FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Functional Safety in KNX

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Marco Steffan
Matrikelnummer 0215884

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao.Univ.Prof.Dr. Wolfgang Kastner
Mitwirkung: Dr. Wolfgang Granzer

Wien, 24.11.2011

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Marco Steffan
Wiesenweg 13, 6170 Zirl

Hiermit erkläre ich, dass ich diese Arbeit selbständig 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 Verfasser)

Abstract

Building automation systems aim at providing a comfortable environment while saving available resources. In case of using fire alarm systems (functional safety) or access-control systems (security) those systems are realized as separate, closed systems interacting with an existing building automation system via dedicated points of interaction. Integrated systems providing functional safety natively are currently hardly available.

This thesis targets an approach to extend the building automation technology KNX with functional safety. In compliance with IEC 61508 (Functional safety of electrical/electronic/programmable electronic safety-related systems) and ISO 13849 (Safety of machinery - Safety-related parts of control systems) an architecture satisfying safety integrity level 3 (SIL3) as defined by IEC 61508 is presented. Security is thereby left unconsidered. SIL3 compliance implies sufficient support on hardware level (fault-tolerance), a standard-conform documentation of all development steps as well as adequate software to detect errors in the hardware and the communication system.

The intention of the thesis is not the provision of a complete implementation of all requirements according to IEC 61508 but rather elaboration of an extension to existing approaches within this field. On that score and in compliance with IEC 61784-3 (Industrielle Kommunikationsnetze - Profile - Teil 3-1: Funktional sichere Übertragung bei Feldbussen) measures to detect errors in the communication system are discussed, architectures for a SIL3 compliant KNX-system are presented and resulting impacts on hard- and software are shown.

Kurzfassung

Gebäudeautomationssysteme dienen in erster Linie der Erzeugung eines komfortablen Raumklimas bei gleichzeitiger, ressourcenschonender Nutzung der zur Verfügung stehenden Energie. Geht man davon aus, dass funktionale Sicherheit (Safety) etwa für Brandmeldeanlagen oder Systemsicherheit (Security) für Zutrittskontrollen erforderlich sind, werden diese Anforderungen durch eigenständige Systeme realisiert, die (im besten Fall) über ausgewählte Schnittstellen mit einem vorhanden Gebäudeautomationssystem kommunizieren. Integrierte Systeme, die bereits "nativ" funktionale Sicherheit zur Verfügung stellen, sind derzeit kaum verfügbar.

Diese Arbeit versucht einen Ansatz zu schaffen, die Gebäudeautomationstechnologie KNX um funktionale Sicherheit zu erweitern. In Übereinstimmung mit den Standards IEC 61508 (Funktionale Sicherheit sicherheits-bezogener elektrischer / elektronischer / programmierbarer elektronischer Systeme) und ISO 13849 (Sicherheit von Maschinen - Sicherheitsbezogene Teile von Steuerungen) wird eine mögliche Architektur erarbeitet, um einen Sicherheitsintegritäts-Level 3 (SIL3) laut IEC 61508 zu erreichen. Systemsicherheit bleibt dabei unberücksichtigt. SIL3 impliziert eine ausreichende Unterstützung der zugrunde liegenden Hardware (Fehlertoleranz), eine Standard-konforme Dokumentation aller Entwicklungsschritte sowie Software, um Fehler in der Hardware und dem Kommunikationssystem zu erkennen.

Ziel dieser Arbeit ist nicht eine vollständige Ausarbeitung aller Erfordernisse gemäß IEC 61508, sondern eine Erweiterung zu bereits bestehenden Ansätzen in diesem Umfeld zu schaffen. Im Zuge dieser Arbeit werden Mechanismen, die Fehler im Kommunikationssystem erkennen, in Abstimmung mit IEC 61784-3 (Industrielle Kommunikationsnetze - Profile - Teil 3-1: Funktional sichere Übertragung bei Feldbussen) diskutiert, Architekturen für ein SIL3 konformes KNX-System vorgestellt und sich daraus ergebende Anforderungen an die Hard- und Software erarbeitet.

Contents

List of Figures

List of Tables

FSoE	Failsafe over CPF 12
CPF	Communication Profile Family
SFF	Safe Failure Fraction
SIL	Safety Integrity Level
PL	Performance Level
PL_r	Required Performance Level
E/E/PES	Electric/Electronic/Programmable Electronic System
SRP	Safety-Related Part
EUC	Equipment under Control
MTTR	Mean Time To Repair
DC	Diagnostic Coverage
CCF	Common Cause Failure
SCL	Safety Communication Layer
PTP	Precision Time Protocol
CRC	Cyclic Redundancy Check
FCS	Frame Checking Sequence
SRDO	Safe Communication Object
SRVT	Safety-relevant Object Validation Time
SA	Safe Address
SPDU	Safety Process Data Unit
APDU	Application Process Data Unit
GSPN	Generalized Stochastic Petri Nets
ANubis	Advanced Network for Unified Building Integration & Services
CPU	Central Processing Unit
OSSD	Output Silicon Switched Device
WCET	Worst Case Execution Time
PES	Programmable Electronic System
MTTF	Mean Time To Failure
FMEA	Failure Mode and Effects Analysis
SCL	Safety Communication Layer
CRC	Cyclical Redundancy Check
CS	Control System
SRESW	Safety-Related Embedded Software
SRASW	Safety-Related Application Software
HVAC	Heating Ventilation Air Conditioning
ROM	Read Only Memory
RAM	Random Access Memory
EPROM	Electrical Erasable Read Only Memory
TMR	Triple Modular Redundancy
BCI	BatiBus Club International
EIBA	European Installation Bus Association
EHSA	European Home System Association
HBA	Home and Building Automation

CSMA	Carrier Sense Multiple Access
TPCI	Transport Layer Protocol Control Information
APCI	Application Layer Protocol Control Information
PDU	Process Data Unit
CAFMS	Computer Aided Facility Management System
FSCP	Functional Safety Communication Profile
SCM	Safety Configuration Manager
CiA	CAN in Automation
SCT	Safeguard Cycle Time

CHAPTER 1

Introduction

1.1 Motivation

Traditionally, Building Automation Systems (BAS) provide basic services like Heating, Ventilation and Air Conditioning (HVAC), lighting and shading. Safety critical applications like fire detection and alarm systems are usually stand-alone units which interact with BAS using dedicated gateways. Increasing requests for BAS in safety-critical environments ask for advanced mechanisms to integrate safety-critical technology into BAS. Therefore, it is necessary to define what safety-critical properties are and what their meaning is - to detect hazardous events in an automation system. These can be failures in hardware, software or the underlying communication-system like a "wrong message" in any way. Such a message can be wrong in a sense of its value-domain or in its time-domain. Detection of the afore mentioned failures requires implementation of certain mechanisms in hardware and software.

The requirements for safety-critical systems are specified in two common standards - ISO 13849 (Safety of machinery - Safety-related parts of control systems) and IEC 61508 (Functional safety of electrical/electronic/programmable electronic safety-related systems). Especially, IEC 61508 presents a very general view on requirements and guidelines for the complete lifecycle of a safety-related device. Requirements to communication systems are presended in detail in IEC 61784-3 (Functional safety fieldbuses).

The thesis follows the approach presented in [20] and tries to extend the KNX protocol to fulfill requirements of SIL 3 as defined by IEC 61508. To achieve this, certain measures regarding hardware and software are required. From a hardware point of view a higher level of safety can be achieved by application of redundancy approaches. Furthermore, software is required which is capable of detecting failures in hardware and the communication system. On that score, the following chapters will give discussions on how to achieve functional safety in the KNX protocol in terms of hardware requirements and involved software.

1.2 Guide through this Thesis

Chapter 2 will give an overview about automation systems and related terms and definitions. Furthermore, the target technology KNX will be described.

Chapter 3 will cover state-of-the-art standards IEC 61508 and ISO 13849 and show the main differences between them. Following IEC 61508, the achievement of specific Safety-Integrity-Levels (SIL) is of importance. SILs define requirements concerning electrical and programming-standards implying the failure rate of a safety-providing device depending on its frequency of use. In the context of this thesis, high demanding devices which allow a maximum of one hazardous failure in 10^7 hours will be of special interest (SIL3).

Existing solutions in BAS and industrial automation will be presented and compared in Chapter 4. Here, special attention is put on potential communication errors as defined by IEC 61784-3.

In Chapter 5, special aspects relevant for this thesis regarding safety will be presented in detail. This will include a discussion on possible hardware architectures, communication issues, clock synchronization, scheduling and hardware self tests.

The closing Chapter 6 will conclude gained knowledge and provides an outlook on further work.

CHAPTER 2

Building Automation Systems

Progress in technology mostly aims at making things more convenient for the user. Focusing on electronic devices, additionally energy efficiency comes into mind. That trend also affects buildings or their building automation systems. When talking about automation, mainly industrial automation comes into mind. Characterized by short reaction times, fast control loops, high precision and occasionally high dependability, an industrial automation system handles tasks where human power is not sufficient, too slow, or not possible due to dangerous environments. Building Automation Systems (BAS) are a special category of industrial automation. In contrast, timings are more relaxed due to long response times from the building. Additionally, a BAS has to take care of energy efficient house keeping and to do that in a most comfortable way.

2.1 Introduction

BAS start at small homes with just a handful of devices and end at large, public buildings like airports or office buildings automatized by some thousand devices. Especially for large buildings the advantage of BAS is clear: A BAS provides central knowledge and control about all processes involved in a building which is also known as Computer Aided Facility Management System (CAFMS). In case of an error, the operator is enabled to gain information about the error and can initiate measures to maintain the system at a very early stage. Another advantage of BAS is the ability to dynamically reconfigure the behaviour of the system. If for example a light switch should control more than the initially installed lamps, it was necessary to re-wire certain parts of the installation in traditional electrical installations. Using a BAS, simply re-binding the switch to more lamps can be done from a PC in far less time. Having knowledge of multiple sensors also enables construction of intelligent buildings. For example, opening a window will turn off the heating or ventilation. Likewise, increasing temperatures in a room will activate sun shadings and climate control. Since the properties of a comfortable room climate are different for each person, smart room controllers in combination with knowledge about who is in the room could control HVAC according to the person's preferences (smart buildings). Against all

advantages, the main disadvantage is the tremendous cost for initial installation. Additionally, operators have to be trained thoroughly.

According to [19], typically the running costs of a building over its lifetime are seven times the initial cost for construction. Considering the whole life cycle of a building, the amount of saved energy during its lifetime makes the use of a BAS economically feasible.

Another topics in BAS are security and safety. These are two completely different concepts, although described by the same word in German language ("Sicherheit").

Security describes protection of a system against malicious attacks. For instance, considering a network, insertion of a malicious message or listening to the contents sent through the network have to be detected or prevented by certain security measures. At the beginning, BAS were designed and implemented as closed systems and missing knowledge of potential intruders on how to break the BAS was protection enough. Advances in wireless technology, networked automation devices in every room in combination with open standards give motivation for development of appropriate measures to close those vulnerabilities.

Safety describes the failure free operation of a system or at least the detection of an error and transferring the system to a safe state. Safety in automation is currently just available for industrial automation solutions (with some minor exceptions). That can be divided into requirements for operator safety and requirements for process safety. For example, an emergency stop information transferred through an automation network is required to be delivered and performed within predefined deadlines. If that requirement cannot be met, the operator working on the machine could sustain injury or the machine could take damage. That means, the information has to be transmitted correctly and in time - no matter what happens, the machine has to be transferred to a safe state. Safety in HBA has been an isolated topic so far, addressing primarily fire alarm systems. Until now, safety providing systems have been mainly constructed as closed systems communicating via dedicated gateways with other systems. The only HBA solution providing functional safety found so far is an extension to LON called SafetyLON.

Automation Networks

Communication in a traditional automation system can be visualized by the three-levelarchitecture as depicted in Figure 2.1.

The *field level* is responsible for direct interaction with the physical environment and collects data from simple sensors and activates actuators. Usually that level is equipped with low-bandwidth networks. The collected data is transferred to the *automation level* which processes and passes data to the management level (*vertical communication*) or issues other devices at field level to take action (*horizontal communication*). The topmost *management level* provides a global view of all data across the BAS. Therefore, control terminals and logging systems are placed on that level. Operators are enabled to (re-)configure the BAS through a control center and perform diagnostic measures on the BAS in case of an error. Typically, the management level is equipped with a high-bandwidth network caused by high amount of data collected by the lower levels. If communication with other automation systems is required, the management level networks are connected via gateways or routers.

As described, the previous approach assumes simple sensors with small processing power to prepare raw data in a very basic way. Development in the microprocessor sector increased

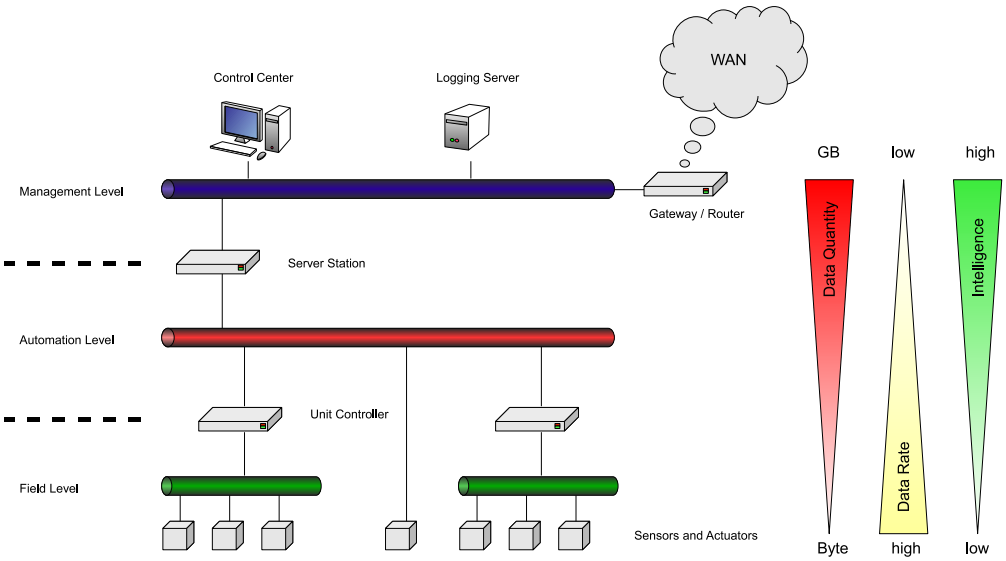

Figure 2.1: Three-tier architecture

processing power to admit advanced techniques to pre-process and transmit sensor values [18]. That simplifies the diagram in Figure 2.1 to the enhanced two-tier architecture depicted in Figure 2.2 by making use of *intelligent devices*.

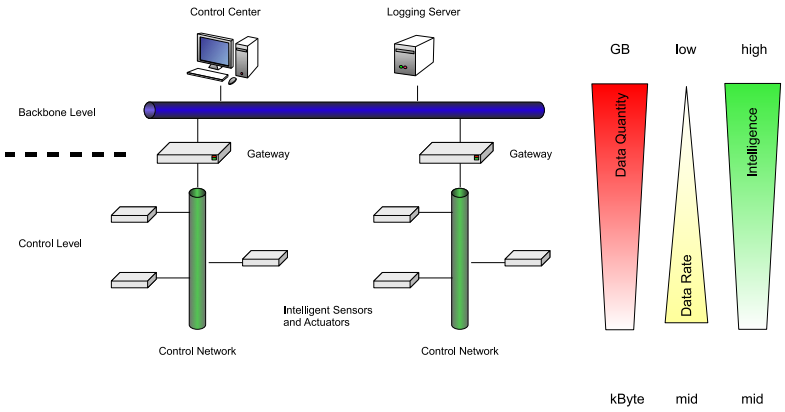

Figure 2.2: Two-tier architecture

Increased intelligence on sensor/actuator level enables integration of communication protocols for direct communication between sensors and actuators through a *control network* what makes a separate automation level obsolete. Communication between different control networks is established via gateways through *backbone networks* providing sufficient bandwidth for inter-

control network communication as well as for management- and logging tasks.

2.2 KNX

In 1996, *BatiBus Club International* (BCI), *European Installation Bus Association* (EIBA) and *European Home System Association* (EHSA) started to develop a common standard for home and building automation. In 1999, leading manufacturers of electrical building equipment such as Siemens, Bosch and Merten, along with some more, founded Konnex Association (also known as KNX Association). The first KNX specification was published in 2002 which was adopted EN 50090 in 2005 and accepted as an international standard ISO/IEC 14543-3 later in 2006.

Basically, KNX defines runtime-characteristics, a toolkit of services as well as mechanisms to manage a network. The building automation system is defined by a distributed application implemented through standardized data-point types and "functional block" objects modelling logical device channels. KNX is platform independent enabling usage of any kind of microprocessor to implement a network device.

Elements of KNX

The KNX framework consists of the following parts:

- An inter-working and (distributed) application model which performs the actual HBA application (lighting, shading, HVAC,...).
- Configuration and management schemes for logical linking or binding of KNX devices. These schemes are structured in a set of configuration modes.
- A communication system which defines communication media, a message protocol and a communication stack. The communication system has to implement required mechanisms for configuration and management and hosts the distributed application. This is typified by the KNX Common Kernel [2].
- A set of device models is summarized in profiles.

An illustration of the afore mentioned components of KNX is depicted in Figure 2.3.

Supported communication media by KNX

KNX offers a wide variety of possible communication media suited to customer's needs and devices to enable interaction between different media.

- Twisted pair is the basic medium in KNX. Main characteristics are: energy and information are transported over the same pair of wires, an asynchronous, character oriented data transfer, half duplex, bi-directional communication. TP1 (9,6 kBit/s) is the basic medium inherited from EIB and allows free choice of topology. On top of TP1 the CSMA/CA protocol is implemented.

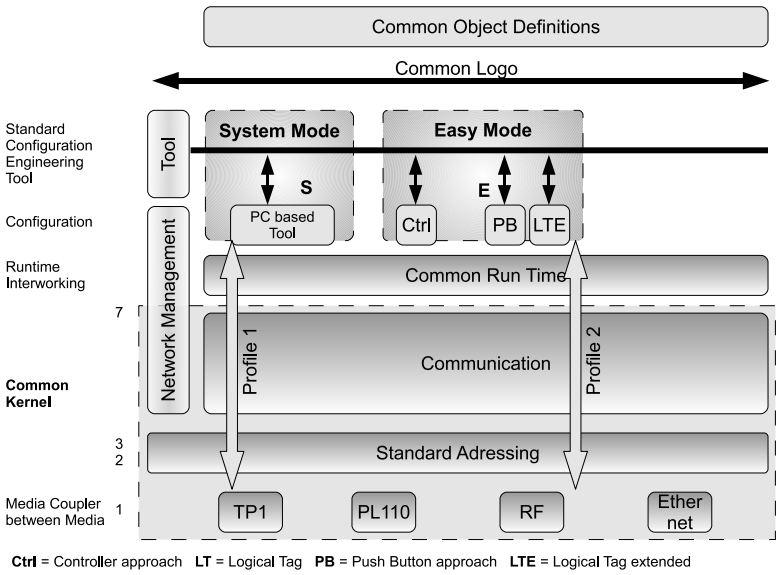

Figure 2.3: KNX model [2]

- Powerline (PL110, 1,2 kBits/s) allows data transmission over main wire. Its characteristics are a central frequency of 110 kHz, spread frequency shift keying signalling, asynchronous transmission of data packets and half duplex, bi-directional communication. PL110 implements CSMA and is EN 50065-1 compliant.
- RF is fully specified within KNX in the 868 MHz bandwidth. Characteristics are: frequency shift keying signalling, asynchronous, half duplex, bi- or unidirectional communication. The central frequency is set to 868,30 MHz using short range device frequency with a duty cycle limited to $< 1\%$ and a data rate of 32 kHz. Medium access is based on CSMA mechanisms [1].
- Furthermore, IP-enabled integration for IEEE 802.2 (LAN), 802.11 (WLAN), IEEE1394 (Firewire) is handled in KNXnet/IP.

The communication is implemented in compliance with the OSI layer model. As in most automation systems, not all seven layers are implemented. KNX uses the following four layers:

- The “Data Link Layer General” is implemented on top of the *Data Link Layer* and provides medium access control and logical link control.
- The *Network Layer* provides a segment wise acknowledge telegram and controls hop count of a frame.
- The *Transport Layer* enables communication relationships between communication points. Supported relations are 1 to N (multicast) connectionless, 1 to all (broadcast) connectionless, 1 to 1 connectionless and 1 to 1 connection-oriented.
- The *Application Layer* offers a toolkit to maintain and run the distributed application.

Topologies

As shown later, a KNX frame supports 16-bit space for individual source and destination addresses. That results in a total of 65535 possible devices on a KNX network. The network can be grouped physically into *lines* of 256 devices each. These lines can be formed by a *main line* into an *area*. A *domain* is a combination of up to 15 areas connected through a *backbone line*. Figure 2.4 gives an illustration of the resulting topology.

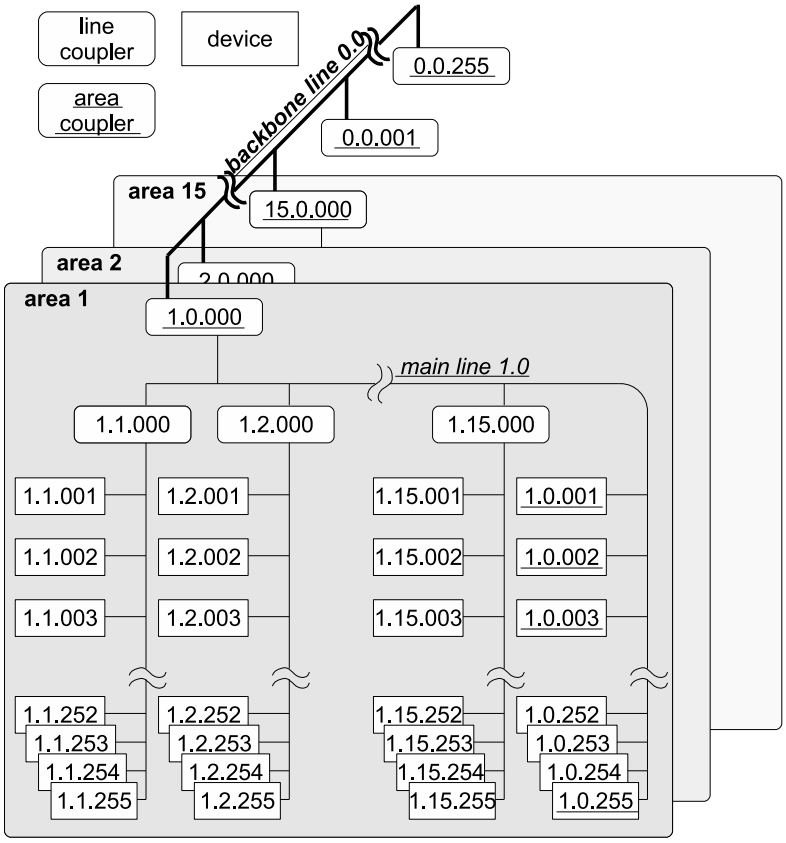

Figure 2.4: KNX topology [2]

Addressing schemes

Central functionality of a network is to enable communication between nodes. Therefore, nodes need to be identified uniquely. In most cases, an installation will be wired and configured afterwards. KNX offers device identification by a unique device serial number or by the device's individual address. Unique serial device numbering is achieved through controlled allocation of number ranges to manufacturers by KNX Association. By knowledge of a devices identification (unique serial number or individual address) it is possible to communicate with that device.

KNX distinguishes system resources keeping configuration information (address-, lookup tables and parameters) and parameters which control the application.

Communication is distinguished between network resource management and run-time communication. Configuration and management tasks usually require direct communication with the related node (point-to-point connection) or require communication with all nodes (broadcast) nodes. In contrast, run-time communication mainly uses multicast communication with other nodes interested in changed values.

In order to achieve inter-working, the data-points have to implement “Standardized Datapoint Types”, grouped into “Functional Blocks”. Communication between nodes is established after “binding” or linking data-points located on different devices to common multicast group addresses. Binding of devices happens either through loose or strict binding rules or depending on semantic information contained in the address. Upon a successful binding process the distributed application is enabled. That is, if a local application on a node writes a data-point value the change notification will be sent across the network with the corresponding address of the sending node. Any node interested in the changed value from that node will receive that value and inform its local application about the new value. The local application on the receiving node will now react depending on its internal state machine and update its own data-points. The communication between nodes transferes multiple local applications into a single, distributed application.

KNX supports the following three binding schemes: free, structured or tagged. Basically, free and structured binding assume free addressing which means that the numerical value of addresses do not contain application semantics. The only assumption is, that all data-points communicating with each other are assigned to the same address. Contrarily, tagged binding assumes the numerical value of an address to contain a semantic (data-point) identifier. Therefore, the *logical tag* or *zoning* part of the address identifies a device’s communication partners on a device level. By assigning data-points to the same zone, they form a group communicating via multicast.

To configure a KNX network, two main configuration modes are specified as depicted in Figure 2.3. Depending on the user’s preferences and application environment these modes provide functionality to configure a device remotely from ETS tool or locally using the push button approach:

- E(asy)-Mode is applied for simple manipulations where devices are configured according to a structured binding without need for separate configuration tools.
  Controller mode (Ctrl) supports installation of a limited number of devices on one logical segment of a physical medium. Such an installation will contain one dedicated node responsible for the configuration process.
  Logical Tag (LT) and Logical Tag Extended (LTE) modes basically enable device configuration using DIP-switches or selectors.
  Push Button mode (PB) is almost equal to Ctrl-mode configuration but without the need for a dedicated configuration device.
- S(ystem)-Mode enables central, free binding and configuration of the installation, typically carried out with the ETS tool.

KNX Frame

The frame of a KNX TP1 telegram is depicted in Figure 2.5. Depending on the communication medium, different preambles might be appended which will be left unconsidered here.

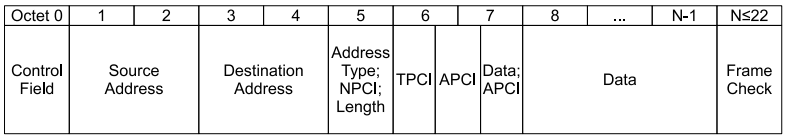

Figure 2.5: KNX LPDU TP1 standard frame structure

The control field determines the priority and distinguishes between standard and extended frame format. The individual source address determines the address of the sending node. The individual (unicast) or group (multicast) destination address determines the address of the receiving node(s). The following byte contains hop-count and address-type-information. The Transport Layer Protocol Control Information (TPCI) controls the transport layer to manage end-to-end connection. The Application Layer Protocol Control Information (APCI) accesses application layer primitives (read, write, response,...). The standard frame ensures compatibility with KNX messages (up to 14 octets of data). Extended frames can contain up to 248 octets of data. The enclosing frame check sequence ensures data consistency.

KNX line access

To access contents sent on the KNX line, special hardware in form of a transceiver is required. Therefore, Siemens provides the TP-UART-IC (Twisted Pair - Universal Asynchronous Receive Transmit - IC).

> This module supports every transmit- and receive - function and also the high ohmic decoupling of energy from bus line. It generates further a stabilized 3.3V or 5V supply to use by a host controller. Up to 256 subscribers can be connected to one bus line [25]

The TP-UART-IC consists of an analog part responsible for level converting on the KNX-line and a digital part providing serial access for communication with connected microcontrollers.

CHAPTER 3

State-of-the-art Standards

This section gives an overview of applicable standards for safety-related systems. First, ISO 13849 for a general approach regarding safety of machinery is presented. A more detailed description of safety-related development is specified by IEC 61508, a standard defining a complete lifecycle model for every development phase of an Electric/Electronic/Programmable Electronic System (E/E/PES). Here, a degree of safety is described by safety integrity levels (SIL) which are assigned depending to the probability of one hazardous failure per hour. In contrast, performance levels (PL) are defined by ISO 13849.

Prior to focusing on the standards, some important terms such as fault, error, failure, risk, hazard, dangerous failure and hazardous event are introduced:

[21] describes faults, errors and failures as a chain depicted in Figure 3.1. A *fault* is the cause of an error and, thus, the indirect cause of failure. In [6], a fault is defined as an unusual condition which leads to loss of ability to perform a desired functionality. An *error* is both, the deviation of an expected result ([6]) or an incorrect internal state, like a corrupted element in the memory ([21]), whereas a *failure* is an event that denotes the deviation between the actual and the intended service, or the loss of ability to perform a demanded functionality, respectively.

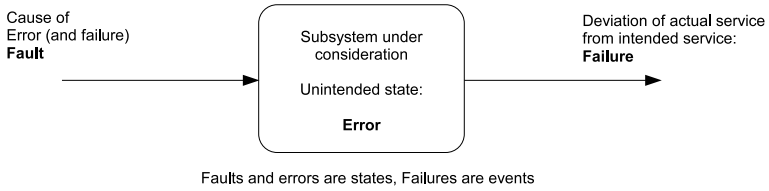

Figure 3.1: Fault chain defined by [21]

Risk is defined as the combination of probability of error and the resulting harm [6].

A *hazard* is a potential source of harm and is specified to define the source (mechanical or electrical harm) or type (fire, cut or electrical shock) of harm [6]. Thus, a *hazardous event* is a situation where a hazard leads to a harm [6].

A *dangerous failure* describes a failure that potentially leads the safety-related system to a dangerous or non-functioning state [6].

In safety-related systems, redundancy is common practice which introduced multiple-channel architectures. Such an architecture can be, for example, a *1oo2 architecture* (one out-of two) describing an approach where one output is chosen among two possible candidates (see Figure 3.2). The expression 1oo2 gives no information about the chosen criteria for either of the two input channels. It is clear, that such an architecture is optionally extendable by more inputs like a 1oo3 or 1oo4 architecture.

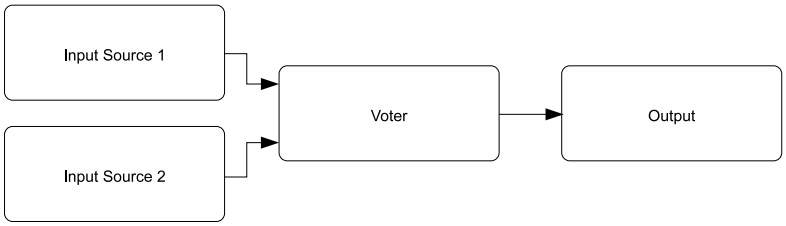

Figure 3.2: One-out-of-two architecture (1oo2)

A clear distinction has to be drawn between safety and security, although it is not always possible in every aspect. Security describes the protection of a system against malicious attacks. Contrarily, safety is defined as the ability of a system to perform its intended behaviour even in case of failure under predefined conditions.

The structure of standards in the domain of safety-related machinery as defined by ISO 12100-1 is as follows:

- Type-A-Standards cover definitions, design guidelines, and general aspects applicable to machinery.
- Type-B-Standards cover a specific safety-aspect or a type of safety equipment that is applicable for a wide range of machinery:
  - Type-B1-Standards for specific safety-aspects like safety margins and temperature levels.
  - Type-B2-Standards for safety equipment.
- Type-C-Standards cover detailed safety requirements for a specific machine or a group of machines.

In case different standards have to be applied, like a Type-A and a Type-C standard, the higher level standard (Type-C in that case) will have to be favoured. By means of that categorization, ISO 13849 is a Type-B1 standard.

3.1 ISO 13849 - Safety of machinery - Safety-related parts of control systems

This section explains some of the basic principles on how to achieve a certain level of safety as defined by ISO 13849-1 [10]. Performance levels (PL) are the base for the following development process. This standard specifies methods to fulfill the requirements for a PL through the terms diagnostic coverage, mean time to failure, common cause failure, and some more key words explained briefly in the following. Furthermore, ISO 13849 defines requirements to the lifecycle of safety-related software. The second part of the standard (ISO 13849-2 [11]) presents guidelines and techniques for the validation of the afore defined safety concept.

All parts of a machine control supplying safety functionality are called "safety-related parts of the control system" (SRP/CS). These parts may be realized in hard- or software. Additionally, such a machine may supply operational functionality. The ability of a device to provide safety-related functionality under predictable conditions is divided into five PLs as shown in Table 3.1. These PLs are defined in terms of probability of a dangerous failure per hour.

Performance Level (PL)	Average probability of a hazardous failure per hour $[1/h]$
a	$\geq 10^{-5}$ until $< 10^{-4}$
b	$\geq 3 * 10^{-6}$ until $< 10^{-5}$
c	$\geq 10^{-6}$ until $< 3 * 10^{-6}$
d	$\geq 10^{-7}$ until $< 10^{-6}$
e	$\geq 10^{-8}$ until $< 10^{-7}$

Table 3.1: Performance Levels (PL)

Probability of a dangerous failure depends on certain parameters. ISO 13849 defines the following criteria which have to be considered:

- Hard- and software structure
- Fault detection mechanisms
- Degree of diagnostic coverage (DC)
- Dependability of used devices ($MTTF_d$)
- Common cause failures (CCF)
- Behaviour at systematic failures
- Behaviour at faults
- Development process
- Load under operational conditions
- Environmental conditions

With regard to the evaluation process of PLs those aspects are grouped into quantifiable (MTTF, DC, CCF, structure) and non-quantifiable, qualitative (all others) principles. Quantifiable aspects of PLs can be estimated by usage of Markov models, generalized stochastic Petri Nets (GSPN)

or reliability block diagrams. In ISO 13849, the determination of PLs under quantifiable aspects is proposed by five different architectures fulfilling predefined characteristics in case of fault. If other architectures are used, detailed calculations on the achieved PLs need to be given. For a description of predefined architectures, please refer to [10].

To achieve a required PL, measures have to be taken to lower risk. These measures are the reduction of the probability of a fault on device level by usage of more reliable devices and by improvement of the structure of the SRP/CS to lower the effect of the fault. Depending on expectable faults, these measures can be applied separately or together, where common cause failures have to be taken into account.

Mean time to failure of a channel $MTTF_d$

Assuming a redundancy approach, a *channel* is defined to be one of the replicated paths. The value of the $MTTF_d$ of each channel is divided into three steps as depicted in Table 3.2 and shall be calculated individually for each channel.

Description for each channel	Range for each channel
low	3 years $\leq MTTF_d <$ 10 years
medium	10 years $\leq MTTF_d <$ 30 years
high	30 years $\leq MTTF_d \leq$ 100 years

Table 3.2: Mean time to failure for a channel $MTTF_d$

$MTTF_d$ for each device has to be determined by gathering information from data-sheets provided by the manufacturer or other methods defined in appendices C and D of [10]. If neither is applicable, a duration of 10 years has to be taken.

Diagnostic coverage DC

In most cases an estimation of the DC will be done by a Failure Mode and Effects Analysis (FMEA) or a similar procedure. Therefore, all relevant faults and failures have to be considered, including a calculation if the PL of the SPR/CS fulfills the required performance level PL_r. ISO 13849 defines four levels of DC as shown in Table 3.3.

Description	Range
none	$DC < 60\%$
low	$60\% \leq DC < 90\%$
medium	$90\% \leq DC < 99\%$
high	$99\% \leq DC$

Table 3.3: Diagnostic coverage (DC)

Requirements to safety-related software

The aim of the software development process is to avoid faults introduced by the software lifecycle. ISO 13849 specifies certain criteria which have to be fulfilled depending on the required performance level. Basically, a consistent documentation of the whole development process falls into these conditions. The standard proposes to use the simplified V-Model for the software lifecycle as shown in Figure 3.3.

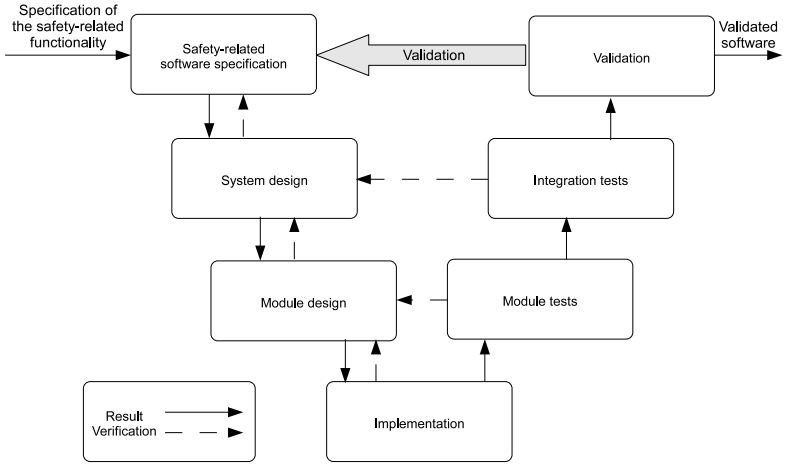

Figure 3.3: Simplified V-Model of the software lifecycle proposed by ISO 13849-1

This standard distinguishes between safety-related embedded software (SRESW) and safety-related application software (SRASW). A subset of the applicable methods for SRESW and SRASW up to the PL_r d is listed in the following:

- Software lifecycle with verification and validation
- Documentation and reasoning of the specification and the design
- Modular and structured development and implementation
- Handling of systematic failures
- Extended functional tests
- Change management including reasoning
- Quality management

Usage of SRASW is subjected to some more requirements depending on the type of programming language and the PL_r:

- Certified toolchain
- Validated libraries
- Criteria to performance (e.g. reaction times)
- Semi-formal methods to describe data and control flow
- Simulation of the implemented code
- Adequate testing

- A complete, consistent, readable, available and understandable documentation
- Verification
- Change management

For detailed requirements, please refer to points 4.6.2 and 4.6.3 in [10].

ISO 13849-1 proposes the parametrization of safety-related software as well. According to this, the inserted parameters need to be examined with respect to their validity. Further, safe data transmission from a configuration tool to the device has to be ensured and the effects of incomplete or incorrect transmitted parameters have to be known in advance. Additionally, the configuration tool needs to comply with the same requirements of SRP/CS as the configured device. Once again, for a detailed description of applicable criteria to parametrize safety-related devices, please refer to point 4.6.4. in [10].

ISO 13849-2 Validation

The standard's second part addresses validation of mechanical, pneumatical, hydraulic and electronic systems. The validation process assumes error lists containing all considered faults. These lists are processed by a predefined validation process and a validation plan. Furthermore, the whole validation process needs to be documented.

Finally, the most important part is the validation of safety-related functionality. In that step validation has to ensure correct operation of the device under different configurations and its reaction to different inputs. Additionally, where applicable, a combination of safety-related devices needs to be validated by analysis or by testing if required.

3.2 IEC 61508 - Functional safety of E/E/PE safety-related systems

IEC 61508 is the de-facto standard for anything concerning safety-related electric/electronic/programmable electronic (E/E/PE) systems. It covers every single step of the development process of safety-related systems starting from the very first concept up to the decommission of the system and provides requirements and methods in order to achieve a specified safety integrity level (SIL).

IEC 61508 is divided into seven technical parts and an additional guide part. The document structure and relation between them are shown in Figure 3.4. Part one covers basic terms, conditions and requirements for the entire safety lifecycle of the development process. The second part addresses special requirements for E/E/PE systems. In the third part, the development of safety-related software is examined in terms of lifecycle, parametrization, extension and upgrading, whereas definitions and abbreviations are defined in the fourth part. Methods for determining the achieved safety integrity level are laid down in part five. The sixth part presents guidelines for the application of parts two and three. Finally, the seventh part gives an overview of techniques and measures for the implementation and validation.

Before details regarding the development of a safety-related system are described, basic definitions of safety and functional safety need to be given. According to IEC 61508-0 [13] the definition of safety is as follows:

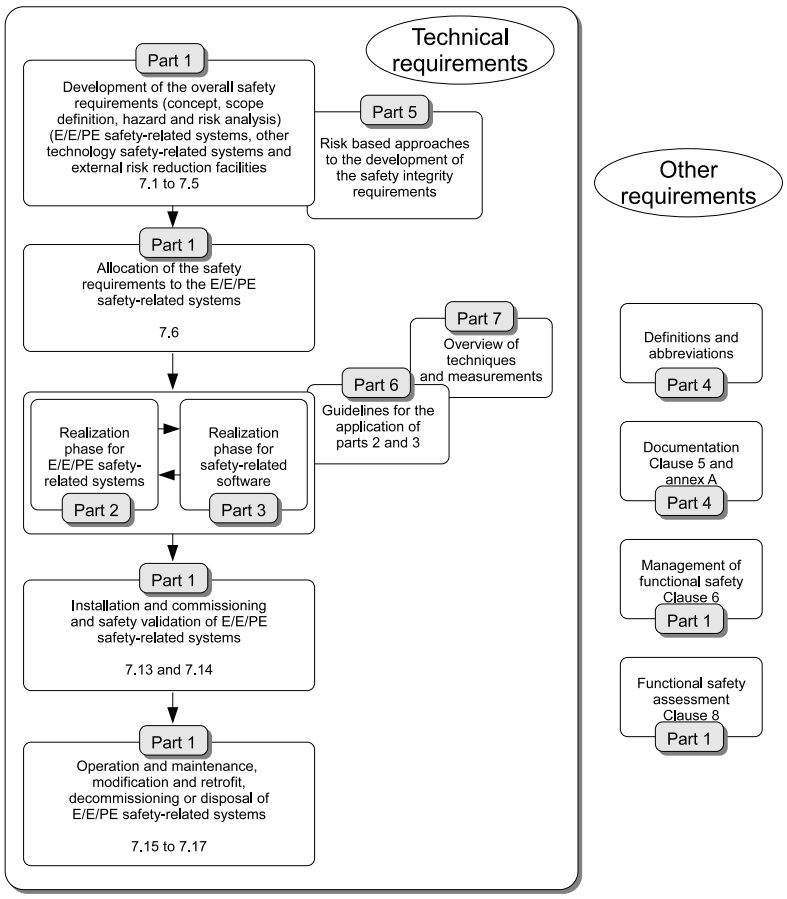

Figure 3.4: Requirements map for parts 1 to 7 of IEC 61508 [13]

> This is the freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment.

Opposite to that, functional safety is defined as:

> Functional safety is part of the overall safety that depends on a system or equipment operating directly in response to its inputs.

Both terms can only be determined by considering the system as a whole together with the environment it is interacting with. The procedure of developing a safety-related device is as follows: First, a hazard analysis needs to be performed. According to this, the necessity of functional safety is determined. If so, adequate measures need to be taken into account during design.

Functional safety means, that it is required to perform a specific function to ensure that risks are kept below a certain level. Therefore, the *safety function requirements* (what the function

does) deriving from the hazard analysis and the *safety integrity requirements* (the probability that the safety function performs as defined) which again derive from the risk assessment need to be determined. The hazard analysis points out what needs to be done to prevent hazardous failures, whereas risk assessment defines the degree of certainty that the safety function will be performed.

The entire safety lifecycle

In order to achieve the required safety integrity, the standard defines a lifecycle model (see Figure 3.5) which covers every step of the lifetime of a safety-related device starting at the first concept and ending by the decommission of the device.

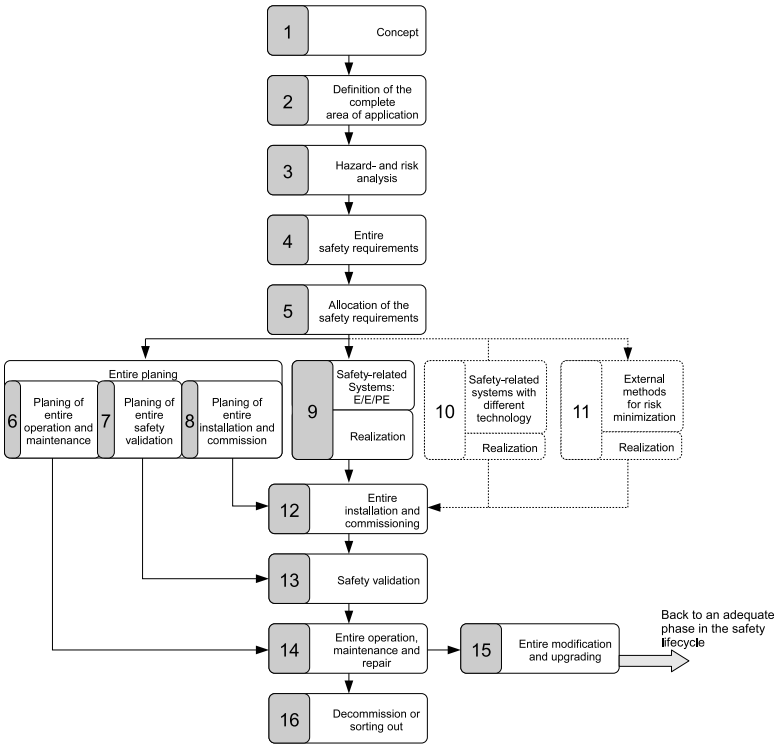

Figure 3.5: Entire safety lifecycle as defined by [3]

To achieve and keep a defined SIL during the design and throughout the further operation, each step must to be documented scrupulously. Additionally, the generated documentation has to be versioned, revisioned and approved. Further, the standard requires defined authorities for the technical and management phases of each cycle in the model, referred to as *management of functional safety*.

A brief description of the single steps of the entire safety lifecycle seems to be helpful:

The concept phase is intended to get knowledge about the equipment under control (EUC) and its environment. This is the base for the consecutive steps, for which reason all possible

sources of hazards and information about them as well as any information from applicable standards have to be pointed out.

The aim of defining the complete area of application is to show the limits of the EUC and the application area for the following hazard and risk analysis, requiring specification of physical devices, external events and subsystems.

The hazard and risk analysis point out hazards, hazardous events and sequences leading to hazardous events. Probability of a hazardous event, its impact and necessary measures to reduce the risk have to be considered. Furthermore, any assumptions during the analysis have to be stated.

The entire safety requirements target the development of safety-related E/E/PES, focussed on the safety functionality and the safety integrity. Therefore, safety functions and necessary risk reduction for every hazardous event have to be defined. Requirements for safety integrity have to be determined for every safety function.

Assignment of safety requirements is intended to map the previously defined safety functions to the safety-related systems and E/E/PES and to assign a SIL to each of these functions. In case the assignment of the safety requirements shows that the required SIL cannot be achieved, the architecture has to be changed and the assignment needs to be re-done. Requirements to safety integrity have to be adequate in order to show that the mean probability of failure or the probability of a hazardous failure per hour is satisfied. Furthermore, common cause failures (CCF) have to be taken into account, unless the single subsystems can be shown to operate independently. Independence is given if

- the subsystems are functionally different,
- they are based on different technologies,
- they do not use common parts, services or supply systems,
- they have no common operational, maintenance or test measures, or
- they are physically separated.

In case one of these requirements cannot be satisfied the subsystems cannot be considered as independent in terms of safety integrity.

Once the mapping has been done, the safety integrity levels have to be assigned according to Table 3.4.

Safety Integrity Level	Operational mode with continuous operation (Probability of a hazardous fault per hour)
4	$\geq 10^{-9}$ until $< 10^{-8}$
3	$\geq 10^{-8}$ until $< 10^{-7}$
2	$\geq 10^{-7}$ until $< 10^{-6}$
1	$\geq 10^{-6}$ until $< 10^{-5}$

Table 3.4: Safety integrity levels for devices with high performance rate [3]

For systems containing of multiple subsystems with different SILs, the whole system will have to be regarded as a system with the lowest SIL among its subsystems, unless it can be shown that sufficient independence between them is present.

The E/E/PES lifecycle model defined by IEC 61508-2

This section describes the lifecycle model for a E/E/PES as a part of the overall IEC 61508-1 lifecycle model in Figure 3.5. Therefore, the component 9 of the model is extracted into further steps as shown in Figure 3.6. The model is kept very general and can be used unchanged for hard- and software development. The sub-lifecycle is organized in six tasks which will be explained in the following.

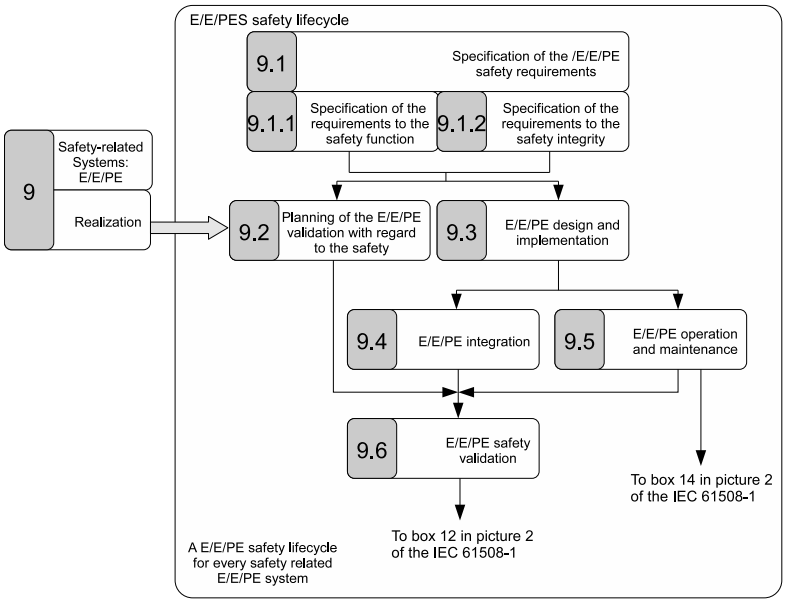

Figure 3.6: E/E/PES safety lifecycle in the realization phase defined by [4]

Specification of the E/E/PES safety requirements

The specification of the requirements to the safety functionality needs to contain the following:

- A description of the provided safety functionality
- Performance requirements like throughput and response times
- Interfaces between the E/E/PES and user interfaces
- Any safety relevant information
- Operational modes like parametrization, automatic, semi-automatic, manual, shut down, maintenance
- All kinds of failure performance, i.e. the reaction of the system in case of failure (e.g. alarm or shut-down)
- The meaning of the hardware/software interaction
- Constraints and limits of the E/E/PE subsystems
- Requirements to the commission and restart of the E/E/PES

Furthermore, the specification of the E/E/PES safety integrity requires to define the SIL for every safety function, the operational mode for every safety function, limits to the environmental conditions and limits to electromagnetic compatibility.

Planning the validation of the safety-related E/E/PES regarding safety

E/E/PES design and implementation

This step presents the most complex part in the development process of a safety related device. For better understanding, it is subdivided into several smaller items:

General requirements The main requirement is that the design needs to fulfill the specification in all points. The design of a safety-related E/E/PES including hard- and software-architecture, sensors, actuators, programmable electronics, embedded- and application software as shown in Figure 3.7 has to be accomplished in order to satisfy all of the following conditions:

- Safety integrity requirements to hardware consisting of the requirements due to the probability of dangerous hardware failures and the constraints of the safety integrity caused by hardware architecture.
- Requirements to the systematic safety integrity consisting of the certificate of approved devices and the requirements to avoidance and handling of systematic failures.
- Requirements to the system behavior when detecting a fault.

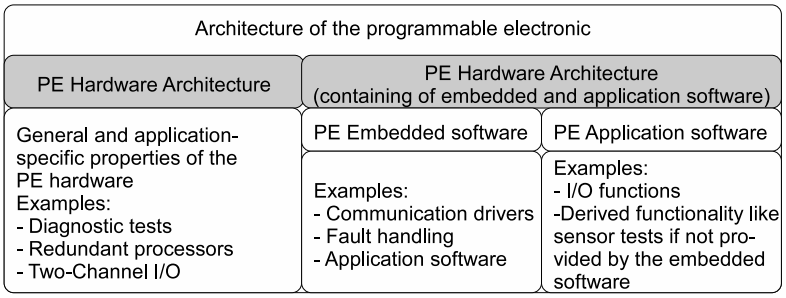

Figure 3.7: Relation between hardware and software architectures of PE [5]

In case a safety-related E/E/PES supplies safety-relevant and non-safety-relevant functionality, the complete hardware and software have to be considered safety-relevant except a proof for the independence of safety and non-safety-related parts of the system can be provided. The arising SIL that has to be satisfied is the highest among all affected devices. That means, a system requiring SIL2 has to contain systems satisfying at least SIL2. If one subsystem just fulfills SIL1, the whole system is considered to have SIL1.

If independence between safety and non-safety functionality is required the methods for achieving the separation and the reasons therefore have to be disclosed.

The developer has to ensure the adequateness of the requirements for the safety-related E/E/PES hardware and software with focus on the safety-functionality, safety-integrity requirements, electrical equipment and user interfaces.

A further step requires documentation and reasoning of the applied procedures and measures in design as well as of hardware-software interaction.

The whole system has to be partitioned into subsystems whereby each of them requires a separate design and verification process. In case a subsystem has multiple outputs it is required to show that no possible combination of states leads to a hazardous failure of the E/E/PES. If possible, all components should be dimensioned for underload.

Constraints to the hardware safety integrity due to architecture The highest achievable SIL in the context of hardware is limited through the fault tolerance of the hardware and the fraction of safe failures in the subsystems. A fault tolerance of N means that the safety functionality will get lost by N+1 faults with the constraint that fault detection mechanisms like diagnosis must not be taken into consideration. Where a fault leads to another fault, these two faults are considered to be a single fault. If certain improbable faults can be excluded from the fault tolerance calculation it has to be reasoned and documented. The fraction of non-hazardous failures (SFF) of a subsystem is defined as the mean rate of non-hazardous faults plus hazardous detected faults divided by the overall failure rate of the subsystem:

$$\text{SFF} = \frac{\text{Safe Faults} + \text{Detected Faults}}{\text{Overall Failure Rate of the Subsystem}} \tag{3.1}$$

The standard defines subsystems of types A and B. Type A is required to be completely specified by means of fault performance of the components, the subsystem itself under a fault and reliable information about process experience. Even if only one requirement is not fulfilled, a subsystem is classified as type B. Depending on the subsystem type either Table 3.5 for type A or Table 3.6 for a type B subsystem have to be taken into consideration. These tables describe the achievable SIL depending on the fault tolerance of the hardware and the fraction of nonhazardous failures. For example, a subsystem of type A with more than 99% of non-hazardous failures can reach SIL4 with a fault tolerance of 1.

Fraction of nonhazardous failures	Fault tolerance of the hardware		
	0	1	2
< 60%	SIL1	SIL2	SIL3
60% - <90%	SIL2	SIL3	SIL4
90% - <99%	SIL3	SIL4	SIL4
≥99%	SIL3	SIL4	SIL4

Table 3.5: Safety integrity of hardware: Constraints to architectures for safety-related type A subsystems [4]

Requirements for the estimation of the failure probability of a safety function due to random hardware faults The probability of loss of the safety functionality due to random hard-

Fraction of nonhazardous failures	Fault tolerance of the hardware		
	0	1	2
< 60%	not allowed	SIL1	SIL2
60% - <90%	SIL1	SIL2	SIL3
90% - <99%	SIL2	SIL3	SIL4
≥99%	SIL3	SIL4	SIL4

Table 3.6: Safety integrity of hardware: Constraints to architectures for safety-related type B subsystems [4]

ware faults has to be less than the specified failure limit and has to be estimated considering the following:

- The architecture of the safety-related subsystem related to the safety function.
- The estimated failure rate/s of each subsystem in every operational mode which leads to a dangerous failure and can/cannot be detected through diagnostic mechanisms.
- The vulnerability to common cause failures.
- The diagnostic coverage of the diagnostic tests.
- The interval of online tests to detect dangerous faults which cannot be detected by diagnostic tests.
- The probability of an undetected failure of any data transmission process.

According to these criteria the diagnostic test-interval has to be set adequately. If for any design the required limits of failure rates cannot be fulfilled, critical components or parameters need to be identified and possibilities for improvements have to be located. Afterwards the improvements have to be applied and the probability of a hardware failure has to be determined again.

Requirements to avoid failures Therefore, appropriate procedures and measures have to be developed and applied. According to the required SIL, these procedures have to be modular and transparent. Furthermore, they have to give a clear and precise description of the provided functionality, the interfaces of the subsystems, the timely order of the information and parallel operation and synchronization. Additionally, a proper documentation as well as validation and verification have to be supported.

Maintenance schemes and integration tests have to be planned during the design phase to ensure that the required SIL can be obtained. If possible, automated tools and integrated development tools should be used.

Requirements to handle systematic failures Systematic failures should already be detected in the design phase. Therefore, the testability and maintainability as well as the human abilities to operate the system have to be taken into account. Thus, the design should ensure that all remaining design errors regarding the hardware, environmental conditions, human errors, all remaining software errors and communication issues are detected.

Requirements to the system behavior at fault detection If a fault has been detected the system either has to go into a safe state and inform the operator about it or, if that is not possible, the fault has to be isolated. If the fault cannot be fixed within the MTTR a predefined action has to take place.

Requirements to E/E/PES implementation The implementation of the safety-related E/E/PES has to be in agreement with the design of the E/E/PES. Every subsystem that is used by a safety function has to be identified and described as a safety-related subsystem. To every safety-related subsystem the following information has to be provided:

- The functional specification of functions and interfaces used by the safety-related subsystem.
- The estimated failure rate/s caused by random hardware errors in every mode leading to a dangerous failure and being detected or not by diagnostic measures.
- The environmental limits of the subsystem.
- The lifetime of the subsystem.
- Maintenance requirements and intervals.
- The diagnostic coverage and test interval.
- Any required information to determine the MTTR.
- Any information to determine the fraction of safe failures.
- Fault tolerance of the hardware.
- All remaining limits applicable to the subsystem to avoid systematic failures.
- The highest SIL that can be consumed by a safety function.
- Any information regarding configuration of the subsystem.
- A confirmation about the verification of the subsystem.

Estimated failure rates for a subsystem caused by random hardware errors can be determined by a failure mode and effects analysis (FMEA) or, if available, by performance information about the subsystem under similar conditions.

Requirements to data communication In case of data communication influencing the safety functionality, the probability of an undetected fault of the communication system has to be estimated. Therefore, transmission errors, repetition, loss, insertion, wrong sequence, corruption, delay and masquerade have to be taken into account. Especially the parameters residual error rate, rate of residual information loss, bitrate and message delay have to be considered for the estimation. The topic of data communication will be discussed in detail in the Section 4.1 describing the IEC 61784-3.

E/E/PES integration

The integration tests of an E/E/PES have to ensure that all modules interact in the specified way and fulfill the intended behaviour. For the execution of the tests, appropriate procedures and measures have to be applied. Furthermore, every modification needs to be evaluated and the tests themselves must be properly documented.

E/E/PES operation and maintenance procedures

That point addresses the routinely procedures for maintenance purposes. It has to be ensured that an unsafe state does not occur during these tasks. Moreover, it requires that irregularities from the normal operation and online test results are documented. Procedures for maintenance have to be defined which are applied in case of failure including procedures for diagnosis, repair, logging and analysis of failures and revalidation. Routinely maintenance procedures have to fulfill systematic methods which have to detect non-detected failures resulting in reduction of the required safety integrity.

Validation of the E/E/PES regarding safety

Validation of the E/E/PES has to be performed according to the previously defined validation plan. Each used measurement device has to be calibrated and verified for its correct functionality. During tests every safety function has to be evaluated according to its intended behaviour and results have to be documented in an adequate way.

E/E/PES modification

If an existing E/E/PES has to be modified the following requirements have to be complied:

- An exact and complete specification of the modification.
- An analysis of the impact on the whole system.
- Approval for all modifications.
- Test-cases of the modified components including data gained by the revalidation process.
- Deviations from the normal operation.
- Required changes to the system behaviour and the documentation.

Once the system has been modified it has to be re-verified and re-validated.

E/E/PES verification

The goal of the verification is to ensure the correctness and consistency of the device with the specification. Therefore, the verification already has to be planned during the development phase of the E/E/PES. That plan has to include strategies and procedures for verification, usage of measurement devices, documentation and analysis of the gained results. For each stage of the design phase it has to be shown that the safety integrity requirements are fulfilled.

SIL 3 in detail

In the previous sections some of the basic requirements for the development of a safety-related E/E/PES have been presented. Basically, these requirements are applicable from SIL 1 to SIL 4. Part 3 of IEC 61508 [5] presents guidelines for every SIL with a special focus on the software of an E/E/PES.

As already mentioned the development of safety-related software has to be executed according to the lifecycle model in Figure 3.6. Besides, some more guidelines for the implementation

of each step are given in Appendix A and B of [5]. For the software specification and design, computer-based specification tools and semi-formal methods are recommended. Appropriate, if possible certified, programming languages, toolchains, compilers, libraries and integrated development environments should be used. As this thesis does not target a fully developed device, not all of the recommendations can be met. For instance, the usage of interrupts and pointers should be avoided although they are some of the basic concepts in microcontroller programming.

According to Table 3.6, a SIL can be achieved by increasing the SFF or the fault tolerance of hardware. For SIL3 that is to detect more than 99% of hazards with a fault tolerance of 0 or to detect 90% to 99% of hazards with a fault tolerance of 1 or to detect 60% to 90% of hazards with a fault tolerance of 2. As shown later, a high SFF can only be gained through extensive online tests and thus high diagnostic coverage. The consequence is to increase hardware fault tolerance. A discussion on different hardware architectures will be given in Section 5.1.

3.3 Conclusions of ISO 13849 and IEC 61508

So far, terms and definitions regarding safety-related systems and the two most important standards in the area of safety-related systems have been presented. Summing up, ISO 13849-1 is kept very generic in some parts of the definition of safety-related devices. There are no mechanisms or methods given on how to realize specific functionality in order to accomplish a certain performance level. Also the second part, ISO 13849-2 is kept generic to be applicable for a wide range of devices.

Contrary, IEC 61508 provides detailed information about the whole development lifecycle of safety-related devices and depicts generic requirements for concept, design, implementation, testing, validation and verification.

Both standards address the development of safety-related systems as a whole and do not provide guidelines on how to implement specific safety-functionality. A more "implementation-oriented" standard is IEC 61784, outlined in the following chapter where measures for safe transmission of data over a network will be presented.

Especially interesting for this thesis is the relation of PLs to SILs which is outlined in Table 3.7.

PL	SIL (high usage)
a	no correspondent
b	1
c	1
d	2
e	3

Table 3.7: Relation between Performance Levels (PL) and Safety Integrity Levels (SIL) as defined by [10]

PL a has no corresponding SIL level and is intended to reduce the risk of slight, usually reversible injury. IEC 61508 defines SIL 4 for possible hazardous accidents in process industry

and is not relevant for the application at machinery. Thus, the highest relevant PL is e which is assigned SIL 3.

CHAPTER 4

Existing Safety Solutions in HBA Systems

Building automation systems have initially been designed for simple applications like lighting, shading and climate control without any safety relevance. If safety was required, separate, closed systems have been installed which were interacting via gateways with the non-safe parts of the automation system.

Increasing demands regarding safety resulted in extensions of existing automation systems with safety functionality. A further requirement has been to enable coexistence of safe and non-safe nodes on the same network. Since also the existing communication media should have been reused, the solution was to implement protocols which were built on top of the existing non-safe ones. Thus, to gain safety requirements, the underlying communication channel is considered as "black channel". That means, theoretically any communication medium, wired or wireless, can be used as long as timing requirements are met which are of major importance. If no guarantees can be given whether a message has arrived or not, timeouts have to be introduced to determine loss of messages. Discussion of these issues will be presented in the following.

However, safety in home and building systems does not mean high dependability. Instead, each safe automation system is assumed to have a safe state. Thus, safety can be gained by detecting faults and transferring the system into a safe state.

In the following four automation solutions will be presented, namely Safety over EtherCAT (SoE), CANopen Safety, SafetyLON and OpenSafety. Although Safety over EtherCAT and CANopen Safety have their origins in industrial automation, they are covered too, since especially SoE becomes more interesting for building automation. However, this thesis does not focus solely on building automation systems. Instead, the communication and hardware technology used is of special interest which brings in SoE and CANopen Safety for comparison.

Preliminary to presenting existing safety solutions in home and building automation systems, a standard defining communication measures for safety-related systems will be presented - IEC 61784-3. The standard describes common communication errors and measures to detect them.

4.1 IEC61784-3 - Functional safety fieldbuses

The IEC 61784-3 standard outlines the general principles for safe message transmission in networks which are conform with afore described IEC 61508. Therefore, communication profiles for different fieldbus networks are specified in parts IEC 61784-3-x and an additional communication/protocol service is extended by a safety layer.

Before the standard will be described in depth some important terms have to be defined. Since all communication profiles defined here base on it, the *black channel principle* will be given special attention:

Black Channel Principle

As defined by [9], that principle states:

> . . . the chosen communications technology does not matter, except for a few basic constraints. . .
>
> . . . none of the error detection mechanisms of the chosen communication technology are taken into account to guarantee the integrity of the transferred process data.
>
> . . . Basically, there are no restrictions with respect to transmission rate, number of bus devices, or transmission technology as long as the parameters are tolerated by the required reaction times of a given safety application.[29]

The black channel principle gives no guarantee whether a sent message has been delivered correctly, in time, or received at all by the receiver. Message transmission is thus just a best effort approach. Any data integrity or safety measures have to be done by the safety layer.

Another term is the *safety communication layer (SCL)*. That is a separate layer in the communication stack which provides measures to ensure safe transmission of messages according to IEC 61508.

Communication Errors

To achieve a certain level of safety in message transmission, all kinds of communication errors have to be taken into account. In the following, IEC 61784-3 defined errors and possible behaviour under black channel conditions will be explained briefly:

Corruption

A message may be corrupted by errors in the communication subsystem or on a node. Such errors are common in networks and usually end up in bit errors (flipped bits).

Unintended repetition

By errors or malfunction, old and out-of-date messages are repeated at wrong time instants. Repetition by sender is common in case an acknowledgement of the receiver is absent.

Incorrect sequence

By errors or malfunction, a sink receives messages in incorrect sequence by means of wrong sequence numbers or timestamps. It is likely that such errors occur in networks with storing elements like routers or gateways where messages are delayed caused by higher prioritized messages.

Loss

By errors or malfunction, a message was not delivered or acknowledged.

Unacceptable delay

Messages are intended to be delivered within a predefined time instant. If a message is delayed due to congestion or errors on the bus, FIFOs in switches, bridges or routers, the message will be delayed.

Insertion

By errors or malfunction, a message from an unintended or unknown source was inserted. Since these messages do not have a valid source they cannot be classified as correct.

Masquerade

Maquerade is similar to insertion, except that the received message comes from a valid source. That means, a non-safety message will be accepted as safety relevant message.

Addressing

Through errors on the communication system a safety relevant message has been received by a wrong node which handles the message as correct.

Deterministic Countermeasures

So far, possible errors on the communication system have been pointed out. They have to be detected by at least one mechanism in the safety communications layer. In the following, countermeasures for deterministic errors as defined by IEC 61784-3 are presented:

Sequence number

Each message is tagged with a continuous increasing number.

Timestamp

Usually, data is only valid for an amount of time. Therefore, each message is tagged with a relative or absolute timestamp. That requires synchronization of clocks across the participating nodes.

Timing expectation

Messages are expected to be received during predefined timeslots. If a message arrives outside a timeslot, an error can be assumed. That requires synchronization, since each participant has to know the time instant of its bus access.

Connection authenticity

Each message contains a unique sender or receiver identifier describing the logical address of the safety relevant participant.

Acknowledge

The message sink replies the reception to the source. Depending on the protocol, that can be a simple acknowledge message, or the message itself to ensure correct reception of data.

Redundancy with crosschecking

Safety relevant data can be packed twice or more times into the same or different messages. On receiver side, the message contents are cross checked to their correctness.

Different data integrity assurance systems

If safety relevant and non-safety relevant messages are transmitted over the same communication system, different data integrity measures like CRC-polynomials or hash functions can be applied. Thus, non-safety messages should not be accepted as safety relevant data.

Relation between errors and safety measures

Table 4.1 shows already described errors on the communication subsystem and possible countermeasures against such errors. It is clear, that any type of error has to be detected by at least one countermeasure. Depending on the implementation of the overall system, the table is ambiguous, since for example incorrect sequence errors can be detected by sequence numbers and timestamps. Thus, not both measures are required to be implemented.

Data integrity and Data security

To ensure integrity of received data, hash functions, parity bits, CRC checks or redundant message sending have to be performed. It has to be mentioned, that the underlying communication channel must not use the same data integrity and data safety mechanisms as the implemented safety communication layer, except special measures against mix up have been met. To relate the degree of safety in the SCL with the required SIL, the residual error rate of the SCL $\Lambda_{SL}(Pe)$, which is a function of the bit error rate Pe, the residual error rate of the safety message $R_{SL}(Pe)$, the maximum number of safety messages per hour v and the maximum number of safety message sinks m, is introduced by IEC 61784-3:

Communication error	Safety Measures							
	Sequence number	Timestamp	Time expectation	Connection authentication	Feedback message	Data integrity assurance	Redundancy with cross-checking	Different data integrity assurance systems
Corruption					X$^{N1)}$	X		
Unintended repetition	X	X					X$^{N1)}$	
Incorrect sequence	X	X					X$^{N1)}$	
Loss	X				X$^{N1)}$		X$^{N1)}$	
Unacceptable delay		X	X$^{c)}$					
Insertion	X			X$^{a)b)}$	X$^{a)}$		X$^{N1)}$	
Masquerade				X$^{a)}$	X$^{a)}$			X
Addressing				X				

a) Application dependent
b) Shows only insertion of an invalid source
c) In any case required
d) Just in case that the residual error rate Λ_{SL} can be shown to meet specified requirements
N1) Under certain conditions

Table 4.1: Communication errors and detection measures by [9]

$$\Lambda_{SL}(Pe) = R_{SL}(Pe) * v * m \tag{4.1}$$

The residual error rate also depends on the maximum number of safety messages per hour which implies bounded reaction times of safety functionality. Regardless of the operational mode (continuous, or low performance, see Table 3.4), relations between the residual error rate of the functional safe communication system and applicable SILs are depicted in Table 4.2.

SIL	Probability of a hazardous failure per hour of the functional safe communication system	Max. allowed residual error rate of the functional safe communication system
4	$< 10^{-10}/h$	$\Lambda < 10^{-10}/h$
3	$< 10^{-9}/h$	$\Lambda < 10^{-9}/h$
2	$< 10^{-8}/h$	$\Lambda < 10^{-8}/h$
1	$< 10^{-7}/h$	$\Lambda < 10^{-7}/h$

Table 4.2: Relation between residual error rate and safety integrity level

Thus, to fulfill SIL 3 the residual error rate per hour has to be less than 10^{-9}.

Additionally to safe transmission of data, security has to be considered as well. According to IEC 61784-3 Point 5.7, security measures have to be implemented in the black channel. Further information will be provided in the upcoming IEC 62443.

The remaining information provided by the standard addresses different communication profiles for fieldbus systems such as Profibus, CIP, EtherCAT and much more which will be explained partially in the following. For further information, refer to [9].

4.2 Industrial Automation solutions

CANopen Safety

This automation solution builds on the well known CAN bus which was originally developed for in-vehicle networks. There are several standardized protocols that make use of the CAN data link protocol which are, for example, CANopen for embedded control systems[1], DeviceNet for factory automation, J1939 based solutions (J1939-71 Isobus, ISO 11992) for trucks and other vehicles and, ISO 15765 for passenger car diagnostics.

CANopen can be extended to be safe by either applying the safety-relevant communication protocol defined in CiA 304 [17] or by using the CANopen safety chip 02 (CSC02) which has been certified by TÜV according to SIL3. Application of CiA 304 describes safety as a property of a device: A device uses all features defined by a communication profile and additionally special safety communication objects. All other, non-safe communication objects remain unchanged. A CSC02 chip contains a complete implementation of standard CiA 301 CANopen application layer [16] and CiA 304 CANopen safety protocol on top of two CAN modules on-chip.

CiA 304 also defines required hardware architecture for SIL3 compliance (see Figure 4.1).

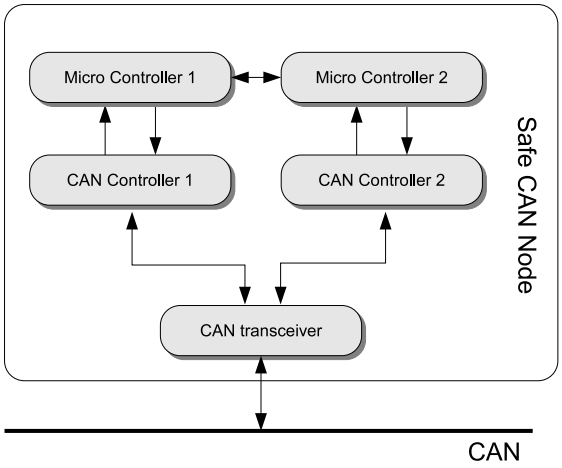

Figure 4.1: C-model for safety-relevant communication networks [17]

CANopen Safety distinguishes sources of safe information (safety switches, light barriers, emergency stops) and consumers of information (relay, valve drives, safety PLCs). Since

[1] http://www.can-cia.org/index.php?id=4

CANopen Safety does not provide request-response communication pattern, it is left up to the data consumers to check data integrity and transfer to safe state in case of incorrect data. The number of information sources (safe inputs) is limited to 64, whereas an unlimited number of information consumers (safe outputs) may listen to the produced safety-relevant data objects (SRDOs).

To increase data safety, an SRDO consists of two standard CAN data frames, where the second data frame transfers the same data as the first one, but in a different bit-ordering, like reverse ordering (Redundancy with cross-checking). Such consecutive CAN frames from the same SRDO have to arrive within the safety-relevant object validation time (SRVT). An example is given in Figure 4.2. Additionally, that mechanisms check whether sufficient network capacity is available. If the second frame is delivered after the SRVT expired, the safety controller shall go into safe state. Likewise, if one of the frames does not satisfy data integrity or data contents of the two frames do not match, the safety controller also has to switch to safe state.

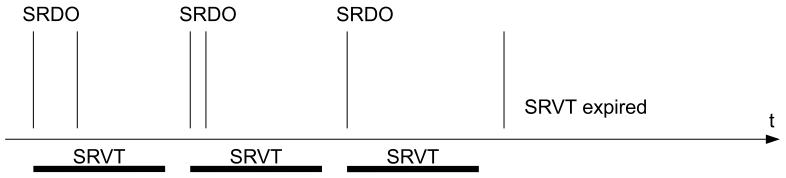

Figure 4.2: Example for SRVT timing [17]

Further, SRDOs are transmitted periodically. The interval between consecutive SRDOs is referred to as Safeguard Cycle Time (SCT). If a message is delivered after the SCT expired, the safety controller shall go into a safe state. Figure 4.3 illustrates an example. Timing expectations require synchronization among safe nodes. Unlike most other safety extensions, CANopen provides synchronization by default.

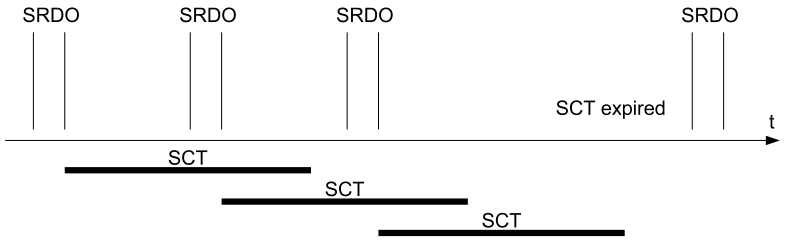

Figure 4.3: Example for SCT timing [17]

CiA 304 also gives a mathematical analysis of the protocol performance: Assuming a maximum of 64 safety relevant devices and an error rate of less than 10^{-9} for SIL3 compliance, 44 SRDOs per second are possible. The calculation is as follows:

The worst case residual error probability of CAN is defined by [8]:

$$P_{Rest} = 7 * 10^{-9} \approx 1 * 10^{-8} \tag{4.2}$$

According to model C defined in [12], sending the safety relevant data twice, the residual error probability results in:

$$P = P_{Rest}^2 \tag{4.3}$$

Applying Equation 4.1 for SIL 3 and assuming the network to consist of 64 devices results in afore mentioned 44 SRDOs per second and, thus, a refresh time of 23ms [17].

A comparison to Table 4.1 is depicted in Table 4.3. It has to be mentioned, that the CANopen specification does not provide all information required to give a complete comparison. Measures depicted here are thus just based on assumptions.

Communicationerror	Safety Measures							
	Sequence number	Timestamp	Time expectation	Connection authentication	Feedback message	Data integrity assurance	Redundancy with cross-checking	Different data integrity assurance systems
Corruption							X	
Unintended repetition		X						
Incorrect sequence		X						
Loss			X					
Unacceptable delay		X	X					
Insertion			X					
Masquerade				X				
Addressing				X				
NOTE: CANopen Safety specification CiA304 forbids usage of safe CAN IDs on non-safe nodes in networks consisting of safe and non-safe nodes.								

Table 4.3: Communication errors and detection measures used by CANopen Safety

Safety over EtherCAT

Safety-over-EtherCAT is defined as communication profile family 12 of IEC 61784-3 [14] and certified for SIL 3 compliance. Like most other industrial safety providing systems, also Safety-over-EtherCAT builds on the black channel principle, and provides safety and non-safety functionality on the same bus.

As depicted in Figure 4.4 Safety-over-EtherCAT uses unique master-slave relationships between FSoE Master (Failsafe over CPF 12) and FSoE Slave called FSoE Connections. Such a FSoE Connection is always established between exactly one FSoE Master and one FSoE Slave.

To ensure data integrity of the safety message transmission, [14] points out the following measures:

- Session-numbers for detecting buffering of a complete startup sequence

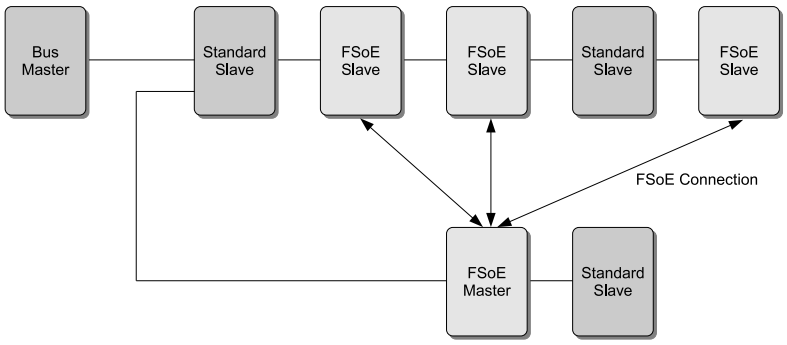

Figure 4.4: Basic FSCP 12/1-System [14]

- Sequence numbers for detecting interchange, repetition, insertion or loss of whole messages
- Unique connection identification for safely detecting miss-routed messages via a unique address relationship
- Watchdog monitoring for safely detecting delays not allowed on the communication path
- Cyclic redundancy checking for data integrity to detecting message corruption from source to sink.

The Safety PDU is embedded into standard Type 12 PDUs as depicted in Figure 4.5. As illustrated, the safety-relevant data is transferred in 2-byte blocks with a separate checksum. The checksum is calculated over the command, two byte safety data, the connection ID, a virtual sequence number, the CRC_0 of the last received safety PDU and three additional zero octets with the CRC polynomial 0x139B7. If only one octet of safety data is transferred, SafeData[1] is skipped in the calculation. The virtual sequence number is a 16-bit value which is separately incremented by the master and the slave each time a new safety PDU is created. Once the sequence number is 65535 it will start again with 1 (0 is left out). In case of faulty checksums, both, the FSoE master and the FSoE slave will switch to a defined safe state. For detailed state diagrams of the Safety-over-EtherCAT nodes please see [14].

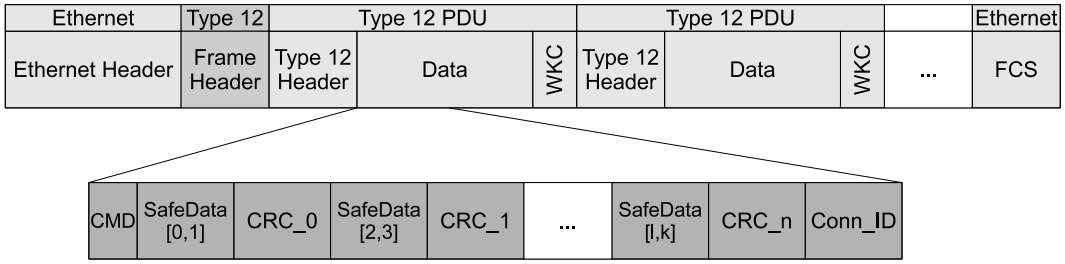

Figure 4.5: Safety PDU for CPF 12 embedded in Type 12 PDU [14]

The communication is organized in FSoE Cycles. The FSoE Master sends a safety master PDU, called SafeOutput, to one of the FSoE Slaves and starts the FSoE Watchdog. The FSoE Slave then handles the data and returns its Safety Slave PDU, also called SafeInput, and starts its own FSoE Watchdog. Once the FSoE Master receives a Safety Slave PDU, it stops the FSoE Watchdog and the FSoE Cycle is finished. An example execution is depicted in Figure 4.6. On expiration of either of these watchdogs, the corresponding node will enter its safe state.

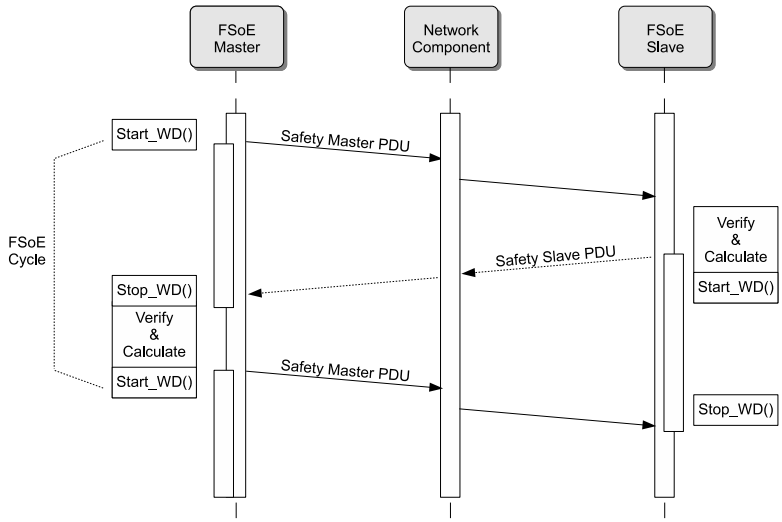

Figure 4.6: FSoE Cycle [14]

Compared to Table 4.1, SoE uses different measures to detect errors on the communication subsystem as depicted in Table 4.4. Note, that in difference to most other solutions presented here, SoE distinguishes cyclic and acyclic measures for error detection.

4.3 OpenSafety

OpenSafety is a new, hardware independent software implementation of a safety stack for automation systems. OpenSafety implements the application layer (layer 7) of the standardized OSI model and considers all lower layers to be a black channel. Thus, it is applicable for practically any underlying hardware architecture. For Sercos III, EtherNet/IP, Modbus-TCP and POWERLINK, OpenSafety is certified for SIL3.

Causes of fault are mostly identified to be routing errors on gateways[2]: One and the same message might be transmitted to the same destination network over two or more gateways resulting in duplicate messages. In contrast, messages might be lost at gateways by not forwarding it at all or forwarding it to wrong destination nets. Long messages sent in packets might be forwarded erroneously, incompletely or delayed at gateways, resulting in corrupt messages. Another source

[2]http://www.open-safety.org/index.php?id=21L=hplzymywy

Communicationerror	Safety Measures							
	Sequence number	Timestamp	Time expectation	Connection authentication	Feedback message	Data integrity assurance	Redundancy with cross-checking	Different data integrity assurance systems
Corruption					Xa	Xa Xc	Xa Xc	
Unintended repetition	Xa	Xc						
Incorrect sequence	Xa	Xc						
Loss	Xa		Xc[a)]		Xa			
Unacceptable delay		Xc	Xc					
Insertion	Xa		Xc[b)]		Xa			
Masquerade			Xc			Xa Xc	Xa Xc	Xa Xc
Addressing				Xa Xc				
a) Missing PDUs shall be detected within maximum reaction time. b) Only one message shall be accepted during a defined time frame. Xc) Errors detected by cyclic measures Xa) Errors detected by A-cyclic measures								

Table 4.4: Communication errors and detection measures used by Safety over EtherCAT

of error is identified to message corruption caused by electro-magnetical interference resulting in flipping bits. Finally, as safe and non-safe nodes may coexist on the same network, non-safe messages might be erroneously identified as safe messages (masquerading or message mix-up).

To detect these identified errors, OpenSafety introduces timestamps as a basic concept. Each sent message is tagged with a timestamp resulting in detection of duplication, delay and mix-up. Each safe receiver is required to reply to a message reception to the sender to indicate that the data link remains established. Additionally, time monitoring detects delayed and lost messages. The latter two mechanisms are referred to as watchdog and are part of the OpenSafety software stack. Message corruption is avoided by tagging each message with a unique 8 or 16 bit identification tag which encodes parts of the messages' address field, telegram type and frame type. Furthermore, CRCs are calculated over every frame and attached to it including the key the calculation was done with. Upon reception, the receiver will recalculate the CRC of the message with the attached key. In case of different checksums, the message will be dropped. As final measurement, each frame is packed twice into one OpenSafety frame as illustrated in Figure 4.7. This increases the probability to detect corrupt messages, since an error would have to occur in both frames at the exactly same position. Also, masquerading is now very unlikely to occur.

Unfortunately, OpenSafety does not provide more detailed information about the exact time-synchronization protocol implemented, but according to the EPSG website (see[3]), a resolution in microsecond range is achievable.

An implementation of OpenSafety on top of the POWERLINK protocol is available under BSD-license at IXXAT[4].

[3]http://www.ethernet-powerlink.org/index.php?id=41

[4]http://www.ixxat.de/ethernet_powerlink_safety_intro_de.html

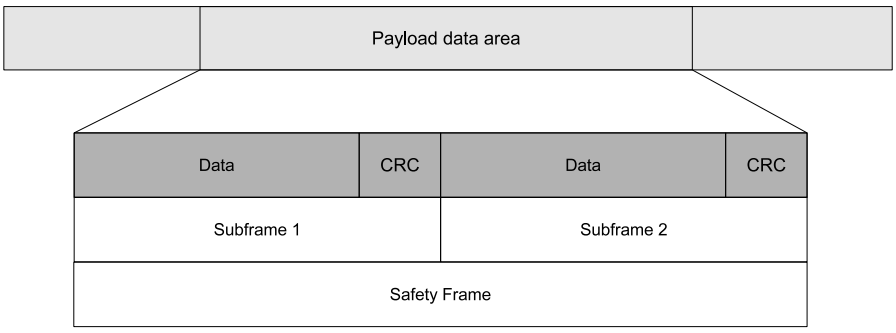

Figure 4.7: OpenSafety safety frame structure

Since OpenSafety just provides a software stack, it is left up to the hardware designer which architecture to use. But to be SIL3 compliant, an architecture similar to the one presented in Figure 4.8 has to be applied.

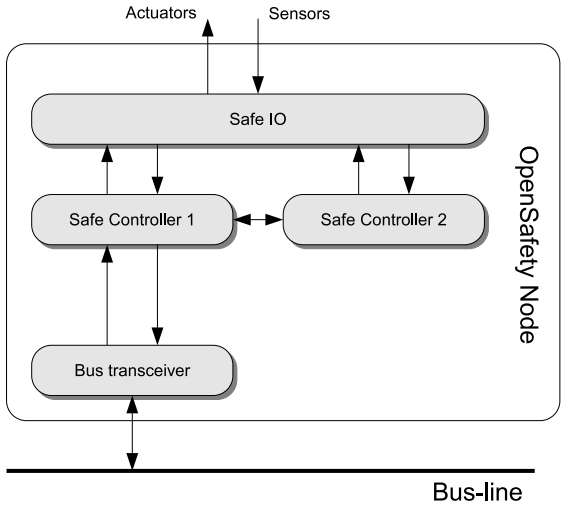

Figure 4.8: Possible hardware architecture for an OpenSafety-Node

An OpenSafety network can handle up to 1023 domains, where each domain may include up to 1023 nodes, whereat safe nodes in a safety domain do not have to operate within the same network. Communication between safety domains is handled through gateways. Since OpenSafety builds on the black channel principle, safe and non-safe nodes are allowed to operate in the same network. Each domain has to contain a Safety Configuration Manager (SCM) to monitor safe nodes. That SCM cyclically sends lifeguard signals to test safe nodes in its domain. If a lifeguard signal is absent, the safe node shall go into safe state.

Compared to safety measures defined by IEC 61784-3 in Table 4.1, OpenSafety uses mea-

sures depicted in Table 4.5.

Faults	Preventive/Corrective measures					
	Time Stamp	Time Monitoring	Identifier	CRC protection	Redundancy with cross-checks	Distinct frame structure
Duplication	X					
Loss		X				
Insertion			X			
Incorrect Sequence	X					
Delay	X	X				
Distortion				X	X	
Mix-up of standard and safety frames						X

Table 4.5: Communication errors and detection measures by OpenSafety

Summing up, OpenSafety is a good option to extend any bus-system with safety functionality since an available software stack is already SIL3 certified. The provided documentation included in the downloadable software stack is a good entry point on how to integrate OpenSafety within an existing implementation. For SIL3 certification, the OpenSafety homepage[5] recommends to contact the EPSG (Ethernet PowerLink Safety Group) for help.

4.4 SafetyLON

SafetyLON is an extension to the building automation system LON developed in the 1990s by Echelon. Since 2008 LON is approved ISO and IEC standard and documented in EN 14908 series. Due to its open and inter-operable specification it is widely used, especially in public buildings. LON nodes base on Neuron Chips, including MAC-, network- and application-CPU. The MAC CPU handles physical access to medium, the network CPU de- and encodes messages to proper format and the application CPU implements the user program. Each neuron chip is identified by a worldwide, unique 48 bit Neuron ID, assigned by the chip manufacturer. The Neuron ID is used for identifying chips in the fieldbus network. The communication between LON nodes is performed according to the LonTalk protocol which is applicable to a wide range of communication media.

Just like other building automation protocols, LON does not provide safety by default. In course of the SafetyLON project, LON has been extended to fulfill safety requirements up to SIL 3. Therefore, the hardware architecture has been extended as similar depicted in Figure 4.8. To access LON networks, the bus transceiver will be replaced by a Neuron chip. Since safe and non-safe nodes shall coexist among the same network, the LonTalk protocol must not be changed which results in application of the black channel principle. Thus, safety relevant data is packed into payload data area of standard LonTalk messages.

[5] http://www.open-safety.com/index.php?id=25&L=wqdrevmxcexvyln

The format of the safe message is depicted in Figure 4.9. To ensure high amount of data integrity and low risk of corruption, the data part of the safe message is sent twice. The message starts with an ID field keeping information about message type followed by a three byte safe address field. The timestamp is divided in two parts which results in a 4 byte timestamp consisting of MSWord (Most Significant Word) for the higher two bytes and LSWord (Least Significant Word) for the lower. SafetyLON uses timestamps for detection of delay, repetition, wrong sequence and in conjunction with safe address, insertion [23]. Finally, two different CRC polynomials are applied and results placed in field CRC 1 and 2. Depending on the message length, either one or two bytes of CRC sums are appended.

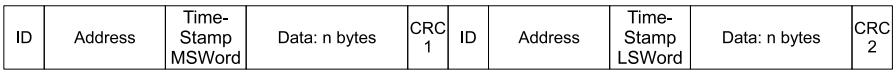

Figure 4.9: SafetyLON protocol Extension

Communication among nodes is performed according to producer-consumer model: Each producer and consumer is assigned a safe address. Additionally, consumers keep a list of safe addresses of producers, from which they are allowed to receive safe messages. When sending a safe message, the producer attaches its own safe address to the message. Upon receiving a safe message, the receiver will only do further processing, if the safe address in the message is contained in its list of valid producers.

Additionally, producers cyclically send heartbeat messages. Consumers check timing intervals between heartbeats and in case of expired timeout, the consumer will enter a defined safe state.

In comparison to Table 4.1, applied measures in SafetyLON are depicted in Table 4.6. It has to be mentioned, that connection authentication is implemented by means of a safe addressing model.

Unfortunately, information about application of SafetyLON can hardly be found. For the time this thesis was written, it was not even possible to examine if SafetyLON was used at all.

Communication error	Safety Measures							
	Sequence number	Timestamp	Time expectation	Connection authentication Safe Addresses	Feedback message	Data integrity assurance	Redundancy with cross-checking	Different data integrity assurance systems
Corruption						X		
Unintended repetition		X						
Incorrect sequence		X						
Loss			X					
Unacceptable delay		X	X					
Insertion		X[a)]		X[a)]				
Masquerade				X				
Addressing				X				
NOTE: Connection authentication is implemented by usage of a safe addressing model. Messages are just processed if the source address in a delivered message is in the list of known source addresses. a) In conjunction of timestamps with safe addresses.								

Table 4.6: Communication errors and detection measures used by Safety LON

CHAPTER 5

KNX Safety

The development of a safety-related device requires to consider every aspect of a device starting from a sensor up to the actuator. The chain is visualized in Figure 5.1. For the further safety considerations, only the red marked units will be taken into account.

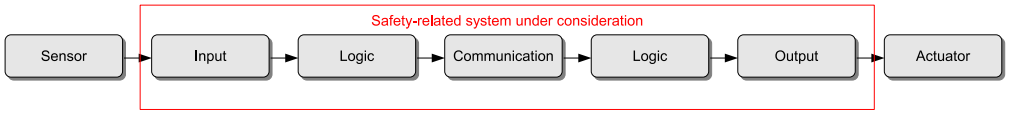

Figure 5.1: System chain - From the sensor to the actuator

First, all possible hazards have to be determined in a hazard and risk analysis. For fieldbus systems, [24] has identified them to:

- Crosstalk due to a coupling fault
- Broken cabling
- Wrong wiring
- Stochastic failures
- Extensive bandwidth allocation
- Transmission of unauthorized messages

These hazards can cause further hazardous events which are:

- Data corruption
- Loss of messages
- Insertion of messages
- Delay, repetition, wrong sequence of messages
- Masquerade: Unsafe messages look like safe messages

Hence, hazards can occur on every component of the microcontroller such as CPU, memory, inputs and outputs. Since it is assumed that not all used components are fully specified with regard to their safety properties, Table 3.6 will be applied. As already stated, a SIL can be achieved by increasing the SFF or fault tolerance of hardware. For SIL 3 that is, according to Table 3.6, to detect more than 99% of hazards with a fault tolerance of 0 or, to detect 90% to 99% of hazards with a fault tolerance of 1 or to detect 60% to 90% of hazards with a fault tolerance of 2. As shown later, a high SFF can only be gained through extensive online tests and thus high diagnostic coverage. The consequence is to increase hardware fault tolerance. Hardware fault tolerance of 1 can be achieved with a 1oo2 architecture as presented in [7].

Up to now, hazards on a microprocessor level have been considered. Failures in the communication subsystem have been discussed in Section 4.1. Safe in- and outputs will be covered in Section 5.9.

In the following sections requirements to extend KNX with functional safety will be presented. This will start with a discussion on possible hardware architectures. After a selection of the best suited hardware architecture, consequences for software will be discussed.

5.1 Hardware Architectures for Safe KNX Nodes

This section discusses possible architectures for safe nodes. We will start with a simple one channel architecture and, by replicating the safe controllers and bus access hardware, end with a triple modular redundancy (TMR) approach. Since most automation systems are assumed to have a safe state, it is sufficient to detect errors and switch to the safe state. This measure lowers requirements to the whole system enormously since complexity of hardware and software of fail-safe and fail-operational systems is magnitudes higher.

Most of the architectures in existing solutions make use of the black channel principle as already described in Section 4.1 which will be assumed here, too. A further requirement for the choice of hardware is reuse of existing wiring-scheme of KNX networks. Thus, full redundant approaches using replicated bus wiring could be left unconsidered.

One Channel Architecture

The one channel architecture is the most simple architecture with just a single controller. The implementation of a SIL 3 compliant device requires a certain degree of safe failures as already presented in previous sections. Since that kind of hardware architecture has a fault tolerance of 0 it is necessary to have a safe failure fraction (SFF) of more than 99% to be SIL 3 compliant (see Table 3.6). This can be achieved by extensive online self tests resulting in a high diagnostic coverage, but it is very resource intensive. To lower the required safe failure fraction it is necessary to increase the fault tolerance of the hardware. Such an approach is presented in the following.

Replicated Safe Controllers on a single TP-UART Chip

The first presented architecture (see Figure 5.3) is a typical master-slave model. The safe controller 1 (SC1) receives messages from the bus and forwards them to the safe controller 2 (SC2).

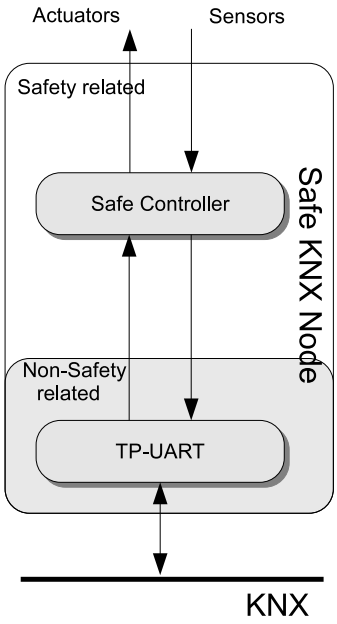

Figure 5.2: One channel architecture

On top of the safe controllers a safe I/O unit decides on the in- and outputs. Referring to Figure 5.14, only the SC1 has to run the complete stack (KNX and TP-UART) whereas the SC2 will only run the KNX Safety stack. This architecture has a fault tolerance of 1 and thus a SFF between 90% and <99% is required. It has to be considered here that in case the SC1 fails, SC2 may still work but has no possibility to continue the further operation since it gets no more messages and cannot send messages.

Thus, the consequence is an architecture shown in Figure 5.4 where SC2 also has bus access. The bus access for SC2 is intended to operate in a cold standby mode which means that it will only be used if SC1 fails. Then SC2 takes over control and performs the communication. The single point of failure SC1 has now moved downwards to the TP-UART which is in the black channel and thus not relevant for the further safety considerations. The problem that arises here is that in case SC1 fails SC2 has to continue the operation where SC1 ended. Thus, the safe controllers have to be synchronized. Additionally, the communication lines between the TP-UART chip and SC1 will have to be physically disconnected since SC1 may fail with a stuck-at error on the bus lines and thus SC2 cannot communicate either. The problem is that everything in the black channel is out of the controlled area of the safe controllers which affects the TP-UART-safe controller connection, too.

Replicated Safe Nodes

A further consequence is to duplicate the TP-UART-Safe Controller line as depicted in Figure 5.5. That approach looks very similar to the afore presented architectures but is very different

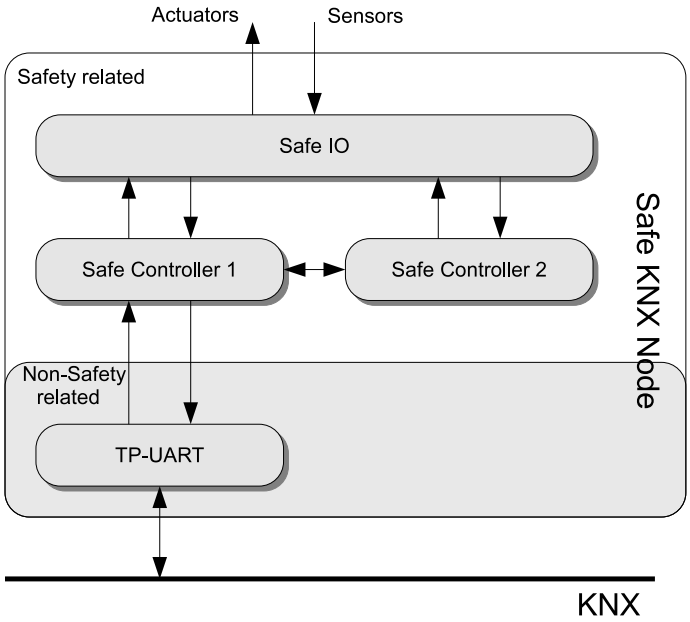

Figure 5.3: Replicated Safe Controllers on a single bus-coupler

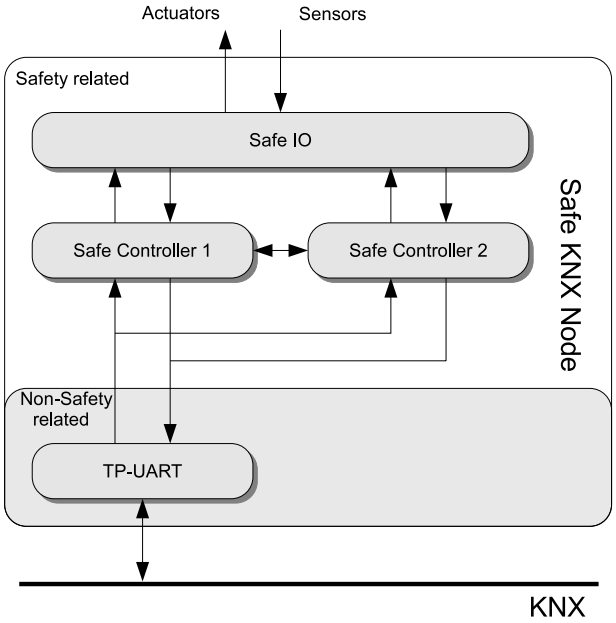

Figure 5.4: Replicated Safe Controllers on a single bus-coupler - Alternative

in terms of the software architecture. The first presented architectures work as master-slave models. SC1 gets messages and forwards them to SC2. So SC2 is just reacting to SC1s actions unlike the second architecture where both SCs have bus access. As soon as both SCs have bus access it is necessary to guarantee determinism among the safe controllers which is also referred to as replica determinism in [21]. This means that the safe controllers have to be synchronized and a protocol which assures that only one of the two SCs actually sends a message has to be implemented. On the other hand, the architecture enables to control whether a message that has to be sent has been sent correctly by immediately reading it while writing. However, the architecture requires certain techniques to synchronize the internal states of the SCs which can be very challenging as pointed out in [21].

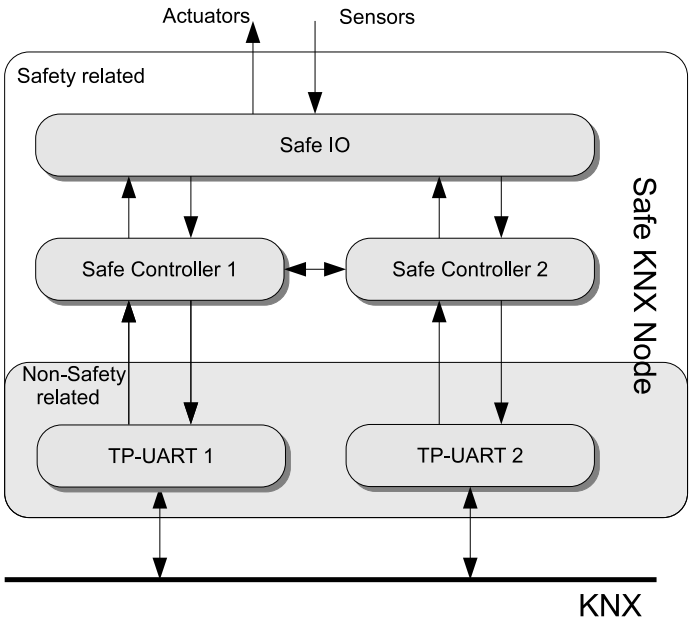

Figure 5.5: Replicated Safe Controllers with replicated bus-couplers

Redundant Nodes on a redundant Bus

For completeness of the architecture discussion redundancy approaches will be presented too. Since these concepts base on duplicated wiring of the fieldbus network they can be disregarded. If the bus lines are wired redundantly the probability of broken wiring will be lowered. In any case, measures have to be implemented to detect broken wiring. The architecture depicted in Figure 5.6 is equal to the architecture depicted in Figure 5.5 from a node-level point of view. Both have a hardware fault tolerance of 1 which requires a safe failure fraction of 90% to <99% for SIL3. Triple modular redundancy (TMR) is achieved by adding a third TP-UART-Safe Controller line to the safe node (See Figure 5.7). That would be the most safe architecture so far with a hardware fault tolerance of 2. According to Table 3.6, the safe failure fraction goes down

to 60% to <90%. The extension brings advantages in the safe I/O unit since now a simple 2oo3 voter can be implemented. On the other hand the hardware cost is significantly higher and for a fieldbus system inapplicable.

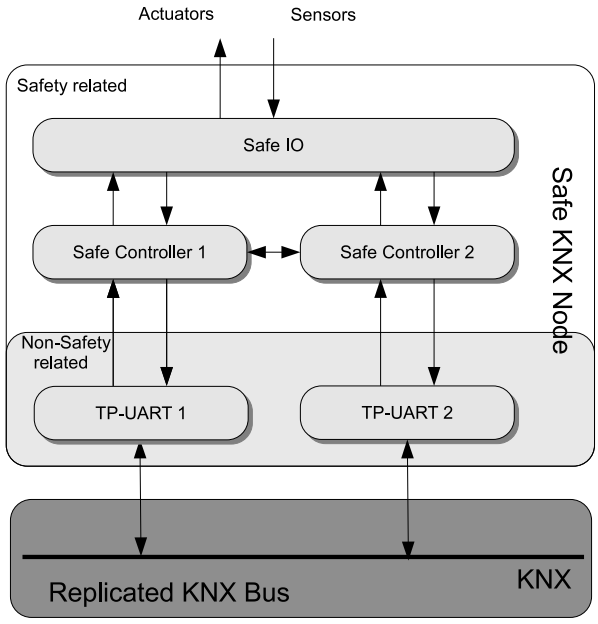

Figure 5.6: Redundant Safe Nodes on a redundant Bus

Conclusion

So far all relevant architectures have been presented with their advantages and disadvantages. The one channel architecture is not useful due to high computation complexity for the required online-tests and the redundancy approaches due to the bus wiring. Thus, the remaining architectures are the replicated SCs with single and replicated bus access depicted in Figures 5.3, 5.4 and 5.5. Since the black channel has to be left unchanged the remaining architecture is the one depicted in Figure 5.3. Additionally, the architecture is advantageous since no synchronization between the single SCs is required. It leaves the black channel completely unchanged and requires no further knowledge about mechanisms working in the black channel.

Depending on the chosen hardware architecture several requirements arise for the software architecture. As the black channel gives no guarantees on the completeness, correctness or timeliness of a sent message, just to name a few, these controls have to be covered by the software. Therefore, the following sections will present approaches to overcome these issues.

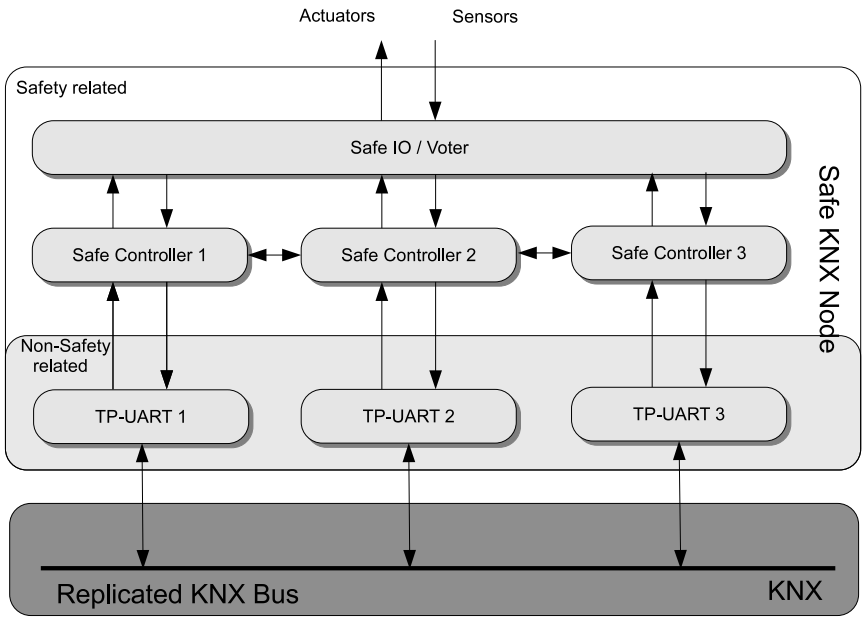

Figure 5.7: Triple modular redundancy - TMR

5.2 Synchronizing Safety Nodes

This section will explain how to gain a global timebase in a distributed system. Therefore, the different concepts of time will be described in detail. Further terms like accuracy, precision and clock drift as defined by [21] will be explained briefly. Two different algorithms, namely Vector clocks and the Precision Time Protocol, will be presented for synchronizing clocks in a distributed system. The closing part will describe the used algorithm in the KNX Safety project.

Basically, a clock is a counter which is increased by the progression of time. Clocks may vary in certain parameters. We call a tick of the reference clock a *microtick* and the time between two microticks the *granularity*. The granularity of a reference clock is the smallest unit of time across all other clocks in the network. Furthermore, we assume an omniscient observer which has access to a reference clock. Each event will be recognized by the observer and tagged with the timestamp of the reference clock.

The **drift** of a clock k between microticks i and $i+1$ is defined as the frequency ration between clock k and the reference clock at the instant of microtick i. Thus, the drift can be calculated by counting the number of microticks of the reference clock z during one granule of clock k and dividing it by the nominal number of microticks n^k of the reference clocks' microticks in a granule [21]:

$$drift_i^k = \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k}$$

Furthermore, the **driftrate** ρ_i^k [21] is defined as

$$\rho_i^k = \left| \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k} - 1 \right|$$

which tends to get zero for perfect clocks.

The **offset** error of two clocks j and k with the same granularity is defined as the time difference between two consecutive microticks of these clocks measured in microticks of the reference clock [21].

$$offset_i^{jk} = |z(microtick_i^j) - z(microtick_i^k)|$$

The **precision** Π_i at microtick i of a given ensemble of clocks $\{1, 2, \ldots, n\}$ is defined as the maximum offset between any of these clocks [21]:

$$\Pi_i = \max_{\forall 1 \leq j,k \leq n} \{offset_i^{jk}\}$$

The maximum offset between any two clocks in an interval of interest is called the precision Π of the ensemble and is measured in microticks of the reference clock.

The **accuracy** is defined as the offset of clock k against the reference clock at microtick i of the reference clock. The maximum offset of clock k in an interval of interest is denoted by $accuracy^k$.

For now the most important terms for clocks are defined. In the following the basic concepts of internal clock synchronization will be explained.

The idea behind clock synchronization is that all correct nodes work within a specified precision Φ regardless of the driftrates of the single clocks. Since every clock works slightly different they need to be synchronized after an interval called *resynchronization interval* R_{int}. An example is shown in Figure 5.8. Here one can see that clocks drift apart (grey shaded areas) and after the duration R_{int} they are resynchronized and the process starts again. The *convergence function* Φ denotes the offset values immediately after resynchronization. The *drift offset* Γ denotes the maximum difference between any two good clocks during a resynchronization interval. As the drift offset depends on R_{int} and ρ it can be calculated by

$$\Gamma = 2\rho R_{int}$$

By looking at Figure 5.8 one can see the *synchronization condition* for an ensemble of clocks:

$$\Phi + \Gamma \leq \Pi$$

This means that starting immediately after a resynchronization, the convergence function corrects the clocks to a specified precision Φ. After that the clocks drift apart by Γ. As the clocks need to stay within a defined interval of precision Π the synchronization condition results in $\Phi + \Gamma \leq \Pi$.

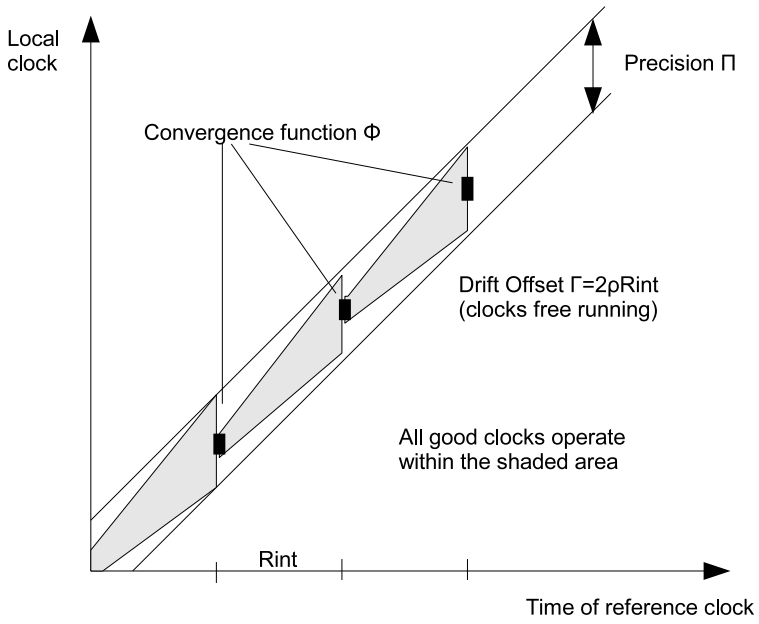

Figure 5.8: Synchronization condition

Now assume we have a central master which periodically sends its current time to all other nodes. The time it takes for the master to read its local clock value, to put into a message and to send it as well as the time it takes for the receiver to unpack the message and to read the sent time is called the *jitter* ϵ. The jitter is most affected through the communication subsystem and thus a non-deterministic function. However, adding the jitter to the precision of the central master synchronization results in

$$\Pi_{central} = \epsilon + \Gamma$$

Precision Time Protocol IEEE1588

The Precision Time Protocol (PTP) is an IEEE standardized protocol for high precise clock synchronization [15]. It is applicable to any communication system supporting multicast communication. The protocol supports a system wide synchronization accuracy to a grandmaster clock in the sub-microsecond range with minimal network and clock computing resources. This section gives a short overview about the terms, constraints, exchanged messages and the communication scheme of the PTP.

Terms of PTP

Grandmaster clock: The grandmaster clock is the source of synchronization within a domain. It is comparable with a reference clock.

PTP Port: A logical access point of a clock for PTP communication to the communications network.

Ordinary clock: A clock that has a single PTP Port. The clock can act as master clock or as a slave clock which synchronizes to a master clock.

Boundary clock: A clock that has multiple PTP Ports in a domain and maintains the timescale used in that domain. That clock may be a master or a slave clock.

Constraints to the network and the implementation

- The network eliminates cyclic forwarding of PTP messages.
- PTP assumes a multicast network model.
- The time accuracy is degraded by asymmetry in the communication paths. This means that it takes different times for messages to be passed to the synchronization source and back.
- PTP tolerates duplicated, missed or out-of-order messages as long as they happen seldom.
- The network should be optimized to forward PTP messages at high priorities to prevent the introduction of jitter.

Exchanged messages

The PTP distinguishes between event messages which contain a timestamp and general messages which do not require accurate timestamps. An event message can be one of:

- Sync
- Delay_Req
- PDelay_Req
- PDelay_Resp

General messages are defined by:

- Announce
- Follow_Up
- Delay_Resp
- PDelay_Resp_Follow_Up
- Management
- Signaling

The PTP defines two ways to measure the propagation delay between PTP ports, namely the delay request-response method for the synchronization of ordinary and boundary clocks and the peer delay mechanism for measuring the link delays. Sync, Delay_Req, Follow_Up and Delay_Resp messages are used in the request-response method whereas PDelay_Req, PDelay_Resp and PDelay_Resp_Follow_Up messages are used to implement the peer delay mechanism. The Announce messages are used to establish a hierarchy between master and slave clocks. Management messages are intended to query and update PTP data sets as well as to customize the PTP system. Signaling messages are defined for communication between clocks regarding all other purposes.

Synchronization Process

The execution of the PTP works in two phases:

- Building a master-slave hierarchy through Announce messages, and
- Synchronization of the clocks

The hierarchy is established by a best master-clock algorithm. The properties of the clocks received in the Announce messages are compared to the already known clocks and the best among them is chosen as master.

Next, the clocks are synchronized according to the message exchange sequence shown in Figure 5.9.

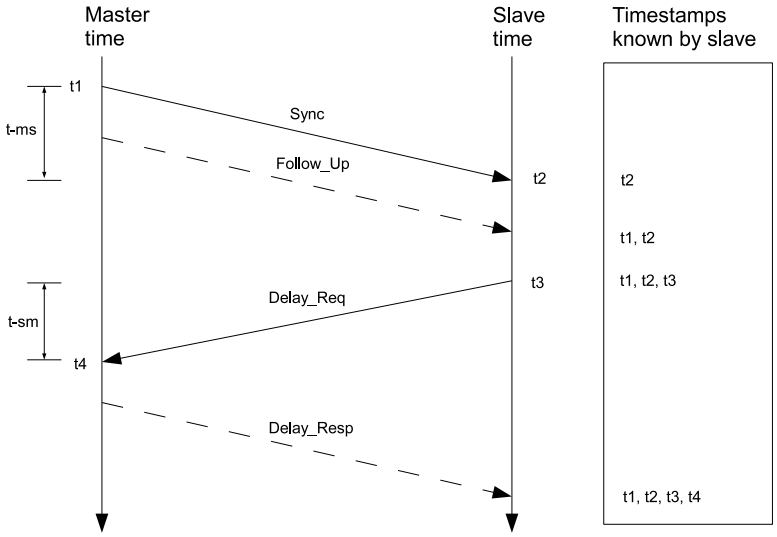

Figure 5.9: Basic synchronization message exchange [15]

- The master initiates the sync-process by sending a Sync message to all its slaves and notes the time t_1 it was sent.
- The slave receives the Sync message and notes the time t_2 it was received.
- The master tells its slaves about the time t_1 when the Sync message has been sent. This can be done either by packing t_1 into the Sync message or by sending a Follow_Up message containing t_1.
- The slave sends a Delay_Req message to the master and notes the time t_3 it was sent.
- The master receives the Delay_Req and notes the time of its reception t_4.
- The master replies with a Delay_Resp containing t_4.

After the synchronization process a slave knows all four times t_1, t_2, t_3 and t_4. First, an offset correction can be applied:

$$offset = t_2 - t_1 - delay$$

where the $Delay$ is not known yet. For the correction of the delay the times t_{ms} (delay master to slave) and t_{sm} (delay slave to master) are assumed to be equal. Otherwise, small errors in the calculation of the link delay will occur.

$$t_{ms} = t_2 - t_1$$

$$t_{sm} = t_4 - t_3$$

$$delay = \frac{t_{ms} + t_{sm}}{2}$$

It is absolutely essential to have the exact times of the sending and receiving instants of the messages. That means that a timestamp is best drawn as late as possible before sending it. Due to the execution of the protocol stack this is not easily possible. Thus, Follow_Up messages are used which contain the sending instant of the previously sent Sync message. The same applies for the receiving times. The timestamp of a received message is best drawn immediately when it is read from the bus. As this is not always possible too, further errors in the offset and link delay calculations will arise.

Vector Clocks

So far the term clock was assumed to be a counter which increases by the progression of time. In the following, we will assume that a clock progresses by the occurrence of events. This means that whenever an event happens the local counter is increased by one. This concept is being referred to as logical clock [22].

Now assume that each node in a network has its own view of the logical times C_i of each other node kept in a vector of length n, where n is the number of processes in the network. At the beginning the vector is initialized with the zero vector. Whenever an event occurs the clock ticks immediately before the event by incrementing the value of its own component:

$$C_i[i] := C_i[i] + 1$$

Each message which is sent across the network contains the timestamp vector of its sender. By receiving a message the timestamps t of the remote vector and the local vector C_i are adapted by the following function:

$$C_i := sup(C_i, t)$$

where *sup* denotes the component-wise maximum operation. The timestamp $C(e)$ of an event e at process P_i is the value of the clock C_i at the moment of the execution of e. An example execution is depicted in Figure 5.10. The vector timestamp $C(e)$ of an event e contains the complete knowledge about previously happened events from which e is potentially dependent.

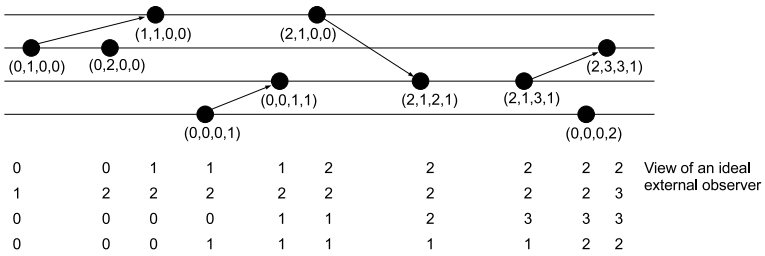

Figure 5.10: Example execution of vector clocks

Conclusion

Summing up, the vector clocks protocol is well applicable for environments where the number of nodes participating the communication is limited since occurance of an event will require to exchange the vector time. Thus, increasing the number of nodes implies increasing the length of messages and increasing the utilization on the bus. However, the vector clocks method is not applicable to KNX Safety due to a simple reason: The protocol extension only allows six bytes of user data. As described, a node has to keep track of all events happening at other nodes which requires to store an array with the length of the number of participating nodes. Assuming that 255 events (1 byte) might happen before the local counters are resetted, only 6 nodes could participate the synchronization which is not sufficient for the requirements of the KNX Safety project.

The Precision Time Protocol is intended for use in environments where highly accurate clock synchronization is required. Therefore, a wide variety of different message types, data types and settings are defined in the IEEE 1588 standard. As one can see later, our approach does neither require nor give the possibility in the implementation of such a high accuracy. Thus, not all properties of the PTP will be implemented. It is sufficient to achieve an accuracy in the millisecond range. Therefore, the basic synchronization algorithm depicted in Figure 5.9 could be implemented.

5.3 Intercommunication - KNX Safety Protocol Extension

To gain functional safety it is not sufficient to just build safe hardware. Instead, also software has to be designed to fulfill safety requirements which involves safety of application software and an applied communication protocol. By employing the mentioned architecture depicted in Figure 5.3 and taking into account requirements of SIL3, a safe failure fraction of 90% to 99% is necessary. That is, more than 90% of all dangerous failures shall be detected. That involves failures in the safety-related part or failures in the black channel which is the standard KNX network.

Safety devices developed from scratch will have no constraints for protocol design. Since this thesis builds on an existing KNX protocol which shall not be altered, protocol safety has

to be gained differently. A widely accepted approach to extend non-safe protocols by safety, is to embed a separate protocol enabling safety requirements into the payload area of the non-safe underlying protocol. Such existing solutions have already been presented in the previous chapter. These solutions make use of the black channel principle stating that the non-safe protocol resides in the black channel and therefore needs not to be taken into account for safety considerations. Instead, the embedded safety-providing protocol has to take care of all mentioned communication errors as specified in Table 4.1. Therefore, the protocol extension as proposed by [20] will be applied (see Figure 5.11).

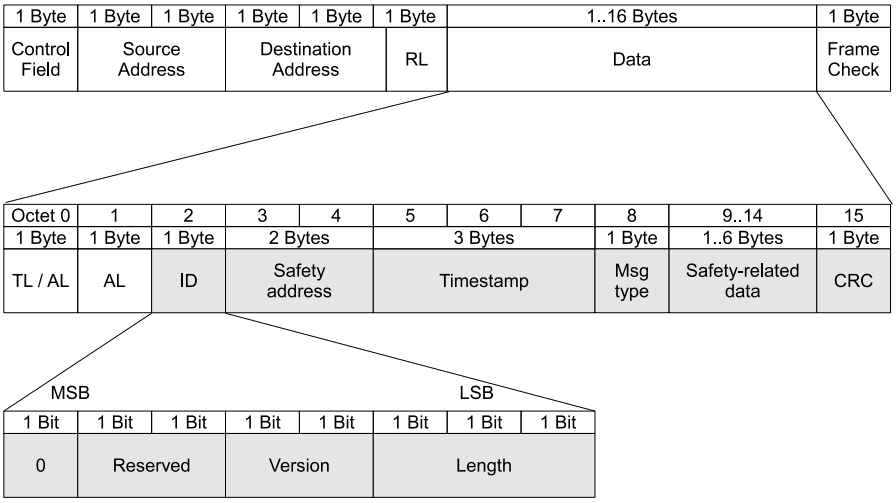

Figure 5.11: Safety providing protocol extension for KNX

- The ID field contains information about the protocol version and the length of the following safety-related data.
- The safety address of the source is encapsulated in every sent message. During commissioning phase, each safety data-point is assigned a safety address. Additionally, each safety data-point receives a list of safety addresses which it is allowed to receive messages from. Thus, a received message will only be processed if the safety address contained in the message is on the list of known safety addresses. Using two bytes for safety addresses, a total of $2^{16} = 65535$ safety data-points is possible.
- Since KNX is purely event-driven and sends messages solely upon changed values, the last received value is assumed to be the most current one which gives no possibility to detect loss or delay of messages. For safety considerations, each message is tagged with a timestamp generated at the time-instant of the causally connected event. To be able to compare timestamps across a network, clock synchronization is required.
- The message type defines the type of the following safety-related data.
- Safety-related data carries the current values captured from the environment.

- Every safety-related frame is checked by a CRC which enables to detect stochastic faults like bit faults resulting in corruption of data.

Implementing the afore mentioned protocol, communication errors as defined in Table 4.1 will be detected by the following measures (see Table 5.1):

Communication error	Safety Measures							
	Sequence number	Timestamp	Time expectation	Connection authentication	Feedback message	Data integrity assurance	Redundancy with cross-checking	Different data integrity assurance systems
Corruption						X		
Unintended repetition		X					X	
Incorrect sequence		X					X	
Loss		X	X					
Unacceptable delay		X	X					
Insertion		X		X			X	
Masquerade				X				X
Addressing				X				

Table 5.1: Communication errors and detection measures used in KNX safety

- Corruption will be detected by CRC in every safety-related data frame.
- Unintended repetition will be detected by timestamps generated once for every sent message. If more than one message from the same source contains the same timestamp, the message has to be neglected.
- Incorrect Sequence: Received messages have to arrive in a strict timely order. That is, assuming events e_1 and e_2 happened at time instants t_1 and t_2, respectively, where t_1 happens before t_2 (denoted as $t_1 < t_2$) then message m_1 sent in accordance to e_1 has to arrive before m_2 sent in accordance to e_2.
- Loss: Safety extension will send messages cyclically after predefined intervals (heartbeat) regardless if values have changed or not. Since every node knows these intervals, a timer (watchdog) will be started upon reception of such a message. If after expiration of that timeout no new heartbeat message has arrived, loss of connection has to be assumed.
- Unacceptable delay will be detected like message loss.
- Insertion: Each safety node is taught a list of safety addresses during the commissioning phase which it is allowed to receive messages from. If a safety address contained in a safety-related message is not in the list of known safety addresses, the message can be assumed to be inserted and has to be neglected.
- Masquerade: Each safety-related message contains the safety address of the sending node and a checksum mechanism to ensure data-integrity. Since the checksum will be generated over the whole safety-related part of the message, it is very unlikely, that a non-safety

message contains information which would represent a valid safety address accepted by the receiving node and, additionally, also the checksum is valid. Thus, it is almost impossible that a non-safety message is interpreted as a safety-related message.

- Addressing will be detected like insertion.
- Additionally, each safety-related message will be received and checked by both safety controllers. Only if both safety controllers agree on the same correct result, the message will be accepted (redundancy with cross-checking). Otherwise, the message will be neglected.

Communication between safety data-points

Addressing of safety nodes cannot be done directly since the safety extension is built on top of an existing KNX protocol. Thus, a safety-relevant message is packed into the payload of a standard KNX message and will have a usual KNX address, too. An addressing scheme is depicted in Figure 5.12.

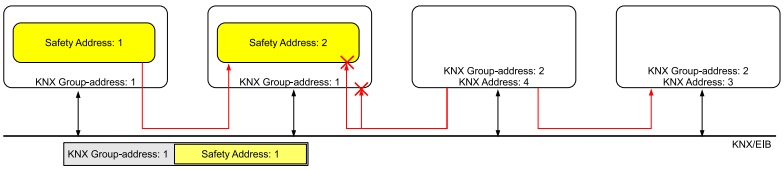

Figure 5.12: Schematic addressing in KNX Safety

Referring to Figure 5.12, if a safety relevant message should be sent from a data-point with safety address 1 (SA) to SA 2, the safety-relevant message will be packed into a standard KNX message with a particular KNX address. Since non-safety devices in the KNX network, such as routers, switches and gateways, do not care about the data content of a message, the safety address cannot be used for addressing. The most simple solution therefore is to define one KNX group where all KNX nodes containing safety data-points are connected. If a message is sent to that group, every node in the same group will receive and accept that message. Thus, also all safety data-points will receive that message. As depicted in Firgure 5.12, if the data-point with safety address 1 tries so send a message, the safety-relevant message will be packed into a standard KNX message containing the KNX group address 1. That message will be received and accepted by any standard KNX node in that particular group with address 1. Any KNX message with a different group address should be neglected by the KNX stack. If a message with an unacceptable group address passes the KNX stack anyway, the safety-related part of the message should not be accepted by the SCL since safety-relevant messages are only accepted if the safety address is in the list of known safety data-points.

As mentioned afore, timing expectation is required to detect loss or delay of messages. To determine a realistic timeout, it is required to know the worst case time it takes for a message to be received by another node. The timeout consists of the processing times required for handling the safety and non-safety protocol, the transfer time from the microcontroller to the TP-UART-chip and from the TP-UART-chip to the KNX-line. Transfer times from microcontroller

to TP-UART and following KNX-line additionally are dependent on KNX-specific retransmit measures. Likewise, handling and transfer times on the receiving node have to be calculated.

- Data sent from the microcontroller to the TP-UART will be transferred at a transmission rate of 19200 bit/s. As soon as the last byte has been transferred to the TP-UART, a checksum will be calculated at the TP-UART and the message will be transmitted to the KNX-line [25]. The time required for checksum calculation will be neglected here. The transfer rate of 19200 bit/s yields in a bit-time of $52,083\mu s$. One data-packet transferred to the TP-UART consists of one start-bit, eight data-bits, one parity-bit and one stop-bit (11 bits). Between consecutive bytes, the bus will be idle for 2 bit-times. Additionally, each data byte sent from the microcontroller to the TP-UART is preceeded by a control byte. To indicate start and end of data-content, special data start and data end octets are transmitted. Assuming the extended KNX frame format, a maximum of 263 bytes of data might be transmitted. Attaching start and end bytes as well as control bytes to each of the transmitted data bytes, this results in 265 * 2 = 530 bytes which is a worst case time of 358,75ms required to transfer data from the microcontroller to the TP-UART.

- Transmission from the TP-UART to the KNX-line: At maximum, 265 bytes (1 control byte, 263 data bytes and 1 checksum byte) will be transferred at a transmission rate of 9600 bit/s which yields in a bit-time of 104μs. Between consecutive bytes, the bus will be idle for 2 bit-times. Before transmission, the sender will wait for 53 bit-times (5,52ms) to ensure that no other sender is currently active. The transmission might fail if a higher prioritized frame is currently in transfer. In that case, the frame will be retransmitted for a maximum of three times. Between retransmit attempts, the sender will wait for 50 bit-times (5,512ms). One data-byte transferred will be enclosed by one start-bit, one parity-bit and one stop-bit which yields in 1,146ms for one data-byte to be transferred. The worst case time for one complete data frame to be transferred would be to initially wait for an empty bus-line (53 bit-times) and failing to send due to higher prioritized frame with a maximum length of 265 bytes. After that, the sender will wait for another 50 bit-times and might fail again. There will be one initial try to transmit followed by 3 retries. Summing up, the overall worst case transfer time results in: 53 bit-times (5,521ms) followed by 265 bytes data (358,07ms), 50 bit-times idle (5,512ms) and 265 bytes data. That is, 5,521ms + 358,07ms +3*(5,512ms + 358,07ms) = 1,454s worst case transmission time from the TP-UART to the KNX-line.

- Transmission from the KNX-line to the TP-UART and the microcontroller: In contrast to the transmission from the microcontroller to the TP-UART and further to the KNX-line, data received by the TP-UART from the KNX-line will be immediately forwarded to the microcontroller after receiving the control-byte from the KNX-line.

- Repeaters, routers, etc. on the KNX-line: KNX supports transmission over a maximum of 5 lines. That means, summing up transmission times from microcontroller to TP-UART (358,75ms) and TP-UART to KNX-line (1,454s) a controller will receive the message at latest after 1812,75ms. Each time a controller forwards a message, that time has to be calculated yielding in 5*1812,75ms = 9,06s until a node in 5 lines distance receives a

message. A complete transmission cycle between two KNX nodes is illustrated in Figure 5.13.

Data exchange between safety data points

To exchange data between safety data points it is required to define the meaning of the exchanged data. Therefore, the KNX safety protocol contains the message type field which indicates how to interpret the following data.

So far identified message types could be encoded as depicted in Table 5.2:

Octet 8								Message type
7	6	5	4	3	2	1	0	
General messages								
0	0	0	0	0	0	0	1	SKNX_A_heartbeat
0	0	0	0	0	0	1	0	SKNX_A_safestate
Time synchronization messages								
0	0	0	0	0	0	1	1	SKNX_A_timesync_sync
0	0	0	0	0	1	0	0	SKNX_A_timesync_follow_up
0	0	0	0	0	1	0	1	SKNX_A_timesync_delay_req
0	0	0	0	0	1	1	0	SKNX_A_timesync_delay_resp
Process data messages								
0	0	0	0	0	1	1	1	SKNX_A_pd_value_read
0	0	0	0	1	0	0	0	SKNX_A_pd_value_write

Table 5.2: Message types for KNX Safety

As mentioned afore, each safety node is required to cyclically send heartbeat messages (`SKNX_A_heartbeat`) to indicate that it is still working. Depending on the intervals required for heartbeat messages and time synchronization processes, the time synchronization process could be used as heartbeat mechanism too.

The `SKNX_A_safestate` message is intended to indicate a global error on a network level. Upon receiving that message, all safety nodes have to transfer into their safe state.

The time synchronization messages have already been presented in the previous Section 5.2.

To indicate process data exchange `SKNX_A_pd_value_read` and `SKNX_A_pd_value_write` messages are defined.

5.4 Software Architecture for a Safety Node

To apply the afore discussed architecture depicted in Figure 5.3, certain requirements arise for the software. Since the chosen architecture consists of two microcontrollers, a communication protocol to exchange messages between them has to be found. To detect errors in the communication system, time synchronization is required. Furthermore, to gain a SFF of more than 90% test mechanisms have to be provided as well. Finally, as stated in [5] if safety-related and non-safety-related software are executed on the same device, it has to be ensured, that the non-safety-related part does not have any influence on the safety-related part of the software.

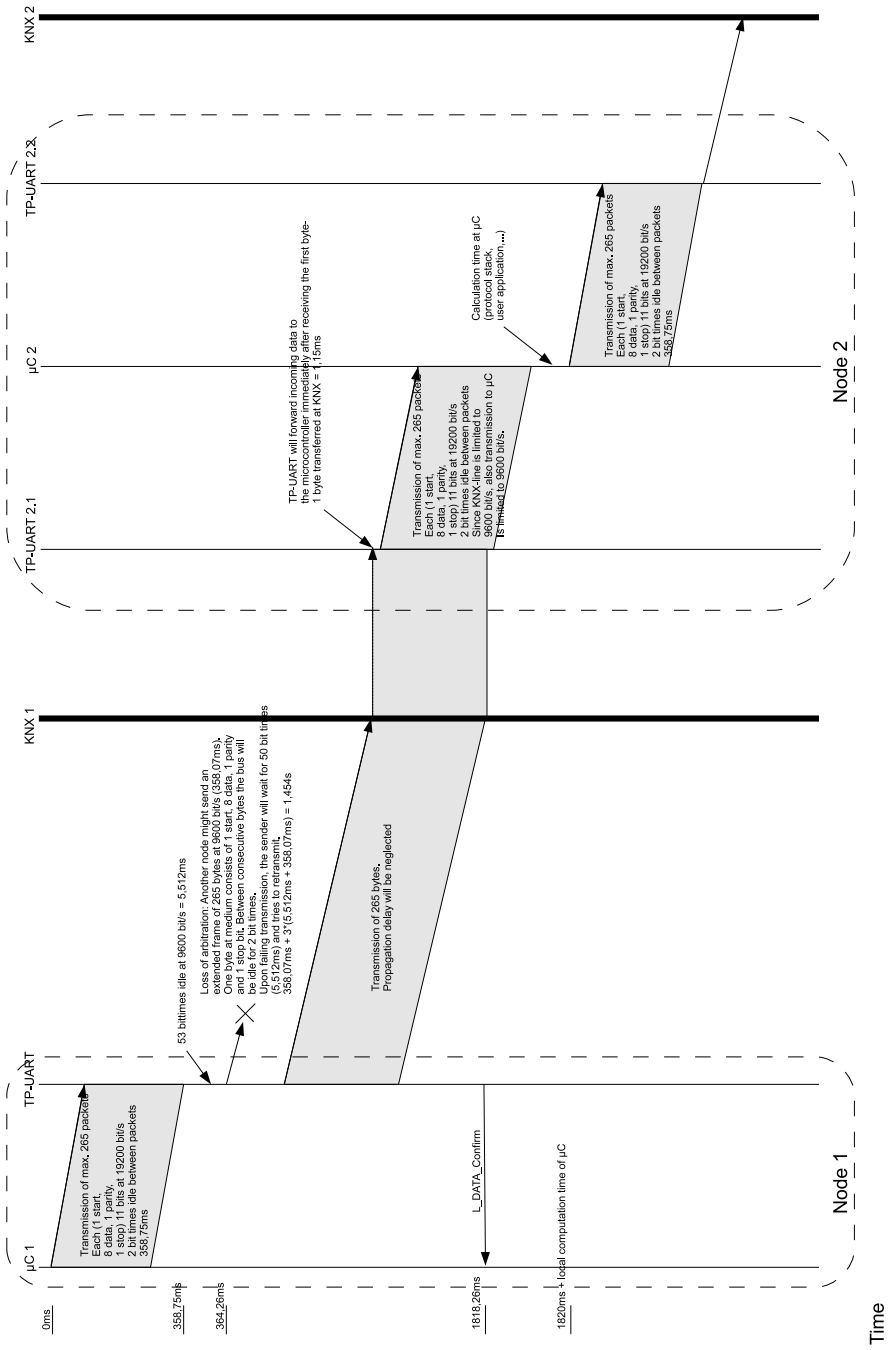

Figure 5.13: Timing diagram of message exchange between KNX nodes

Therefore, an operating system or scheduler has to be developed which ensures independence of safety-related and non-safety-related parts of the software.

A possible software architecture for KNX Safety is depicted in Figure 5.14. As one can see, the KNX Safety API is the core of intelligence of a safety node. The KNX Safety API provides an operating system responsible for executing cyclical tasks like time synchronization, online self tests and serial communication between safe controllers 1 and 2. Furthermore, both, the KNX application and the KNX Safety application access the KNX Safety API by means of its included operating system. Depending on the received message type either a task will be scheduled to the KNX application or the KNX Safety application, respectively. It has to be mentioned, that in case a safety-related task and a non-safety-related task are to be executed, the non-safety-related task will be preempted. As one can see, the KNX application will just be executed on safe controller 1. Since there is no requirement for non-safety-related messages to be cross-checked with safe controller 2, safe controller 2 will never receive a non-safety-related message. Thus, safe controller 2 cannot run the non-safety-related application.

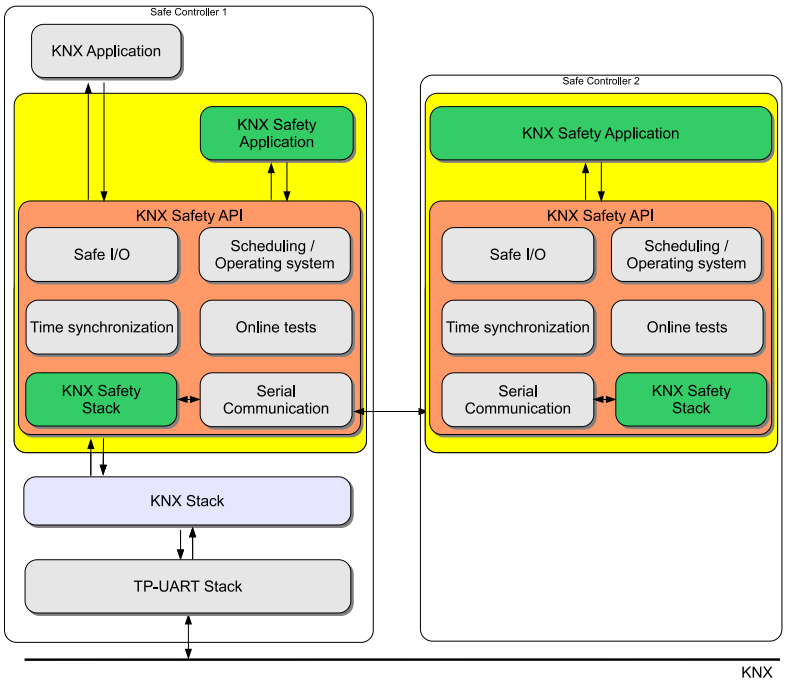

Figure 5.14: Software architecture of a safe KNX node

The following sections will provide discussions on the afore mentioned issues regarding a communication protocol between safe controllers 1 and 2 (Section 5.5), an operating system (Section 5.8) and online test mechanisms (Section 5.7).

5.5 Intracommunication - Communication between Safe Controllers

Messages are received from the TP-UART Chip only at one processor, namely the coordinator. As we have a two channel architecture (see Figure 5.3) the other processor, namely the participant needs to get the message too. Therefore, this section describes possible methods for a reliable message exchange between two or more processors.

Simple Acknowledge

The first protocol is a simple transmission of data with a following acknowledge as shown in Figure 5.15. Using this protocol the coordinator can be sure that the participant got the message if it replied with an ACK. The other way round, the participant cannot be sure, whether the coordinator actually got the ACK to the just received message. This means, that the participant cannot be sure if the coordinator actually knows that the participant got the message. Thus, a more sophisticated way of a message exchange has to be applied.

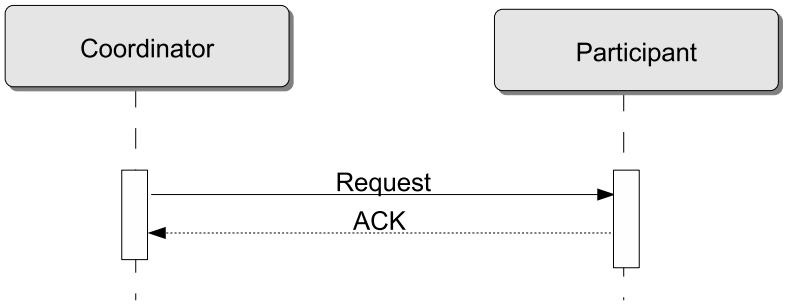

Figure 5.15: Simple acknowledge transmission protocol

Two-Phase-Commit Protocol 2PC

The logical consequence to the afore mentioned problem is the introduction of a second communication phase which ensures that a participant gets an acknowledge to the previously sent ACK as presented by [27]. The presented 2PC is at first based on the assumption that no failures occur. Afterwards, possible scenarios are described where the coordinator or participants may fail. Example executions of 2PC are depicted in Figures 5.16 and 5.17:

- The coordinator sends a `VOTE_REQUEST` to all its participants and expects to get either `VOTE_COMMIT` or `VOTE_ABORT`.
- If a participant receives a `VOTE_REQUEST` it either returns a `VOTE_COMMIT` to indicate that it is locally prepared to commit a transaction or it replies a `VOTE_ABORT`.

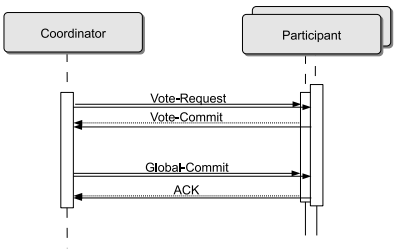

Figure 5.16: Sequence diagram of a successful Two-Phase-Commit Protocol

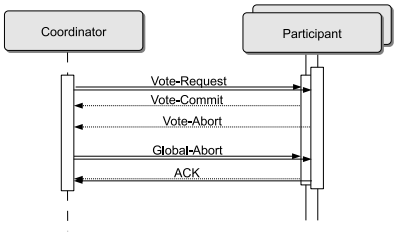

Figure 5.17: Sequence diagram of a failed Two-Phase-Commit Protocol

- Upon receiving a `VOTE_COMMIT` from all its participants the coordinator will reply with a `GLOBAL_COMMIT` to notify the participants to commit the transaction.
- If one of the participants replies with a `VOTE_ABORT` the coordinator will broadcast a `GLOBAL_ABORT` to indicate that the transaction has failed.

In a failure free scenario the 2PC-protocol can ensure that a participant got a previously sent message, and the coordinator will know that the participant actually got the message. As the assumption of a failure free environment is not sufficient for this thesis, we assume that fail-stop failures may occur. This can be explained best using the state diagrams depicted in Figures 5.18 for the coordinator and 5.19 for the participants.

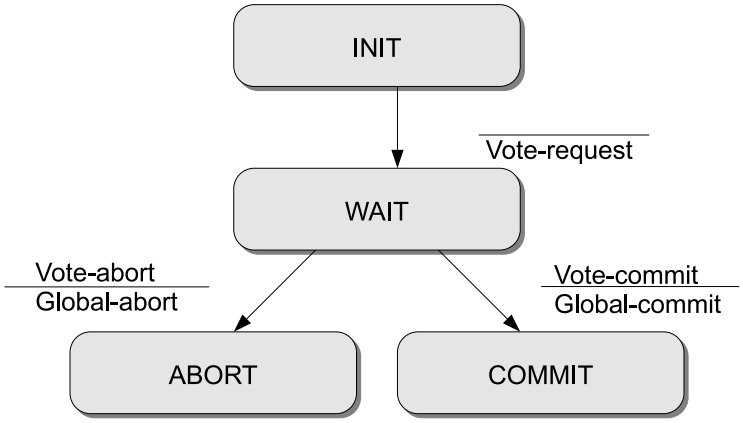

Figure 5.18: State diagram of the coordinator in the Two-Phase-Commit Protocol

First, assume all participants are in `INIT` state and the coordinator crashes. The participants will wait for a `VOTE_REQUEST`. Since the coordinator has crashed such a message will never be received and thus the participants will be blocked in `INIT` state. To detect such a case participants will wait until a timeout happens and send a `VOTE_ABORT` to the coordinator and cancel the current transaction locally.

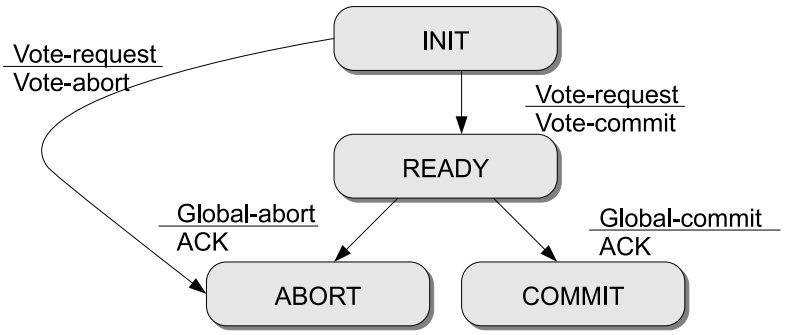

Figure 5.19: State diagram of a participant in the Two-Phase-Commit Protocol

A similar case can be observed if the coordinator is in state `WAIT`, and waits for votes from its participants. If not all participants replied within a certain time, the coordinator has to distribute a `GLOBAL_ABORT` to all of them.

Finally, assume a participant in state `READY` waiting for a `GLOBAL_COMMIT` or a `GLOBAL_ABORT` from the coordinator. Furthermore, assume the coordinator has crashed. This means that the coordinator may have sent a `VOTE_REQUEST` or a `GLOBAL_COMMIT`, whereas either of those messages has not been delivered to all participants. In such a case the participant may not simply abort the transaction. Instead, it has to find out in which state the other participants are and decide according to their states to either abort or commit the transaction.

In any case this model assumes that faulty processes recover within a finite time. To enable local recovery the current state of the process needs to be written to a persistent memory. For instance a participant which has crashed in states `COMMIT` or `ABORT` without having returned an ACK to the coordinator, can recover to its last logged state and inform the coordinator about its decision.

Problems arise if a participant crashes in state `READY`. After recovery it can not safely decide to abort or commit the transaction without checking the decisions of other participants.

If the coordinator crashes in state `WAIT` it has to ensure, that it has not missed any `COMMIT`-messages. Therefore, a safe solution is to retransmit the `VOTE_REQUEST`. Likewise, if a decision has already been taken it is sufficient to retransmit it when recovering.

Here one can observe that a participant may block until the coordinator has recovered. Such a scenario is present if all participants have received the `VOTE_REQUEST` and the coordinator crashes. If so, the participants cannot cooperatively decide on a final result. A possible solution to avoid the blocking issue is resolved in the Three-Phase-Commit protocol described in the following section.

Three-Phase-Commit Protocol 3PC

As described before if the coordinator crashes, the participants may not be able to reach a final decision. Therefore, [26] has extended 2PC to avoid blocking processes in crash-stop scenarios. To achieve this, the following two constraints have to be fulfilled:

- There is no single state from which it is possible to directly reach one of the states `COMMIT` or `ABORT`.
- From each state it is possible to reach a final decision and from which a transition to `COMMIT` can be made.

The execution of the 3PC is quite similar to 2PC but with the difference that an additional pre-commit phase is now introduced. The coordinator starts again by multicasting a `VOTE_REQUEST` to its participants and expects to receive `VOTE_COMMIT` messages. Once the coordinator got all `VOTE_COMMIT` messages it broadcasts a `PREPARE_COMMIT`. After receiving all acknowledges the coordinator will now send the `GLOBAL_COMMIT` message to actually commit the transaction.

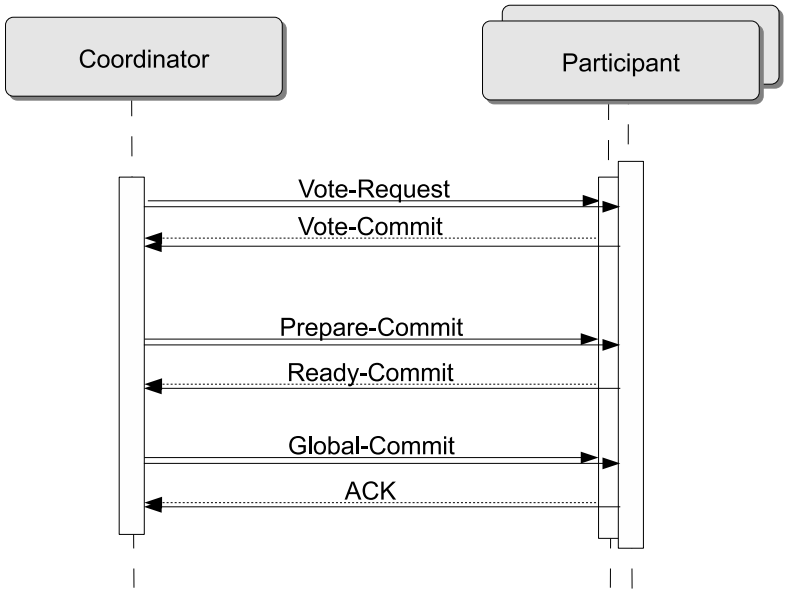

Figure 5.20: Sequence diagram of the Three-Phase-Commit Protocol

Once again, there are possible scenarios where processors may block waiting for incoming messages. Therefore, state diagrams for the coordinator and the participant in Figures 5.21 and 5.22 illustrate that behaviour.

Like in the 2PC, a participant may wait for a VOTE_REQUEST until it times out and eventually aborts the transaction. Analogously, the coordinator may stay in WAIT state waiting for votes from the participants. On a timeout the coordinator will assume that one or more participants have crashed, abort the transaction and broadcast a GLOBAL_ABORT.

Now assume that the coordinator is blocked in state PRECOMMIT. Since all participants must have voted for committing the transaction before - otherwise the coordinator would not have reached the PRECOMMIT state - the coordinator can now safely commit the transaction by multicasting a GLOBAL_COMMIT message.

A participant may block in one of the states READY or PRECOMMIT. On a timeout the participant has to ask its neighbours for their states. If all of them are in state COMMIT or ABORT the participant should move to one of those states, too. If all neighbours are in state PRECOMMIT the transaction can safely be commited.

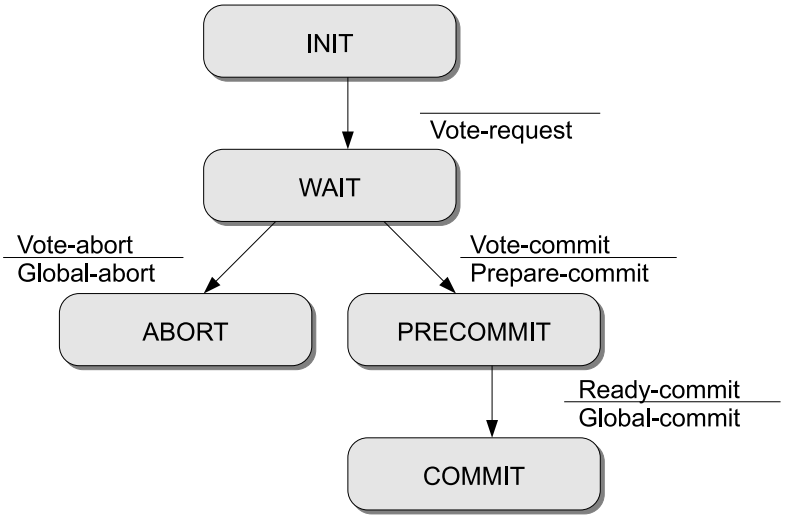

Figure 5.21: State diagram of the coordinator in the Three-Phase-Commit Protocol

Conclusion

This section has described possible methods for the exchange of messages between safe controllers on a single node - hence the name intracommunication. The simple acknowledge protocol has been shown to not supply the required level of safety. The applied architecture as depicted in Figure 5.3 makes use of just two safe controllers. The three-phase commit protocol is assumed to rely on a majority of correct working controllers, which can not be guaranteed with only two processors. Hence, the 3PC can be taken out of consideration. Thus, the two-phase

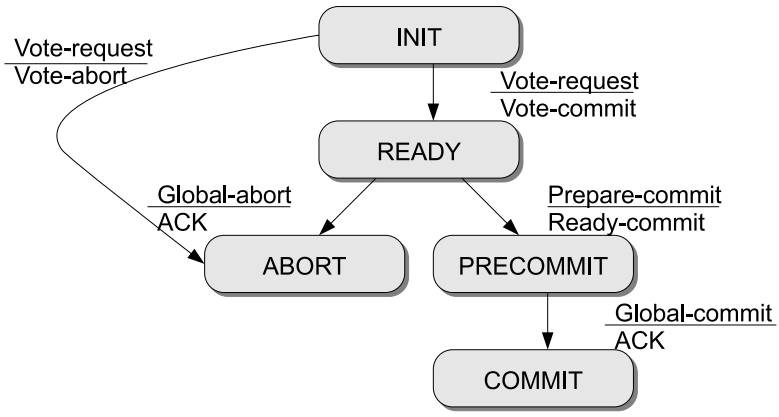

Figure 5.22: State diagram of a participant in the Three-Phase-Commit Protocol

commit protocol is the only remaining protocol which might be applied for communication between safe controllers 1 and 2 in KNX Safety.

5.6 KNX Safety Application

The actual user application is carried out in the KNX Safety Application. The application programmer defines how KNX Safety messages `SKNX_A_pd_value_read` and `SKNX_A_pd_value_write` as defined in Table 5.2 have to be interpreted. Therefore, the programmer has access to the KNX Safety stack to fetch the content of a safety message and to create safety messages. To enable interaction with the environment, the programmer has access to the interface of the safe I/O unit. It has to be mentioned, that it is up to the underlying operating system to run cyclical processes like time synchronization and intracommunication. The KNX Safety application is solely responsible for reading and writing output values according to the user application.

Similar to KNX, the concept of data points will be applied here too. Here, a *KNX Safety Data Point* can be read using a `SKNX_A_pd_value_read` message while writing a safety data point will be handled by an `SKNX_A_pd_value_write` message. If KNX-mechanisms like `A_GroupValue_Read-PDU` and `A_GroupValue_Write-PDU` are required too, Table 5.2 has to be extended by the required message types.

5.7 Hardware self tests

As already stated, a certain SIL can be gained through increasing fault tolerance of hardware or increasing detection of failures which is mainly gathered by software. While hardware architectures have already been discussed thoroughly in previous chapters, test mechanisms in software will be covered in this section.

Basically, there are two ways of test executions: First, to run the system for a predefined time and fully test it afterwards (*offline test*). A second possibility is to test the system cyclically in running mode (*online test*). Figure 5.23 illustrates the required test intervals of on- and offline tests.

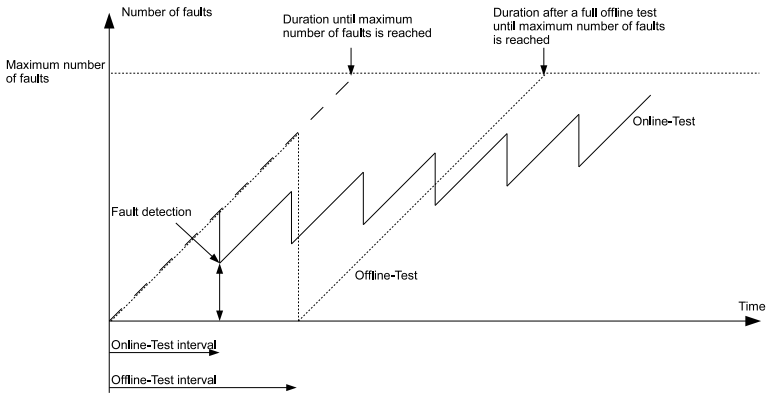

Figure 5.23: Online and Offline test intervals. Slightly modified illustration from [28]

During operation the number of errors will increase linear. If no error detection measures are performed, the system will run until a maximum number of faults is reached where the system cannot be assumed to work safely any further. At that time instant, the system has to be fully tested. After that test the system is in a theoretical new state which means that theoretically every error has been detected and repaired. In practice not every error will be detected nor repaired.

If cyclical tests are performed, a fraction of errors can be detected and repaired. Since not every error can be detected or repaired by online tests (for instance mechanical wear problems) some errors will remain what requires to perform a full test additionally.

It is clear, that only combination of online and offline tests yield in an optimal solution. Eventually, information about best test strategies gives an FMEA analysis.

Errors in memory

Any CPU requires various memory elements to process data. Therefore, data is stored in memory elements and retrieved later to be processed. During that time, information in memory might be corrupted through hardware defects. Basically, memory can be divided into two categories: *Read Only Memory (ROM)* and *Random Access Memory (RAM)*. ROM keeps the operating system, bootstrap loader and application code while RAM contains working information like registers. Thus, errors in memory may occur in various ways resulting in marginal deviated stored values or in hazardous program execution. Therefore, it is of importance to ensure a correct working behaviour of memory. This can only be achieved through repetitive memory tests. The test intervals will depend on manufacturer specific MTTF of the memory.

Memory elements are organized in units of bytes (8 bit) or words (16 bit) and can be accessed by addressing the memory element followed by a write or read command. Memory elements are addressed by an address decoder controlled by the CPU. The contents of the addressed memory cell are then made accessible by an I/O driver which is controlled by an access logic deciding if the cell has to be read or written. An illustration of a memory structure including possible errors is given in Figure 5.28.

An error is present if the memory access deviates from the intended behaviour. An error free memory element will behave like depicted in Figure 5.24.

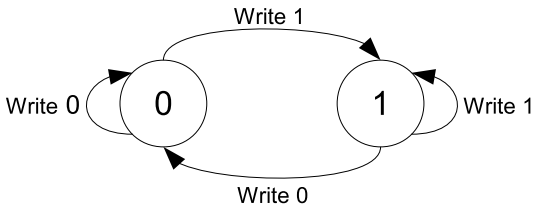

Figure 5.24: State diagram of a correct working memory cell

If the current value of the cell is 0, and the next value written will be 0 too, the resulting value of the cell will be 0. If the current value is 0 and the written value is 1, the resulting value will be 1. The same applies for an initial value of 1. Any deviation of the afore depicted state diagram is an error.

Such an error might be a *stuck-at-error* as depicted in Figures 5.25 and 5.26 resulting in an unchanged memory cell. If the current value of the cell is 0 it will remain 0 regardless of the written value. The same applies to a cell value of 1 and a written value of 0.

Similarly, a memory cell can be in a dominant state resulting in an unchangeable state once the cell resides in that dominant state (see Figure 5.27).

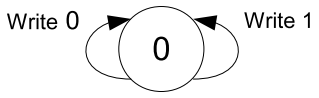

Figure 5.25: State diagram of a stuck-at zero error in a memory cell

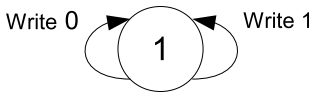

Figure 5.26: State diagram of a stuck-at one error in a memory cell

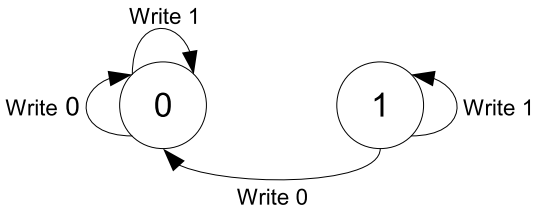

Figure 5.27: State diagram of a state transition error of memory cell

So far, errors in memory have been considered to occur in the memory cell only. Since memory cells are accessed via addresses, the address decoder or access logic might behave erroneous, too. Potential errors are depicted in Figure 5.28.

- A short circuit between address lines will result in replicated data in memory since addresses will occur twice. The replicated cells will depend on either a logical 0 or 1 in the address decoder will become dominant.
- Likewise, a stuck-at to ground will result in replicated data in memory.
- A short after the decoder will result in replicated memory, too.
- A short in the data area will result in identical bytes in a word. If the short resides on the output side, any data will be affected. There is also the possibility that the short resides in the memory cell itself which can be detected by direct addressing the cell.
- Timing errors can be assumed to occur sporadically. But if they occur, a total failure of the device can be assumed.
- Defects in the memory cell can be caused by manufacturing process or occur during operation.
- Open circuits might result in no access at all or might affect neighbor cells.

RAM tests

As described so far, errors in memory may occur in the memory cell or in the access logic for the cell. Thus, memory testing should cover all parts of the memory element to ensure a high diagnostic coverage. Basically, memory tests work by writing test patterns to memory and

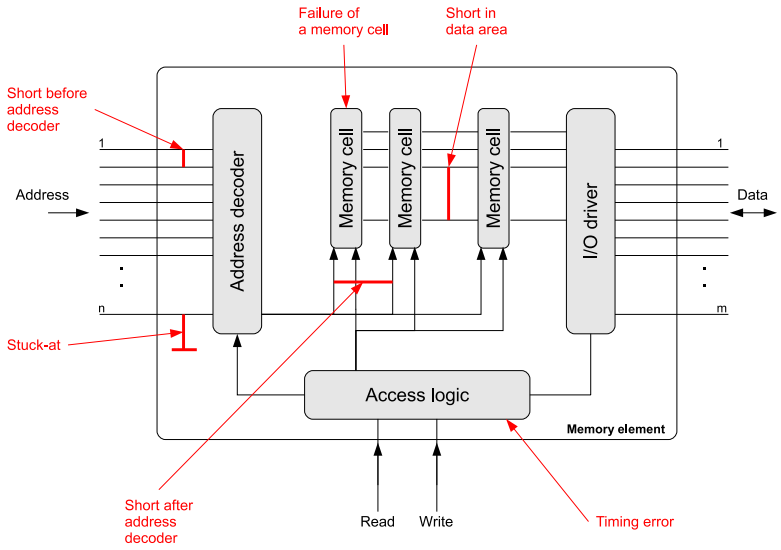

Figure 5.28: Potential errors in a memory block

reading them back afterwards. If the written value differs from the read one, the memory cell has to be assumed erroneous. It is clear, that this kind of tests can only be performed on writable memories.

The *Marching-Bit-Test* assumes an empty memory (all cells set to 0) and writes sequentially, beginning by the first memory address ones to each bit of the memory. Before writing the new value, each bit is checked to be 0. In a second run, the test will be performed with inverted data beginning from the last memory cell.

Similarly, the *Checkerboard-Pattern-Test* initially writes alternating 0/1 bits to memory. Afterwards, all bits are read back and checked for their correct value. A second run is performed with inverted data.

The *Walkpat-Pattern-Test* assumes a memory initialized with defined bit-patterns. The first step will be to invert the first bit and test all other bits for their validity. After that, the first bit will be set to its initial value again and the procedure will be performed for the second bit. In a second run, the whole memory will be inverted and tested again.

Finally, the *Galpat-Pattern-Test* (galloping patterns) is a variation of Walkpat-Pattern-Test where a single 1 passes an initially empty (all bits set to 0) memory. After inverting a single bit, all (including the currently set bit) bits are read and tested for validity. Additionally, after reading a (0-) bit-cell also the inverted (1-) bit-cell will be checked for validity. Thus, the Galpat-Pattern-Test also detects errors yielding from unexpected writing after reading a bit-cell. After every bit-cell has been inverted, a second run will be performed starting with a 1-initialized

memory where a single bit is set to 0. Figure 5.29 illustrates a sample test execution.

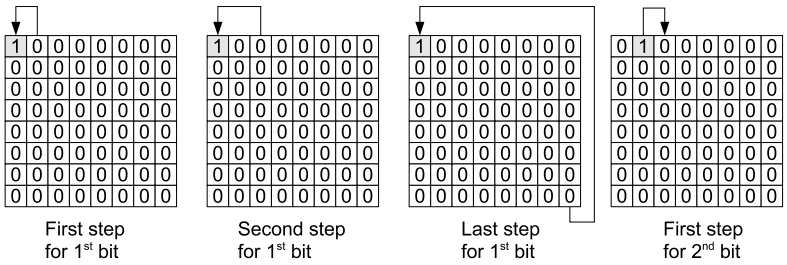

Figure 5.29: Sample execution of Galpat-Pattern-Test

Memory Test Performance and Test Strategies

Depending on the chosen test pattern, memory tests can be very time consuming. For example, simply setting all bits to 0, reading the memory, writing all bits to 1 and reading the complete memory again already results in test length of $4*2^N$ where N is the number of address bits where diagnostic coverage (DC) is less than 50% since neither the decoder errors nor short circuits are detected. The more exhaustive Galpat-Pattern-Test provides very high diagnostic coverage but yields in test length of $2*(2^N+2*n^2)$ where N is the number of address bits and n the number of memory cells (bits). Table 5.3 gives an overview of tests and the resulting DC.

Test / Diagnostic method	**Diagnostic Coverage**
Checkerboard-Pattern-Test	low
Marching-Bit-Test	low
Walkpat-Pattern-Test	medium
Galpat-Pattern-Test	high
Parity Bit	low
Mirrored memory with constant bit comparison on every read- and write access	high

Table 5.3: RAM test methods and resulting DC

Since processing time can be assumed to be limited, it is almost impossible to test the whole memory with a high DC in a single test execution. Instead, memory has to be divided into several smaller segments and a test manager has to take care of running memory tests part-wise. Since all presented memory test algorithms are data destroying, the values before test execution have to be mirrored and written back again. Additionally, the currently in use memory has to be tested, too. To ensure correctness of mirrored data, it is required to calculate a checksum over

the mirrored memory area before the test starts, recalculate the checksum after writing back the memory segment and finally compare the checksums.

Read Only Memory Tests

Contents of read only memory are usually written only once by the manufacturer to ROM or during activation or maintenance to EEPROM (Electrical Erasable Programmable Read Only Memory). Possible errors can therefore be reduced to random errors occurring at putting on supply voltage. Thus, ROM checks should be performed immediately at startup of the system. Since errors can occur during operation of the system, online-tests are necessary.

The simplest test for ROMs are calculation of parity bits (even of odd) and store the parity information in a separate word. Alternatively, checksums of ROM can be taken and stored. Additionally, overflow bits might be considered or not.

A more safe method is to calculate a CRC where the complete memory is assumed to be a polynomial. Therefore, every byte of the memory is attached to a chain. Using that chain a CRC is calculated where the remainder is kept in memory. It is clear, that CRC calculation requires check polynomials guaranteeing Hamming-distance and complete coding. Figure 5.30 illustrates a simple example assuming two bytes of memory and additionally, one byte for the CRC checksum. To increase data integrity, an appropriate check polynomial has to be chosen. As depicted in Figure 5.30, combining 16 bits differently results in 65536 possible combinations of data. Including 8 bits for checksum, 24 bits of information resulting in 16777216 possible data combinations are available. That is, 16777216 possible combinations against 65535 valid combinations. Probability to not detect an error is thus $65535/16777216 = 1/256$ which yields in DC of more than 99%.

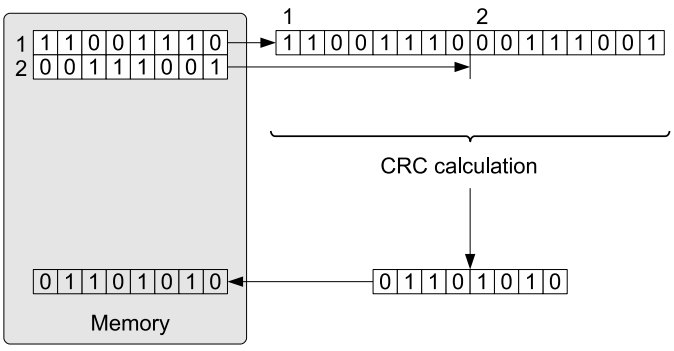

Figure 5.30: Sample calculation of CRC

Summing up, ROM error detection mechanisms differ in simplicity and thus error detection coverage. Parity checks will fail even on even numbers of flipped bits, where for failing CRC at

least 4 bits have to flip. Table 5.4 gives an overview of ROM test measures and resulting DC.

Test / Diagnostic method	**Diagnostic Coverage**
Parity bit	low
Double word checksum	medium
CRC with guaranteed Hamming-distance	high

Table 5.4: ROM test methods and resulting DC

Errors in the CPU

A CPU (Central Processing Unit) is the core of any processor or microcontroller and is responsible for executing software. Usually, CPU consists of an ALU (Arithmetic Logic Unit), instruction counter, registers and instruction decoder. Microcontrollers are microprocessors extended by peripheral components like I/O ports, clock generator, watchdog or communication ports. To thoroughly test a microcontroller implementations have to be provided to test every component separately. It is clear, that only those components have to be tested which are required for the safety functionality.

There is still doubt about the effectiveness of online CPU tests since the question arises which errors in a processor could be detected by an erroneous processor and if there is a possibility to transfer it to a safe state upon detecting an error. Therefore, the following conditions have been defined [28]:

- A test should detect random errors.
- A test should detect errors in production lots.
- The DC is derived from error models and not from error combinations.
- Errors are limited in their effects. Even in case of an error, there is still possibility to transfer the system into a safe state.

[4] presents requirements for error models in single components of microcontrollers and defines according DC upon detecting an error. An example would be an emergency-stop signal which is fired only rarely. Software is implemented to execute the according handler which works on different registers. Caused by rare usage of those methods and switching to associated registers, it might be the case that the handler method is not executed correctly.

Therefore, every seldom used method or hardware component has to be tested dynamically. Furthermore, test results have to be compared with a predefined expectancy value. It is clear, that self tests consume a lot of time, but components do not have to be tested concurrently. Test routines can be executed serially, where overall consuming time to test the whole device should be between one and two hours [28]. For systems which are restarted regularly it might be sufficient to execute self-tests at startup.

Execution of tests are coordinated by a test manager. The manager has to take care, that none of the test routines consumes more time than provided. Therefore tests have to be designed to be short enough to fit that constraint. If a test routine takes more time it has to be divided into several smaller jobs.

Checking the Stack

The stack memory size is assigned at development time. Depending on the usage of interrupts stack memory can reserve some bytes up to some kilobytes. The exact size can be determined after extensive tests. Monitoring stack memory can be done as illustrated in Figure 5.31: The maximum stack memory element is allocated a fixed value. Following stack memory entries are filled top-down. Thus, a stack underflow can only occur if more “pop”-commands are executed than “push”-commands. A stack overflow might occur if (nested) interrupt routines are executed multiple times. Therefore, a buffer area in the stack memory is reserved. Stack memory is initialized on startup and has to be tested during runtime like other memory.

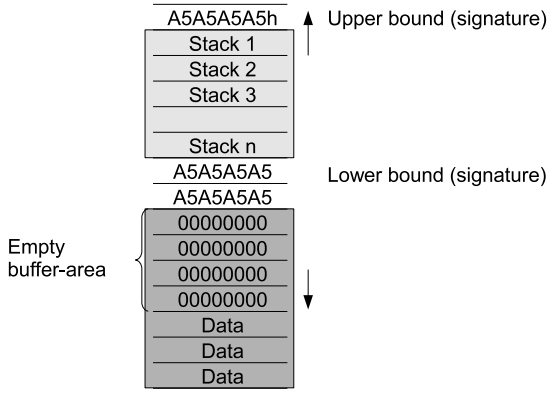

Figure 5.31: Structure of stack memory

Implementing test routines

For different test routines it might be required to temporarily disable interrupts. It has to be verified, that enabling and disabling interrupts has been executed successfully. If components are required constantly it might not be necessary to test them separately since deviations might already be detected during regular operation. For example, sending and receiving registers of a serial communications port which cyclically sends and receives data do not have to be tested. Especially if protocols containing checksums are used, any error in the communication system will be detected by higher level software.

5.8 Scheduling tasks on a Microprocessor

So far, most required tasks for safety-related nodes have been identified and discussed thoroughly:

- (Cyclical) Message exchange between safety data-points (intercommunication)
- (Cyclical) Message exchange between safety controllers (intracommunication)
- (Cyclical) Clock synchronization between safety nodes
- Running the safety-related application software
- Running the non-safety-related application software
- Running hardware self tests

To ensure in time execution of each of these tasks, scheduling strategies are of major interest. On personal computers, scheduling is implemented by the operating system. On most microcontrollers, scheduling is not available by default. Since the application programmer should not take care about the execution of the afore mentioned tasks, an operating system or at least a simple scheduling mechanism should be provided which takes care of executing the basic tasks required for running the safety-related and non-safety-related software.

In the following, some available operating systems have been evaluated for possible reuse and will be described briefly[1] . The criteria for the choice of an existing product has been the existence of a port to the MSP430f149 since an implementation of the hardware drivers, the TP-UART stack and the KNX stack is already available.

- MicroC/OSII

  MicroC/OSII has been developed by Micriμm and advertises its safety-capability, especially with its SIL3 and even SIL4 compliance:

  > "...it is successfully implemented in some of the highest level safety-critical devices, including those certified for avionics DO-178B Level A, and EUROCAE ED-12B, medical FDA pre-market notification (510(k)), and pre-market approval (PMA), and SIL 3/SIL4 IEC for transportation and nuclear systems.[2]
  > "

  Micriμm offers a 45 day trial license for evaluation after which the product needs to be purchased. There exists a port to the MSP430x5xx processor series, but unfortunately not for the used MSP430f149.

[1] http://processors.wiki.ti.com/index.php/MSP430_Real_Time_Operating_Systems_Overview
[2] http://micrium.com/page/products/rtos/os-ii

- TinyOS

  TinyOS[3] is an open-source embedded operating system mainly developed for sensor nodes (motes). The extension SafeTinyOS[4] adds further functionality regarding datatype and memory safety at runtime. The communication protocol is predefined. Thus, the only thing to do for the programmer is to handle the inputs and outputs. This is done in the module based language NCC. Reuse of existing C-code is not provided thus the complete KNX stack and the TPUART driver would have to be recoded. However, TinyOS supports the MSP430f149 in the Telos mote, but recoding KNX would go beyond the scope of this thesis.

- FreeRTOS

  The RTOS[5] family is available in three different versions:

  - FreeRTOS is open source and royalty free.
  - OpenRTOS is the commercially licensed and supported version or FreeRTOS. It supplies further functionality such as USB and TCP/IP components.
  - SafeRTOS is a SIL3 certified version with a complete development/safety lifecycle documentation for compliance with IEC 61508.

  There exists a FreeRTOS port for the MSP430f149 in combination with the MSPGCC toolchain. However, the last supported version of MSPGCC is dated back in 2004 which is quite too old.

- Other remaining operating systems either do not offer an active support, are not freely available (embOS, IAR PowerPAC) or do not support the MSPGCC toolchain (Salvo, CMX-Tiny+).

So far, operating systems for the MSP430 family have been evaluated for reuse in the KNX safety project. Since no OS met all requirements, scheduling needs to be developed from scratch. In the following, an approach for a simple scheduling of safety and non-safety-related tasks will be presented.

Simple Scheduling for KNX Safety

Scheduling is a very wide area of research and development. Since a detailed discussion on different scheduling mechanisms would go far beyond the scope of that thesis, basic appropriate scheduling mechanisms will be presented only. For choice of a scheduler in KNX Safety, the following tasks and issues have to be taken into consideration:

[3] http://tinyos.net/
[4] http://docs.tinyos.net/index.php/Safe_TinyOS
[5] http://www.freertos.org/

- Safety-related user-application has to be prevented from starvation.
- Intercommunication task
- Intracommunication task by means of commit protocol.
- Clock synchronization task should no be preempted, since even smallest protocol-stack jitter lowers the precision of the synchronization protocol. Therefore, a required clock synchronization process (as clock master) has to be processed immediately. Clock synchronization requests (as clock slave) received from synchronization master also have to be answered as fast as possible.
- A test-manager has to take care of execution times of online self-test routines including internal processor tests (RAM, ROM tests) and external tests (safe IO). Such a routine has to be short enough to prevent other processes from starvation caused by exhaustive CPU usage from test-manager.
- Non-safety-related user-application has to be prevented from starvation.

A common scheduling strategy to prevent starvation of tasks is round-robin which will be outlined in the following. However, a round-robin scheduler does not fully support requirements as defined previously. Especially time-critical tasks might be problematic.

Round-Robin scheduling

A basic round-robin scheduler implements a preemptive, first-come-first-serve (FCFS) strategy with fixed time intervals. A dispatcher will assign each queued process a slot of CPU-time. If the process requires less time than it was assigned, it will release the CPU and the next process in queue will proceed immediately. If the process requires more time, it will be preempted, the dispatcher will choose the next process and the preempted process will be moved back in the queue.

For instance, a simple execution of round-robin scheduler could be as P1 takes 25ms, P2 takes 3ms, P3 takes 15ms, P4 takes 20ms, with timeslot each 10ms. Processes arrive in order P1, P2, P3, P4. Scheduling would be preformed as depicted in the following Table:

A round-robin scheduler prevents tasks from starvation, but does not support requirement to immediately respond to clock synchronization requests. For the sake of simplicity and the fact, that a KNX message might take up to 8 seconds to be received by a remote node (see Section 5.3), clock synchronization cannot be assumed to be as precise as PTP would provide under best circumstances. Furthermore, transfer times for messages are non-deterministic caused by delays at routers, gateways and higher prioritized frames on KNX-line. Therefore, scheduling can be simplified to the approach presented in the following.

Process	CPU cycles	Queue	Information
P1	0..10	P2, P3, P4, P1	P1 preempted, 15ms remaining
P2	10..13	P3, P4, P1	P2 is done
P3	13..23	P4, P1, P3	P3 preempted, 5ms remaining
P4	23..33	P1, P3, P4	P4 preempted, 10ms remaining
P1	33..43	P3, P4, P1	P1 preempted, 5ms remaining
P3	43..48	P4, P1	P3 done
P4	48..58	P1	P4 done
P1	58..63		P1 done

Table 5.5: Example round-robin scheduling

Simplified priority scheduling

Afore mentioned tasks can be reduced to

- Execution of safe user application
- Execution of non-safe user application
- Handling of clock synchronization messages
- Executing self tests

These tasks will be executed cyclically depending whether it is required to execute the task or not. For example, upon receiving a safety-related message, a flag will indicate the reception of the message and during the next round of executing all available tasks serially, the safety message handling task will be executed. To ensure execution of the safety-related tasks, two-level prioritized (safe and non-safe priority levels) will be introduced privileging a safety-related task instead of a non-safety-related task. Furthermore, a watchdog is started up along with starting a task which is configured long enough to execute the task entirely. If a task takes longer than that timeout, it will be preempted. Therefore, tasks have to be designed to fit that timeout or vice-versa.

5.9 Building Safe Hardware

Safe Inputs and Outputs

Safety-related systems are required to perform their defined behaviour under any circumstances. That is not only fail-safe communication between safety nodes and evaluation of data, but also reading and setting according values from the environment via sensors and actuators. Therefore,

mechanisms are required to detect erroneous inputs or outputs. That can be gathered, like already presented in previous chapters, by single-channel approaches or redundant solutions.

Sensors and input devices

The most simple sensor would be a switch connected to an input of the microcontroller. Sensors are not restricted to return binary values but can also provide analog values like a temperature or rotational speed. Single channel approaches might fail if the sensor fails, the connecting wire breaks or the input port of the microprocessor fails, which results in loss of whole safety functionality. Therefore, at least two-channel architectures are of importance. Such architectures are depicted in Figures 5.32 and 5.33. The architecture presented in Figure 5.32 has to be considered too since it might not always be possible to mount more than one sensor. It does not detect a failing sensor, but errors in one of the input stages. To increase the level of safety a second sensor has to be used and in best case both make use of a different technology. If for example both sensors are implemented by the same technology and one sensor fails caused by an unexpected reason, the second sensor would likely fail, too. The usage of different technologies eliminates such faults.

Figure 5.32: Single sensor on replicated input stages

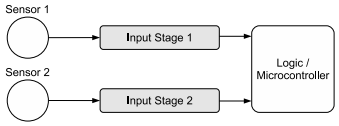

Figure 5.33: Replicated sensors on replicated input stages

Binary sensors like switches have to be handled thoroughly. By looking at Figure 5.34 one can see that safety functionality is completely lost if wiring is shorted. A short between *Signal 1* and *Signal 2* as depicted in Figure 5.35 reduces the circuit to evaluate only one channel. To prevent this, each switch has to be wired separately and connected to different in- and outputs of the logic. Furthermore, instead of providing direct voltage, the signal sent to the switch can be pulsed, where each of the switches gets different pulse patterns. Such pulsed patterns are also referred to as OSSD (Output Silicon Switched Device). Reading back the inputs and comparing the sent pulse pattern with the received one enables to detect shorts between wiring of the switches (see Figure 5.36). It has to be mentioned, that evaluation of the switch state can only be performed if the switch is closed. That is sufficient since it is assumed that an open switch indicates a safe state.

If sensed values can not be applied to the microcontroller directly, input stages have to be used. Depending on the applied architecture, input stages may become single points of failure and therefore have to be tested as well. Since the output signal of the logic is pulsed, the sensor as well as the input stage is tested implicitly. Referring to Figure 5.37, wiring the input stage with logic twice increases possibility to detect errors in the wiring or logic inputs.

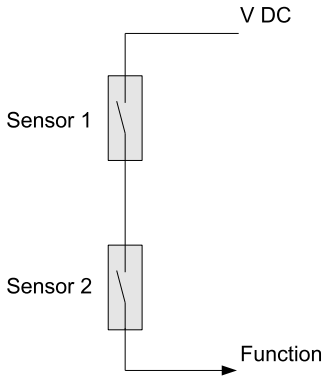

Figure 5.34: Example of connecting two switches in line

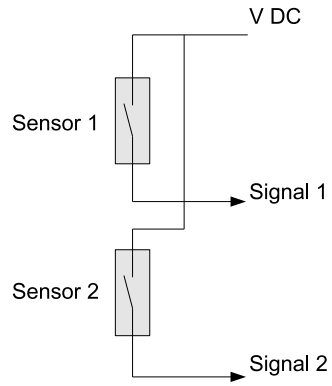

Figure 5.35: Example of connecting two switches parallel

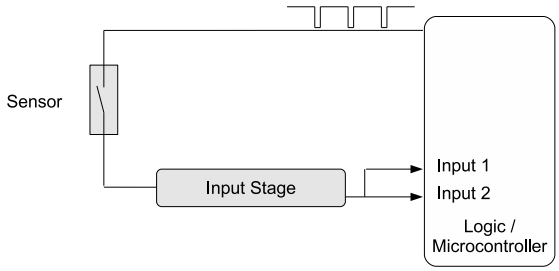

Figure 5.36: Monitoring sensors using pulsed voltage

Since not every sensor provides potential free contacts, it might not be possible to use afore mentioned pulsed pattern technique to ensure correctness of operation. That might be the case for sensors with separate power supply providing an active signal themselves like proximity sensors (see Figure 5.38). Here one can see that the input stage receives a pulsed signal from the logic which cyclically tests if the input stage still reacts by forwarding the pulses to the logic. It is clear, that such an approach assumes correctness of the sensor itself. It can only be tested if the input stage behaves according to its specification.

Safe outputs

In safety-relevant automation systems it is often required to safely turn off a device like a motor for example. Here it is not sufficient to use a simple switch to execute a safety function since in case of a stuck-at error, it might not be possible to disconnect the device and transfer the system into a safe state. Therefore, multiple switches connected serially as depicted in Figure 5.39 are applied. If one of the switches fails to disconnect, there is still another one to execute the safety

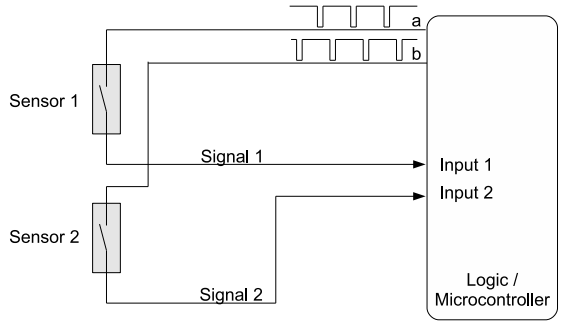

Figure 5.37: Test in a closed circuit

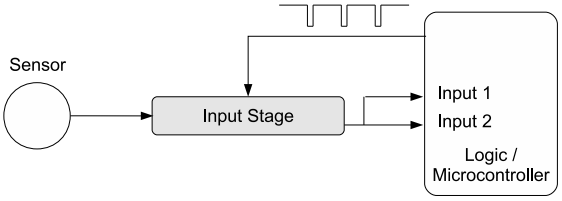

Figure 5.38: Testable input stage in a closed circuit

function and transfer the system to a safe state. To ensure if the switch is actually disconnected, reading back values gives information about successful execution of the switch-command.

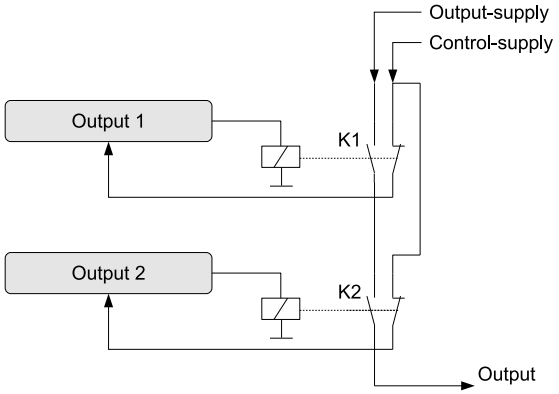

Figure 5.39: Serially connected switches with read-back switch state

A more sophisticated circuit fulfilling even highest level safety requirements is depicted in Figure 5.40. Just like in the afore presented circuit two switching elements are used to safely disconnect the device. To ensure correctness of the operation, both semiconductors (T2, T3)

have to be tested cyclically. Therefore, microcontroller 1 informs microcontroller 2 about an upcoming test and sends an impulse via R1 to T1. That pulse will be recognized by microcontroller 2 (via R4) which will inform microcontroller 1 about correct operation of T2. The same mechanism applies to testing T3 from microcontroller 2. A further measure to increase safety is usage of a fail-safe unit which is enables control voltage for T1. The fail-safe unit is implemented to be controlled dynamically and just enables output signal if both microcontrollers give the same input. That will ensure safe switching off the output even if one of the microcontrollers has failed. It is clear, that test pulses have to be short enough to not affect correct operation of the device connected to the output. Furthermore, semiconductors T2 and T3 are driven by different technology to overcome simultaneous failing of both driving units. Summing up, the following measures for safely switching off have been applied:

- Usage of two microcontrollers (two-channel-architecture)
- Cyclical tests of main semiconductors using test-pulses and reading back pulses crosswise.
- Usage of different driving technology for main semiconductors
- Usage of dynamically controlled fail-safe unit

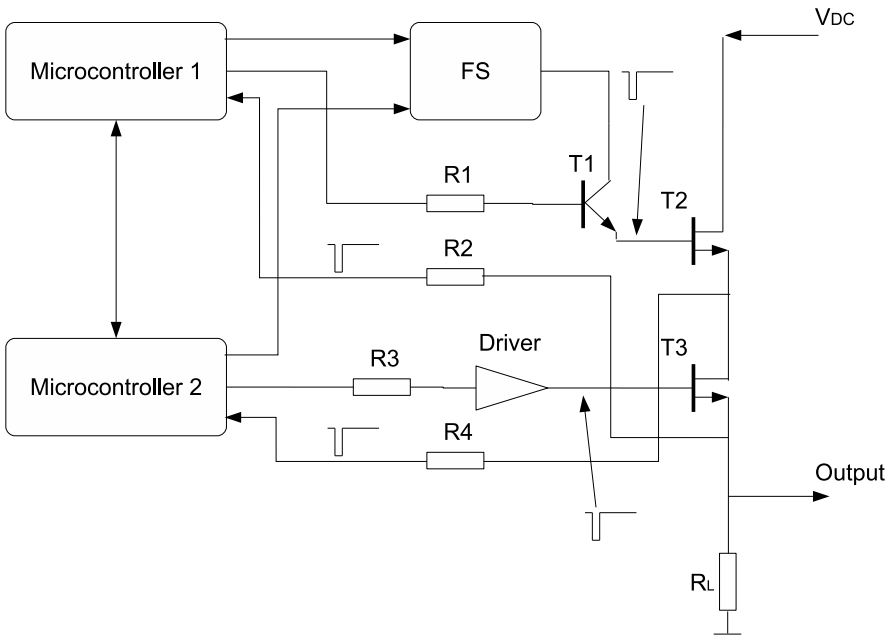

Figure 5.40: Two-channel output using semiconductors

Fail-safe unit As mentioned, a fail-safe unit is required to enable controlling voltage for T1. The unit will only return a valid output if both its inputs are controlled according to predefined

dynamic signals. Otherwise, an output indicating safe state will be provided. An example of a fail-safe output unit is depicted in Figure 5.41. Microcontroller 1 provides constant voltage via high-side driver 1 while microcontroller 2 provides pulsed voltage to drive the transformer via low-side driver 2. Additionally, both microcontrollers provide alternating test pulses. The transformer including filter D1, C1 are dimensioned that microcontrollers are enabled to read back test pulses via R1 or R2, respectively. Following D2 and C2 are dimensioned that output of the fail-safe unit is pulse-free direct voltage. If either of the microcontrollers fails, the output of the fail-safe unit turns to zero since valid outputs of both microcontrollers are required to drive the transformer.

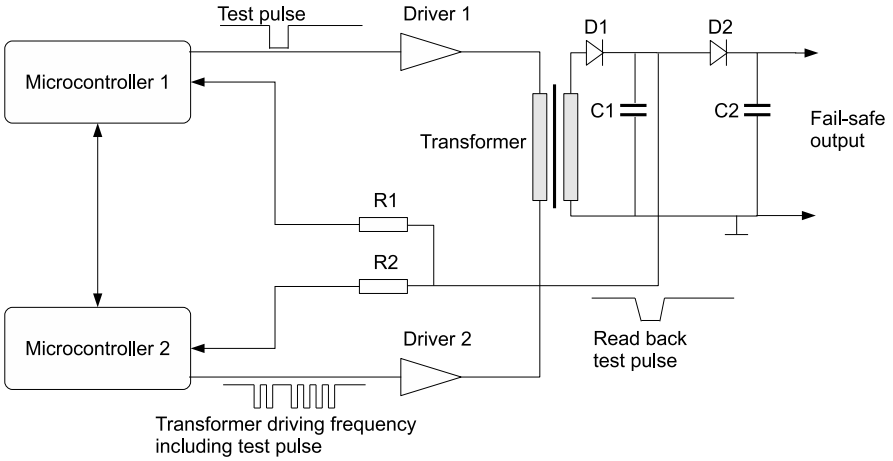

Figure 5.41: Fail-safe unit

CHAPTER 6

Conclusion

The thesis started by explaining basic terms and definitions required for building automation systems and KNX protocol in depth. Chapter 3 gave an outline on state-of-the-art standards IEC 61508 and ISO 13849 concerning functional safety in electrical systems and pointed out requirements for functional safe devices from a development lifecycle as well as hard- and software point of view. Basically, IEC 61508 states that higher safety integrity levels can either be gained by increasing fault tolerance of hardware or by increasing fraction of errors which can be detected by system itself through extensive self tests.

In Chapter 4, a selection of existing solutions for industrial and home and building automation providing functional safety has been presented and compared in terms of their protocol safety. Possible errors in communication systems and measures to detect and prevent them are defined in IEC 61784-3. Communication errors have been identified in case of corruption, unintended repetition, incorrect sequence, loss, unacceptable delay, insertion, masquerade and wrong addressing. Detection of timing-related errors requires implementation of clock synchronization mechanisms to ensure global notion of time and therefore ability to order messages by their occurrences. Due to limited bandwidth of the KNX bus line, existing synchronization protocols, namely vector clocks and precision time protocol, have been evaluated with a focus on reuse in the thesis.

To gain safety integrity level 3 as specified by IEC 61508, application of extensive hardware self tests or hardware redundancy is required. Due to limited processing power of microcontrollers, a hardware redundancy approach using two microcontrollers and a single TP-UART-IC for KNX-line access has been chosen. Even implementation of hardware redundancy requires hardware self tests, but to gain SIL 3 a safe failure fraction of more than 90% is sufficient which can be gained by memory tests as presented in Chapter 5.7. To enable application of redundant controllers, a reliable communication protocol between safe controllers had to be applied.

Furthermore, safe hardware also includes safe in- and outputs for interaction with environment which has been outlined in Section 5.9. Since non-safety and safety-related applications should operate on the same node, IEC 61508 requires to show sufficient independence of safety-related and non-safety-related application which requires implementation of scheduling mechanisms.

Summing up, developing a safety-capable device requires much more than just building redundant hardware. Instead, especially protocol safety is of major concern, which even gets more important if existing wiring has to be kept and safety and non-safety nodes should coexist on the same network. To solve this problem, almost any existing solution for safety-related automation systems relies on the black-channel-principle stating, that the safety-related protocol itself has to take care about correct transmission, reception and detection of errors of a message without relying on potentially implemented error detection mechanisms of the underlying non-safety-related protocol. Further issues become safety of hardware itself which includes implementation of safe interaction with the environment via safe inputs and outputs as well as communication between redundant safe controllers on a safe node itself. Finally, the hardware has to be checked cyclically for correctness of operation where especially errors in memory have to be detected using different memory-check algorithms.

6.1 Outlook and further work

Process data exchange has been explained only very briefly. KNX supports a variety of mechanisms to exchange data between nodes and how nodes are seen from their data-point-of-view by means of their functional block description. To allow a simpler implementation of a KNX Safety network, KNX Safety should be extended to support functional blocks as well as more sophisticated methods to work with safety data points.

Rapid spreading of wireless devices with potential safety-related functionality entails another topic which has to be considered too i.e. security. A safety-related device might operate safely in closed circuits, but if an unauthorized person gains access to a safety-related system, the system itself cannot be considered to safely operate anymore. Therefore, integration of security measures into safety-related devices and vice-versa has to be a major concern.

Bibliography

[1] Konnex Association. *KNX Handbook, Version 2.0*. Konnex Association, 2009.

[2] Konnex Association. *KNX System Specifications, Architecture, v3.0*. Konnex Association, 2009.

[3] CENELEC Europäisches Komitee für Elektrotechnische Normung. DIN EN 61508-1 (VDE 0803 Teil 1) Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme - Teil 1: Allgemeine Anforderungen (IEC 61508-1:1996 + Corrigendum 1999); Deutsche Fassung EN 61508-1:2001. *IEC*, 2001.

[4] CENELEC Europäisches Komitee für Elektrotechnische Normung. DIN EN 61508-2 (VDE 0803 Teil 2) Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme - Teil 2: Anforderungen an sicherheitsbezogene elektrische/elektronische/programmierbare elektronische Systeme (IEC 61508-2:2000); Deutsche Fassung EN 61508-2:2001. *IEC*, 2001.

[5] CENELEC Europäisches Komitee für Elektrotechnische Normung. DIN EN 61508-3 (VDE 0803 Teil 3) Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme - Teil 3: Anforderungen an Software (IEC 61508-3:1998 + Corrigendum 1999); Deutsche Fassung EN 61508-3:2001. *IEC*, 2001.

[6] CENELEC Europäisches Komitee für Elektrotechnische Normung. DIN EN 61508-4 (VDE 0803 Teil 4) Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme - Teil 4: Begriffe und Abkürzungen (IEC 61508-4:1998 + Corrigendum 1999); Deutsche Fassung EN 61508-4:2001. *IEC*, 2001.

[7] CENELEC Europäisches Komitee für Elektrotechnische Normung. DIN EN 61508-6 (VDE 0803 Teil 6) Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme - Teil 6: Anwendungsrichtlinie für IEC 61508-2 und IEC 61508-3 (IEC 61508-6:2000); Deutsche Fassung EN 61508-6:2001. *IEC*, 2001.

[8] Charzinsiki. Bewertung der Fehlersicherungsverfahren im CAN Protokoll. *Universität Stuttgart*, 1991.

[9] DEK Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE. DIN IEC 61784-3 (VDE 0803-500) Industrielle Kommunikationsnetze - Profile - Teil 3: Funktional sichere Übertragung bei Feldbussen (IEC65C/500/CD:2008. *IEC*, 2008.

[10] DIN Deutsches Institut für Normung e.V. DIN EN 13849-1 Sicherheit von Maschinen - Sicherheitsbezogene Teile von Steuerungen - Teil 1: Allgemeine Gestaltungsleitsätze (ISO 13849-1:2006); Deutsche Fassung EN ISO 13849-1:2008. *DIN*, 2008.

[11] DIN Deutsches Institut für Normung e.V. DIN EN 13849-2 Sicherheit von Maschinen - Sicherheitsbezogene Teile von Steuerungen - Teil 2: Validierung (ISO 13849-2:2003); Deutsche Fassung EN ISO 13849-2:2008. *DIN*, 2008.

[12] Fachausschuss Elektrotechnik. Grundsatz für die Prüfung und Zertifizierung von Bussystemen für die Übertragung sicherheitsrelevanter Nachrichten, 2002.

[13] IEC International Electrotechnical Commission. IEC 61508-0 Part 0: Functional safety and IEC 61508. *IEC*, 2005.

[14] IEC International Electrotechnical Commission. IEC 61784-3-12 Industrial communication networks - Profiles - Part 3-12: Functional safety fieldbuses - Additional specifications for CPF 12 (Safety-over-EtherCAT). *IEC*, 2010.

[15] IEEE Instrumentation and Measurement Society. IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE*, Revision of IEEE Std 1588-2002.

[16] CAN in Automation e.V. CANopen - Application Layer and Communication Profile - CiA Draft Standard 301. *CAN in Automation e.V.*, 2002.

[17] CAN in Automation e.V. CANopen - Framework for safety-relevant communication - CiA Draft Standard 304. *CAN in Automation e.V.*, 2005.

[18] Wolfgang Kastner and Georg Neugschwandtner. Datenkommunikation in der verteilten gebäudeautomation. *Bulletin SEV/VSE*, 2006.

[19] Wolfgang Kastner, Georg Neugschwandtner, Stefan Soucek, and H. Michael Newman. Communication systems for building automation and control. In *Proceedings of the IEEE*, volume 93, pages 1178–1203, 2005.

[20] Wolfgang Kastner and Thomas Novak. Functional safety in building automation. In *In Proc. of 14th IEEE Conference on Emerging Technologies and Factory Automation (ETFA '09)*, pages 1–8, September 2009.

[21] Hermann Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Kluwer Academic Publichers, 2003.

[22] Friedemann Mattern. On the relativistic structure of logical time in distributed systems. In *Elsevier Science Publishers B.V in Parallel and Distributed Algorithms*, pages 215–226, 1992.

[23] Thomas Novak and Thomas Tamandl. Architecture of a safe node for a fieldbus system. In *5th IEEE International Conference on Industrial Informatics*, pages 101–106, 2007.

[24] Dietmar Reinert and Dietmar Schaefer. *Sichere Bussysteme für die Automation*. Hüthig, 2001.

[25] Siemens. *Technical data EIB-TP-UART-IC*, 2001.

[26] Dale Skeen and Michael Stonebraker. A formal model of crash recovery in a distributed systems. *IEEE Transactions on Software Engineering*, pages 219–228, 1983.

[27] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems - Principles and Paradigms*. Prentice Hall, 2002.

[28] Peter Wratil and Michael Kieviet. *Sicherheitstechnik für Komponenten und Systeme*. Hüthig, 2007.

[29] Richard Zurawski. *The industrial communication technology handbook*. CRC Press, 2005.