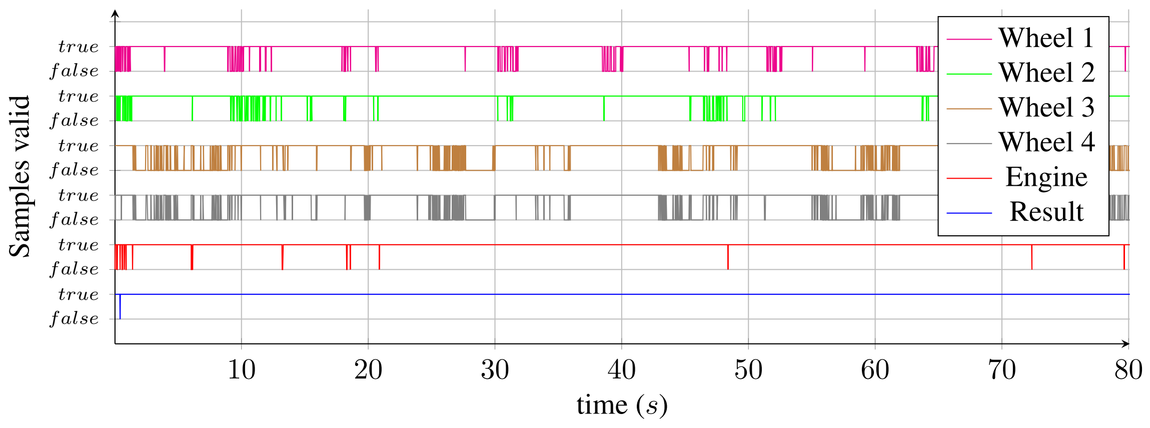

Figure 6.6: Result of Boxplot – Method (no faults injected)

As it is depicted in table 6.3 nearly no speed values found no majority and therefore no data was neglected. As a small amount of data loss is tolerable, it would have been possible to reduce the multiplication factor even more. This would produce the good effect, that the algorithm gets more sensitive for injected faults. The result of this race is pictured in figure 6.7 where the resulting speed value is very close to the five input speeds. Also a lot of faulty speed values got neglected.

	Percent	Matched Samples	All Samples
Wheel1 samples valid	92.71%	4565	4924
Wheel2 samples valid	96.71%	4762	4924
Wheel3 samples valid	68.89%	3392	4924
Wheel4 samples valid	70.82%	3487	4924
Engine samples valid	99.37%	4893	4924
Result valid	**99.96%**	**4922**	**4924**

Table 6.3: Result of Boxplot – Method (no faults injected)

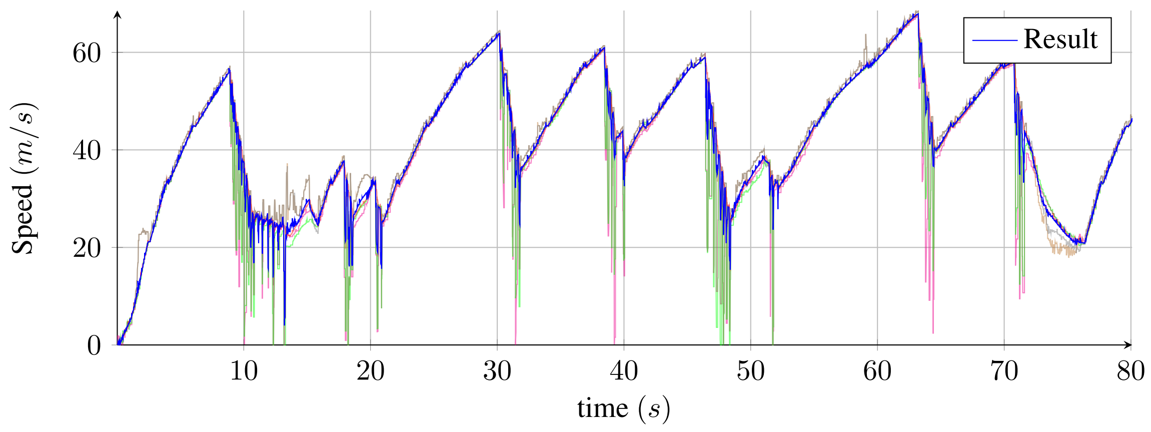

Figure 6.7: Resulting speed value of the Boxplot – Algorithm (no faults injected)

The second experiment is carried out during a fault injection scenario with faults on wheel1, wheel2, wheel4 and the engine. This is the same scenario as it was used with the first algorithm. Again around 4900 data samples are generated and used as an input for testing the algorithm. The following figure 6.8 and table 6.4 show the result of this scenario. Most of the injected faults and even the injection of two faults at the same time is detected very efficiently. The small amount of wrong detection happens during a braking manoeuvre while two faults got injected at the same time. As the engine sensor is weighted three times more, even this faulty behaviour reduces the impact of this wrong detection.

A possible improvement of this result is described in the algorithm refinement section but as this algorithm is very application dependent it needs a lot of parameter tuning to get perfect results.

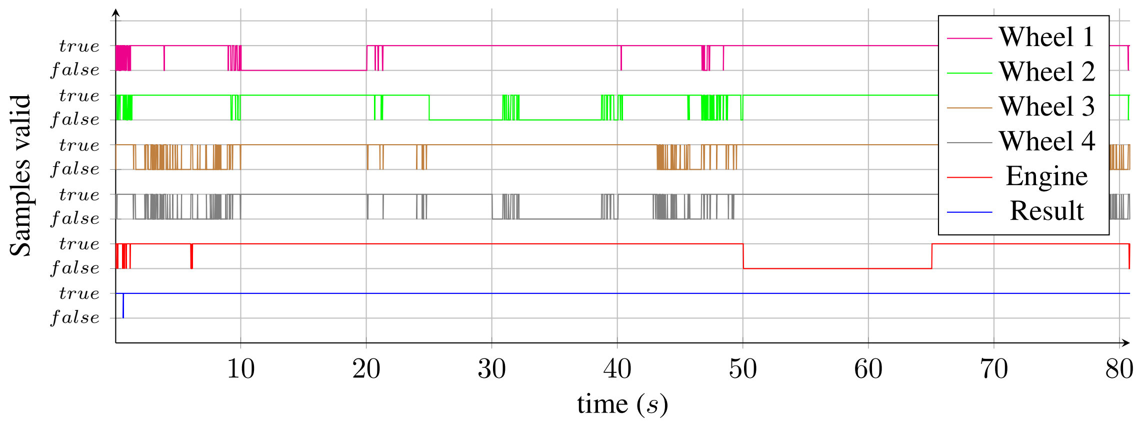

Figure 6.8: Result of the Boxplot – Method (faults injected on W1, W2, W4 and Eng.)

	Percent	Matched Samples	All Samples
Wheel1 samples valid	84.39%	4092	4849
Wheel2 samples valid	80.55%	3906	4849
Wheel3 samples valid	81.09%	3932	4849
Wheel4 samples valid	72.74%	3527	4849
Engine samples valid	81.07%	3931	4849
Result valid	**99.98%**	**4848**	**4849**
Wheel1 fault wrong det.	0.00%	0	597
Wheel2 fault wrong det.	8.96%	81	904
Wheel3 fault wrong det.	0.00%	0	0
Wheel4 fault wrong det.	13.48%	81	601
Engine fault wrong det.	0.00%	0	900

Table 6.4: Result of Boxplot – Method (faults injected on W1, W2, W4 and Eng.)

The resulting speed value of this fault injection scenario is given in figure 6.9. There the resulting speed of the algorithm is pictured by the blue line and the five input speed values are printed transparent in the background. Single injected faults get detected without a problem and also produce adequate speed results. Also because of the high detection rate of the majority of values, nearly all the time a valid result is found. An interesting attribute of this result is that in a breaking event of the car the IQR increases and as a consequence the algorithm gets more tolerant for faulty values. This is visible in a fast change of speed in second 31 to second 33.

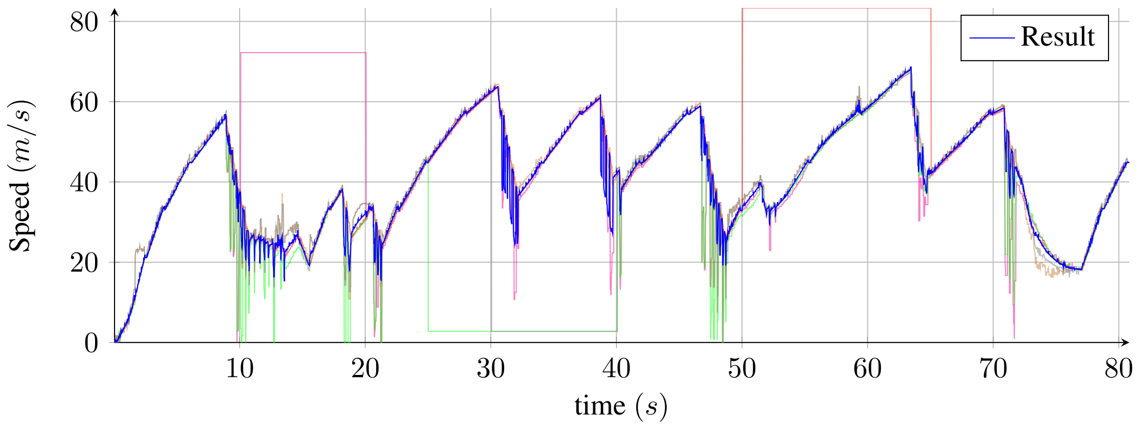

Figure 6.9: Resulting speed value of the Boxplot – Algorithm (faults injected on W1, W2, W4 and Eng.)

Algorithm Refinement

The main problem by using this kind of technique to find outliers is, that an average value is built which includes all available values in the calculation. If now two values get modified at the same time these average value also changes what can have significant effects. This impact particularly is very huge when the number of data samples is limited as it is the case in this application. The results would have been significantly better when double or even more sources are available. Then also a lower value of the IQR multiplication factor would have been possible because the result would have been more obvious.

An interesting aspect of this algorithm is the always variable threshold value to detect the outliers. For example, during braking the values of the front and back wheels deviate from each other. In this case this algorithm also produces a bigger IQR and therefore gets more tolerant for faults. On the other hand if all speed values are close to each other, as it is during acceleration of the car, the algorithm gets more sensitive when a fault is injected. This is a very interesting behaviour and can achieve even better results in applications with a large number of data samples.

6.3 Histogram Method

This algorithm is the simplest but also fastest implementation of the problem statement. It creates a set of bins and assigns the speed values to it. The main advantage of it is the runtime which is depicted to be $O(n)$. (details see section 4.3) The first run of this algorithm is given in figure 6.10 and table 6.5.

It produces similar results as the first k^{th} nearest neighbour method. But there is a slight difference to it. As the buckets are created statically in the init procedure of the algorithm they have a fixed size. This size should not be too small because otherwise a majority can't be found in most of the cases. Therefore the main aspect for such a short runtime is also a major drawback because it is not as accurate as others. The size and range of the bins is also application dependent and has to be chosen individually.

The scenarios where no majority was found were about 4% which is a very good result. As less valid rounds would also produce a good speed value, a change in the size of the bins would have been still possible.

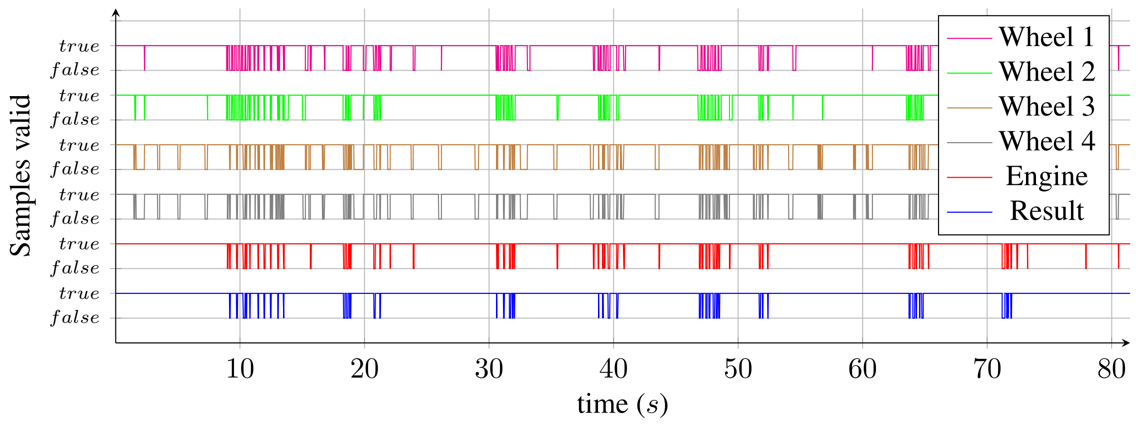

Figure 6.10: Result of the Histogram – Method (no faults injected)

	Percent	Matched Samples	All Samples
Wheel1 samples valid	88.12%	4404	4998
Wheel2 samples valid	89.68%	4482	4998
Wheel3 samples valid	84.75%	4236	4998
Wheel4 samples valid	85.51%	4274	4998
Engine samples valid	94.74%	4735	4998
Result valid	**96.04%**	**4800**	**4998**

Table 6.5: Result of Histogram – Method (no faults injected)

The second experiment is carried out by again inducing three faults at the wheel sensors and one at the engine. As all of these faults are far away from the real speed of the car they get properly detected and the faulty values are neglected. The result of this simulation is given in figure 6.11 and table 6.6. It is easy to see that a small fault injection scenario immediately decreases the number of valid samples. In this case about 7% less results were found. This is still enough for a very good result but it clearly indicates that this algorithm is not as stable as the previously mentioned approaches.

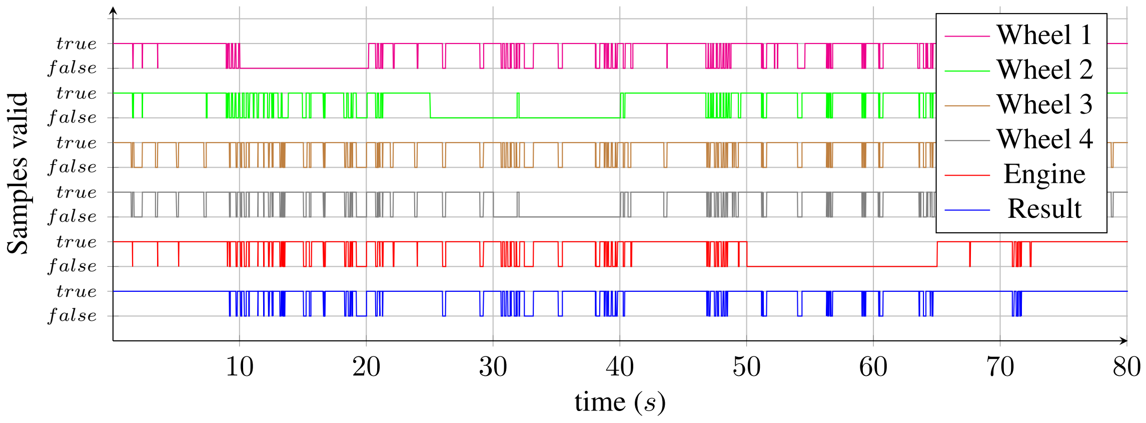

Figure 6.11: Result of the Histogram – Method (faults injected on W1, W2, W4 and Eng.)

	Percent	Matched Samples	All Samples
Wheel1 samples valid	75.03%	3719	4957
Wheel2 samples valid	69.64%	3452	4957
Wheel3 samples valid	83.20%	4124	4957
Wheel4 samples valid	74.90%	3713	4957
Engine samples valid	71.47%	3543	4957
Result valid	**89.19%**	**4421**	**4957**
Wheel1 fault wrong det.	0.00%	0	609
Wheel2 fault wrong det.	0.85%	8	939
Wheel3 fault wrong det.	0.00%	0	0
Wheel4 fault wrong det.	1.28%	8	624
Engine fault wrong det.	0.00%	0	939

Table 6.6: Result of Histogram – Method (faults injected on W1, W2, W4 and Eng.)

The following figure 6.12 shows the resulting speed value of the fault injection experiment drawn by the blue line. As these faults deviate quite much from the original speed of the car, all faults got properly detected and a valid speed value was found in most cases. The results produced with this easy approach is still quite effective because even with a lack of accuracy with faults that are very close to the real value, they are not influencing the result value that much. Therefore all values with the biggest impact on the end result are detected.

As there are still some scenarios where no majority is found, (see second $18 - 20$) the result has some inaccuracies which are not that influential on the odometer value.

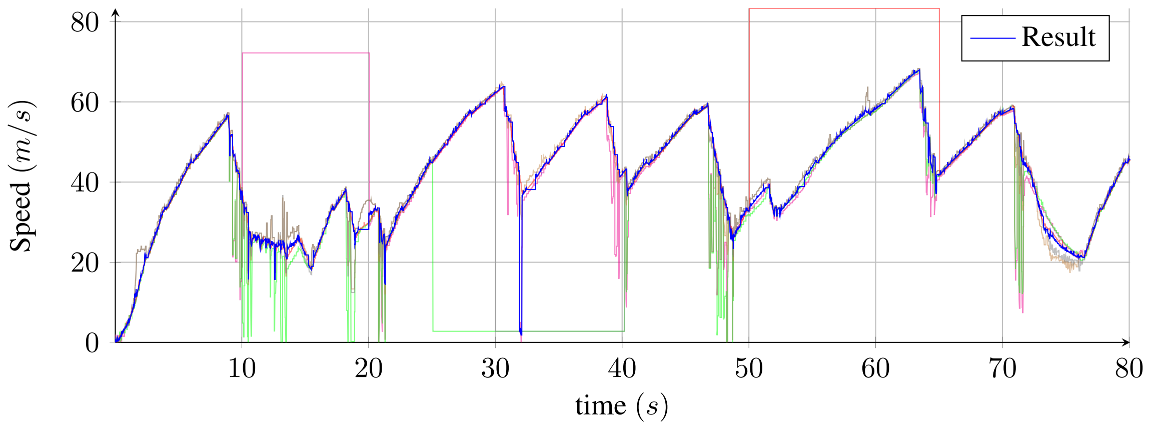

Figure 6.12: Resulting speed value of Histogram – Method (faults injected on W1, W2, W4 and Eng.)

6.4 Single-Linkage Clustering

This algorithm produces a consensus by putting all available data sets into a cluster and merges the closest clusters with each other until a majority is found or the limit is reached. The details about the algorithm are given in section 4.3 where also a small example of Johnson's algorithm is carried out [Joh67]. A normal run of the algorithm without any injected faults is pictured in the following figure 6.13 and table 6.7. As the merge limit is chosen to be two remaining clusters and the algorithm stops after a majority has been found, there are always neglected values. These neglected values are the speeds with the largest distance to each other and therefore the most inexact ones.

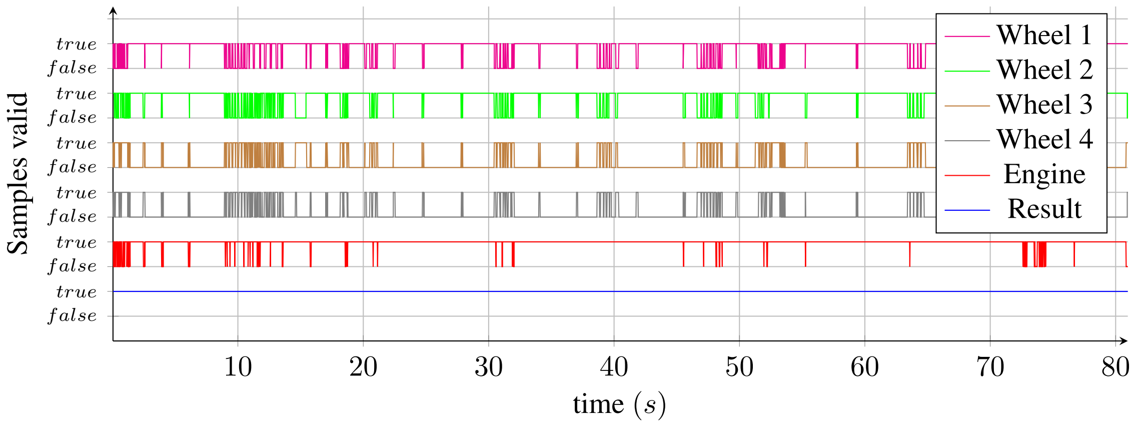

Figure 6.13: Result of the Singe-Link Clustering (normal race)

As this algorithm merges the clusters until a majority is found and as there are no faults injected in this simulation, the algorithm produces a valid result in each case. Changing this is discussed in the algorithm refinement section. It also stops merging when a majority is reached and therefore neglects also values which are close to the real speed but not needed in the majority. This is visible in 6.7 by the percent values of the wheel sensors. The values of the back-wheels get neglected in about 80% of all test cases.

	Percent	Matched Samples	All Samples
Wheel1 samples valid	83.35%	4144	4972
Wheel2 samples valid	84.67%	4210	4972
Wheel3 samples valid	20.76%	1032	4972
Wheel4 samples valid	19.73%	981	4972
Engine samples valid	96.58%	4802	4972
Result valid	**100.00%**	**4972**	**4972**

Table 6.7: Result of Singe-Linkage Clustering – Method (no faults injected)

The next discussed simulation is a race where again four faults got injected. One on wheel1, the second one on wheel2 and wheel4 at the same time and the last one on the engine sensor. The result of this race is depicted in figure 6.14 and table 6.8.

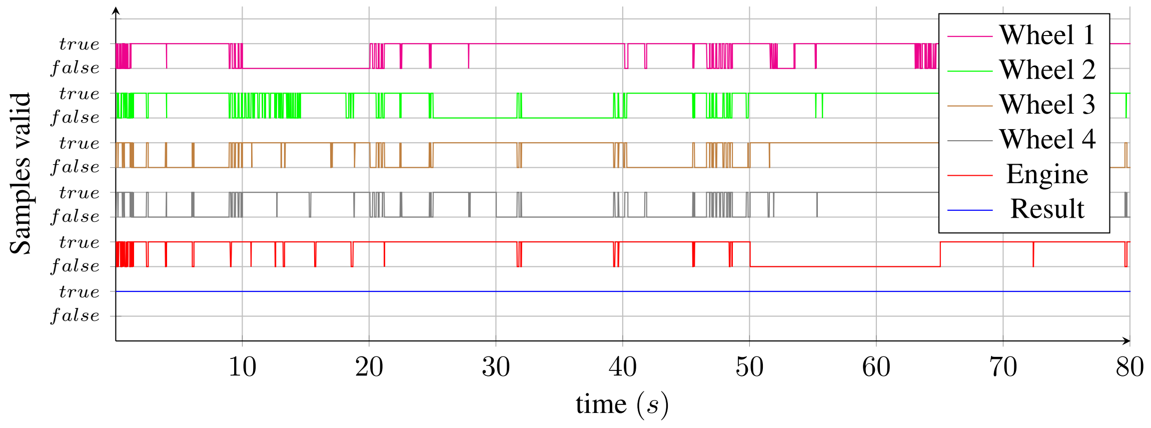

Figure 6.14: Result of the Singe-Linkage Clustering (faults injected on W1, W2, W4 and Eng.)

During all test cases a valid result is found and also all injections got properly detected. As the engine is still weighted three times more than all the other values, the result is very dependant on it. The algorithm again tolerates up to two faults at a time and is able to detect more faults if they are not the same. The small amount of wrong detections is produced during a breaking manoeuvre in which the front wheels locked-up and produced a very small speed value. As the injected faults are also small values they formed a majority and produced a wrong result. As the lock-up of the wheels normally happens for a short time this is not really influential on the result value.

	Percent	Matched Samples	All Samples
Wheel1 samples valid	76.86%	3756	4887
Wheel2 samples valid	71.91%	3514	4887
Wheel3 samples valid	58.07%	2838	4887
Wheel4 samples valid	46.57%	2276	4887
Engine samples valid	78.27%	3825	4887
Result valid	**100.00%**	**4887**	**4887**
Wheel1 fault wrong det.	0.00%	0	592
Wheel2 fault wrong det.	2.63%	24	913
Wheel3 fault wrong det.	0.00%	0	0
Wheel4 fault wrong det.	3.75%	23	613
Engine fault wrong det.	0.00%	0	927

Table 6.8: Result of Singe-Linkage Clustering (faults injected on W1, W2, W4 and Eng.)

In figure 6.15 the resulting speed value is depicted by the blue line. The five sensor values with the injected faults are pictured transparent in the background.

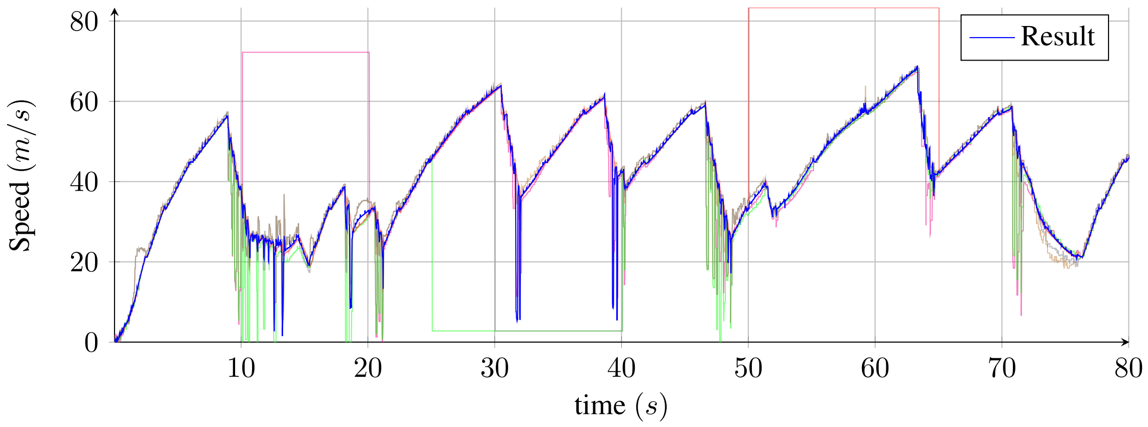

Figure 6.15: Resulting speed value of Singe-Linkage Clustering (faults injected on W1, W2, W4 and Eng.)

Algorithm Refinement

As the result of this algorithm is still quite good and there are no definable scaling factors an other way to improve it has to be found. An interesting way to do that would be the introduction of an additional threshold value which defines a merge limit of two clusters. As the previous version merges clusters as long as a majority is found or the limit is reached, this would make the algorithm more sensitive for manipulation. It would produce more unresolved conversions, but also would never detect an invalid value as valid.

CHAPTER 7

Conclusion

In this last chapter the conclusions and insights are presented which were gained by the implementation of the Validation Middleware (VaM) as a fault tolerant approach for the upgrade of information integrity in an upstream information flow.

By using the *deterministic* communication channels of the TTNoC it is guaranteed that all messages belong to their corresponding data set and have a *consistent delivery order* with respect to multiple encapsulated communication channels. [Pau08, Page 83] It also prevents erroneous data sources from flooding the communication channels which would block the whole communication on the NoC. Spatial protection of each component is usually achieved by using memory management units. Because of the spatial separation of every micro component in the TTSoC this is provided by design. These properties make the TTSoC well applicable for the use in a mixed criticality system which is shown in section 3.4 where this architecture is compared to the MILS architecture. [Rus81]

By the use of Totel's model as a integrity policy for guarding the information flow between principals with different security clearances, upstream communication is prevented. This impedes data from a low integrity level to interfere with higher ones and therefore the induction of faults in a secure component is prevented. For the scenarios where upstream communication is still needed, the Validation Middleware (VaM) is used to upgrade the integrity of this information flow. For this, the VaM needs *diverse* redundant inputs to upgrade the information integrity with the fault tolerant mechanism inside the VaM. This integrity upgrade can be done by realising inexact voting algorithms and the use of the basic concepts of anomaly detection. (see section 4.3) As it is depicted in section 6, the online detection of possible faults with this kind of algorithms works quite well, but there are limits:

Fallout or manipulation of wheel sensors is tolerable as long as a majority of sensors is still correct. If a majority of values is incorrect and not the same, the behaviour of each algorithm is different. The k^{th} nearest neighbour method, the probabilistic boxplot method and the his-

togram method find no result and reuse the last valid value. The single-linkage clustering algorithm merges the values until a majority is found, even if other algorithms would have marked some values as wrong. (see section 4.3 for a detailed description of the algorithms) Because of this, the clustering algorithm always produces an output. The optimal solution for dealing with a missing majority is different and depends on the application. On the one hand, the time where no majority is found can be very long and therefore the last valid value has to be reused for a long period. This would cause a huge deviation to the original value and can produce a wrong result.
On the other hand, if an algorithm always finds a solution it maybe has to use values whose deviation from the real data is so huge that the end result is changed by it. As the average of all values in the majority is built and as there are also some correct values in it, the deviation of the end result is reduced by the correct values in the majority.

The worst case of the whole fault scenario happens when wrong values accidentally are the same and form a majority. Then the valid values are a minority and the faulty values fully determine the result.

Looking at the exemplary automotive application with the five speed sensors, two faulty sensors can be detected. The main problem in this application is the breaking scenario, where a huge deviation between the different speed values can arise. This can happen because of the locked-up or spinning wheels during braking. If there are two ongoing fault injections, both with a low value and if also the wheels are blocking and produce a low value, the result speed of the car can get nearly zero even if it is still driving with a high speed. This kind of scenario is pictured in figure 6.15 in second 32 and points out a possible problem which exists with all realisations of the inexact voters in this thesis. How this kind of scenario can be detected is discussed in the following section 7.1.

7.1 Outlook

All of the selection techniques used in this thesis look at the data instances of each round separately. Therefore each algorithm is not aware of the behaviour of the system in the previous round and consists of no memory. This is done on purpose because the equipment of an algorithm with this kind of memory makes the prediction of each decision very difficult. As all of these approaches need to be certified to a high integrity level, a memory inside the algorithms would be hard to certify.

Because of this, the main outlook for this approach is to find a solution for the voting problem which is easy to certify and able to detect a fault scenario like pointed out in the previous example. Some possible solutions for this kind of problem are neural networks and genetic algorithm techniques. [CSB+95] An interesting approach was created at the *University of Illinois* in 2006 where Hidden Markov Models (HMMs) where used to capture the error and attack free scenarios of the environment. [BGKI06]
Also the approach of Subramaniam et al. in 2006 [SPP+06] is very interesting, where the un-

derlying distribution function of the sensor values in a sliding window is estimated. This data distribution allows to combine the sensor data and to find possible outlier values.

Most of this approaches learn the normal behaviour of the system and continuously compare the ongoing behaviour with the past. If the ongoing data deviates from the saved manner, an error is detected and arrangements can be made to select out the faulty data. As this kind of algorithms can learn each behaviour in every situation, they are well applicable for the use as a reusable fault tolerant approach inside the VaM, with a limit. As this kind of algorithm also needs to be certified and consists of an inner memory, an approach has to be found which is able to certify. Because of the large-scale of these approaches this is a very hard challenge.

APPENDIX A

Simulation Environment – Setup

This guide should provide a full instruction, how the Validation Middleware (VaM) simulation environment has to be built up in order to work properly. The chapter starts by activating the FPGA development board and by establishing the serial connection to the host PC. It also provides a detailed guide for setting up the software on the host PC and explains how to download the program onto the hardware.

A.1 Hardware – Setup

In order to build up the hardware a few components are needed, which are listed as following:

- An *Altera* Stratix III FPGA Development Board[1] (FPGA number *EP3SL150F1152*). Including a standard USB-Cable for downloading the user software and for debugging.
- A *Terasic* HSMC to GPIO Daughter Board[2]
- An *EXSYS* (EX-1334) USB 1.1 to 4S Serial Ports Converter.
- Four standard Serial Cables.
- 20 x *Measuring Leads*[3] with two insulated sockets for 0.64 mm round pillar and 0.64 mm rectangular pillar.
- A x86 or x64 host PC with Linux or Windows operating system. This thesis just covers the setup by using a Linux operating system. (Ubuntu 8.04 LTS is used)

[1]Manual at: `http://www.altera.com/literature/hb/stx3/stx3_siii5v1.pdf`
[2]Manual at: `http://www.terasic.com.tw/en/`
[3]Manual at: `http://at.rs-online.com/web/0775732.html`

- A *Maxim* MAX232CWE Chip[4] to transform the low voltage VCC from the HSMC Daughter Board, to the needed 12 Volts of the serial device. The *Maxim* Chip and the needed circuit is used in form of a pre-made device, pictured in figure A.1.

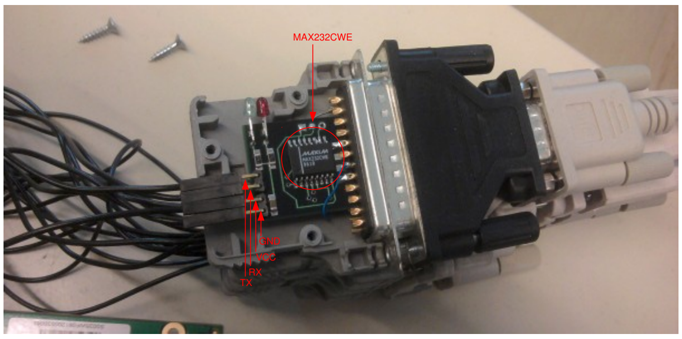

Figure A.1: Maxim MAX232CWE Chip with wiring

The heart of the hardware setup is the Altera Stratix III^{TM} Development Kit. It is connected with the host PC over a standard USB-Cable and five serial devices. The setup of the serial devices is covered in appendix A.1. The FPGA Board provides two extension slots called High-Speed Mezzanine Card (HSMC) ports. The *Terasic* Daughter Board has to be connected at the HSMC Port A of the FPGA Board. To power up both boards a 12 Volts / 4.3 Ampere power adapter is needed.

Serial Devices

As already mentioned, five serial-interfaces are needed, in order to connect the four wheel nodes and the engine node to the host PC. These interfaces are not provided by the FPGA development board and have to be created manually. Therefore each processor, which needs a serial interface, gets an additional module, called Avalon Jtag UART. This module needs a RX-Pin as an input and a TX-Pin as an output. All RX and TX pins are mapped to their corresponding PINs on the HSMC extension board as listed in table A.1.

These ten pins get connected with the five *Maxim* devices, by using the black *Measuring Leads*. To create a fully functional serial device, also VCC and GND need to be connected with the other ten *Measuring Leads*. After this is done, it is possible to connect the *Maxim* devices with the host PC, by using standard serial cables and the USB 1.1 to 4S Serial Ports Converter. As the host PC provides one serial port, the engine serial is directly connected to the host PC without using the USB to serial converter. As the output connectors of the *Maxim* devices are old obsolete parallel ports, five parallel to serial converters have to be interconnected. After establishing the serial connection, it is recommended to test the interface by interposing an oscilloscope. If sending and receiving data is working, it is possible to continue with setting up the host PC, which is described in section A.2.

[4]Manual at: `http://datasheet.octopart.com/MAX232CWE-Maxim-datasheet-3759.pdf`

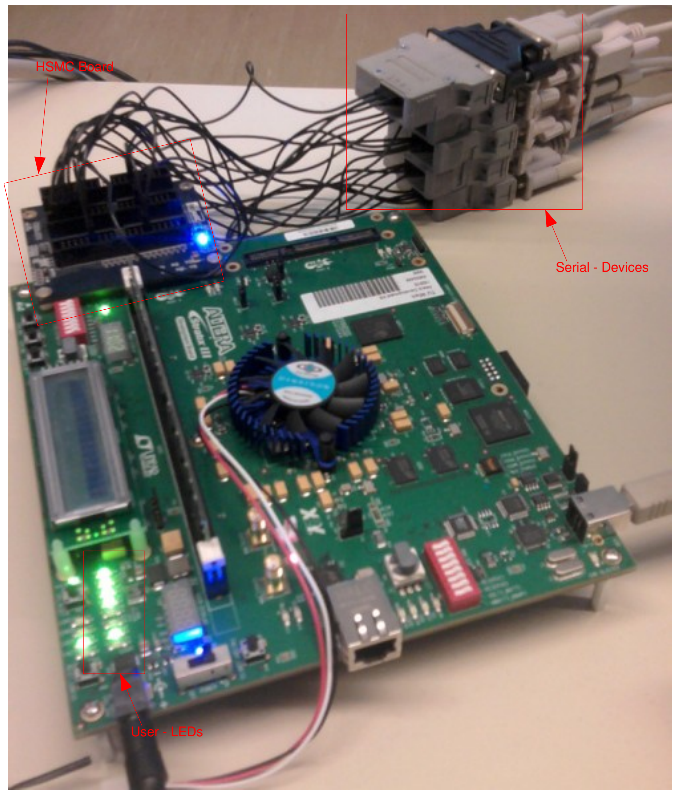

Figure A.2: The FPGA board with the HSMC expansion board and the serial devices

A.2 Host PC – Setup

The operating system, which is running on the host PC can be chosen autonomous, as *Altera* provides either a Windows or a Linux Version of the *Quartus Development Suite*. However this guide just covers the set-up of the environment by using a Linux operating system. In our case it is the distribution *Ubuntu 8.04 LTS*. If you are planning to use Windows, a few things like the software installation are different, but the basic idea should be the same.

The first thing, that has to be done after installing the operating system, is to prepare everything for the installation of the Quartus Development Suite. The main steps have to be performed as following:

- Install the package *tcsh*.
- Find out which shell currently is used, by typing: "*ls -la /bin/sh*"
- Change the shell to bash by typing: "*sudo rm /bin/sh*" and "*sudo ln -s bash /bin/sh*"

After successfully switching the shell, the current running shell window has to be restarted, to make the changes effective. Then the installation process of the *Quartus Development Suite*

Signal Name	FPGA Pin Name	HSMC Pin Name
rxd_to_uart_0	PIN_AC11	HSMC_TX_P0
txd_from_uart_0	PIN_AC9	HSMC_TX_P1
rxd_to_uart_1	PIN_AB10	HSMC_TX_N0
txd_from_uart_1	PIN_AC8	HSMC_TX_N1
rxd_to_uart_2	PIN_AJ4	HSMC_RX_P0
txd_from_uart_2	PIN_AH5	HSMC_TX_P2
rxd_to_uart_3	PIN_AJ3	HSMC_RX_N0
txd_from_uart_3	PIN_AH4	HSMC_TX_N2
rxd_to_uart_4	PIN_AG4	HSMC_RX_P1
txd_from_uart_4	PIN_AE8	HSMC_TX_P3

Table A.1: Pin assignment internal FPGA pins to HSMC board pins

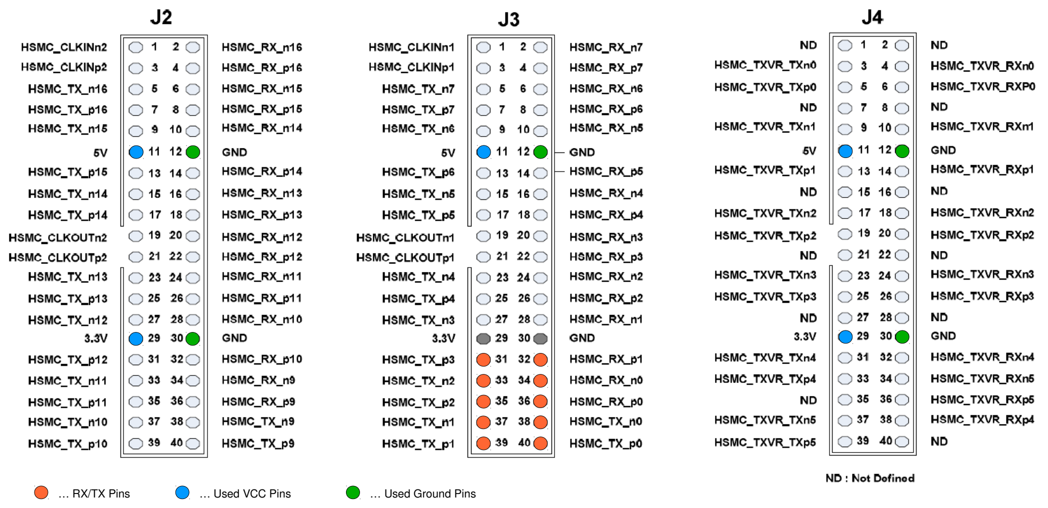
J2
HSMC_CLKINn2 1 2 HSMC_RX_n16
HSMC_CLKINp2 3 4 HSMC_RX_p16
HSMC_TX_n16 5 6 HSMC_RX_n15
HSMC_TX_p16 7 8 HSMC_RX_p15
HSMC_TX_n15 9 10 HSMC_RX_n14
5V 11 12 GND
HSMC_TX_p15 13 14 HSMC_RX_p14
HSMC_TX_n14 15 16 HSMC_RX_n13
HSMC_TX_p14 17 18 HSMC_RX_p13
HSMC_CLKOUTn2 19 20 HSMC_RX_n12
HSMC_CLKOUTp2 21 22 HSMC_RX_p12
HSMC_TX_n13 23 24 HSMC_RX_n11
HSMC_TX_p13 25 26 HSMC_RX_p11
HSMC_TX_n12 27 28 HSMC_RX_n10
3.3V 29 30 GND
HSMC_TX_p12 31 32 HSMC_RX_p10
HSMC_TX_n11 33 34 HSMC_RX_n9
HSMC_TX_p11 35 36 HSMC_RX_p9
HSMC_TX_n10 37 38 HSMC_TX_n9
HSMC_TX_p10 39 40 HSMC_TX_p9
J3
HSMC_CLKINn1 1 2 HSMC_RX_n7
HSMC_CLKINp1 3 4 HSMC_RX_p7
HSMC_TX_n7 5 6 HSMC_RX_n6
HSMC_TX_p7 7 8 HSMC_RX_p6
HSMC_TX_n6 9 10 HSMC_RX_n5
5V 11 12 GND
HSMC_TX_p6 13 14 HSMC_RX_p5
HSMC_TX_n5 15 16 HSMC_RX_n4
HSMC_TX_p5 17 18 HSMC_RX_p4
HSMC_CLKOUTn1 19 20 HSMC_RX_n3
HSMC_CLKOUTp1 21 22 HSMC_RX_p3
HSMC_TX_n4 23 24 HSMC_RX_n2
HSMC_TX_p4 25 26 HSMC_RX_p2
HSMC_TX_n3 27 28 HSMC_RX_n1
3.3V 29 30 GND
HSMC_TX_p3 31 32 HSMC_RX_p1
HSMC_TX_n2 33 34 HSMC_RX_n0
HSMC_TX_p2 35 36 HSMC_RX_p0
HSMC_TX_n1 37 38 HSMC_TX_n0
HSMC_TX_p1 39 40 HSMC_TX_p0
J4
ND 1 2 ND
HSMC_TXVR_TXn0 3 4 HSMC_TXVR_RXn0
HSMC_TXVR_TXp0 5 6 HSMC_TXVR_RXP0
ND 7 8 ND
HSMC_TXVR_TXn1 9 10 HSMC_TXVR_RXn1
5V 11 12 GND
HSMC_TXVR_TXp1 13 14 HSMC_TXVR_RXp1
ND 15 16 ND
HSMC_TXVR_TXn2 17 18 HSMC_TXVR_RXn2
HSMC_TXVR_TXp2 19 20 HSMC_TXVR_RXp2
ND 21 22 ND
HSMC_TXVR_TXn3 23 24 HSMC_TXVR_RXn3
HSMC_TXVR_TXp3 25 26 HSMC_TXVR_RXp3
ND 27 28 ND
3.3V 29 30 GND
HSMC_TXVR_TXn4 31 32 HSMC_TXVR_RXn4
HSMC_TXVR_TXp4 33 34 HSMC_TXVR_RXn5
ND 35 36 HSMC_TXVR_RXp5
HSMC_TXVR_TXn5 37 38 HSMC_TXVR_RXp4
HSMC_TXVR_TXp5 39 40 ND
ND : Not Defined
● ... RX/TX Pins
● ... Used VCC Pins
● ... Used Ground Pins

Figure A.3: Possible wiring of the HSMC expansion board

9.0 Sp2 can begin. It is very important to use exactly this version of the environment. When the setup program is asking for the installation path of the program, a path with no spaces is needed. Otherwise the installation will not work properly.

After the installation process is completed, the permissions of the USB-Devices still need an adjustment. This can be done by inserting the following line at the end of /etc/fstab:

```
procbususb /proc/bus/usb usbfs auto 0 0
```

Finally some global system wide variables need to be set. There is a file included in the project folder called *.bash_profile*. Copy this file into the home directory and insert the following lines into your *.bashrc* file. Modifying the .bashrc file is not necessary, if another distribution than *Ubuntu* is used.

```
if [ -f $HOME/.bash_profile ]; then
   . $HOME/.bash_profile
fi
```

A look inside of *.bash_profile* is recommended, because this is the place where all system variables are set. The three most important paths are the TTSOC_DESIGN, SOPC_KIT_NIOS2 and the QUARTUS_ROOTDIR. They have to be adjusted as it is needed in the operating system. In order to download a program to the FPGA, the jtagd daemon needs to run. With this daemon running, it is possible to check if the connected hardware is detected properly, by using the command *jtagconfig -n*. If this program is returning: *Unable to lock chain*, insufficient rights for the USB devices are granted. This can be solved by using the following command:

```
$ sudo chmod -R a+rw /proc/bus/usb/
```

Now all requirements are fulfilled to compile and download the TTSoC design. This is done by using the Makefile and can take an hour or more, depending on the CPU speed. As already mentioned in section 5.2 the TTSoC design consists of seven IP-Cores, which are realized through a CPU called the *Nios II processor*. This processor is automatically generated during compile time and because of it's modular design, it can be easily extended by using a program called the *SOPC Builder*.

The actual tasks are executed on this processors. They are ready to be downloaded, after the TTSoC design is on the hardware. To do so an editor called *nios2-ide* should be used. When this editor starts up the workplace has to be switched into the directory called: *$TTSOC_DESIGN/usr/software*. This directory is the home of the software, that is executed on the processors. After switching the workplace, the projects for each of the seven processors need to be imported. Each processor software consists of two projects, the main program and the library. While importing the library, it is required to add the properties of the processor by selecting the file "userland.ptf" which is located in the folder called "usr/sopc". In addition, the compile flag "MY_CPU_ID=$ID" needs to be added as a preprocessor in the project properties. To compile the processor software, simply run the "build all" command in the editor. This compilation takes a few minutes. After it is finished the result can be downloaded to the processors by using the Makefile. As mentioned in section 5.4 a program called repeater is needed on the host PC to read data from the serial. This program is located inside the "pcsim/repeater" folder and has to be compiled and executed before the processor software is downloaded. Otherwise the processors start to infinity listen on the serial and freeze. If the permission to the serial devices is denied, the file inside the folder "udev/" has to be copied into the folder /etc/udev/rules.d/. In order to make the changes effective, the udev daemon has to be restarted and the devices have to be reconnected. Restart udev with the command:

```
$ sudo /etc/init.d/udev restart
```

Now the whole system is running in idle mode, because no race currently is in progress. To start a new race a car simulation called TORCS is used, which has to be configured as described in the following chapter.

Compiling The Open Racing Car Simulator

The functionality of this car simulator is described in detail in section 5.6. The source code of the program is located inside the "pcsim/torcs-1.3.1" folder. In order to compile TORCS, a few dependencies need to be fulfilled by the used operating system:

- Hardware accelerated OpenGL (usually provided by your distribution)
- GLUT 3.7
- PLIB version $\geq$ 1.8.5
- OpenAL
- libpng and zlib (usually provided by your distribution)

Ubuntu 8.04 satisfies all dependencies, by simply installing the libraries from the package manager, with one exception. The version of the PLIB library is too old. To install this library in the correct version, the source code for compiling it is added to the project in the "lib/torcs/plib-1.8.5" directory. If the compilation of this library fails and a x64 processor is used, the following compile flags are needed:

```
$ export CFLAGS="-fPIC"
$ export CPPFLAGS="-fPIC"
$ export CXXFLAGS="-fPIC"
```

After the required dependencies are fulfilled, TORCS can be compiled by using the following commands:

```
$ cd torcs-1.3.1
$ ./configure        # --prefix="target dir", --enable-debug or --disable-
    xrandr might be of interest
$ make
$ make install
```

The version of TORCS, which is used in the project, includes a robot called “flow” in the "torcs-1.3.1/src/drivers/flow" directory. To drive with this robot, “Practice” has to be chosen in the main menu of the game. In the following configuration menu, the robot and the tracks can be selected. Finally everything is adjusted to start the first race.

The output of the simulation is logged by the repeater program, by writing the data into a file inside the "race_results" directory. As already described in section 5.5, there also is a program called the fault injector. It is located inside the "pcsim/faultinjector" directory and can be easily compiled by using the corresponding Makefile. While using this program, it is possible to switch the VaM algorithm and inject faults into the sensors.

Debugging the Simulation Environment

The debug output of the program can be activated by executing a shell script called "*readalluarts.sh*" in the project folder. This script opens seven windows, one for each processor and displays the debug output from the TTSoC design. If this windows print no output, the compile flag ALT_RELEASE has to be changed to ALT_DEBUG. This has to be done in the project options of the "*nios2-ide*" and also in the first line of the Makefile. As printing the debug output is a very resource consuming activity, running the TTSoC with a period of 32 milliseconds is too fast. To slow down this period, the parameter called MSB_PERIODBIT inside the file "etc/vam_CONFIG" has to be set to 30 or higher. In order to make these changes effective, the following command has to be executed before recompiling the whole TTSoC design:

```
$ make writecfg
```

This command executes a python script, which imports some configuration files from the original TTSoC project. This only can be done when the "ttsoc-ng" repository is checked out in the same folder as the "flow" repository.

Debugging with Modelsim

It is also possible to debug the TTSoC design without using the FPGA at all. Therefore the simulation environment called *Modelsim SE* provided by *Mentor* is needed. The version 6.3j of the simulation environment is recommended and tested. After installing this software, the path to the program has to be adjusted inside the already mentioned configuration file called ".bash_profile". Once this is done correctly, the simulation can be started by using the makefile with the command "make sim". This command regenerates the processors and starts Modelsim SE afterwards. If the generation of the processors is already completed and the simulation just needs to be started, this can be done by using the following command:

```
$ vsim usr/sopc/userland_sim/setup_sim.do
```

The simulation environment provides the following options:

- s – Load all design (HDL) files
- c – Re-compile memory contents
- w – Sets-up waveforms for this design
- l – Sets-up list waveforms for this design

APPENDIX B

Implementation Specific Details

This chapter covers additional information, needed to work with the simulation environment of the Validation Middleware (VaM). First the user interface on the FPGA is explained and second the format of the messages sent over the NoC is illustrated.

B.1 FPGA User Interface

As mentioned in the previous chapter, printing debug messages over the Avalon Jtag UART is too slow for normal operations. To provide status output on the development board in realtime, some user programmable interfaces are used. The used peripheries are four Seven Segment Displays and eight user LEDs, which are marked red in figure B.1.

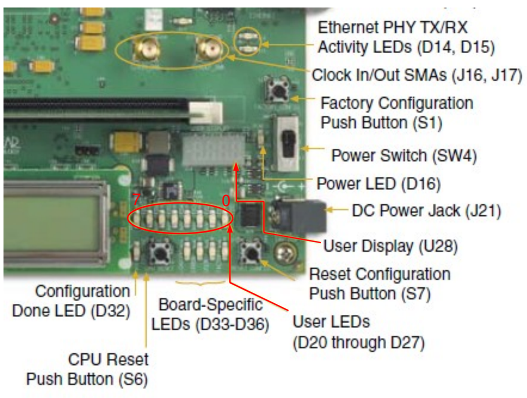
Ethernet PHY TX/RX Activity LEDs (D14, D15)
Clock In/Out SMAs (J16, J17)
Factory Configuration Push Button (S1)
Power Switch (SW4)
Power LED (D16)
DC Power Jack (J21)
7
0
User Display (U28)
Reset Configuration Push Button (S7)
Configuration Done LED (D32)
Board-Specific LEDs (D33-D36)
User LEDs (D20 through D27)
CPU Reset Push Button (S6)

Figure B.1: The user interface which is used on the FPGA

Seven Segment Display: This display is used to print the current odometer value. As there are just four digits available, the accuracy is limited to two decimal places. To make it possible to write all four displays at once, they have to be multiplexed. Responsible for this is an extra hardware module called "*sevensegmux.vhd*", which is interconnected between four eight-bit PIO-Modules in the odometer processor and the seven segment pins on the board.

User LEDs: This eight LEDs are used to indicate status information of the current ongoing race. The function of each LED maps exactly to the first eight bits of the status word pictured in figure 5.4. Therefore LED zero indicates if a race is started, LED one shows the ABS status, LED 2-3 which VaM algorithm is currently activated and the last four LEDs show if a wheel sensor is on or off.

B.2 Sending Floating-point values over the NoC

As described in section 5.2, the five speed values from the wheels and the engine are sent over the Network-on-Chip. At first it has to be mentioned that the On-Chip Memory (OCM) is very limited on the IP-Cores of the TTSoC. A simple call of floating point library functions, often is not possible because of the high amount of static memory needed. So the use of library functions is avoided as much as possible.

The speed values from the serial devices are sent as ASCII characters and saved as a string at the receiving side. In order to get the speed values out of the string, integer functions are used to extract the pre-decimal position and the decimal places separately. This is done because integer library functions use much less static memory then floating point functions. Both integer values get separately transmitted over the NoC and assembled at the ABS and odometer tasks. Therefore floating point values are only needed at the main processors, where more memory is available. Figure B.2 shows the exact format of the messages, sent from the wheel IP-Cores to the odometer and ABS cores.

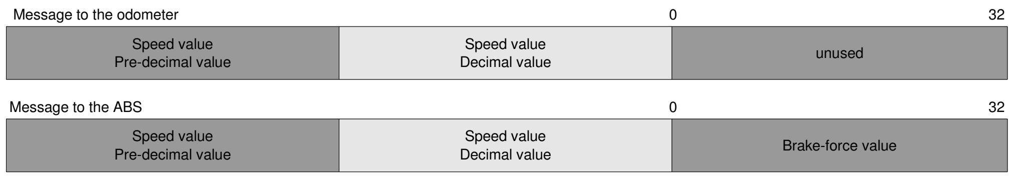

Figure B.2: Format of the messages sent from the wheels to the odometer and ABS

B.3 Logging Messages

This section focuses on how the results from the VaM are transmitted to the host PC, in order to evaluate and compare them. As it is easy to see in figure 5.2, the IP-Core of the odometer has no direct connection to the host PC and also no backloop to the wheel nodes is available. Therefore two additional logging messages are defined, which need to be sent over the NoC to the engine core. From there it is possible to send the values to the host PC, by using the local serial-interfaces. The structure of the logging messages is given in figure B.3. The first message simply consists of the speed values of the four wheel sensors. In this 32 bit word the upper half is used for the pre-decimal position and the lower half for the decimal places. The second message consists of the speed value from the engine, the result word of the VaM and the status word which has already been described in figure 5.4.

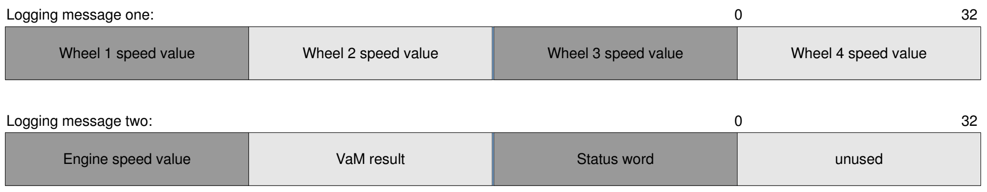

Figure B.3: The logging messages from the odometer core

The VaM result word in the second message provides the information, which values are used to form the majority and if the set of speed value has a majority at all. Details about the bit structure are given in figure B.4.

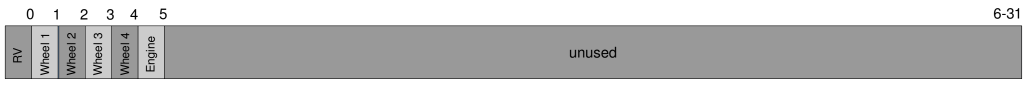

Figure B.4: The structure of the VaM result status word

Bit zero indicates if the result is valid in a whole and bit one to five show which speed value is used in the evaluation. The information from the status messages are used to create the statistics, stored in the directory called the "race_results/". The format of three data samples in the race result file is given as following:

```
6.103980 28.465300 1 1 27.994508 1 1 27.668401 1 1 27.774242 1 1 28.127008 1
    1 1
6.127411 28.109314 1 1 28.934540 1 1 29.999817 1 1 29.999817 1 1 28.127008 1
    1 1
6.140967 28.146800 1 1 28.135593 1 1 27.989921 1 1 28.294250 1 1 28.127008 1
    1 1
```

The first value is the time in nanoseconds, when this measurement sample is taken with respect to when the race is started. Then there are the five speed values with two boolean values added after each speed value. The first boolean value indicates if this speed value is actually used in the calculation and is therefore the output of the VaM algorithm. The second boolean value is one if the wheel sensor was enabled and zero it if was disabled. A wheel sensor is disabled if a fault is currently injected at this sensor. The last boolean value in the result is one if the result is valid in a whole or zero when the round is discarded.

As this format is not use-able with the pgfplots[1] latex libraries, a python script called *splitter.py* helps with preparing this result file for plotting in latex. It simply creates a data file for every speed and result value, with an added timestamp. Additionally a latex table is created with a small evaluation of all data samples.

[1] Manual at: `http://www.iro.umontreal.ca/~simardr/pgfplots.pdf`

List of Acronyms

A/C IS	Aircraft Information System
A/C OM	Aircraft Operation and Maintenance
ABS	Anti-lock Braking System
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
CDI	Constrained Data Items
COTS	Commercial off the Shelf
Doctor	Integrated Software Fault Injection Environment
ECU	Electronic Control Unit
EE	Execution Environment
FCR	Fault Containment Region
Ferrari	Fault and Error Automatic Real-Time Injection
FIST	Fault Injection System for Study of Transient Fault Effect
FM	Flight Management
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
HMMs	Hidden Markov Models
HSMC	High-Speed Mezzanine Card
IVPs	Integrity Verification Procedures
MARS	Maintainable Real-Time System
MEDL	Message Descriptor List
MILS	Multiple Independent Layers of Security
MLO	Multi Level Object
MMU	Memory Management Unit
NoC	Network-on-Chip
NVP	N-version programming
NVS	N-version software
OCM	On-Chip Memory
OW	Open World
RATE	Region Approximation and Temperature Estimation
RB	Recovery Blocks
RMA	Resource Management Authority
SLO	Single Level Object
SoC	System-on-Chip

TDMA	Time Division Multiple Access
TISS	Trusted Interface Sub-System
TNA	Trusted Network Authority
TORCS	The Open Racing Car Simulator
TPs	Transformation Procedures
TS	Timeslot
TSS	Trusted Sub-System
TT	Time-Triggered
TTA	Time-Triggered Architecture
TTE	Time-Triggered Ethernet
TTNoC	Time-Triggered Network-On-Chip
TTSoC	Time-Triggered System-On-Chip
UART	Universal Asynchronous Receiver Transmitter
UDI	Unconstrained Data Items
VaM	Validation Middleware
VaO	Validation Object

Bibliography

[AC77] Algirdas A. Avizienis and L. Chen. On the Implementation of N-Version Programming for Software Fault Tolerance During Execution. *In Proc. IEEE COMPSAC 77* , pages 149–155, November 1977.

[ACL89] Jean Arlat, Yves Crouzet, and Jean-Claude Lapire. Fault Injection for Dependability Validation of Fault-Tolerant Computing Systems. *Proc. 19th Ann. Int'l Symp. Fault-Tolerant Computing*, pages 348–355, 1989.

[AFOTH06] J. Alves-Foss, P. W. Oman, C. Taylor, and W. S. Harrison. The MILS Architecture for High-Assurance Embedded Systems. *International Journal of Embedded Systems*, 2:239–247, 2006.

[AGK$^+$85] A. Avizienis, P. Gunningberg, J. P. J. Kelly, L. Stirigini, P. J. Traverse, K. S. Tso, and U. Voges. The ucla dedix system: A distributed testbed for multiple-version software. *Digest of FTCS-15, the 15th International Symposium on Fault-Tolerant Computing*, pages 126–134, 1985.

[ALS$^+$87] Algirdas A. Avizienis, M. R. T. Lyu, W. Schütz, K.-S. Tso, and U. Voges. *Software Diversity in Computerized Control Systems*, volume 2, chapter DEDIX 87 - A Supervisory System for Design Diversity Experiments at UCLA, pages 129–182. Springer-Verlag Wien, October 1987.

[Avi95] Algirdas A. Avizienis. The Methodology of N-Version Programming. *Software Fault Tolerance edited by M. Lyu, John Wiley & Sons*, pages 23–46, 1995.

[BCWW07] A. Bogdanov, D. Carluccio, A. Weimerskirch, and T. Wollinger. Embedded Security Solutions for Automotive Applications. *Advanced Microsystems for Automotive Applications*, pages 1–10, 2007.

[BDN$^+$04] David Burton, Amanda Delaney, Stuart Newstead, David Logan, and Brian Fields. Effectiveness of ABS and Vehicle Stability Control Systems. Technical report, Royal Automobile Club of Victoria (RACV) Ltd, April 2004.

[Bel74] D. E. Bell. Secure Computer Systems: A Refinement of the Mathematical Model. *Electronic Systems Division, Air Force Systems Command*, III, April 1974.

[BGKI06] Claudio Basile, Meeta Gupta, Zbigniew Kalbarczyk, and Ravi K. Iyer. An approach for detecting and distinguishing errors versus attacks in sensor networks. *Proceeding of the 2006 International Conference on Dependable Systems and Networks*, 2006.

[Bib77] K. J. Biba. Integrity Considerations For Secure Computer Systems. Technical report, Mitre Corporation, April 1977.

[BK09] Arindam Banerjee and Vipin Kumar. Anomaly Detection: A Survey. Technical report, ACM Computing Survey, September 2009.

[BL75] D. E. Bell and L. J. LaPadula. Computer Security Model: Unified Exposition And Multics Interpretation. Technical report, MITRE Corp., Bedford, June 1975.

[BM06] Tobias Bjerregaard and Shankar Mahadevan. A Survey of Research and Practices of Network-on-chip. *ACM Computing Surveys (CSUR)*, (1), March 2006.

[CSB$^+$95] P.R. Croll, A.J.C. Sharkey, J.M. Bass, N.E. Sharkey, and P.J. Fleming. *Engineering Applications of Artificial Intelligence*, volume 8, research article Dependable, Intelligent Voting for Real-time Control Software, pages 615–623. Elsevier, December 1995.

[CW87] David D. Clark and David R. Wilson. A Comparison of Commercial and Military Computer Security Policies. *IEEE Symposium on Security and Privacy*, pages 184–194, May 1987.

[EBB01] Donald L. Evans, Phillip J. Bond, and Arden L. Bement. Security Requireents for Cryptographic Modules. *Federal Information Processing Stabdards Publication (Supercedes FIPS PUB 140-1)*, 2001.

[EI03] Wilfried Elmenreich and Richard Ipp. Introduction to TTP/C and TTP/A. *Proceedings of the Workshop on Time-Triggered and Real-Time Communication Systems*, pages 1–9, 2003.

[ELOGV$^+$11] Luis Entrena, Celia López-Ongil, Mario García-Valderas, Marta Portela-García, and Michael Nicolaidis. *Soft Errors in Modern Electronic Systems*, volume 41, chapter Hardware Fault Injection, pages 141–166. 2011.

[ES07] Christian El-Salloum. *Interface Design in the Time-Triggered System-on-Chip Architecture*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, December 2007.

[Gal87] Patrick R. Gallagher. A Guide to Understanding Discretionary Access Control in Trusted Systems. *National Computer Security Center*, 1987.

[GKT89] O. Gunnetlo, J. Karlsson, and J. Tonn. Evaluation of Error Detection Schemes Using Fault Injection by Heavy-Ion Radiation. *Proc. 19th Ann. Int'l Symp. Fault-Tolerant Computing*, pages 340–347, 1989.

[HSR95] S. Han, K.G. Shin, and H.A. Rosenberg. Doctor: An Integrated Software Fault-Injection Environment for Distributed Real-Time Systems. *Proc. Second Annual IEEE Int'l Computer Performance and Dependability Symp.*, pages 204–213, 1995.

[HTI97] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer. Fault Injection Techniques and Tools. *Coordinated Sci. Lab., Illinois Univ., Urbana, IL* , 30 Issue:4:75 – 82, April 1997.

[Hub08] Bernhard Huber. *Resource Management in an Integrated Time-Triggered Architecture*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, January 2008.

[IM03] Terrance R. Ingoldsby and Christine McLellan. Creating secure systems through attack tree modeling. Technical report, Amenaza Technologies Limited, 550, 1000, 8^{th} Ave SW, Calgary, AB, Canada, June 2003.

[IT96] Ravishankar K. Iyer and Dong Tang. Experimental Analysis of Computer System Dependability. *Fault-Tolerant Computer System Design* , pages 282–392, 1996.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys (CSUR)*, 31, Issue 3:264–323, 1999.

[Joh67] Stephen C. Johnson. *Psychometrika*, volume 32, chapter Hierarchical Clustering Schemes, pages 241–254. Springer, 1967.

[KAGS05] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. The Time-Triggered Ethernet (TTE) Design. *Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 1–12, 2005.

[KB03] Hermann Kopetz and Günther Bauer. The Time-Triggered Architecture. *Proceeding of the IEEE*, 91:112–124, January 2003.

[KFG^{+}92] Hermann Kopetz, Gerhard Fohler, Günter Grünsteidl, Heinz Kantz, Gustav Pospischil, Peter Puschner, Johannes Reisinger, Ralf Schlatterbeck, Werner Schütz, Alexander Vrchoticky, and Ralph Zainlinger. The Distributed, Fault-Tolerant Real-Time Operating System MARS. *IEEE Operating Systems Newsletter*, 6(1), 1992.

[KKA92] G.A. Kanawati, N.A. Kanawati, and J.A. Abraham. Ferrari: A Tool for the Validation of System Dependability Properties. *Proc. 22nd Ann. Int'l Symp. Fault-Tolerant Computing*, pages 336–344, 1992.

[KNT00] E. M. Knorr, R. T. Ng, and Tukakov. Distance-based Outliers: Algorithms and Applications. *The VLDB Journal 8*, pages 237–253, 2000.

[Kop92] Hermann Kopetz. Sparse Time versus Dense Time in Distributed Real-Time Systems. *In 12th Int. Conf. on Distributed Computing Systems*, pages 460–468, 1992.

[Kop97] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Number 978-0792398943. Springer, April 1997.

[Kop08] Hermann Kopetz. The Complexity Challenge in Embedded System Design. *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, May 2008.

[KR07] Ian Kuon and Jonathan Rose. Measuring the gap between fpgas and asics. *In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 203–215, February 2007.

[Lam69] B.W. Lampson. Dynamic Protection Structures. *AFIPS Conf. Proc.*, 35:27–38, 1969.

[LCE89] Paul R. Lorczak, Alper K. Caglayan, and Dave E. Eckhardt. A Theoretical Investigation of Generalized Voters for Redundant Systems. *Digest of Papers FTCS-19:The Nineteenth International Symposium on Fault-Tolerant Computing*, pages 444–450, 1989.

[LDPA09] Youssef Laarouchi, Yves Deswarte, David Powell, and Jean Arlat. Connecting Commercial Computers to Avionics Systems. *28th Digital Avionics Systems Conference*, pages 6.D.1–(1–9), December 2009.

[LH94] J.H. Lala and R.E. Harper. Architectural Principles for Safty-Critical Real-Time Applications. *In Proceedings of the IEEE*, 82:25–40, January 1994.

[Mat00] Matteo Matteucci. Hierarchical clustering algorithms, 2000. Available at: `http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html`.

[MTL78] Robert Mcgill, John W. Tukey, and Wayne A. Larsen. Variations of Box Plots. *The American Statistician*, 32:12–16, 1978.

[OPHES06] Roman Obermaisser, Philipp Peti, Bernhard Huber, and Christian El-Salloum. DECOS: An Integrated Time-Triggered Architecture. *Journal of the Austrian professional Institution for Electronic and Information Engineering*, 3:83–95, March 2006.

[Pau08] Christian Paukovits. *The Time-Triggered System-on-Chip Architecture*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, December 2008.

[PK09] Christian Paukovits and Hermann Kopetz. Building Encapsulated Communication Channels in the Time-Triggered System-on-Chip Architecture. Research Report 8/2009, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2009.

[PNMV05] Tan P.-N., Steinbach M., and Kumar V. Introduction to Data Mining. *Addison-Wesley*, 2005. Chapter 2.

[Pri07] Paul J . Prisaznuk. *ARIC Specification 653, Avionics Application Software Standard Interface*. Number 978-0-8493-8438-7. Cary R . Spitzer, 2007.

[Ran75] Brian Randell. System Structure for Software Fault Tolerance. *IEEE-Software Eng.*, SE-1:220–232, 1975.

[Rus81] J. Rushby. Design and Verification of Secure Systems. *Proceedings of the 8th ACM Sympsium on Operating System Principles*, 15(5):12–21, December 1981.

[SPP$^+$06] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online Outlier Detection in Sensor Data Using Non-Parametric Models. pages 187–197, 2006.

[SWJR07] X. Song, M. Wu, C. Jermaine, and S. Ranka. Conditional Anomaly Detection. *IEEE Transactions on Knowledge and Data Engineering*, 19:631–645, 2007.

[TBDB$^+$01] Eric Totel, Ljerka Beus-Dukic, Jean-Paul Blanquart, Yves Deswarte, and David Powell. *A Generic Fault-Tolerant Architecture for Real-Time Dependable Systems*. Number 0-7923-7295-6. Kluwer Academic Publishers, 2001.

[TBDP00] Eric Totel, Jean-Paul Blanquart, Yves Deswarte, and David Powell. Supporting Multiple Levels of Criticality. *ESPRIT project 20716: GUARDS*, 2000.

[TCL90] H. Teng, K. Chen, and S. Lu. Adaptive Real-Time Anomaly Detection Using Inductively Generated Sequentianl Patterns. *In Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy*, pages 278–284, 1990.

[Tra88] P. Traverse. *Use of Diversity in Experimental Reactor Safety Systems*, chapter AIRBUS and ATR System Architecture and Specification, pages 95–104. Springer Wien, New York, 1988.

[Tuk77] John Wilder Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[VM97] Jeffrey Voas and Gary McGraw. *Software Fault Injection: Innoculating Programs Against Errors*. Number ISBN 0-471-18381-4. John Wiley & Sons, 1997.

[Wei69] Clark Weissman. Security Controls in the ADEPT-50 Time-Sharing System. *AFIPS Conf. Proc. 35*, pages 119–133, 1969.

[WESK10] Armin Wasicek, Christian El-Salloum, and Hermann Kopetz. A System-on-a-Chip Platform for Mixed-Criticality Applications. *13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, pages 210–216, May 2010.

[WYF83] J. F. Williams, L. J. Yount, and J. B. Flannigan. Advanced autopilot flight director system computer architecture for boeing 737-300 aircraft. *In AIAA/IEEE 5th Digital Avionics Systems Conference*, November 1983.

Index