FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Tunneling Ethernet Traffic via Redundant Low Bandwidth Communication Links

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Jürgen Schober
Matrikelnummer 0926506

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof.Dr. Wolfgang Kastner
Mitwirkung: Univ.Ass.Dipl.-Ing.Dr. Lukas Krammer

Wien, 13.08.2015

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Tunneling Ethernet Traffic via Redundant Low Bandwidth Communication Links

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Jürgen Schober
Registration Number 0926506

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof.Dr. Wolfgang Kastner
Assistance: Univ.Ass.Dipl.-Ing.Dr. Lukas Krammer

Vienna, 13.08.2015

(Signature of Author)	(Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Jürgen Schober
Orionstraße 14, 4030 Linz

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)

Acknowledgements

First, I would like to express my gratitude to my advisors Wolfgang Kastner and Lukas Krammer for the useful comments, remarks and engagement through the learning process of my Master's thesis and the possibility to write this thesis at the Automation System Group.

Furthermore, a special thanks to all my colleagues and friends from the Vienna University of Technology for sharing many moments of joy with me over the last years.

I would like to thank my loved one, Christina-Anna, who has supported me throughout the entire process, both by keeping me harmonious and helping me putting pieces together. I will be grateful forever for your love.

Last but not the least, I would like to thank my parents, Ingrid and Gerhard, as well as my sister, Doris, for supporting me spiritually throughout writing this thesis and my life in general.

Abstract

A Network Control System (NCS) is a control system where control loop signals are exchanged via digital communication networks. The aim of those control systems is to eliminate discrete cabling. Hence, the complexity and the total cost for the development and implementation are reduced. If sensors and actuators are added, it is easy to modify and extend them at low cost and without significant change of the system. As communication networks, classical fieldbus systems, Industrial Ethernet or wireless networks are used. The distance covered between two network devices is often larger than 100 meters. However, this distance equals the maximum possible distance between two Ethernet network devices.

An existing platform solves this problem by using TIA/EIA-485. TIA/EIA-485 is a half-duplex differential network bus being able to reach a bus length of up to 1200 meters. Moreover, it is an efficient and robust solution. A micro-controller is utilized to realize an interface between an Ethernet network and TIA/EIA-485. This allows to build wide spread networks, which are often part of NCS. The difficulty still is to transmit Ethernet data transparently.

Within the scope of this thesis, a more efficient protocol for the transmission of Ethernet frames via redundant TIA/EIA-485 interfaces is developed. Thereby, the data throughput is significantly improved compared to the existing protocol. A micro-controller receives data from an Ethernet endpoint and forwards it in the UART data format via two TIA/EIA-485 interfaces. If an error occurs during transmission or one of the two TIA/EIA-485 interfaces fails, it is detected and only a part of the Ethernet frame has to be retransmitted. This is achieved by splitting the Ethernet frames into smaller frames. It has to be assured that all frames are only transmitted and received once. Nevertheless, the system continues the operation with the remaining TIA/EIA-485 interface. The data rate of the TIA/EIA-485 interface is configurable depending on the network length.

The protocol is successfully implemented as a proof-of-concept on Microchip development boards Explorer 16. A sound evaluation shows the feasibility and efficiency of the proposed approach. At the end of the thesis, a detailed comparison with the initial approach is given to show the improvements concerning performance and reliability of the concept.

Kurzfassung

In vernetzten Regelsystemen werden Regelkreise durch ein Kommunikationsnetzwerk geschlossen. Sinn und Zweck solcher Regelsysteme ist es Verkabelung zu eliminieren. Dadurch werden die Komplexität und die Gesamtkosten von Entwicklung und Implementierung reduziert. Fügt man Sensoren und Aktuatoren hinzu, können diese sehr einfach mit geringen Kosten und ohne signifikante Veränderungen der Systeme modifiziert und erweitert werden. Als Kommunikationsnetzwerk werden klassische Feldbusse, industrielles Ethernet oder drahtlose Netzwerke verwendet. In vernetzten Regelsystemen ist die Distanz zwischen zwei Steuergeräten oft größer als 100 Meter. Diese Distanz ist aber gleichzeitig die maximal zulässige Distanz zwischen zwei Ethernet-Netzwerkgeräten.

Eine existierende Plattform löst dieses Problem, in dem sie Telecommunications Industry Association/Electronic Industries Alliance (TIA/EIA)-485 verwendet. TIA/EIA-485 ist ein differenzieller, halb-duplex Bus, mit dem Buslängen von bis zu 1200 Metern erreicht werden können. Zusätzlich ist es eine effiziente und vor allem robuste Lösung. Ein Mikrocontroller wird dafür eingesetzt, um die Schnittstelle zwischen dem Ethernet-Netzwerk und TIA/EIA-485 zu realisieren. Dies ermöglicht ein weit ausgedehntes Netzwerk, wie es in vernetzten Regelsystemen oft Bestandteil ist. Die Schwierigkeit dabei ist, die Ethernet-Daten nachvollziehbar via TIA/EIA-485 zu übertragen.

Im Rahmen dieser Diplomarbeit wird ein effizientes Protokoll für die Übertragung von Ethernet-Frames über redundante TIA/EIA-485-Leitungen entwickelt. Dabei wurde der Datendurchsatz im Vergleich zu einem bestehenden Protokoll erheblich verbessert. Ein Mikrocontroller empfängt die Daten von einem Ethernet-Endpunkt und leitet diese im Universal Asynchronous Receiver Transmitter (UART)-Datenformat über zwei TIA/EIA-485-Schnittstellen weiter. Wenn während der Übertragung ein Fehler auftritt oder eine der beiden TIA/EIA-485-Schnittstellen ausfällt, wird diese Situation erkannt und es muss nur ein Teil des Ethernet-Frames neu übertragen werden. Das ist möglich, indem Ethernet-Frames in kleine Frames aufgespalten werden. Jeder Frame darf nur ein Mal gesendet und empfangen werden. Das System soll trotzdem weiterhin mit der verbleibenden Verbindung funktionieren. Wie hoch die Datenrate ist, kann abhängig von der Leitungslänge der TIA/EIA-485-Schnittstelle konfiguriert werden.

Das neu entworfene Protokoll wurde erfolgreich auf der Microchip-Entwicklungsplatine Explorer 16 implementiert. Wie effizient das Konzept ist, wird im Evaluierungsteil bewertet. Am Ende der Diplomarbeit wird das alte und neue Protokoll verglichen, um die Verbesserung zu verdeutlichen.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

A *NCS* is a control system where control loops are closed through a communication network. Four basic elements are used to establish the functionality of a NCS. Sensors acquire information, controllers provide decisions and commands, actuators execute the control commands and the communication network enables the exchange of information.

NCSs eliminate unnecessary wiring, through which the complexity and the overall cost in designing and implementing the control systems is reduced. By adding sensors and actuators to them, they can be easily modified or upgraded with relatively low cost and no significant changes in their structure. The types of communication networks used in NCSs are fieldbusses, IEEE 802.3/Ethernet [1] and wireless networks. In NCSs, the distance covered between the control instances is often more than 100 meters. This is the maximum possible distance between Ethernet devices.

An existing hardware platform helps solving this problem by using TIA/EIA-485 [2], which is defined as a half-duplex differential network bus, to reach network lengths of up to 1200 meters. The TIA/EIA-485 interface is an efficient and robust solution, often used in NCSs.

A micro-controller is utilized to implement a tunneling interface between an Ethernet network and TIA/EIA-485, which allows building wide spread networks being deployed in NCSs. The difficulty is getting transparent Ethernet traffic over TIA/EIA-485.

A micro-controller receives data from an Ethernet endpoint and forwards the data using the UART data format over two TIA/EIA-485 interfaces. If one of the two TIA/EIA-485 lines breaks down, the system continues the operation with the remaining TIA/EIA-485 interfaces. The TIA/EIA-485 transceiver provides different data rates depending on the network length. Furthermore, it has to take care of splitting the data into small frames and assuring that all frames are only transmitted and received once.

It does not matter which interface is utilized between the Ethernet endpoint and the micro-controller as long as it is much faster than the TIA/EIA-485 interface.

The initial protocol for this application scenario is very inefficient and has low data throughput. Therefore, the new protocol is designed to allow an efficient transmission of Ethernet frames - with sizes ranging from 64 bytes to 1530 bytes - and to provide reliability by using the redundant hardware.

1.2 Aim of the Thesis

The thesis aims at developing an efficient protocol to carry Ethernet frames over redundant TIA/EIA-485 interfaces. Thereby, the data throughput shall be significantly improved compared to the initial protocol. If an error occurs during transmission, it has to be detected and only a part of the Ethernet frames has to be retransmitted. This shall be achieved by splitting the Ethernet frames into smaller frames. The data rate of the TIA/EIA-485 interface shall be configurable depending on the network length.

The approach developed in this thesis will be capable of dealing with physical redundant TIA/EIA-485 interfaces. An interruption of one of the TIA/EIA-485 lines between two micro-controllers or an error of both TIA/EIA-485 connections has to be detected. After the problem on the communication channel is fixed, both channels shall be ready for the transmission.

Another objective of this thesis is the successful implementation of the protocol as a proof-of-concept on Microchip development boards Explorer 16 [3]. A sound evaluation will show the feasibility and efficiency of the proposed approach.

1.3 Methodology and Approach

In a first step, a literature research is done to find out how such a protocol can look like. The basics of the International Organization for Standardization (ISO)-Open Systems Interconnection (OSI) layer model and the physical media are studied. Different existing media access schemes are examined and checked concerning their suitability for the hardware setup. Furthermore, several data protection techniques and communication models are investigated.Many papers with topics similar to this thesis are researched to find the best possible way, how such a protocol can look like.

To reach the expected results, the next step is getting familiar with the existing hardware platform and the development board Explorer 16 as well as the PIC micro-controller itself. Therefore, the used interfaces between the devices are identified and the data sheets of the devices are studied. The existing protocol is analyzed.

In third step, the collected knowledge is used to develop a protocol fitting the requirements. Hence, the discussed media access schemes are compared and then one, which suits the application scenario best, is chosen. Additionally, methods for a reliable communication are elaborated. Based on this knowledge, an advanced concept for the protocol is formulated, where all the details are specified, e.g. by using flow charts to show the procedure of the protocol.

To prove the feasibility and to analyze the performance of the protocol, it is implemented on the development platform Explorer 16. Therefore, several tests using unidirectional and bidirectional data exchange with different Ethernet frame sizes are executed. Based on the outcomes, the performance of the new protocol is examined. In a final step, the implementation of the newly created protocol is compared with the initial protocol to show the gain of the recently developed protocol.

1.4 Structure of the Work

In Chapter 2, the state-of-the-art knowledge gained by the literature research is summarized. The ISO-OSI layer model and the physical media for a data transmission including TIA/EIA-485 are explained. Furthermore, the assets and drawbacks of four different multiple access schemes and three different data protection mechanisms are discussed. To understand the Ethernet data, which will be tunneled, a brief insight into Ethernet is given. At the end of this chapter, a short overview about related work completes the state-of-the-art.

In Chapter 3, the developed protocol is described. The prerequisites of the new protocol are stated and the design considerations are presented by explaining all necessary details. Furthermore, the developed protocol is introduced as well as some mechanisms to detect faults occurring during transmission are illustrated.

Chapter 4 deals with the implementation and evaluation of the designed protocol. Further on, the interfaces used in this thesis are illustrated. The hardware of the initial approach is analyzed for enabling the comparison between the initial protocol and the new protocol. Next, the implementation of the developed protocol is explained. In addition, the chapter informs about the evaluation of the received results.

Finally, the thesis concludes with a summary and open tasks for further research.

CHAPTER 2

Industrial Communication

This chapter provides the theoretical background and the basic principles which are necessary for the development of a new protocol to send Ethernet traffic over TIA/EIA-485 [2]. As the thesis is concerned with the development of such a protocol, the first section explains the ISO-OSI layer model [4]. Subsequently, relevant mechanisms and technologies, like for example physical media including TIA/EIA-485, access schemes and communication models are described according to the layers of the OSI reference model with some examples of the area of NCS due to the usage of TIA/EIA-485. Other important topics for this thesis, which are summarized, are Cyclic Redundancy Check (CRC) [5] and Institute of Electrical and Electronics Engineers (IEEE) 802.3/Ethernet. Afterwards, the existing hardware platform and the current protocol will be discussed to allow a comparison to the new development. Finally, related work is discussed for being adopted for the new concept.

2.1 Basics

A network system can be separated into the user program and the communication system. To cope with the complexity, the communication system can be also split up in several functions. This was done by the ISO with the *Basic Reference Model for OSI*, the so called ISO-OSI layer model or OSI reference model, maintained under the standard ISO/IEC 7498-1 [4].

The necessary operations for the communication between two entities are divided into seven layers with distinctive functions (see Table 2.1). Every layer provides its functions solely to the immediate above layer and only uses the services of the immediate underlying layer.

A lot of networking technologies are based on packet switching involving the creation of small pieces of data which are sent over a network. The word *packet* appears in the name of this communication method, but the data chunks sent between devices in a network are mostly named messages. *Packet* is one of several terms which is used in various contexts to refer to messages transmitted between devices.

These different names can be sometimes very useful, like for example, the used term for the specific message can tell something about the content of the message. In the specific case,

#	**Layer**			**Data unit**
7	Application Layer			Data
6	Presentation Layer			Data
5	Session Layer			Data
4	Transport Layer			Segments
3	Network Layer			Datagram
2	Data Link Layer	2b	LLC	Frame
		2a	MAC	Bit
1	Physical Layer			Bit

Table 2.1: The seven layer of the ISO-OSI-reference model [6]

various message names are associated with protocols and technologies, which operate at specific layers of the OSI reference model. Therefore, the usage of different names can simplify discussions involving several protocols of various layers, but it can also cause confusion because the terms are not always used consistently.

The most common terms which are used for messages are the following [7]:

- Frame: This term refers to messages at low levels of the OSI reference model. In particular, it is mostly used as reference to data link layer messages. A frame is created by taking higher-level packets or datagrams and inserts additional header information, which is needed at the lower level.

- Packet: This term is used for a message which is sent by protocols operating at the network layer of the ISO-OSI layer model, e.g. IP packets. It refers also commonly to any type of message.

- Datagram: This term is sometimes synonymous with *packet*. It refers to network layer technologies, but is also often used to refer to a message that is sent at a higher level of the OSI reference model.

- Segment: If the transport protocol is Transmission Control Protocol (TCP) [8], the unit of data sent from TCP to network layer is called segment.

If the user program from a participant A is going to send data to the user program of participant B, then it uses a service of the application layer (layer 7). Subsequently, the data passes all layers, whereby every layer adds control information to the data of the user and utilizes the service of the underlying layer to pass it down to the physical layer (layer 1), which accomplishes the physical transmission. The data passes through the layers at the participant B in reverse direction. Hence, the physical data are received using layer 1 and are passed up to layer 7.

The terms Protocol Data Unit (PDU) and Service Data Unit (SDU) [4] refer to protocol messages, which are used in the OSI reference model. A PDU at layer N is a message sent between protocols at layer N. It consists of layer N header information and an encapsulated message from layer $N + 1$, which is called both the layer N SDU and the layer $N + 1$ PDU.

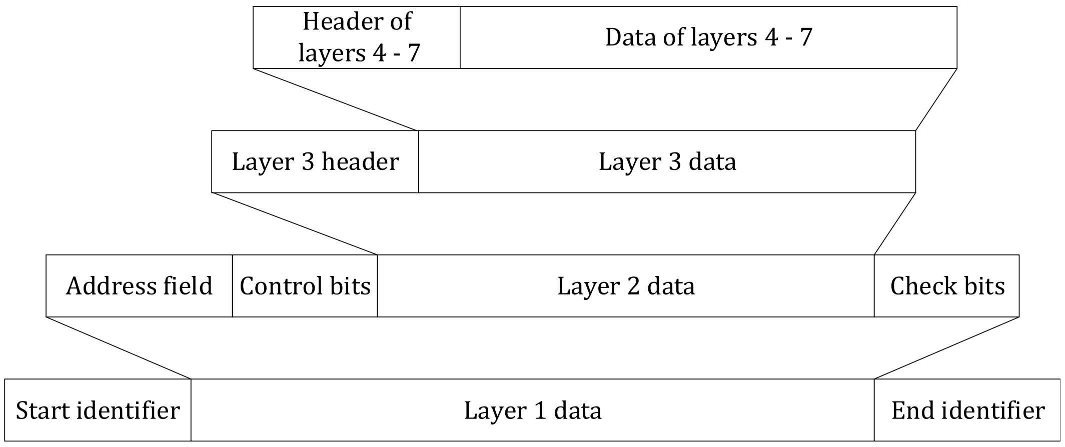


Figure 2.1: Encapsulation of a telegram [6]

The addition of layer specific control information leads to nested telegrams. There are two remarks to Figure 2.1:

1. It is just a matter of interpretation, whether the start- and end-identifier are represented as elements of layer 1 or they are added as control information of layer 2 to the telegram frame.
2. While layers 1 and 2 are creating a frame with header and trailer control information, the control information of the layers 3 to are packed as header in front of the data.

Layer 1 covers the physical realization of the transmission [9]. It defines, for instance, what kind of cable and which encoding for the data transmission is used. These subjects are discussed further in Subsection *Physical Media*.

Layer 2 includes two functions and both are used to send and receive data. For that matter, they allow the data transmission. The kind of medium access (Medium Access Control (MAC) [10]) determines the way how a station which requests access gets the transmission authorization. It must be guaranteed, that two stations accessing the same shared medium never send simultaneously. Otherwise, data would be destroyed due to collisions. These topics are shown in Subsection *Communication Models*.

The second task of layer 2 is to add a frame, which is predefined for a specific communication system, to the data. Usually, a frame with the following elements is built (see Figure 2.1):

- an address field,
- several control bits, which are, for example, used to indicate the length of the payload,
- the data containing the payload and the headers of the upper layers 3 to 7,
- optionally additional information for checking the integrity of the frame.

The frame contains important information besides the payload. There are also telegrams without any payload, e.g., the data field is empty. For example, such telegrams request to send data or to acknowledge the reception of data. The control information, which is added from layer N in participant A, is removed from the data and analyzed in the same layer N in participant B. The design of the control information and the actions, which are determined by them, follow specific rules. For example, in layer 2 the arrangement of the address field and the approach to determine the receiver are determined.

The seven layers can be described as following [11]:

- Layer 1 is concerned with the transmission and reception of the unstructured raw bit stream over a physical medium. It describes the electrical/optical, mechanical, and functional interfaces to the physical medium, and carries the signals for all of the higher layers.
- Layer 2 provides transfer of data frames from one node to another over the physical layer, allowing layers above it to assume virtually error-free transmission over the link.
- Layer 3 controls the operation of the subnet, deciding which physical path the data should take based on network conditions, priority of service, and other factors.
- Layer 4 ensures that messages are delivered without errors, in sequence, and with no losses or duplications between two peers. It relieves the higher layer protocols from any concern with the transfer of data between them and their peers.
- Layer 5 allows session establishment between processes running on different stations.
- Layer 6 formats the data to be presented to the application layer. It can be viewed as the translator for the network. This layer may translate data from a format used by the application layer into a common format at the sending station, then translate the common format to a format known to the application layer at the receiving station.
- Layer 7 serves as the window for users and application processes to access network services.

The Ethernet protocol (compare Section 2.3) includes only layer 1 and 2. On top of that, higher protocols, like the protocol Internet Protocol (IP) [12] for layer 3, relies on the Ethernet protocol.

Every layer produces some overhead. Sending and especially processing the headers consumes a lot of time. As short transmission times are desired in some systems, like for example in fieldbus systems, layers are skipped sometimes. However, essential functions, which are normally located in layers 3 to 6, are moved to layer 7.

2.2 Control Networks

After the basics of communication in general, the following section discusses different types of physical media and multiple access schemes occurring in industrial communication especially in fieldbus systems. Furthermore, data protection procedures and communication models are presented.

The services in a fieldbus system can be divided into three fields of functions:

1. Before the data transfer phase can be started, initial operations have to be done. The individual stations will be initialized and parametrized, perhaps programs will be downloaded and started. Altogether, the stations are prepared for the following periodical transmission of the control data.

2. The data of sensors and actuators have to be transferred in periodic time intervals for the process control. Additionally, short control information, which is also requested periodically and shows whether the devices are working correctly, has to be sent. This can be called the periodical transmission of control data in the data transfer phase.

3. It has to be possible during the data transfer phase to request detailed diagnostic information from a station and to change the control parameters of a station. Therefore, an aperiodic transmission of messages must be possible besides the periodical transmission of control data.

The second task must be done as efficient as possible with minimal overhead. Hence, primarily the services from layer 2 are used for this task. Layer 7 is only used to pass the data to the user and the layer 7 header like in Figure 2.1 is very short or not existing.

The other two tasks only differ in the matter of the time of occurrence, before or during the data transfer phase. The services are very similar or partially the same. Because of the sporadic transmission, the efficiency is less important. Capable layer 7 services, which take over the necessary tasks in such a way, that information is provided in resulting telegrams defined by a standard for easy access to the data, are essential for users.

In a powerful fieldbus system both is equal necessary, the efficient periodical transmission of control data and the capable, through standards defined manifestation of layer 7 with the corresponding services and telegrams for the initial operation, parametrization, configuration, diagnosis and maintenance.

Physical Media

In general, a communication system is not limited to a specific embodiment of the wire between stations and a specific encoding for the binary values 0 and 1 with corresponding voltage levels on this wire. Therefore, it needs to be possible to exchange the realization of layer 1 by keeping the remaining stack as it is. Alternative realizations for the layer 1 in communication systems and thereby for the transmission medium are available.

Usually, there are preferred transmission media for particular protocols as shown in Table 2.2. Twisted two-wire cables are the most used media, e.g. used in Ethernet [1], PROFIBUS [13], Controller Area Network (CAN) [14], Local Operating Network (LON), Modbus and Actuator Sensor Interface (AS-i). Fiber optical cables are also in use. They have advantages at high data rates and no problem with electromagnetic interference, but they are not as sturdy as

Bus system	Transmission line	Transmission standard
Ethernet	Twisted pair (two or more twisted two-wire cable), shielded or unshielded	IEEE 802.3
PROFIBUS-FMS	Twisted two-wire cable, shielded	TIA/EIA-485
PROFIBUS-DP	Twisted two-wire cable, shielded	TIA/EIA-485
PROFIBUS-PA	Twisted two-wire cable, shielded or unshielded	MBP
Foundation-Fieldbus	Twisted two-wire cable, shielded	MBP
CAN	Twisted two-wire cable, shielded	TIA/EIA-485, modified
LON	E.g. twisted two-wire cable	E.g. TIA/EIA-485
Modbus-RTU	Twisted two-wire cable, shielded	TIA/EIA-485
Interbus	Twisted five-wire cable	TIA/EIA-485
AS-i	Twisted two-wire cable, unshielded	specific procedure

Table 2.2: Preferred used transmission medium

twisted two-wire or coaxial cables. Wireless transmission can also be used, but is vulnerable to electromagnetic interferences.

Modulation

A decision about the encoding of the significances 0 and 1 of a bit has to be made due to the realization in hardware of a fieldbus. For this purpose, different line codes will be considered.

- The first option is the unipolar representation like in Figure 2.2 a), e.g. the binary 1 is encoded through a positive voltage V^+ and the binary 0 is encoded through voltage 0. The mean value is floating and it is $\pm$ 0.5 V observed over an infinitely long time. If the power supply and the fieldbus coupling of a device are realized with only one cable pair, the signal is superimposed the supply voltage. Thus, the value of the supply voltage is not constant over time.
- If a bipolar encoding is used, as shown in Figure 2.2 b), the signal has zero-mean. The signal does not return to zero and hence, it is called Non-return-to-zero (NRZ) line code or NRZ signal. Like in the unipolar representation, the signal does not contain any clock information. If there are several consecutive bits of the same value, no falling or rising edge appears. However, this is necessary for clock synchronization. It is possible to return

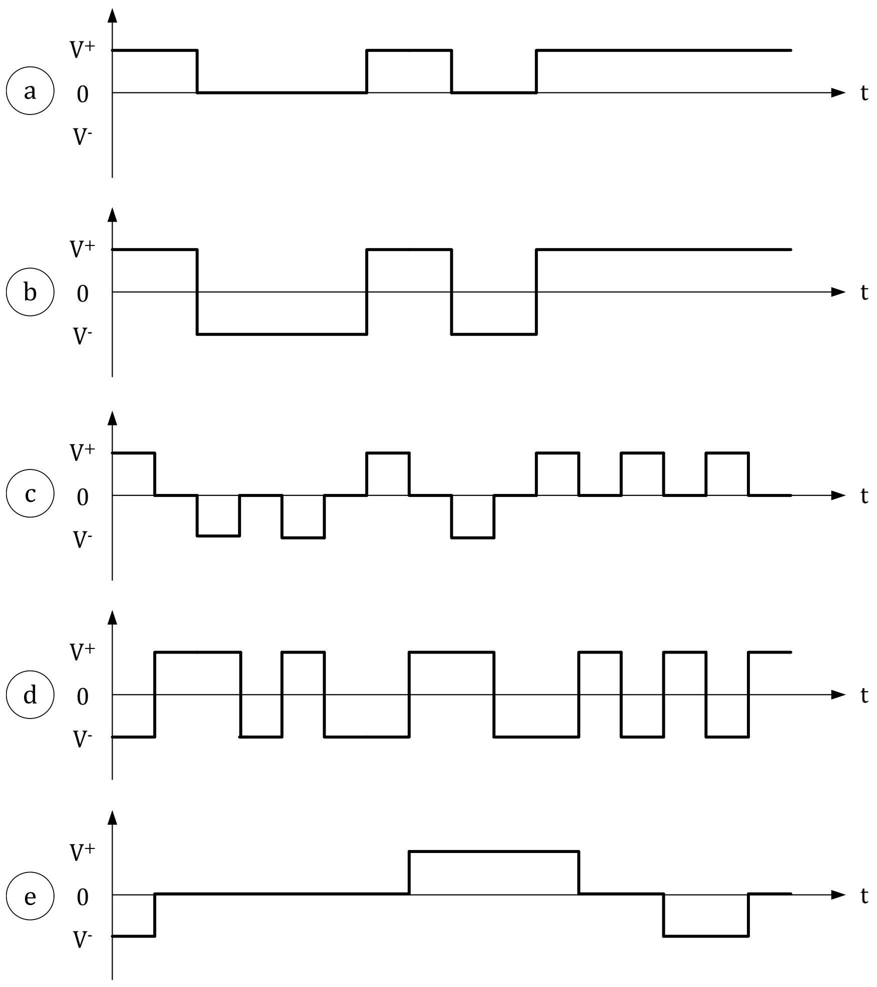


Figure 2.2: Line codes: a) unipolar, b) bipolar NRZ, c) bipolar, RZ, d) Manchester-code, e) MLT-3 code [6]

in the middle of every bit of the signal to level 0 (Return-to-zero (RZ)), like in Figure 2.2 c), to solve this problem. The information about the transmit clock is then contained entirely in the signal.

- A signal with zero-mean and complete clock information can also be generated by using the Manchester-code (see Figure 2.2 d). A binary 1 has a rising edge in the in the middle of a bit (from V^- to V^+) and a binary 0 has a falling edge in the middle of a bit (from V^+ to V^-). In order to preserve the possibility for these two different transitions for every bit, an additional edge has to be inserted on the bit border of two consecutive bits of the same binary value. Compared to the RZ-signal, the Manchester-encoded signal uses only two voltage levels.

It is worth mentioning that a binary 1 is not necessarily assigned to a positive voltage level or a rising edge. The voltage levels can be inverted and therefore the method of the encoding does not change.

Fieldbus systems mainly use the NRZ-encoding according to TIA/EIA-485 [2], but the Manchester-code is also used in, e.g., Foundation Fieldbus, PROFIBUS-PA and AS-i. Originally, Ethernet also used the Manchester-encoding, but it got disadvantageous with increasing data rates. The bandwidth of the transmission signal doubled because of the additional edges on the border between two bits and the edges in the middle of the bits.

Therefore, Fast Ethernet with a transmission rate of 100 Mbit/s uses the Multi-Level Transmit with 3 Levels (MLT-3)-encoding, which uses three different signal levels (see Figure 2.2 e). A binary 1 changes the signal level in the middle of a bit from V^- to 0, then from 0 to V^+, again back to 0 and so on. A binary 0 does not change the signal level. The transmission bandwidth of the signal using MLT-3 is much smaller than using the Manchester-code, which can be seen by comparing Figure 2.2 e) to Figure 2.2 d). However, the complete information about the clock is lost. Especially, with several consecutive zeros it is not possible for the receiver to synchronize its clock. Hence, Fast Ethernet uses an upstream change of the encoding. Every bit which has to be transmitted is split up into two 4-bit-nibbles. Every 4-bit-nibble is assigned to a five bit long code according to the 4B5B-encoding (see Table 2.3). The 5-bit-code is especially encoded, so that there are no three consecutive zeros. To transmit five instead of four bits per nibble increases the required bandwidth, but not as much as the Manchester-Encoding.

4-bit-nibble	**5-bit-code**	**4-bit-nibble**	**5-bit-code**
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

Table 2.3: 4B5B-encoding [6]

Asynchronous and Synchronous Protocols

Another distinctive feature of fieldbusses is the synchronous and asynchronous transmission. Typically, the characters are encoded in UART-format (see Figure 2.3). Therefore, the character being transmitted is framed with a start bit, e.g. 0, an optional parity bit and one or two stop bits, e.g. with the value 1. The duration of a single bit is defined through the clock generator in the transmitter by sending the next bit with the next clock pulse. The receiver samples the incoming bit pattern with an asynchronous clock signal which runs at a multiple of the data rate, like for example 16 times the bit rate. If the receiver is waiting for the next bit, it samples the edge of the start bit and recognizes that a new character is received. The number of bits per character and

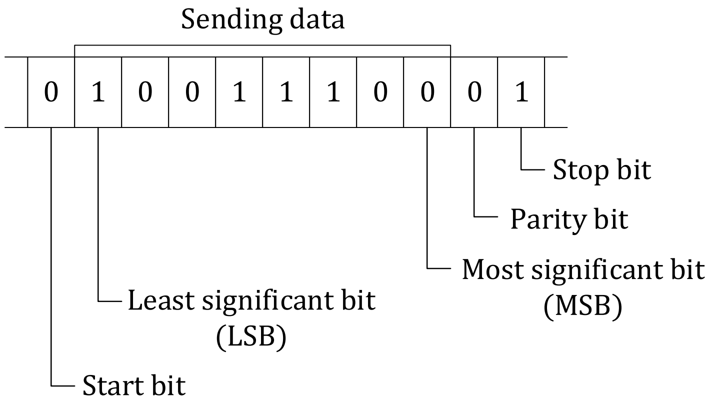


Figure 2.3: UART bits for sending a byte (e.g. 0x39) [6]

the bit rate must be configured equally for the transmitter and the receiver. Hence, the receiver knows how many bits it has to expect and can determine the value of the bits by sampling.

The asynchronous transmission is used for low data rates and for sending a few bits, but it is also possible to send longer telegrams via a UART. It is shown in Figure 2.3, that every byte of the to transmitting data is sent with a frame of a start bit, a parity bit and a stop bit. The start bit of the next byte follows the stop bit of the previous bit immediately without any break. For eight bits there are three additional bits to send and hence, the throughput efficiency is reduced. It takes 38 % [6] longer to send the UART character in comparison to a byte. This time is lost for the transmission of other data. For that reason, it is helpful for longer telegrams to use a synchronous transmission.

Characteristic for a synchronous transmission is that the transmitter and the receiver use the same clock frequency and that with every telegram a clock synchronization takes place at the receiver for correct sampling. This synchronization has problems with parts of the telegram which have several bits of the same value, because the transmission with a NRZ-signal does not change the signal level (no rising or falling edges) and therefore, is not possible to recognize the clock frequency in the signal. Especially, at the beginning of a telegram, when the receiver has to synchronize to the clock of the transmitter, it is necessary to send a special start identifier which, for example, consisting of an alternating bit pattern (e.g. for Ethernet 01010101, 0x55 [1]). The clock information is contained entirely in such a pattern, such that the receiver and the transmitter can synchronize. If the telegram is not too long, the receiver can keep the clock synchronous until the end of the telegram. It is possible, that the transmitter has to send a stuffed bit with a complementary value after a maximum acceptable number of bits with the same value. This method is used, e.g. in CAN, for re-synchronization within a telegram.

The Manchester-encoding is especially suitable for a synchronous transmission, because there is at least one edge per bit, such that the clock information is available during the entire telegram.

TIA/EIA-485

Fieldbusses in the field of NCS use often the standard TIA/EIA-485. This norm is designed for digital multipoint systems (multidrop systems).

For a configuration without a repeater, there are following guidelines [2]:

- A linear bus is used as network topology, where both wires are connected with resistors to power and ground having a defined voltage level on the bus in the idle state. Proper cable termination is essential for highly reliable applications with reduced reflections in the transmission to achieve higher data rates and longer cables. [2]
- Shielded or unshielded twisted two-wire cable with a minimal signal attenuation are used as transmission medium.
- A binary 0 is a differential voltage between conductor A (V^+) and conductor B (V^-) in the range of $1.5V <= V^{AB} <= 5V$ and a binary 1 is in the range of $-5V <= V^{AB} <= -1.5V$ (see Figure 2.2 b, only negated).

  It must be guaranteed that there is a steep enough edge at the transition from one voltage level to another. The time interval for a transition must be less than 30 % of the time which is available for the transmission of a bit.
- The maximal number of participants is 32. Every participant on the bus is counted. If a participant has a higher or lower load for the bus as the norm load, than this number can vary.
- The maximum data rate is 12 Mbit/s, but only on very short distances (up to a few meters). Usually the data rates are in the range of hundred kbit/s.
- The longer the cable, the higher is the attenuation of the signal. Therefore, the cable length is limited in general. Moreover, the signal attenuation is frequency dependent and also depends on the bale type, in particular on the cross-section of the wires. Thicker cables have smaller attenuation. Hence, the used standard TIA/EIA-422 defines the permitted cable length in dependency of the data rate [2]. For PROFIBUS, the following pairs of data rates and cable length are given:
  - for 9.6 or 19.2 or 93.75 kbit/s until 1200m,
  - for 500 kbit/s until 400m,
  - for 1.5 Mbit/s until 200m,
  - for 12 Mbit/s until 100m.

An increase of the cable length and/or number of the participants is possible with active hubs or repeaters, but since only two wires are used, the signal flow direction has to be changed dynamically. The number of participants is limited by the medium access procedure or by the maximum address range of the telegram. Another important usage of repeaters is the extension of the line structure in a tree structure.

Multiple Access Schemes

In general, every participant connected to a shared medium, e.g., the line of a bus structure, receives every packet. As long as the permissible number of participants and therefore, the permissible electrical load for the sender is not exceeded, the receivers are not disturbed in any way. The time at which a participant can access the bus and is able to send its telegram must be defined. If two participants send simultaneously, then the telegrams overlap in such a way, that both telegrams are destroyed.

The MAC from layer 2a in the ISO-OSI reference model is used to solve this problem. Many procedures have been developed to control the bus access. Some, which are mainly used in fieldbus systems, are discussed here: Master-Slave procedure, Token-Passing procedure, Carrier Sense Multiple Access with Collision Detect (CSMA/CD) and Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA).

The multiple access scheme characterizes the bus system decisively. It influences how complex the bus interface has to be, which data rates and which cable lengths can be used. The access scheme also decides the real-time capability of a fieldbus.

The multiple access schemes of the described fieldbus systems are shown in Table 2.4. There are two different types of handling the medium access: First, the access protocol can detect or avoid data packet collisions (*access according to demand*), if an access method, which is based on packet mode contention, is used. A second way is to reserve resources, e.g., time slots or frequencies, to establish a logical channel (*access according to assignment*), if a circuit-switched or channelization-based access method is used.

Bus system	**Access according to**	**Access scheme**
Ethernet	Demand	CSMA/CD
PROFIBUS	Assignment	Token-passing combined with Master-Slave protocol
Foundation-Fieldbus	Assignment	Arbitrator-producer-consumer procedure
CAN	Demand	CSMA/CA
LON	Demand	CSMA/CD, modified
Modbus-RTU	Assignment	Master-Slave protocol
Interbus	Assignment	Summation frame protocol
AS-i	Assignment	Master-Slave protocol

Table 2.4: Fieldbusses according their access schemes [6]

In systems using the *access according to assignment*, there are participants, which have a higher ranking and control the access to the bus of other participants (usually, only for a single telegram). Most of the time, the bus access takes places periodically, e.g. every participant receives the permission to send telegrams after a cycle. Hence, the bus access is almost deterministic, because it is defined at the time of configuration of the bus, in which order the participants are allowed to send.

The approach of the *access scheme according to demand* is quite different. All participants have equal rights. If a participant wants to send a telegram, it waits until the medium is free and then sends its telegram via the bus. Hence, they always access the communication channel, when they have a demand. This demand can happen from two participants at the same time. Then a collision of these two telegrams occurs. Therefore, the participants have to retransmit their telegram again. However, it is not guaranteed, that the retransmitted telegram does not collide with another telegram again.

It can be seen, that a participant, which has to send a telegram urgently, has to wait until it is its turn again. This waiting time is at worst a cycle time. Every participant can send immediately at the access scheme according to demand, but it is not guaranteed, that the telegram is transferred without a collision to the receiver. The probability, that a participant can send its telegram after a very short waiting time, is (at least at) a normal bus utilization very high. The probability for longer waiting periods is inversely proportional to the length of the waiting time, e.g., a correct transfer is less likely for longer waiting times.

In summary, there is less utilization by means of access according to assignment, but it is safer. The risk is higher when using access schemes according to demand, but there are shorter waiting times. The worst case is a very high bus utilization, because the probability for a lot of collisions is high.

Master-Slave procedure

A single participant called master controls the access to the bus of all other participants called slaves at the Master-Slave procedure. Therefore, the master can send a telegram on its own or it can pass the permission to send data to another participant.

The communication is based on a command-response scheme. [6] Either the master sends a command data to a slave and a slave responses the correct reception for the data or the master requests data from the slave using a command. Additionally, it is possible, that data are transferred from a master to a slave and back from the slave to the master in one command response cycle, which is especially useful at remote input/output operations.

Usually, every slave is addressed strictly periodically. It is possible, that data are exchanged with one slave once within a time frame, whereas another slave is addressed several times in the same time frame. Hence, different time intervals can be realized by this method and this can be used, for example, to use different sampling times for getting the control variables (letting the relating slave send) and sending the manipulated variables (from the master to the related slave) of a controller. This approach increases the transmission efficiency.

Advantages of the Master-Slave procedure

- The control algorithm is implemented in the master and therefore, the bus interface of the slaves is very simple.
- Data has a maximum time period within it is sent, because every participant has a fixed timeslot to access the bus. The requirement for real-time capability is partly fulfilled.

Disadvantages of the Master-Slave procedure

- If the master fails due to an error, the whole bus system breaks down.

- The transmission between slaves to be handled over the master is time consuming.
- The cycle time is getting longer with an increasing number of participants.

Token-Passing procedure

Whereas only one master exists in a bus system at the Master-Slave procedure, all participants are eligible to control the bus access at the Token-Passing procedure. Therefore, the bus access authorization is passed around, in such a way that only one participant is master at a certain period of time. A so called token, which is a short telegram, is used to coordinate this procedure. It is sent around all participants. If a station holds the token, it can access the bus and send its telegrams.

If the token holder has nothing to send, it passes the token to the next participant. Otherwise, it sends one or multiple telegrams in a row. The token holder is not allowed to keep the token for an arbitrary time period. There is a maximum Token Hold Time (TTH) to guarantee, that a participant gets the sending authorization for a predefined and configured time period.

The order of participants getting the token is defined at installation time. Hence, every participant knows to whom it has to deliver the token next. The token is passed in virtual circles. Typically, it is just a logical ring and the stations do not have to be arranged in a physical ring structure.

There must be a special master at the Token-Passing procedure, which has specific controlling tasks, like for example, after the start of the bus system, a token has to be created and if after a failure a token was lost, a new one has to be generated.

The Token-Passing procedure can be a multi-master procedure, but it is similar to the Master-Slave procedure from the point of view of the user, because the slaves are addressed in cyclic order from the master at the Master-Slave procedure and the stations get the token in cyclic order.

Advantages of the Token-Passing procedure

- Participants can directly communicate with each other without including a master device.
- If a participant does not have anything to send, it can pass the token such that the token slot is used by another participant.
- A master can monitor the neighbor. If an error occurs at the neighbor, the faulty station is excluded from the possession of the token and the remaining stations can use the bus further.
- The real-time capability is fulfilled, because a maximum token rotation time is guaranteed.

Disadvantages of the Token-Passing procedure

- The management of the bus is quite complicated and every participant needs a comparatively expensive bus interface.

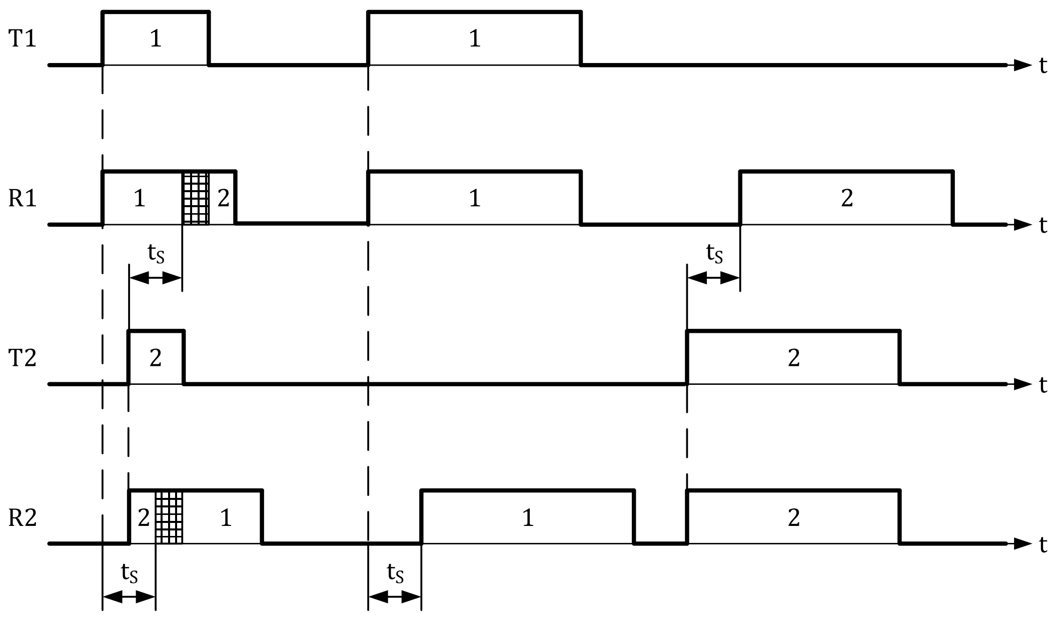


Figure 2.4: Sending and receiving of single bits using CSMA/CD [6]

- The time, which a station has to wait until it can send again, gets longer with an increasing number of participants.

CSMA

The used methods of the described fieldbus systems for the procedures according to demand are all based on Carrier Sense Multiple Access (CSMA) procedures. All bus stations can take over the function of a master. If a station wants to send, it listens to the shared medium. If the bus is free, i.e., no other station is sending, the station can start to send its telegrams.

Hence, there is a serious problem. In a spatial extended arrangement, the signal propagation delay cannot be neglected. For example, a cable length $l = 100m$ results already in a signal propagation delay $t_s = 0.5\mu s$.

If two stations waited until a third one has finished to send its telegram and then try to access the bus simultaneously to send their telegrams, they will detect the collision at the earliest after the signal propagation delay t_s. Every station recognizes, that its telegram is manipulated, because everyone monitors the bus on its own (listen while talk principle).

In Figure 2.4, it is shown for station 1 and station 2, how telegrams are sent on a shared medium (transmitter T1 and T2) and what they receive while monitoring the bus (receiver R1 and R2). A collision occurs in the left part of the picture, because both transmitters start to send telegrams within a short time period. Postponed by the signal propagation delay t_s, there is a superposition of the telegrams at the receivers of both stations (crosshatched in Figure 2.4).

There are several ways, how the sending stations can react to such collisions [15].

CSMA/CD

By using CSMA/CD, collisions between telegrams will be detected as soon as possible. Both transmitters abort their telegrams and release the bus (see left part of Figure 2.4). They try to retransmit their telegrams after a random time (see right part of Figure 2.4), named backoff period.

For example, this procedure is used by IEEE 802.3/Ethernet [1] in the area of Local Area Network (LAN). The telegrams must not be arbitrary long to guarantee, that the procedure works in larger networks and therefore, with long cable lengths and with long signal propagation delays. Hence, the minimum telegram length in IEEE 802.3/Ethernet is defined by 64 bytes.

Advantages of CSMA/CD

- For the use in the area of LAN, it is important, that plenty of participants can be attached to the network, because the transmission takes place on demand and not in a cyclic manner.
- A new participant can be added or removed without a reconfiguration of the bus.

Disadvantages of CSMA/CD

- The transmission efficiency decreases in high load situations, because a lot of telegrams may be aborted. It must be remembered, that in the time while sending a telegram, which will be aborted, there is no utilizable transmission on the bus. The more telegrams get aborted, the less is the number of successful transmitted ones.
- If a telegram is aborted, it takes a random time until a retransmit is started. It can happen, that this retransmission also is aborted and then again a random time is waited. Therefore, a transmission in time is not guaranteed and this procedure is not real-time capable.

CSMA/CA

CSMA/CA is used to improve the performance of the CSMA method by preventing collisions instead of just detecting them. Prior to transmitting, a station first listens to the shared medium to check if it is free, like in all CSMA procedures [16].

A *Request to Send* or *Clear to Send* may optionally be used at this point to signalize the start of a transmission. If a shared medium was identified as being idle or the station received a *Clear to Send* to indicate it can send, it starts the transmission of the message. Afterwards, the station waits for the reception of an acknowledgement to indicate the message was received correctly and the checksum was accurate. If such an acknowledgement does not arrive after a defined time, it is assumed that the message collided with some other transmission, which forces the node into a period of binary exponential back-off before starting a restransmit. [16]

For example, this procedure is used by CAN [17] which modifies TIA/EIA-485 in such a way, that a binary 0 is dominant, e.g., if a binary 0 from one transmitter and a binary 1 from another transmitter collide on the bus, the dominant binary 0 enforces the recessive binary 1 and the bus is set to 0. [18] *Request to Send* and *Clear to Send* are not used.

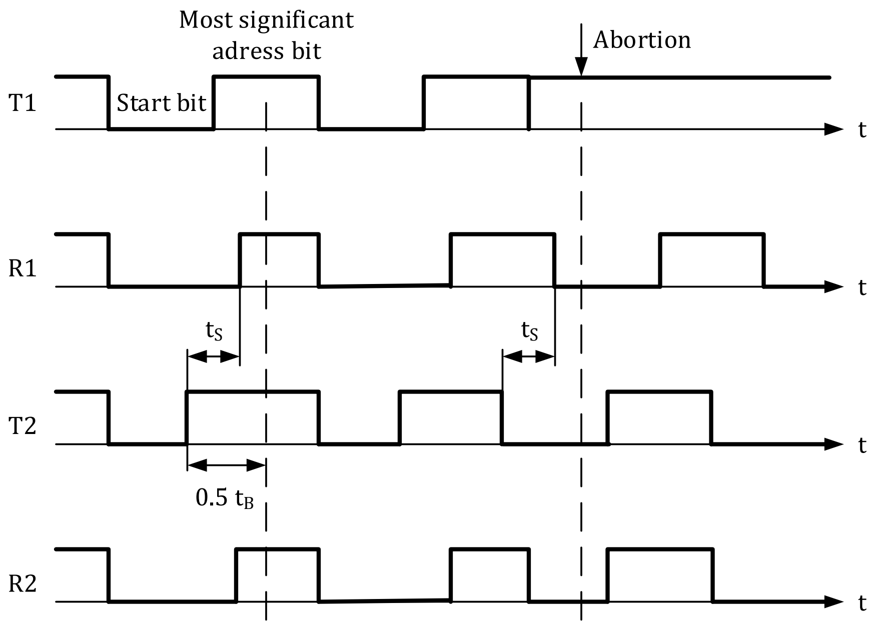


Figure 2.5: Sending and receiving of single bits using CSMA/CA [6]

Hence, if two stations detect simultaneously, that the bus is idle, they put one bit of the telegram after another on the bus. At first, every station sends a dominant start bit (see Figure 2.5, transmitter T1 and T2). Afterwards an identifier follows. If a binary 0 and a binary 1 collide within the address bits, which is sent with the most significant bit first, the dominant binary 0 enforces on the bus. Station 1 sends the identifier 1011... and station 2 sends the identifier 1010... in Figure 2.5, hence, station 2 wins the arbitration. Because both stations monitor simultaneously the bus as well (receiver R1 and R2), station 1 recognizes, that its binary 1 was changed to a binary 0 and aborts its transmission. The other transmitter, whose telegram was not changed, continues the transmission undisturbed. After the end of the transmission, the outnumbered station can retry to send its telegram right away. [19].

Therefore, it can be seen, that a telegram is sent more reliable at the first try, if the identifier of a station is low. Hence, the address functions like a priority. The smaller the value of the address, the faster the telegram is sent in case of a conflict.

CAN, which uses this procedure, uses the identifiers to identify, like it is common at the Producer-Consumer-Model, not for a single station, but for single objects, i.e., a measured values for a temperature provided by a station. Priorities can be assigned to these objects depending on their importance like with the tasks of a real-time system.

Advantages of CSMA/CA

- Like in other CSMA procedures, a number of participants can be attached to the bus.

- A new participant can be added or removed without a reconfiguration of the bus.
- The requirement of the consistency of the output data is easily grantable, because all participants receive the same data simultaneously.
- In the case of a collision, a telegram always gets in. Hence, there is no down time, where no transmission happens. There is no loss of efficiency in high load situations.
- Because priorities can be assigned to single telegrams, the telegram with the highest priority is sent with minimal delay. The real-time capability is given in optimum manner.

Disadvantages of CSMA/CA

- CSMA/CD only works as long as single bits of any two competitive telegrams are put on the bus within a specified time. Every station sends a bit during the time t_B (see Figure 2.5). The stations analyze the receiving marks in the middle of a bit being sent by the transmitter. The signal propagation delay t_s must be smaller than $\frac{1}{2}t_B$:

  $$t_s = \frac{l}{v} \ll t_B = \frac{1}{r} \tag{2.1}$$

  with the length of the bus l, velocity of propagation v and the transmission rate r.

  From this relation it follows, that

  $$l * r \ll v. \tag{2.2}$$

  The product *cable length · transmission rate* must be smaller than a certain bound, because the velocity of propagation v is constant. Like for example, the maximum cable length of 25 m is proposed for r = 1 Mbit/s [6].

Data Protection

With a very low probability, it is possible, that a bit of the telegram can change its value during transmission. Moreover, it can happen, that several bits of a telegram change. The main reason of such interferences are electromagnetic fields, which cannot be avoided in technical facilities with, e.g., electrical drives, power electrics and transformers.

Voltage being induced through an electromagnetic field overlays the primary signal, which could have a very low voltage level, especially at wide spread networks and networks with many participants (see Figure 2.6). The input stage at the receiver differentiates between binary 0 and binary 1 with the help of threshold values. Therefore, it may happen, that the sum of signal and interference at the moment of signal sampling exceeds the threshold value (see Figure 2.6 e to 2.6 g) and the input stage at the receiver determines the wrong value for the incoming bit. Several bits can be affected at a longer ongoing interference.

Several countermeasures depending the transmission medium can be taken against this interferences. One option is to use a higher signal voltage level at least at none intrinsically safe

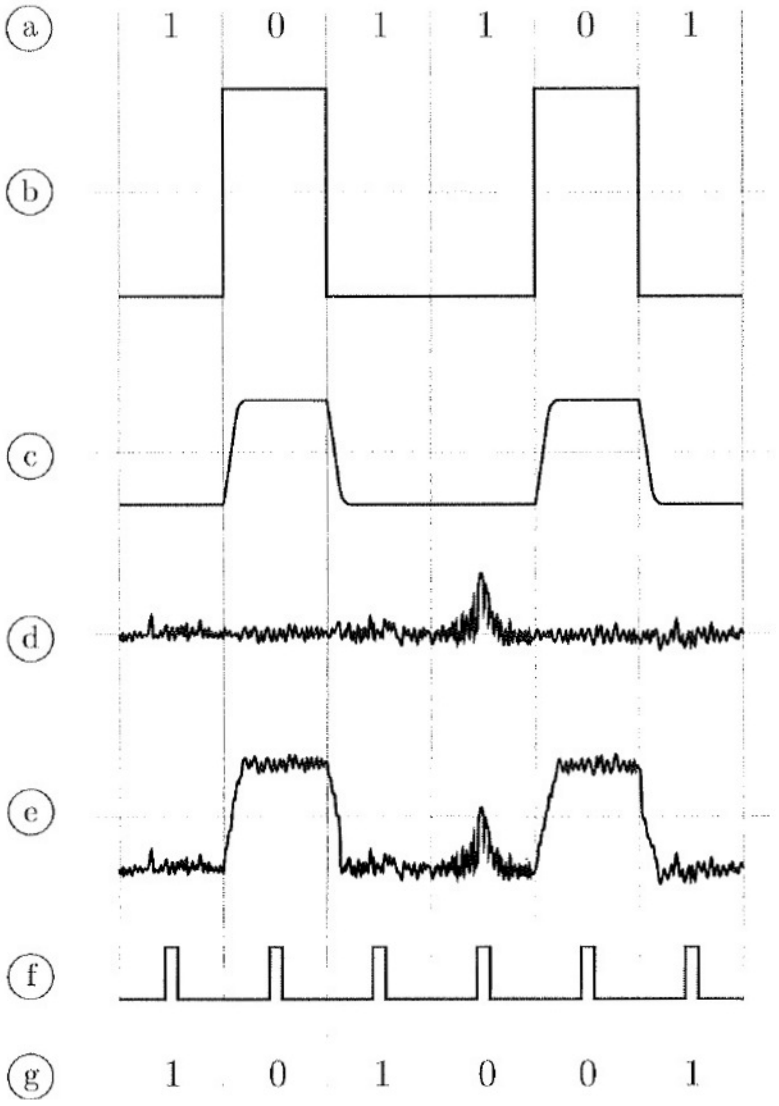


Figure 2.6: Development of a faulty bit: a) data to send, b) sending signal, c) damped signal, d) interference, e) damped and interferenced signal, f) signal sampling, g) received data. [6]

electrical systems. Additionally, shielded cables or even fiber optic cables can be used to reduce the probability of occurrence, but it is not possible to totally exclude the possibility of faults. For example, it is assumed, that one bit changes its value at the transmission of 10,000 bits in average while using shielded cables. [6, p. 89]

Transmission errors in fieldbusses can have bad consequences, like for example, a device is switched on instead of switched off. Hence, a product may be faulty, the plant may be damaged or a dangerous state for the operator may happen. To avoid these problems, counter measures must be taken.

If the change of a bit during transmission cannot be prevented, it must be tried to detect the error. This is done by the insertion of redundant check bits in the telegram of the transmitter. The receiver can determine if the transmission was successful using the check bits.

The more redundant bits are used for the verification, the higher is the transmission reliability. It is possible to add check bits to a telegram, such that a receiver can identify and even correct the errors. However, the transmission efficiency decreases with an increasing number of check bits. Hence, a trade-off between safety and efficiency has to be found.

In general, error correction needs considerably more bits than error detection. Hence, fieldbusses rely on the recognition of a correct or a faulty telegram. Typically, the receiver reports back the error with a negative acknowledgement and data need to be retransmitted.

If then data are also interfered, the procedure is repeated again until the information is received correctly or a maximum number of transmission attempts has been reached. It can happen, that the wire or the receiving participant has a permanent defect. The bus must not be jammed by the transmitter sending the same data again and again. Instead, the transmitter should notify, that a non-recoverable error occurred and the bus should be freed for other data of participants.

For error detection, fieldbusses use different methods (see Table 2.5): parity bits, checksum, message authentication code and CRC.

Bus system	**Data protection**	**Transmission**
Ethernet	CRC	synchronous
PROFIBUS-FMS	Parity bit + check sum	asynchronous
PROFIBUS-DP	Parity bit + check sum	asynchronous
PROFIBUS-PA	CRC	synchronous
Foundation-Fieldbus	CRC	synchronous
CAN	CRC	synchronous
LON	CRC	synchronous
Modbus-RTU	CRC	asynchronous
Interbus-S	CRC	asynchronous
ASi	Parity bit	asynchronous

Table 2.5: Used data protection [6]

The three data protection procedures have different efficiency, e.g., the gain of reliability per additional transmitted bit is not the same at all three procedures. Every procedure, which will be discussed, can certainly identify a single bit error of the telegram as faulty. It is crucial, if two, three or more errors coincide in a telegram. The most important point is the efficiency of a method, e.g., the maximal number of errors within a telegram, which can be detected to ensure the telegram is faulty or not. Hence, the term Hamming Distance (HD) comes into play:

The Hamming Distance, which is the minimum possible number of bit inversions that must be injected into a message to create an error that is undetectable by that message's data projection method. [20]

Parity bits

Parity bits are a data protection procedure being used for a very long time in asynchronous transmissions in the UART format (see Figure 2.3).

There are two variants of parity bits: even parity bit and odd parity bit.

In the case of even parity, the number of bits whose value is a binary 1 in a given byte are counted. If that total is odd, the value of the parity bit is set to binary 1, making the total count of binary 1's an even number. If the count of ones in a given set of bits is already even, the parity bit's value remains binary 0.

In the case of odd parity, the situation is reversed. Instead, if the sum of bits with a value of binary 1 is odd, the value of the parity bit is set to binary 0. And if the sum of bits with a value of binary 1 is even, the parity bit value is set to binary 1, making the total count of binary 1's an odd number.

If two bits in one UART character are changing their value simultaneously, it is possible, that the error is not recognized with the parity bit. In general, one has to bear in mind that check bits can also be distorted like the data bits.

The code of two characters is different in at least two bits. This is called, the hamming distance d is 2 in the field of coding theory. With the data protection procedure *parity bit* a code is produced with

$$d = 2.$$

In general, the amount e of reliable detectable errors in such a code is shown be the relation:

$$e = d - 1$$

The greater the hamming distance, the more errors can be detected in a telegram and the greater is the reliability, that a telegram is recognized as faulty. Therefore, the value

$$e = 1$$

is obtained for the data protection procedure *parity bit*, which satisfies the above considerations.

Checksum

Telegrams on a fieldbus can be protected with checksums as well. All data bytes and a part of the bytes from the frame are summed up with modulo-operation depending on the length of the checksum. The arising checksum, which is calculated in that way, is added after the sending data to the telegram by the transmitter. The receiver executes the same modulo-operation with the received data and checks, if the calculated and the received checksums are equal. If this is not the case, a transmission error occurred. This method is typically combined with the parity bit procedure.

If just one single error occurs in such a telegram within a byte, it is detected by the parity bit of this byte, independently of the telegram. Every odd number of errors will be detected safely by the parity bit. If two bits in one byte are changing during a transmission, the error is

- a) Sending data:

	Data bits	parity
Data byte 1	00111001	0
Data byte 2	11110100	1
Checksum	00101101	0

- b) Receiving Data:

	Data bits	parity
Data byte 1	$0011\underline{01}01$	0
Data byte 2	$1111\underline{10}00$	1
Checksum	00101101	0

Figure 2.7: Combination of parity bit and checksum at four bit errors. [6]

not recognized, but the checksum over the bytes of telegram is not valid anymore, assuming the remaining part of the telegram is not interfered.

Two errors within the telegram will be detected, no matter if they occur together in one byte or in two different bytes. Three errors will be detected, no matter how the errors are spread over the bytes, if there is at least in one byte an odd number of errors, which is shown by the assigned parity bit. If four errors occur, there are combinations which cannot be detected by the procedure with parity bit and checksum. An example is shown in Figure 2.7, where an 8-bit checksum with modulo-256-addition is computed by adding data bits with even parity. The remaining telegram should not be interfered and is without relevance.

In Figure 2.7 a), the correct values, which were sent by the transmitter, are shown. The received bits are shown in Figure 2.7 b), where the interfered bits are underlined. The four errors compensate at this error arrangement with two errors per byte and each with the same significance. Hence, neither the parity bit nor the checksum can detect the error.

There are many other arrangements with four interfered bits, where the distortion is detected, but all arrangements of errors have to be detected safely. Therefore, the number of recognized errors e is 3 at the combination of parity bit and checksum.

Additionally, this result has been reached by viewing the variation of the code like in the section *Parity bit*. The whole telegram is viewed as a pattern of the code. If all to protecting bits are binary 0, then also all parity bits and the bits of the checksum are 0. If this telegram is compared with another one, in which only a single bit is 1 within a data byte, then the two telegrams respectively code patterns are different in four bits: the bit in the data byte, the parity bit of this data byte, the bit of the corresponding significance of the checksum and the parity bit of the checksum. The hamming distance is

$$d = 4$$

and the number of safely detectable errors

$$e = 3,$$

which is the same as in the above considerations.

Cyclic Redundancy Check

CRC is used quite often (see Table 2.5), because it is the most efficient data protection procedure, but also the most complicated one. Hence, just a quick glance is given here.

All parts of the telegram are interpreted as a binary number T to determine the check bits. The produced n check bits are inserted after the k bits of the binary value T. Therefore, n bits with the value 0 are attached to the binary number T, which is the same as a multiplication of T with 2^n.

The in such a way modified binary number $T \cdot 2^n$ is divided by a so called generator polynomial G, which is given for a bus system. The subtractions, which are done at the multiplication, are performed modulo-2, thus without carry.

The division of the modified binary number follows the relation

$$\frac{T \cdot 2^n}{G} = Q + \frac{R}{G} \tag{2.3}$$

with the result of the division Q and the remainder R. The result Q is not used any longer and the remainder R is attached to the telegram as a check sequence, which is usually called Frame Check Sequence (FCS). If n check bits have to be produced, the generator polynomial has to be n+1 bits long. The binary number R contains maximal n bits without the trailing zeros because of the modulo-2-subtraction.

If a generator polynomial is used, which has a binary one as highest and lowest priority bit and is at least three bits long, then it can be shown in general [6, p. 99], that the occurrence of the following errors are detected in a telegram:

- a single error,
- two errors at the usual telegram length,
- an odd number of errors,
- an accumulation of errors within an otherwise error free telegram, where the number of bits from the first until the last error bit is smaller than the number of check bits n. It is possible, that two or even more of such accumulations with a dimension smaller than n appear in a telegram. There must be error free parts of at least n bits between the accumulations of errors,
- error groups with a length greater than n, except the error arrangements, which can be divided by G without a remainder.

The proofs of these claims can be found in appropriate literature (e.g. [20] and [21]).

There are more or less suitable generator polynomials. The usual representation of these polynomials marks the position, being a binary one in the binary representation, as a power of x. Koopman [20] gives an overview on generator polynomials and compares them with each other.

For example, Interbus-S uses a 16-bit generator polynomial, which is recommended by the Comité Consultatif International Téléphonique et Télégraphique (CCITT)

$$G = x^{16} + x^{12} + x^5 + 1 \equiv 0b1000100000010000 = 0x8810 \tag{2.4}$$

with x^{16} as the highest bit and an implicit +1 term. PROFIBUS-PA and LON also use this polynomial.

Otherwise, CAN uses the 15-bit generator polynomial

$$G = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1 \equiv 0b0110001011001100 = 0x62CC. \tag{2.5}$$

The number of detectable errors and error accumulations can be found by using the above considerations for $n = 15$. To be able to compare the different test methods in regard of efficiency, a claim about the safely detectable errors or the hamming distance is needed. CAN can detect

$$e = 5$$

errors safely and therefore, the hamming distance is

$$d = 6.$$

Hence, these values are higher than the values from the parity bit procedure and the check sum procedure, which confirms the higher efficiency of CRC.

The claim for e and d can be determined by a simulation at the about 100 bits of the CAN telegram. Like it is shown in the previous section, this claim can be found by running through the error patterns or by looking at the variation of the code. [20]

The CRC mechanism seems to be complicated, but it can be used in hardware and software easily, which contributes with the high error detection rate to the wide spread of this procedure.

Computation of a CRC resembles long division of a binary message string with a fixed number of zeroes appended by the "generator polynomial" string except that exclusive OR operations replace subtractions. Divisions of this type are efficiently realized in hardware by a modified shift register and in software by a series of equivalent algorithms, starting with simple code close to the mathematics and becoming faster through byte-wise parallelism and space-time tradeoffs.

Communication models

Application layer models

The different functional levels in NCS have different needs in communication. Most of the NCS communication systems are able to use different kinds, which depend on the needed functionality. In the following paragraph, three communication models [22], which will be described, are summarized in Table 2.6.

	Client-server	Publisher-subscriber	Producer-consumer
Transfer method	dialog, connection oriented	multi-cast, connectionless	broadcast, connectionless
Physical structure	point-to-point	multipoint	multipoint
Logical structure	master-slave, peer-to-peer	peer-to-peer	peer-to-peer
Application	request, set value, parametrize	change of value/state	alarm/event message, change of value/state

Table 2.6: Overview of the communication models [22]

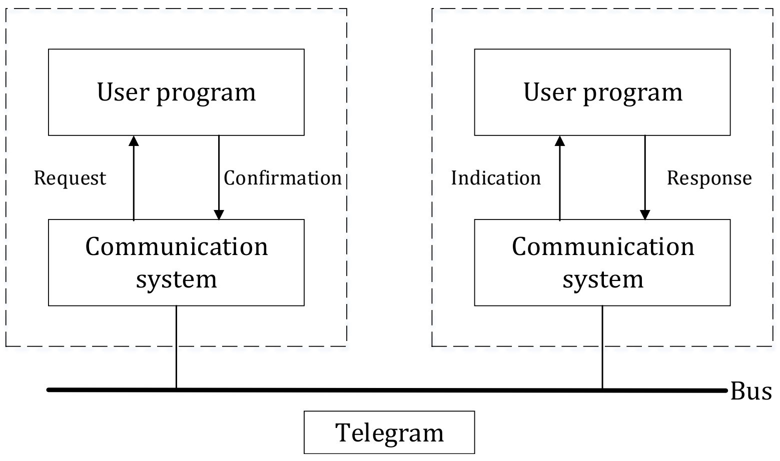


Figure 2.8: Communication example between two participants [6]

The client-server model is well-known in Information Technology (IT). In a Distributed Control System (DCS) there is always a lot of data transfer between the participants of a fieldbus, where, for example, Field Control Station (FCST) requests measured data or state information from sensors and receive the desired data as a response. The FCST also sends the output data to actuators and receive state information from the actuators. Finally, an FCST can communicate with another FCST via the fieldbus and the data transfer is in general bidirectional.

All these concrete transmissions can be represented like in Figure 2.8, where a user program runs in participant A, which produces data for participant B or otherwise requests data of participant B. [23]

The user program transfers output data requests and input data requests with a *Request* to the communication system. Both is packed in a telegram and sent via the fieldbus to the communication system in participant B. The data and the request sent from participant A are hand over to the user program of participant B as part of the *Indication*. With a *Response* the user program passes the desired data to the communication system in participant B. The data is again packed in a telegram and transferred to communication system of participant A. Finally, the user program in A receives the request data with a *Confirmation*.

An important point in this view is the separation of user program and communication system.

The communication system connects the participants via a medium and allows an easy data transmission between them. The user program deals with the specific tasks of the participant: This task in a sensor consists in determining the measured value and eventually add data, like for example, the degree of pollution. The user program in a FCST is almost equivalent with a program for open and closed-loop control of a process and in case of a display or operator component it provides the process state and responds to inputs of the operator.

The distinctive separation of the user program and the communication system is a expression of the goal, that the user program is completely independent of the realization of the transmission via the fieldbus. The user program uses the communication system with the help of the above described services.

A disadvantage of the client-server model is the overhead, which is created through the request-response mechanism. Overhead is always a problem in fieldbus systems, because the data rate is very limited.

In the publisher-subscriber model, the subscriber can enroll for a service at a publisher under a specific event. If this event occurs, the publisher sends the information to all subscribers, which have subscribed for this event. The subscription is valid until the subscriber cancels it or a certain time period is reached. The publisher informs all subscribers at the same time with a single communication which reduces the network traffic significantly, but there is additional administration overhead for the control of the subscription. This can be a disadvantage in field devices, because of the restricted resources.

An alternative approach in terms of hardware solution is the producer-consumer model. The producer of the information distributes the data over the network as a multicast or broadcast and all consumers which are interested in this information receive it. The reception of the data depends on the client, which can choose if it wants to receive the data. The relation between the information type, the producer and the consumer is established during the initialization phase. There is no additional administration effort during the operation regarding addresses, registration of devices and so on.

Acknowledged and Non-Acknowledged Services

There are services using an acknowledgement to confirm the correct reception of a message of participant A to participant B. Because of efficiency reasons the proof of proper transmission is done by the communication system of participant B without involvement of the user program. Hence, the service *Response* is not necessary. If an error is detected, the communication system of participant A starts automatically a retransmit of the data until the transmission was successful or a specific number of retransmits has been reached. Only if the maximum number of retransmits was reached, the user program of participant A has to decide what happens further.

If a non-acknowledged service is used, the efficiency of the data transmission is higher, because the response telegram has not to be sent. But the safety of the system is worse, because the transmitter does not know, if the data was received successfully.

Duplex modes

A duplex communication system is a point-to-point system, which is composed of two connected participants that can communicate with each other in both directions. It has two clearly defined paths. One path carries information from A to B and the other one from B to A. There are two types of duplex communication systems, full-duplex and half-duplex. [23]

In a full duplex system, both participants can communicate to each other simultaneously, like for example, in telephones and in the Internet. Two participant can send and receive data at the same time, because there are two communication channels between them.

In a half-duplex system, there are also two participants with defined paths and each participants can communicate to the other but not at the same time, like for example, in a walkie-talkie two-way radio. If a user wants to speak, he has to push a button and the transmitter is turned on, whereas the receiver is turned off. Hence, other users cannot be heard. The button has to be released, which turns on the receiver but turns off the transmitter, so that the users can listen.

2.3 IEEE 802.3/Ethernet

Ethernet [1] was developed in the mid 70's by the company Xerox, first standardized in 1983 as IEEE 802.3 and is the dominating wired LAN technology replacing other protocols, like for example, token ring, Fiber Distributed Data Interface (FDDI) and Attached Resource Computer NETwork (ARCNET). Until today, it has been refined to support higher bit rates and longer link distances. The data rates reaches from the original 10 Mbit/s up to 100 Gbit/s [1].

Issues of the access scheme CSMA/CD

A problem of Ethernet is the medium access scheme. It uses the CSMA/CD procedure, which is described in Section 2.2 and has two big disadvantages:

- Two participants can start to send after they checked if the bus is free, such that the telegram can collide and be destroyed. If the two participants restart their telegram transmission after a random time, their telegrams can collide with another participant again. Therefore, no upper bound, which guarantees the maximum transmission time, exists and it is not real-time capable.

  Hence, extra measures should be done to avoid collisions.

- If a participant sends a short telegram, it can happen, that the station is done with sending the telegram before a telegram of another participant, which sends in parallel, arrives. Therefore, the collision is not be recognized while at another point of the network the two telegrams collide. The length of an Ethernet telegram is at least 64 bytes long without the preamble and the start identifier.

Physical medium

For the physical medium, there are many different alternatives. Originally, coaxial cables have been used, where the participants were wired in a line or tree structure. The problem with these

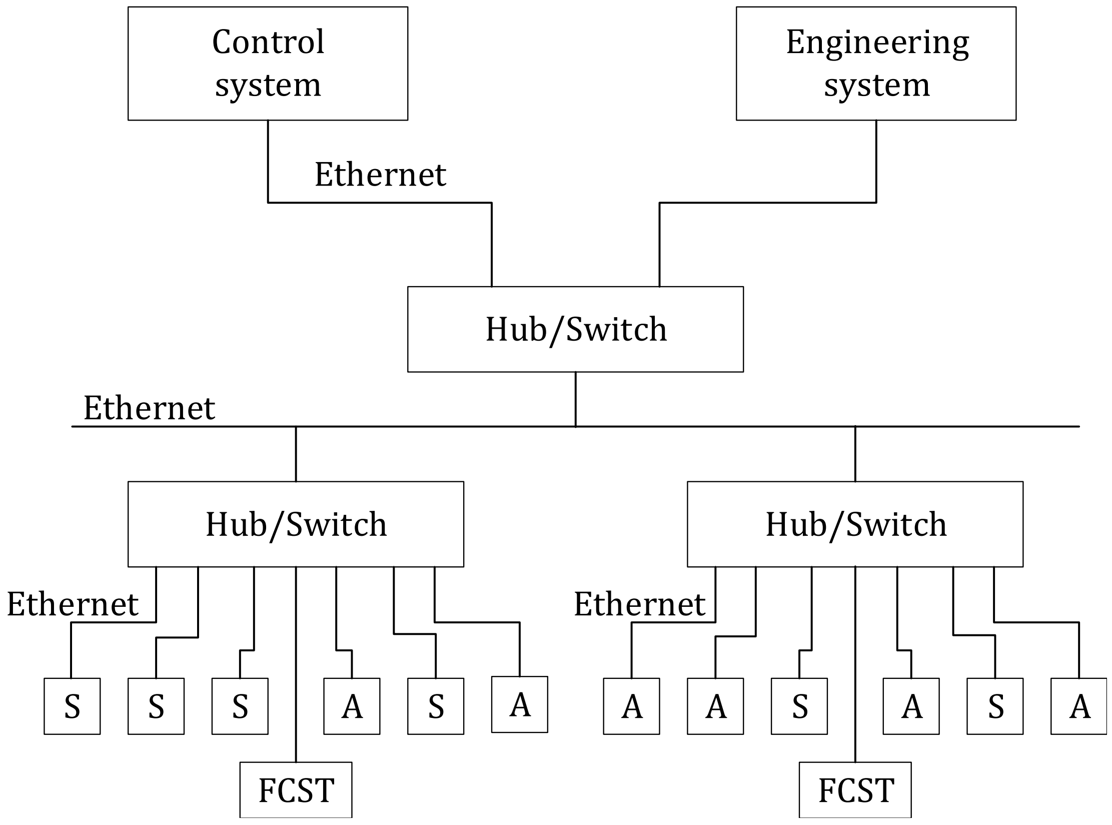


Figure 2.9: Star structure with hubs/switches

structures was, that through the increasing amount of data, the amount of collisions increased and hence the transmission efficiency decreased dramatically. Therefore, most of the line structures were replaced by the star topology. Figure 2.9 illustrates, how such a bus topology can look like in the automation and field level.

Every participant is connected to a hub with an individual line in such a star structure. The hub is again connected to the rest of the network. The connections of the hubs is called backbone. If a switch, which can buffer the telegrams and forwards them to just one participant, is used instead of a hub, a collision between two participants, which are connected to the same switch, cannot occur in a configuration shown in Figure 2.9. The two participants, which exchange data, have their own lines. Usually, the connections to the switch uses at least a four-wire cable, so that each participant has two wires for each transmission direction. Hence, a full-duplex mode is possible.

Cables in different types with several cable pairs, i.e., twisted two-wire cables with four pairs, are in use for the connections with the switch. The cables are classified in different cate-

Pre	SD	DA	SA	Tag (optional)	Len	Data	Pad	CRC

Pre	Preamble	7 bytes	0x55
SD	Start of frame delimiter	1 byte	0x5D
DA	Destination address	6 bytes	0xB827EB14A1F2
SA	Source address	6 bytes	0xB827EB874076
Tag	optional, only at Tagged-MAC-frame	4 bytes	
Len	Length of the data field or Ethernet type	2 bytes	0x0800
Data	Payload	0 to 1500 bytes	0x45....
Pad	Pad field	0 to 46 bytes	
CRC	Frame check sequence, 32-bit CRC	4 bytes	0x5E7D

Figure 2.10: Telegram of Ethernet

gories according to the maximum allowed transmission rate. Cables of category 5, abbreviated CAT-5 [24], or higher have to be used for Fast Ethernet. Fiber optical cables are mostly used for long-distance ranges. Wireless transmission (Wireless Local Area Network (WLAN)) is standardized by IEEE 802.11.

Besides the change of the wiring, also the encoding of the signals changed. While previously the Manchester code was used, Fast Ethernet uses MLT-3 and 4B5B encoding (see Section 2.2).

The switches have to be especially powerful devices and hence, are quite expensive. Additionally, the connectors and cables used in the office area are not suited for rough industrial environment and hence, special types in form of Industrial Ethernet are used. [25]

Telegram Structure

The telegram structure of Ethernet starts with a preamble consisting of seven bytes (see Figure 2.10). The start identifier follows the preamble. The preamble and the start identifier are actually not necessary in Fast Ethernet, because of the star topology and the use of switches the lines are always in steady state, but both fields retain due to compatibility reasons.

The source and destination addresses are 48-bit long numbers and are called Ethernet addresses, MAC addresses or physical addresses. Each Ethernet interface card receives a worldwide unique address at the manufacturing process. The two most significant bits are reserved for special functions. If, for example, the destination address starts with a binary 0, then a single participant is addressed. The telegram is sent to a group with a binary 1 at the first bit, whereas every participant has to know to which group it belongs. The highest address (0xFFFFFF) is reserved for broadcasts, the associated message is sent to all participants in the communication system. The second bit decides, whether the consecutive address is assigned according to global rules (bit = 0) or it is a local address (bit = 1). A local address can be chosen completely free. One has only to take care, that it is unique within a closed area. If a global address is used indicated with a binary 0 at the second bit, then the remaining bits of the address are split up in a field with the identification of the manufacturer and in a 24-bit field with the identification of the network card.

The length field in the telegram indicates the size of the payload in bytes, where the value is in the range of 0 to 1500. If the value is higher than 1500, then this field indicates the protocol type of the above layers. For example, the hexadecimal value 0x0800 is used for the Internet protocol IP. The length of the payload is encoded in the payload within the control information of IP. The so called Tagged-MAC-frame uses the hexadecimal value 0x8100 and signals that additional 4 bytes are used in the Ethernet frame. The protocol type is shifted to the last two bytes of the four bytes of the Tag field. The first two bytes contains among other things a priority from 0 to 7, which is used, for example, at PROFINET. [26]

If the payload contains less than 46 bytes, then a so called pad field is used to add so many bytes, that the payload and the pad field are 46 bytes long. The pad field ensures a minimum length of the telegram to identify collisions reliably.

The frame check sequence contains a 32-bit CRC to detect corrupted data (compare Section 2.2) at the end of the Ethernet telegram.

2.4 Related Literature

Carrier Sense Multiple Access

A number of papers in the field of wireless communication are discussed in this section, because wireless communication is similar to a half-duplex communication via a wired shared medium, like the TIA/EIA-485 interface in this thesis. In both fields, two stations are able to communicate with each other, but they are not able to send and receive data at the same time.

A method for tunneling Ethernet frames over General Packet Radio Service (GPRS)/Enhanced Data rates for GSM Evolution (EDGE) is proposed by Fernando [27], where a micro-controller is used to control an Ethernet controller and to forward the Ethernet data in a special data frame called Netmos. Therefore, an RS-232 interface is used to connect the micro-controller with a GPRS/EDGE module.

The basis for the new approach is an efficient medium access protocol. There are different kinds of CSMA to control the access of the stations on a single shared medium, like shown in Section 2.2. One possibility is CSMA/CD with a suitable back-off algorithm for collision resolution. Pantazi [15] developed a Markovian model of the back-off algorithm, which is used in the slotted 1-persistent CSMA/CD protocol, in order to analyze the performance of such a network, like for example IEEE 802.3/Ethernet. Because of the high complexity, the existing models of the CSMA/CD protocol do not incorporate the effects of the back-off algorithm. In the approach of Pantazi, an approximate Markovian model of the system with a multi-dimensional state vector is developed and the stability behavior of the system is extracted. [15]

The IEEE 802.11 standard for WLANs is based on the CSMA/CA protocol which supports asynchronous data transfers. CSMA/CA uses an acknowledgement mechanism to verify successful transmissions and optionally, a handshaking mechanism to decrease the collision overhead. An exponential back-off mechanism is used in both cases. The work of Ziouva [19] investigates the theoretical performance of both mechanisms in terms of throughput and delay under traffic conditions that correspond to the maximum load that the network can support in stable conditions.

CSMA/CA is a fully distributed, asynchronous protocol that does not need a centralized controller to synchronize the operations on the shared medium. It can naturally support variable length data packets without any sophisticated segmentation and reassembly. These two features make the unslotted CSMA/CA very simple and scalable. For the implementation of the protocol, however, important issues including fairness scheduling and effects of implementation parameters like buffer size have to be investigated. Four different scheduling algorithms are evaluated by Kim [16]: Random Select, Destination Priority Queuing, Longest Queue First, and Shortest Packet First, which are designed for the unslotted CSMA/CA with a back-off MAC protocol to address the issues of fairness and bandwidth efficiency.

Many papers have studied the theory of distributed scheduling in wireless networks, including topics like stability or utility maximizing random access. Several publications in 2008 showed an adaptive CSMA that in theory can approach a bandwidth optimum without any message passing under a number of assumptions. The paper of Lee [28] reports the results from the first deployment of such random access algorithms through an implementation over conventional IEEE 802.11 hardware. It is shown that Utility Optimal CSMA may work well in practice even with an implementation over legacy equipment, and a wide array of gaps between theory and practice in the field of wireless scheduling. Therefore, that paper also brainstorms the discovery of and bridging over these gaps, and the implementation-inspired questions on modeling and analysis of scheduling algorithms. [28]

An additional important point is the transmission buffer, which can also be improved as shown by Ninagawa [29] with a transmission buffer framework for CSMA/CA type control networks for resource-limited networked embedded systems. The unique points of the buffer framework are prioritized push-out and packet concatenation with limited resources. The effectiveness of the new buffer implementation is validated in comparison with a conventional FIFO buffer using a network traffic simulator for CSMA/CA control networks.

Another approach of CSMA/CA is analyzed in industrial products, more specific in modular power plants. Many such systems depend on cheap micro-controllers. A common approach, which is often used in networks, has been based on the combination of the available standard serial communication controller (UART) normally included in such devices and TIA/EIA-485 drivers. However UART-based network protocols have some potential disadvantages. CAN offers an attractive alternative because of the availability and low cost of CAN based devices. This paper compares these two approaches in the design of a serial bus and describes the migration process from the first to the second solution. [18]

Two algorithms, namely time-based frame aggregation and selective frames, to improve the multiple-access method in beacon-enabled mode (slotted CSMA/CA) are proposed by Baz [30]. In slotted CSMA/CA, a network coordinator sends two timing parameters in regular beacon frames: Beacon Order (BO) and Superframe Order (SO). Devices compute the interval between beacons (BI) and Superframe Duration (SD) at the end of which devices go to a low power mode. The structure of the superframe is shown in Figure 2.11.

SD can be split up into a Contention Access Period (CAP) and a Contention Free Period (CFP). In the CAP, devices compete with each other to access the shared channel. A device can choose the method of sending a frame depending on the type of frame. There are four types of frames. Beacon frames are used to allocate the coordinator, acknowledgments are used when a

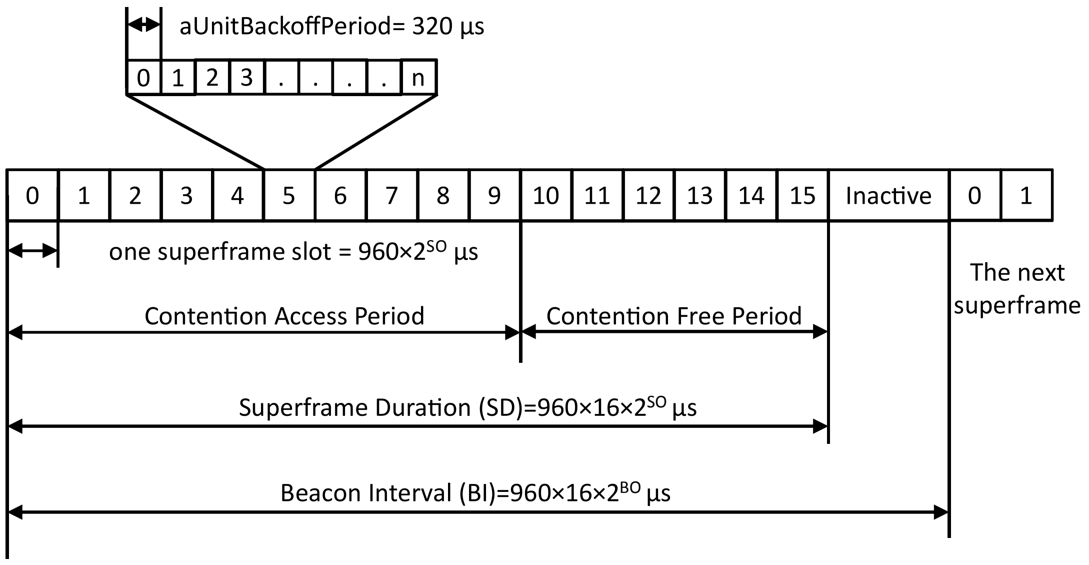

aUnitBackoffPeriod= 320 µs
0 1 2 3 . . . . n
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Inactive 0 1
one superframe slot = 960×2^SO µs
Contention Access Period
Contention Free Period
The next superframe
Superframe Duration (SD)=960×16×2^SO µs
Beacon Interval (BI)=960×16×2^BO µs

Figure 2.11: Structure of a superframe of slotted CSMA/CA [30]

device needs a confirmation of reception, data frames are used to encapsulate data arriving from higher layers, and command frames are used to build and maintain the network topology. [30] The CFP is optional in the standard and devices start the communication directly during time slots which were reserved without any channel assessment. SD can be divided into smaller time periods, which are called UnitBackoffPeriods or slots and they are used to manage transmission activities in slotted CSMA/CA protocol. Time-based frame aggregation tries to reduce waste in waiting times caused by different long frames. It combines multiple frames to reach a multiple of a slot duration, that are 10 octets, which reduces the transmission overhead, increases the throughput and channel utilization, and reduces delay. The second proposal, selective frames, resolves a blocking state that a device enters when it attempts to transmit a frame longer than the remaining time in the allocated transmission period. This algorithm enables a device to select an alternative frame which can be fitted in the reminder of the period, thereby giving higher throughput and lower delay compared to the original slotted CSMA/CA protocol. [30]

The work of Miškowicz [31] shows a performance analysis of the fixed-window CSMA/CA media access protocol with the geometric probability distribution of choosing the slot within the contention window. This scheme was initially proposed for large-scale wireless sensor networks. The primary benefit of geometric CSMA/CA is high channel utilization for broad range of numbers of contending station. The goal of his paper was to compute the standard performance measures of geometrically distributed slotted-CSMA both in the context of data-centric dense sensor networks and node-centric industrial automation networks. The stochastic analysis of the throughput/channel utilization, the collision rate and the mean access delay is analyzed. In the end, the protocol performance is compared with some conventional CSMA schemes with uniform distribution.

Tay proposed a protocol with the name CSMA/p* [32], which is non-persistent CSMA with a carefully chosen nonuniform probability distribution p*, that is used by nodes to randomly select contention slots. He shows that CSMA/p* is optimal in the sense that p* is the unique probability distribution that minimizes collisions between contending stations. CSMA/p* has knowledge of N, which is the number of stations in a network. He concludes with an exploration of how p* could be used to build a more practical medium access control protocol via a probability distribution with no knowledge of N that approximates p*.

Master-Slave procedure

Another way to control the access on a shared medium is the Master-Slave procedure. Miorandi considers the behavior of a Master-Slave protocol placed on top of the IEEE 802.11 wireless LAN [33]. More precisely, he proposes a prototype of a Master-Slave protocol and discusses how it could be implemented using the communication services supplied by IEEE 802.11. The protocol does not require any change to the IEEE 802.11 specification and it may make use of commonly available network components, which are suitable for industrial applications. Following the structure proposed by such a standard for the interface to the upper layers of the communication stack, he decided to use the Logic Link Control (LLC) services to map the Master-Slave functions on IEEE 802.11. In particular, he considered the acknowledged connection-less services as the most suitable to handle both the types of traffic he has analyzed: cyclic and acyclic. The proposed model specifies a simple polling scheme for the exchange of cyclic data, whereas three different techniques for handling acyclic requests have been considered.

A Master-Slave distributed network monitoring and control system for different kinds of high frequency vibrating screens is designed by Ren [34]. Hence, an Industrial Personal Computer (IPC)) is used as master, PIC micro-controllers are used as slaves and the communication interface is TIA/EIA-485. To make data transmit possible on the network, a communication protocol had to be developed for the data link layer. The protocol is defined in such a way, that there are two types of commands for the salves. Command 1 contents the command word, the slave address, the string length, given values for the eight vibration exciter amplitudes, control bits for starting or stopping the slave computer and a CRC and command 2 contents the command word, the slave address, string length and a CRC. There are again two different commands from a slave to the master according to received command. The answer 1 contents the code 102, which indicates a successful reception by the slave, and the answer 2 contents the command word, the slave address, string length, the measured values of the eight vibration exciter amplitudes and a CRC. [34]

Token-Passing procedure

The fourth medium access scheme, which is discussed in Section 2.2, is the Token-Passing procedure. The paper of Gershon [35] proposes a low-level network architecture for reliable powerline communications, using a Token-Passing protocol supporting network access and device response under worst-case powerline interference. The approach bases on a spread spectrum physical layer enabling short-duration powerline transmissions and adaptation to changing powerline conditions and on a packet structure, which is designed for error correction. Several key features of a data link layer, which are required for reliable operation of large networks on the powerline, are the decomposition of larger packets to powerline frames to create reliable communications, error correction and detection, effective adaptive equalization and reliable transfer of control. [35] There are a lot of corrupted transmissions on the powerline and hence, a transmissions of short frames is suggested. To ensure the integrity of any frame of data, both error correcting and detecting codes are used to detection errors and reduce the number of retransmissions. Each frame is acknowledged by the receiver before the transmission proceeds to the next frame. To implement this low-level link protocol, the higher level packet is broken up into such short frames. It is also discussed, why Token-Passing is preferred over CSMA/CD.

A timed-token protocol for real-time communications is proposed by Malcolm [36]. There are two classes of messages in his approach: synchronous and asynchronous. Synchronous messages are used for real-time communication and can have deadline constraints. The idea behind the protocol is to control the token rotation time. A protocol parameter called the Target Token Rotation Time (TTRT) gives the expected token rotation time. A portion of the TTRT, which is the maximum time a station is permitted to send synchronous messages every time it receives the token, is assigned to each station. When a station receives the token, it transmits its synchronous messages for a period not longer than its assigned TTH. If the time elapsed since the previous token departure from the same station is less than the value of TTRT, it can then transmit its asynchronous messages. This only happens if the token arrived earlier than expected. To guarantee that synchronous-message deadlines are met, system designers must carefully choose network parameters such as the TTRT and the buffer size. The buffer size is especially important, because messages in a network can be lost if there is insufficient buffer space at either the sending or the receiving station. The send buffer holds messages waiting to be transmitted to other stations and the receive buffer holds messages that have been received from other stations and are waiting to be processed by the host. [36]

The paper of Husein describes a technique for improving the response time of a Token-Passing network without using high transmission rates. [37] A Token-Passing network, which is introduced as the HRT-Net (Hard Real Time Network) operating at 1 MBit/s, is proposed and evaluated. The HRT-Net was developed in order to meet the requirements of hard real time control environments, especially where extremely fast response times are required and it uses two tokens instead of one of the Token-Passing procedure. This reduces the TTRT for each server and therefore, it improves the network response time. Additionally an implicit Token-Passing (tokenless) technique is used to conquer the effects of the Token-Passing delay. The two servers are assigned unique channels within the media by simply multiplexing the channels in frequency domain in the HRT-Net. These channels are named Normal Channel (N-Channel) and High Priority Channel (HP-Channel). The two servers with a Normal Token and a High Priority Token are associated with each of these channels. Hence, the HRT-Net consists of two logical rings. The Normal Token provides all the services, which are required for a Token-Passing network. Therefore, all active nodes, like for example, nodes on the logical ring, receive the Normal Token, whereas, the High Priority Token only provides services to nodes generating high priority messages. [37]

The performance of a timer-controlled Token-Passing mechanism in an industrial communication network is analyzed by Kim. [38] Several real network parameters, like for example finite buffers and a finite TTH, which generally determine the overall performance of a network, are considered. The performance of fieldbus or other timer-controlled Token-Passing network is evaluated using a matrix equation between the queue length distribution and the token rotation time, since finite size buffers and finite TTH are considered. The approximation error is shown to be small by computer simulation. The probability distribution of frame inter-arrival time follows a Poisson distribution. Blocking occurs if the queue is full. In this case, the blocked frames are assumed to be lost. [38]

CHAPTER 3

Conceptual Design

This chapter contains the prerequisites and the approach for the design of the new developed protocol. Different media access schemes are discussed and the one suiting best is chosen. The protocol is described using a flow chart. Additionally, a failure model is given.

3.1 Prerequisites

A system, which is used for the detailed description of the protocol, is shown in Figure 3.1. For the design of the new protocol, several assumptions are made:

- The implementation of the protocol is executed on a high-performance micro-controller running at a speed of 40 MHz.
- The micro-controller has two UART interfaces, which are both connected via TIA/EIA-485 transmitters. The half-duplex TIA/EIA-485 interface can operate with 1.25 Mbit/s at a cable length of up to 600 m and with 625 kbit/s at a cable length of up to 1,200 m.
- One micro-controller is always connected to another micro-controller with the same firmware via two TIA/EIA-485 interfaces. One of the micro-controllers can have a distinguished configuration, e.g., a master function or token monitoring.

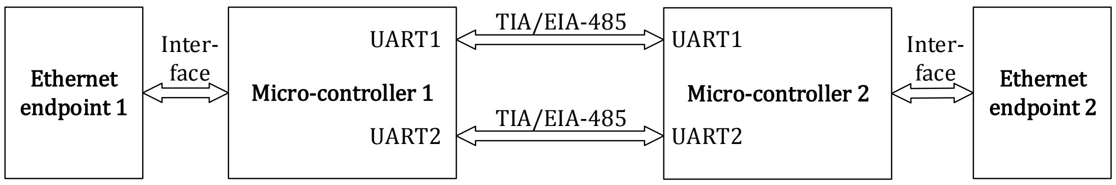


Figure 3.1: Example system for describing the protocol

- The Ethernet frames are transmitted from an Ethernet endpoint, which can be, e.g., a switch or PC to the micro-controller and vice versa via an interface, that is much faster than the TIA/EIA-485 interface, like for example Media Independent Interface (MII) or Serial Peripheral Interface (SPI). Hence, it does not matter, which interface is used, but it is important that the TIA/EIA-485 interface is still the bottleneck of the system.
- The Ethernet frame size ranges from 64 bytes to 1536 bytes according to IEEE 802.3 [1]. Ethernet frames with less than the minimum 64 bytes (14 byte header + payload + 4 byte FCS) are padded to 64 bytes, i.e. if there is less than 46 bytes of payload, extra padding data is added to the frame. Additionally, there are seven bytes of the preamble and one byte for the start frame delimiter.

Another important fact besides the prerequisites are the errors, that can occur during transmission. The possible errors are the following:

- One connection between one of the TIA/EIA-485 interfaces can fail.
- Both connections between the TIA/EIA-485 interfaces can fail.
- A transmission error can happen, where several bits change their values during transmission.
- A TIA/EIA-485 transmitter can fail. Therefore, no communication via TIA/EIA-485 is possible.
- A micro-controller can break down.
- The interface between the micro-controller and the Ethernet endpoint fails.
- The Ethernet endpoint breaks down or sends always data (babbling idiot).
- The size of the Ethernet frame is out of the defined range. Hence, the Ethernet frame is too big to transfer.

3.2 Protocol Design: Ethernet to TIA/EIA-485

Based on the state-of-the-art analysis (see Chapter 2), a protocol for TIA/EIA-485 is developed. Therefore, the physical medium is given by using TIA/EIA-485 (see Section 2.2). The next step is to choose the multiple access scheme for controlling the bus access of two communication partners. The following procedures which are shown in Section 2.2 are considered:

- Master-Slave procedure
- CSMA/CD
- CSMA/CA
- Token-Passing procedure

The Master-Slave procedure is typically used for controlling a number of slaves on a bus. The existing hardware system has a point-to-point connection between two participants. Hence, the control algorithm used in the master has too much communication overhead for one of the two participants, which has to be the master.

CSMA/CD is also not used, because the transmission efficiency decreases in high load situations due to a lot of aborted telegrams. It is shown in Section 2.2, that in the time where a telegram, which will be aborted, is sent, there is no utilizable transmission on the bus. The more telegrams get aborted, the less is the number of successful transmitted ones. This is a problem in case of safety critical applications, because there is a lot of traffic on the communication medium, which cannot be transmitted due to collisions.

CSMA/CA cannot be used with the given hardware configuration because of the long cable length of up to 1200 m. The velocity of propagation v is a constant and therefore, the product *cable length* $\cdot$ *transmission rate* must be smaller than a specific bound. Though this is not a problem in small dimensions, the transmission rate has to be reduced, if this bus access procedure is used in wide spread facilities,.

Finally, a Token-Passing approach is chosen as medium access control scheme, because of several advantages. If a participant does not have anything to send, it can pass the token such that the token slot is used by another participant. Additionally, a master can supervise the neighbor. If an error occurs at the neighbor, the faulty connection can be detected. An additional benefit is, that there is no handshake necessary prior a transmission.

There are two possibilities to utilize the two parallel TIA/EIA-485 communication channels. The first way is to use the two channels for improving the bandwidth by splitting up the Ethernet frame. One half of the Ethernet frame is sent via the first interface and the second half via the other to send at both channels at the same time. The second possibility is to use it for reliability. Hence, a single Ethernet frame is sent on both channels. If an error occurs on a channel, the error-free frame from the other channel can be used. The first option is used for the new protocol, because the performance is very important due to the low data rate of the TIA/EIA-485 interface. Reliability is also very important and therefore, this is achieved in another way, which will be described later in this section.

The procedure of the new protocol is described using a system, which is shown in Figure 3.1, where micro-controller 1 reads an Ethernet frame from Ethernet endpoint 1 and sends is to micro-controller 2 via TIA/EIA-485, which forwards the Ethernet frame to Ethernet endpoint 2.

Micro-controller 1 receives Ethernet frames from an Ethernet endpoint 1 via an interface, which is assumed to be much faster than TIA/EIA-485, such that TIA/EIA-485 is the bottleneck of the system and saves them in the internal Random-Access Memory (RAM). The Ethernet frame size is in the range from 64 bytes to 1530 bytes, which is assumed as maximum received frame size. Ethernet frames are split up into smaller numbered frames during saving into the RAM to simplify the transfer via TIA/EIA-485.

The transmitting micro-controller 1 sends several frames via both UART interfaces over TIA/EIA-485 while holding the token and keeps track of the chronological order of the sent frame numbers. A CRC of every sent frame in a token slot is computed and is sent with the frame number as a frame header before every frame (see Table 3.2). The receiving micro-controller 2 checks every received frame for errors using the CRC. If a frame is received correctly, it will be acknowledged at the start of the next token slot. If the receiving micro-controller 2 receives the token, it becomes the transmitting one and starts its transmission by sending a token start header, shown in Table 3.1, including a start identifier, the number of frames in the token slot, two status bits, the length of the transmitted Ethernet frame (computed by the total number of frames of the Ethernet frame + the number of bytes of the last frame) and the frame numbers of the received frames being acknowledged. The length of the token start header is four bytes (for the start identifier, the number of frames in the token slot, two status bits and the length of the transmitted Ethernet frame) plus a byte for every number of an acknowledged frame indicated by the after the start identifier received number of frames in the token slot. The status bits indicate if the TIA/EIA-485 lines operate correctly. If no data has to be transferred, an empty frame is sent as heartbeat frame being signaled by the status bites. Micro-controller 1 receives this token start header and can check which frames have not been transmitted correctly and retransmits those frames in its next token slot.

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	1	1	1	1	1	1	1	1	1	Number of frames					
Status		Number of bytes of last frame						Total number of frames							
Ack frame number 1								Ack frame number 2							
Ack frame number 3								Ack frame number 3							

Table 3.1: Token start header

If a complete Ethernet frame is correctly received by micro-controller 2, the Ethernet frame is written to the Ethernet endpoint.

The data frame header, which is sent before every frame is shown in Table 3.2. It includes the frame number and a 10-bit CRC. The CRC is used to detect bit errors, which happen during transmission. A parity bit is not suitable for so many bits being sent in a frame and the checksum method does not perform as well as the CRC method (see Section 2.2). Additionally, the parity bit is used by UART for protecting one byte.

Three parameters CRC size, message length, and Hamming Distance are explored for choosing the CRC. The polynomial 0x319 [20] is chosen for the CRC, because it provides a Hamming Distance of 4 ($HD = 4$) for a length of up to 501 bits. The maximum message length indicates the maximum number of bits being protected by CRC without any repeating CRC value. That means, that up to three bit errors are detected safely with a message length of up to 62 bytes.

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Frame number						CRC									

Table 3.2: Data frame header

Table 3.3 shows the time in milliseconds for sending several frames per token slot with different frame sizes using one TIA/EIA-485 channel. The baud rate is assumed to be 0.625 MBit/s (i.e. 78.125 kByte/s). The frame header is 2 bytes like described above. An Ethernet frame of size 1024 bytes is transmitted in 13.1 milliseconds without any header. In the right section of the table, there is an additional column with the overhead of the headers transmitting 1024 bytes in percent.

Hence, this Table 3.3 is used to determine the choice for the number of frames per token slot and frame size. The fields show the transmission time with different frame sizes in the columns and varying number of frames in the rows. The grey background highlights the entries where 1024 bytes of data are transmitted and show how long it takes to transmit the token start header, the frame header and the payload. The smaller the frame size and the smaller the number of frames per token slot, the longer takes the transmission and the higher is the overhead of the headers due to the several headers. For example, a transmission with one frame per token slot and only 16 bytes per frame has the highest overhead. The blue row highlights the row with four frames per token slot, which will be used.

Number of frames per token slot	16	32	64	128	256	512	1024	Frame size in bytes	Overhead in percent
1	0.23	0.44	0.84	1.66	3.30	6.68	13.13	time in ms	0.20
2	0.46	0.87	1.69	3.33	26.42	13.16		time in ms	0.39
4	0.92	1.74	3.38	6.66	13.21	26.32		time in ms	0.78
8	1.84	3.48	6.76	13.31	26.42	52.63		time in ms	1.56
16	3.69	6.96	13.52	26.62	52.84	105.27		time in ms	3.13
32	7.37	13.93	27.03	53.25	105.68	210.53		time in ms	6.25
64	14.75	27.85	54.07	106.50	211.35	421.07		time in ms	12.50

Table 3.3: Comparison of different frame sizes and data packets per token slot

If the number of frames per token slot and the frame size are chosen too high, then non-utilized bytes have to be transmitted, e.g., if frame size is 1024 and there are only 500 bytes to send, then 524 not utilized bytes have to be send, too. On the other side, if the numbers are chosen too low, then there is a lot of overhead due to the headers.

Finally, the frame size is set to 32 and four frames will be transmitted per token slot. The entry in the table for the chosen transmission parameters is highlighted in light green. The values are chosen quite small, because it is assumed, that mainly small Ethernet packages are utilized in NCSs.

Figure 3.2 shows the maximal utilized bandwidth with the chosen parameters (TIA/EIA-485 bandwidth of 625 kbit/s, 32 bytes per frame and 4 frames per token slot). With the minimal Ethernet frame size of 64 bytes the transmission takes 1.74 milliseconds resulting in a data rate of 294 kbit/s. The transmission of 96 and 128 bytes also takes 1.74 milliseconds resulting to a data rate of 441 kbit/s respectively 588 kbit/s, because always 128 bytes per token slot are transmitted. The maximal data rate remains the same at 588 kbit/s for the other frame sizes (256, 512, 1024 and 1280 bytes) and is only marginally lower for 1500 bytes at 581 kbit/s.

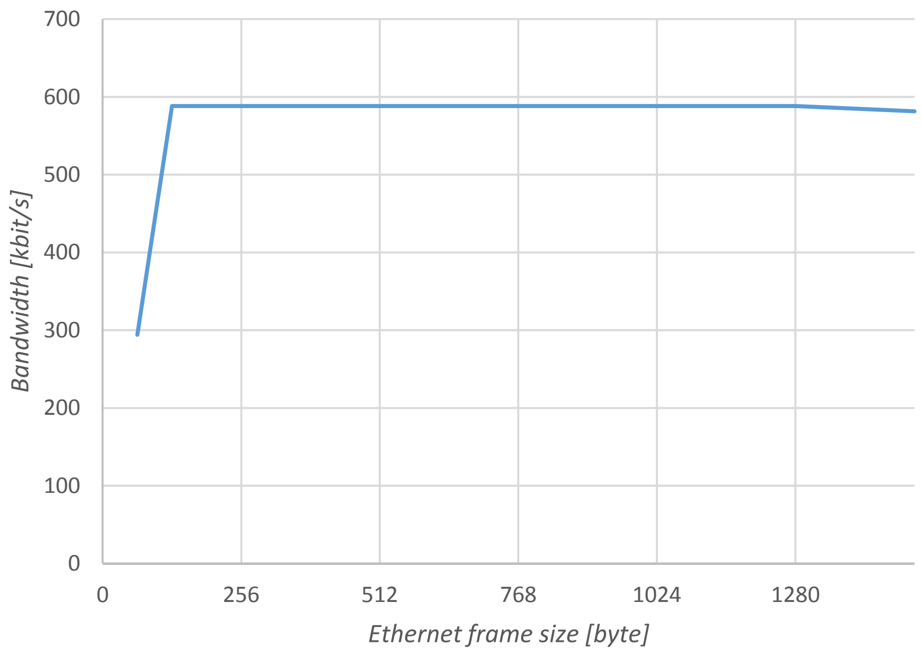


Figure 3.2: Maximal bandwidth of the protocol

3.3 Protocol Description

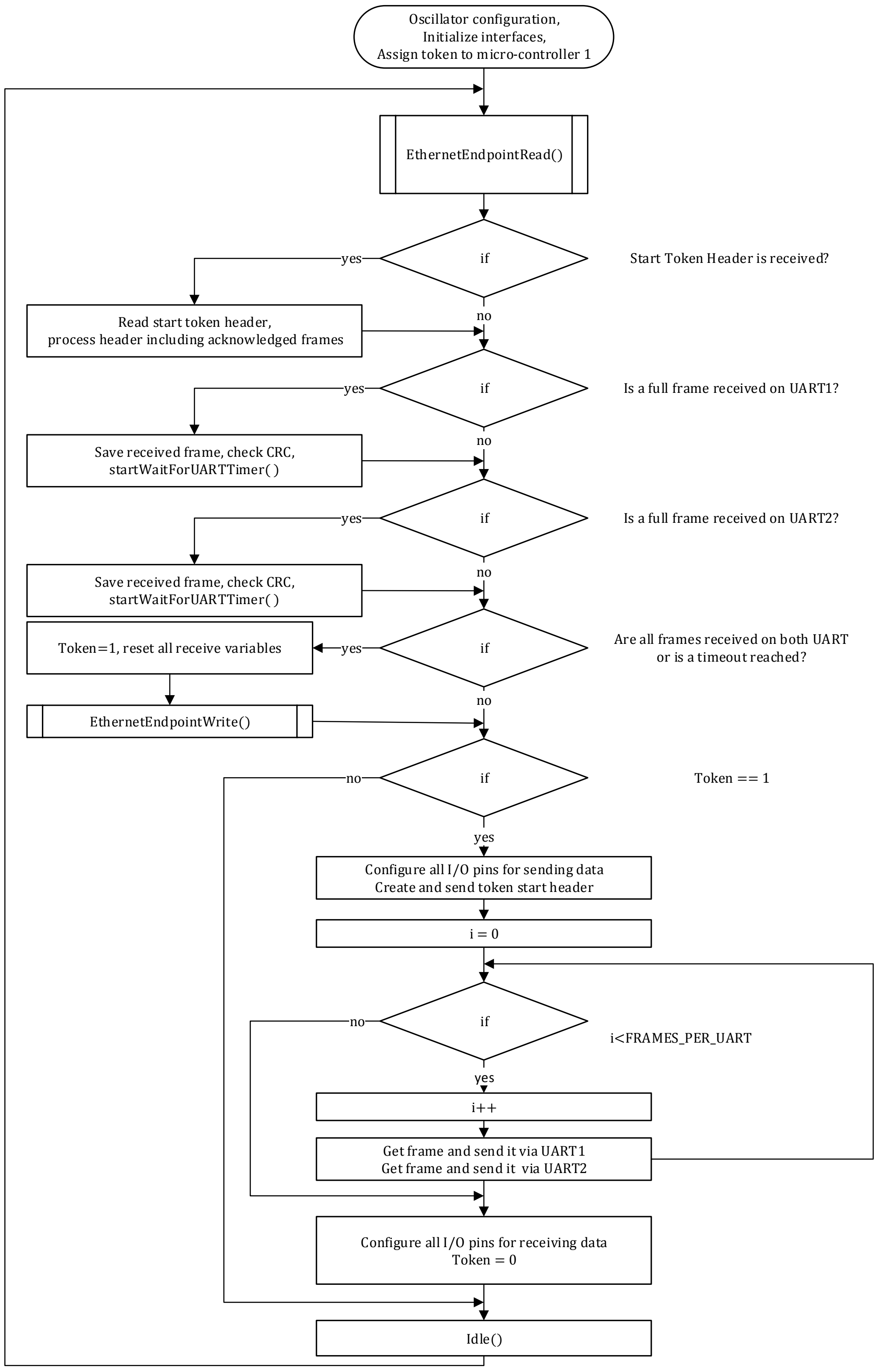
The flow chart of the main program is shown in Figure 4.9. The configuration of the oscillator, the initializes the I/O ports and other peripherals like for example the UART interface and the interface to the Ethernet endpoint are done at the start of the micro-controller. Furthermore, micro-controller 1 monitoring the token, gets assigned the token.

The while loop can be split into four different parts:

- The procedure of the function *EthernetEndpointRead()* for reading Ethernet frames via the interface between the micro-controller and the Ethernet endpoint and saving them into a buffer is called if the flag *readyForEthernetReceive* is set to true.

- The UART receive procedure checks repeatedly how many bytes have been received on the two UART interfaces.

  First, it is evaluated, if the token start header is received on one of the two UART interfaces. If this is the case, it is evaluated, if the received bytes are the start header by checking for the token start header identifier (compare Section 3.2, Table 3.1). If a token start header is received, it is read and processed by setting the number of frames in this token slot, the token status and the length of the currently transmitted Ethernet frame. Additionally, the acknowledgement of the frames being sent in the previous token slot is



Oscillator configuration,
Initialize interfaces,
Assign token to micro-controller 1
EthernetEndpointRead()
if
yes
no
Start Token Header is received?
Read start token header,
process header including acknowledged frames
if
yes
no
Is a full frame received on UART1?
Save received frame, check CRC,
startWaitForUARTTimer( )
if
yes
no
Is a full frame received on UART2?
Save received frame, check CRC,
startWaitForUARTTimer( )
Token=1, reset all receive variables
yes
if
no
Are all frames received on both UART
or is a timeout reached?
EthernetEndpointWrite()
no
if
Token == 1
yes
Configure all I/O pins for sending data
Create and send token start header
i = 0
no
if
i<FRAMES_PER_UART
yes
i++
Get frame and send it via UART1
Get frame and send it via UART2
Configure all I/O pins for receiving data
Token = 0
Idle()

Figure 3.3: Flowchart of the main loop

checked by reading the frame numbers and verifying which have been correctly received by the second micro-controller.

Afterwards, it is checked, whether a frame is received on UART 1 or UART 2. The received frame is saved into a buffer. Additionally, the correctness of the CRC of the frame is evaluated. A timer with the maximum timeout period for the successful reception of the frames on the other UART interface is started to detected problems on the TIA/EIA-485 lines, like for example, if two frames are received via the UART 1, then the specified time is waited for the reception of the two frames on UART 2. If the frames on UART 2 arrive in time, then the timer is stopped, otherwise the timer interrupt service routine is executed and it is assumed that there is a problem with the TIA/EIA-485 lines of the belonging UART interface. It is also kept tracked how many frames have been successfully received on each UART interface.

Finally, if the all frames are received on both UART interfaces or a receive timeout happened, then the token is assigned. In case, that a timeout occurred, a necessary error handling is done and the receive buffers of the belonging UART are cleared. Additionally, it is checked, if all frames of an Ethernet frame have been received, because then the write process to the Ethernet endpoint can be started.

- After a complete Ethernet frame is received via UART, it will be written via an interface to the Ethernet endpoint with the function *EthernetEndpointWrite()*.
- The UART sending procedure is only started if the micro-controller possesses the token. The two TIA/EIA-485 transmit lines are enabled for being eligible to send data via TIA/EIA-485.

  Before the actual data can be transmitted, the token start header has to be created, as described in Section 3.2. Afterwards, the created header is sent via UART.

  Then, a frame is read from teh send buffer and sent via UART 1 and subsequently the same happens for UART 2. This is repeated until the number of frames per token slot is reached, e.g., four frames per token slot. The frame header (see Section 3.2) including the frame number and the CRC is created. If there are no more frames to send, a frame with the highest possible frame number 63 is sent. These frames are used as heartbeat frames being able to detect problems with the TIA/EIA-485 interfaces even if no payload needs to be transmitted.

  At the end of the sending procedure, the two TIA/EIA-485 transmit lines are disabled. If all frames of an Ethernet frame have been sent, then the transmit buffer is reset and a new Ethernet frame can be read from the Ethernet endpoint.

3.4 Fault Detection

The micro-controller has two UART interfaces, which are both connected to TIA/EIA-485 drivers. Hence, there are two independent communication channels for transmitting the decomposed Ethernet frames during normal operation. If one of the channels fails due to, e.g., a broken line, then it is recognized by the system like it is described in Section 3.3. This is done by using a timer, which is started if the defined number of frames is received by one of the two channels, like for example if there are four frames per token slot sent, then the timer is started after two received frames at one of the two UART interfaces. If the second channel also receives the defined number of frames in time, then the timer is stopped and the operation is continued. Otherwise, if the timer expires, it is assumed, that there is a problem with the channel.

It is also recognized, when both channels break down, because then a micro-controller does not receive any bytes via the UART interfaces. This is never the case in correct operating mode, because either the decomposed Ethernet frames or heart beat frames with a frame number of 63 and empty payload are transmitted. This is also the case, when a micro-controller or a TIA/EIA-485 transmitter including both TIA/EIA-485 interfaces breaks down.

Bit changes during transmission are detected by two different mechanisms:

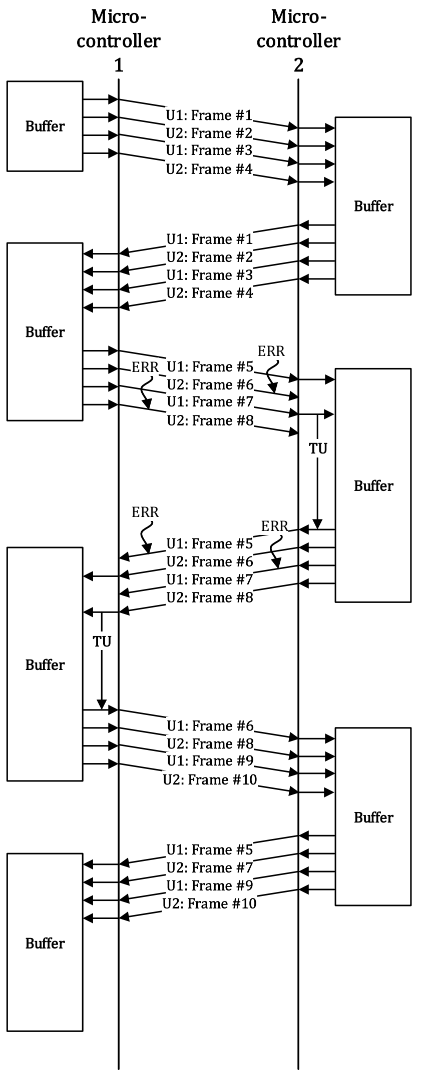
- The UART interface provides an option for using a parity bit for the transmission of every byte, which is used by the micro-controller. The parity bit detects any odd number of errors like it is shown in Section 2.2.
- Every 32-bit frame is sent with a leading frame header (see Section 3.2) including the frame number and a 10-bit CRC. The polynomial 0x319 [20] is chosen for the 10-bit CRC, because it provides a good hamming distance of 4 ($HD = 4$) for a length of up to 501 bits. That means, that up to three bit errors are detected safely with a message length of up to 62 bytes. A hamming distance of 4 for a 10-bit is feasible, because the 16-bit CRC CCITT-16 with the polynomial 0x8810 also has a hamming distance of 4, but uses 6 bits more. [20]

An error, which can not be detected by the micro-controller, is the breakdown of the interface to the Ethernet endpoint or the Ethernet endpoint itself, because then no data is read by the micro-controller. The micro-controller cannot decide if these outages occurred or if there is just no data to transmit.

If the micro-controller receives an Ethernet frame, whose frame size is bigger than the defined length of 1536 bytes, then the buffer of the micro-controller is too small to save this Ethernet frame due to the limited number of RAM. Therefore, this frame cannot be transmitted correctly.

Figure 3.4 shows two sequence diagrams with occurring errors on one or both TIA/EIA-485 interfaces. Only one TIA/EIA-485 interface respectively UART interface at the third and forth transmission is interfered in the left diagram. A transmission from a micro-controller to another with four frames represents a token slot. The first two token slots are transmitted correctly. Both micro-controllers acknowledge the successful transmission of the frames at the beginning of their token slot with the token start header. Frames with the frame number 1 to 4 are sent from micro-controller 1 to micro-controller 2 and vice versa. At the third token slot an error occurs at UART 2. Therefore, the frames with the number 6 and 8 are not received at micro-controller 2. A timer with the timeout UART *TU* is started after the successful reception of two frames from UART 1. After the timer is triggered, micro-controller 2 assumes, that there was an error during transmission on UART 2 and starts sending new frames. If no error occurs the timer is stopped after the reception of the other two frames on the UART 2. At the fourth token slot, an error occurs at UART 1. Micro-controller 1 receives two frames at UART 2 and starts again the timer *TU*. After the timer is triggered, micro-controller 1 can start to send frames again. The not successful frames with the number 6 and 8 are sent along with the two new frames at the fifth token slot. At this time, there is no error during transmission. Finally, micro-controller 2 sends the at the former errornous frames with the number 5 and 7 again along with two new frames.

In the right sequence diagram, both TIA/EIA-485 interfaces at the third transmission are interfered. Again, the first two token slots are transmitted correctly. Both micro-controllers acknowledge the successful transmission of the frames at the begging of their token slot with the token start header. Frames with the frame number 1 to 4 are sent from micro-controller 1 to micro-controller 2 and vice versa. After the token is passed to micro-controller 1, a timer with the timeout token *TT* is started being used to monitor the token. This timer is always reset, if the token is received. At the third token slot errors occur at UART 1 and UART 2. Therefore, the frames with the number 5 to 8 are not received at micro-controller 2. After the timer *TT* is triggered, micro-controller 1 assumes, that an error during transmission occurred on UART 1 and UART 2. Hence, the token was lost and a new one is created. Then the micro-controller starts sending new frames. The frames with the number 5 to 8 are sent again at the fourth token slot. At this time, there is no error and the transmission continues.



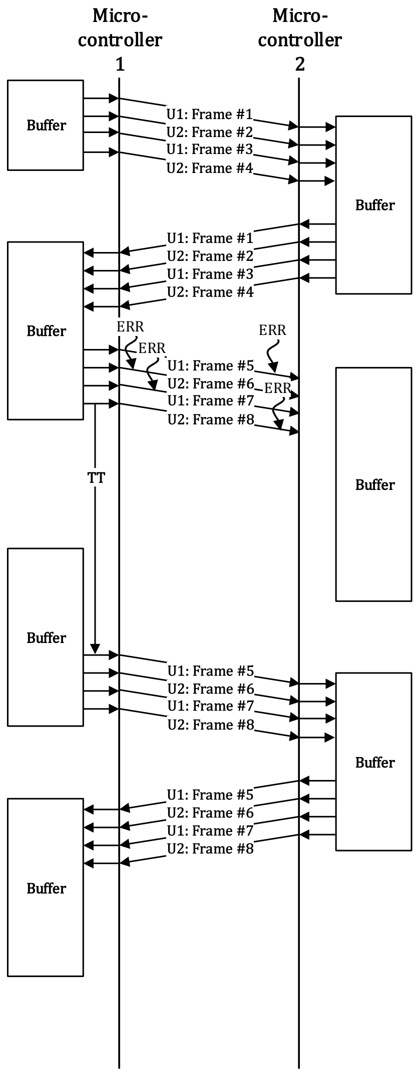


Figure 3.4: Two sequence examples: occurring of an error on one UART (left) and on both UARTs (right)

CHAPTER 4

Evaluation

In order to evaluate the previously proposed concept, this chapter discusses the used hardware and interfaces. An analysis of the initial protocol is given. Besides the implementation of the protocol and the realization of the proof-of-concept, the results of the comparison of the two protocols are presented.

4.1 Hardware

The initial protocol is running on a special hardware. Therefore, a short overview of the hardware is given to provide a glance of the requirements of the system.

In Figure 4.1, the hardware structure is shown. A Marvell 88E3016-Fast-Ethernet switch [39] receives the Ethernet frames and provides them via MII defined by IEEE 802.3 [1]. A PIC micro-controller PIC24HJ64GP206 [40] is used with 64 kB Flash, 8 kB RAM and a clock frequency of 40 MHz. All the algorithms of the protocol are implemented on this controller. There are two separate PIC micro-controllers to provide two different communication channels, but only one of them is able to configure the switch via the Serial Management Interface (SMI) [1]. Both PICs can send status messages to the Central Processing Unit (CPU) via the Synchronous Serial Interface (SSI) [2]. SSI is not relevant for the development and the evaluation of the protocol and hence, it is no longer considered.

The PIC micro-controller reads and writes the Ethernet frames via MII from an Ethernet endpoint. Additionally, it transfers the data to the MAX3293, which is a low-power, high-speed, half-duplex transmitter for TIA/EIA-485/TIA/EIA-422 interfaces.

4.2 Interfaces

Media Independent Interface (MII)

All Ethernet transmissions defined in IEEE 802.3 are using the MAC protocol (see Section 2.1). Before transmission, the MAC has to check if the medium is free and if there is a collision

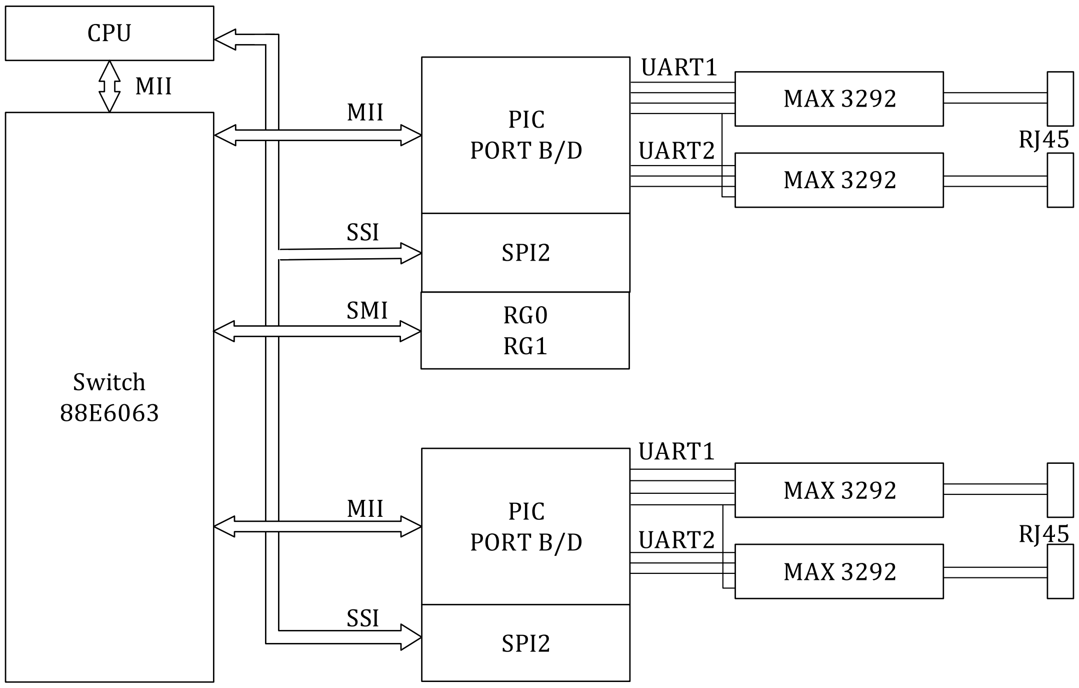


Figure 4.1: Hardware overview

during the transmission. Different physical layers can be used by the MAC through the common MII including sensing and transmission capabilities. The MII is also defined in IEEE 802.3. This separation is very important, because different physical media can be used in such a way. Additionally, a management interface, the SMI, which will be described later, is defined as part

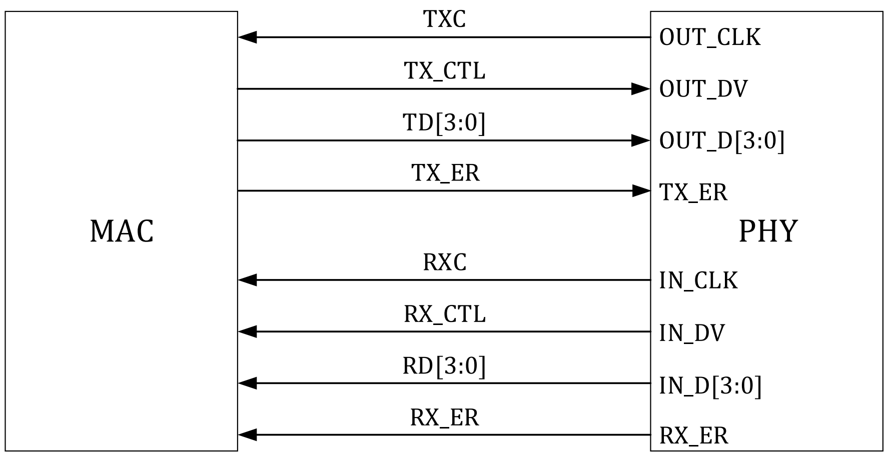


Figure 4.2: MII signals [39]

of the MII besides the usage for media-sensing and data transfer.

The data transmission lines of the MII (see Figure 4.2) include transmit clock (TX_CLK, TXC, OUT_CLK), transmit enable (TX_CTL, TX_CTRL, OUT_DV), transmit error (TX_ER) and four bits of transmit data (TD[3:0], OUT_D[3:0]) in the transmit direction and receive clock (RX_CLK, RXC, IN_CLK), receive enable (RX_CTL, RX_CTRL, IN_DV), receive error (RX_ER) and four bits of receive data (RD[3:0], IN_D[3:0]) in the receive direction. MII provides full duplex operation because of the separate transmit and receive lines. Additionally, there is a carrier sense line (CRS) and a collision detection line (COL) for media sensing.

The clock TX_CLK is supplied by the Physical Layer (PHY) and can be present all the time. The TX_CLK frequency ranges from DC to a quarter of the bit rate of an Ethernet transmission, i.e., up to 2.5 MHz for 10 Mbps Ethernet and up to 25 MHz for 100 Mbps Ethernet, because there are four data lines, which are written simultaneously. The same holds for the clock RX_CLK, but for reading direction.

The transmit enable (TX_EN) coincides with the presence of data bits (TD[3:0]). Meanwhile, the carrier sense (CRS) detects the same transmission on the media by the PHY and becomes active after a little delay. Any error which is detected during the transmission is relayed to the MAC over the TX_ER pin.

The receive data valid (RX_DV) starts during the preamble and no later than the start frame delimiter (SFD). Any error being detected during the reception is relayed to the MAC over the RX_ER pin.

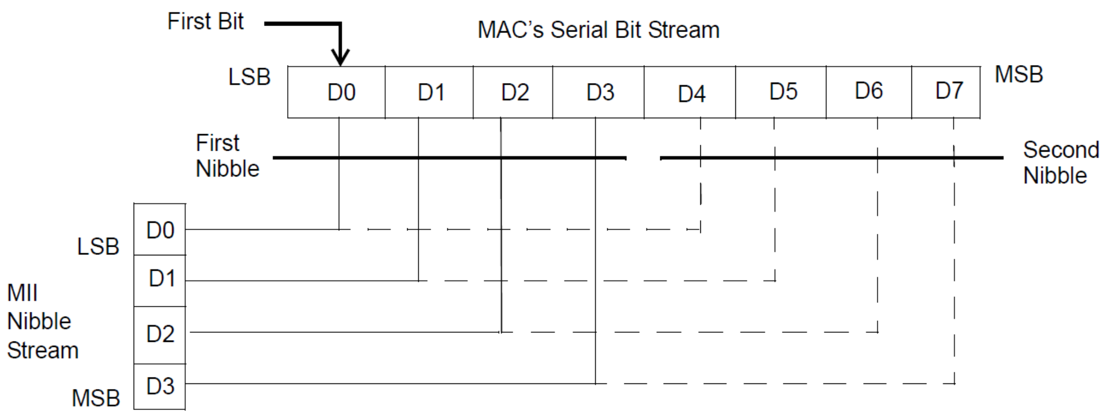


Figure 4.3: MII byte/nibble transmit and receive order [1]

Transmission and reception of a byte of data is done a nibble at a time according to the order of nibble transmission and reception shown in Figure 4.3 for the MII. The bits of each byte are transmitted and received as two nibbles, where bits 0 to 3 of the byte correspond to the bits 0 to 3 of the first nibble and bits 4 to 7 correspond to the bits 0 to 3 of the second nibble.

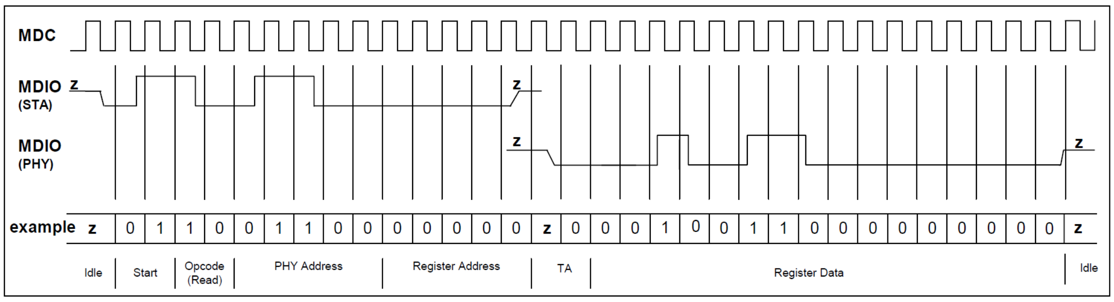


Figure 4.4: Typical SMI read operation via MDC and MDIO [39]

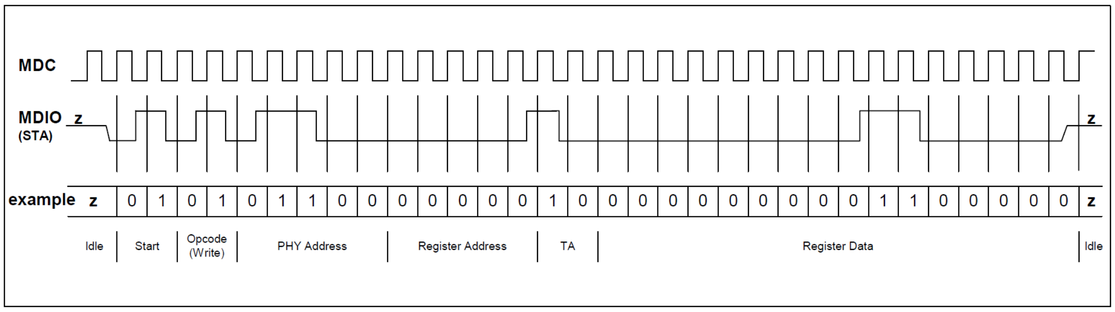


Figure 4.5: Typical SMI write operation via MDC and MDIO [39]

Serial Management Interface (SMI)

The SMI provides access to internal registers of the MAC via the Management Data Clock (MDC) and Management Data Input/Output (MDIO) lines and is conformal to IEEE 802.3u. MDC can run from DC to a maximum rate of 8.33 MHz and is driven by the MAC device to the PHY. MDIO is a bi-directional signal that runs synchronously to MDC. It is driven by the PHY to provide register data at the end of a read operation. The MDIO pin requires a 1.5 kΩ pull-up register to pull the MDIO high during idle and turnaround times.

Typical read and write operations are shown in Figure 4.4 and 4.5.

During a write command, the MAC provides address and data. When the MAC drives the MDIO line, it has to guarantee a stable value 10 ns (setup time) before the rising edge of the clock MDC. Further, MDIO has to remain stable 10 ns (hold time) after the rising edge of MDC. [39]

The PHY takes over the bus at the end of the address bits transmission to supply the MAC with the register data requested for a read command. When the PHY drives the MDIO line, the PHY has to provide the MDIO signal between 0 and 300 ns after the rising edge of the clock. [39] Therefore, with a minimum clock period of 400 ns (2.5 MHz maximum clock rate) the MAC can safely sample MDIO during the second half of the low cycle of the clock.

Serial Peripheral Interface (SPI)

The SPI is a synchronous serial interface used for the communication with other peripherals or micro-controller devices.

The SPI bus specifies four signals, which are shown in Figure 4.6 [40]:

- Serial Clock (SCLK output from master).
- Master Output, Slave Input (MOSI output from master, SDI input for slave).
- Master Input, Slave Output (MISO input from master, SDO output from slave).
- Slave Select (SS, active low, output from master).

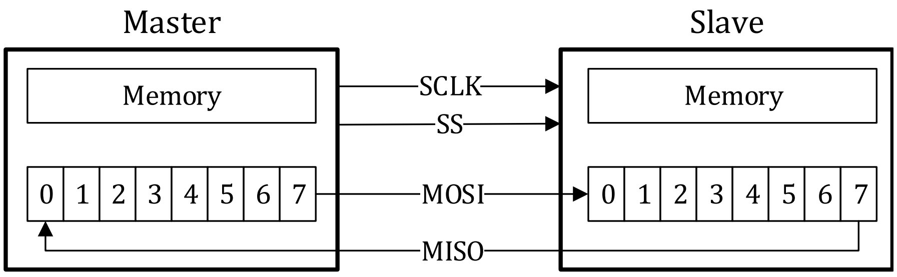


Figure 4.6: A typical SPI hardware setup using two shift registers to create a circular buffer [40]

SPI devices communicate in full duplex mode using the Master-Slave procedure with a single master. The master sends the frame for reading and writing. During each SPI clock cycle, the master or the slave sends a byte and has vice versa always to receive a byte. This happens even if only one-directional data transfer is intended. Transmission involves two shift registers of some given word size, like for example as 8 or 16 bits, as shown in Figure 4.6. Multiple slave devices are supported by using a selection with individual slave select (SS) lines.

At the beginning of the communication, the master configures the clock, using a frequency supported by the slave device, typically up to 32 MHz. The master then selects the slave device with a binary 0 on the select line.

In addition to the setting of the clock frequency, the master must also configure the Clock Polarity (CKP) and Clock Edge (CKE) with respect to the data. There are four possible configurations [40]:

- At CKP=0, the idle state is a low level. Active state is a high level:
  - For CKE=0, data is captured on the clock's rising edge and data is propagated on a falling edge.
  - For CKE=1, data is captured on the clock's falling edge and data is propagated on a rising edge.
- At CKP=1, the idle state is a high level. Active state is a low level:

- For CKE=0, data is captured on clock's falling edge and data is propagated on a rising edge.
- For CKE=1, data is captured on clock's rising edge and data is propagated on a falling edge.

For the proof-of-concept, a Raspberry Pi is used as master and a PIC micro-controller as slave. WiringPi [41] provides a small library, which is utilized for opening and sending bytes to/from SPI devices on the Raspberry Pi. The configured SPI frequency of 10 MHz is provided by the master. The communication is byte-wide. Hence, an 8 bit shift register is utilized. CKP is set to 0 and the idle state is a low level. CKE is set to 1 and data is captured on the clock's falling edge.

4.3 Analysis of Initial Approach

The initial approach, which is replaced by a new one, is based on two UARTs of the PIC as described in Section 4.1. The data transfer uses UART characters (compare Section 2.2). There are two types of characters:

- **Protocol Characters (PC)** are used to organize the protocol sequence and to manage the two channels of each PIC. The structure of the protocol is shown in Table 4.1, where the field *Health (H)* is used to exchange information about the state of the channel. The values reach from 0, which is the best health condition, to 3, which is the worst one. Every participant keeps track of the health state on its own.

  The field operation marks the four types of operation:

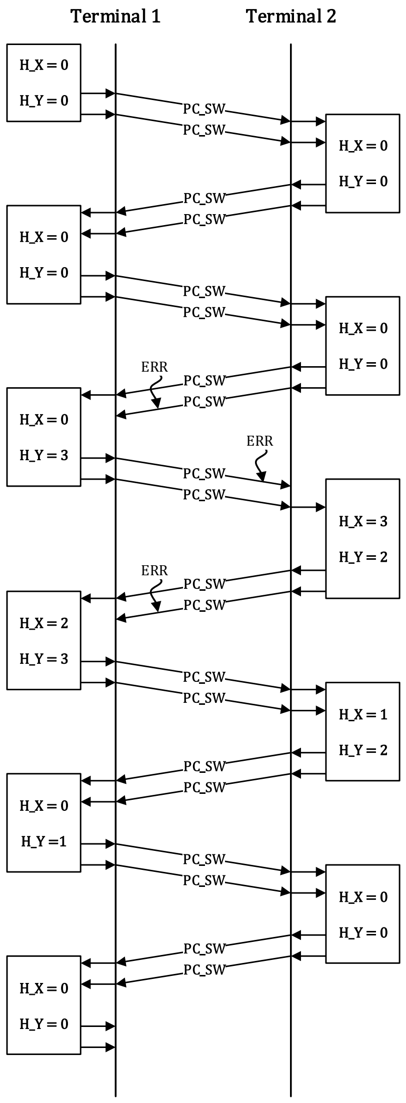
  - *Switch* (SW, value 3): No frame transfers are pending.
  - *Request to send* (RS, value 2): Signals that the transmitter has an Ethernet frame to send.
  - *Clear to receive* (CR, value 1): Signals that the receiver is ready to receive an Ethernet frame.
  - *Frame end* (FE, value 0): The end of a frame was detected by the receiver.

  The fields *Health* and *Operation* are protected with the field Hamming Complement to receive a hamming distance of 4. Additionally, the parity bit of the UART is used and hence, it is guaranteed, that the handshake protocol is protected against interferences.

Start	B0	B1	B2	B3	B4	B5	B6	B7	Parity bit	Stop
	Health		Operation		Hamming Complement					

Table 4.1: Structure of protocol characters

- **Frame Characters (FC)** are utilized to transfer Ethernet frames using the MII. Therefore, two received nibbles are put in one frame character for the transmission via the UART.



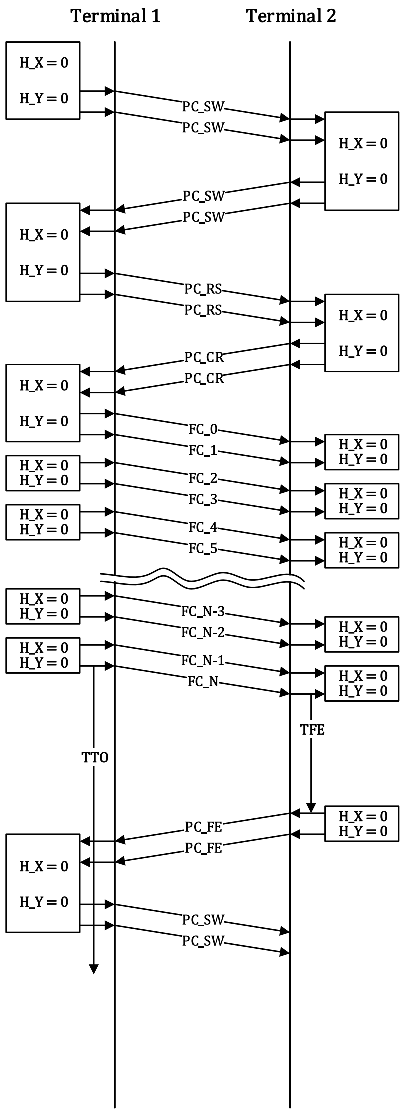


Figure 4.7: Two sequence examples: occurring of an error (left), transmission of data(right)

Start	B0	B1	B2	B3	B4	B5	B6	B7	Parity bit	Stop
	Ethernet frame data (2 nibbles from MII)									

Table 4.2: Structure of frame characters

Figure 4.7 shows two sample sequences. In the first example, nothing happens at the beginning and hence, only switch messages (PC_SW, protocol character switch) are exchanged. The health states of both channels is 0 (H_X=0 and H_Y=0). Then an error occurs at the fourth transmission of the channel Y and the health state of Y is set to 3 (H_Y=3). A second error happens in the next frame, but this time on channel X and the health state of X is set to 3 (H_X=3), where the health state of Y is decreased by one (H_Y=2), because of an error-free transmission on channel Y. After that, there is an error on channel Y and therefore, the health state of Y is set to 3 (H_Y=3), where the health state of X is decreased by one (H_X=2). Henceforth, no more errors occur and the two health states are decreased by one after each successful transmission until both are 0 (H_X=0 and H_Y=0).

In the second example, a data transmission is shown. There are two switch operations in the first two transmissions (PC_SW), where nothing is to send. Terminal 1 signals with a *Request to send* message (PC_RS) that there is data to transfer. Then terminal 2 replies with a *Clear to send* message (PC_CR) and terminal 1 can start the transmission with the next frame. It sends all its data while terminal 2 receives all the data (FC_0 to FC_N). After the transmission terminal 2 recognizes the end, if there was no data for a defined duration *TFE* and sends a *Frame end* message (PC_FE).

Therefore, there are some aspects in the initial protocol, which can be improved:

- Time is wasted with the handshake message before starting a transmission, which leads to a problem at short Ethernet frames, because they produce a lot of overhead.
- The used time out at the end of every transmission of frame characters wastes also a lot of time since the channel is idle.
- If only one frame character is lost during transmission, the Ethernet frame is incomplete and hence, dismissed at the receiving participant. The sending participant has to retransmit the Ethernet packet again.
- The same holds, if a frame character is interfered. Then the Ethernet frame is not correct and again the whole frame has to be dismissed.

4.4 Protocol Implementation

Component Overview Diagram

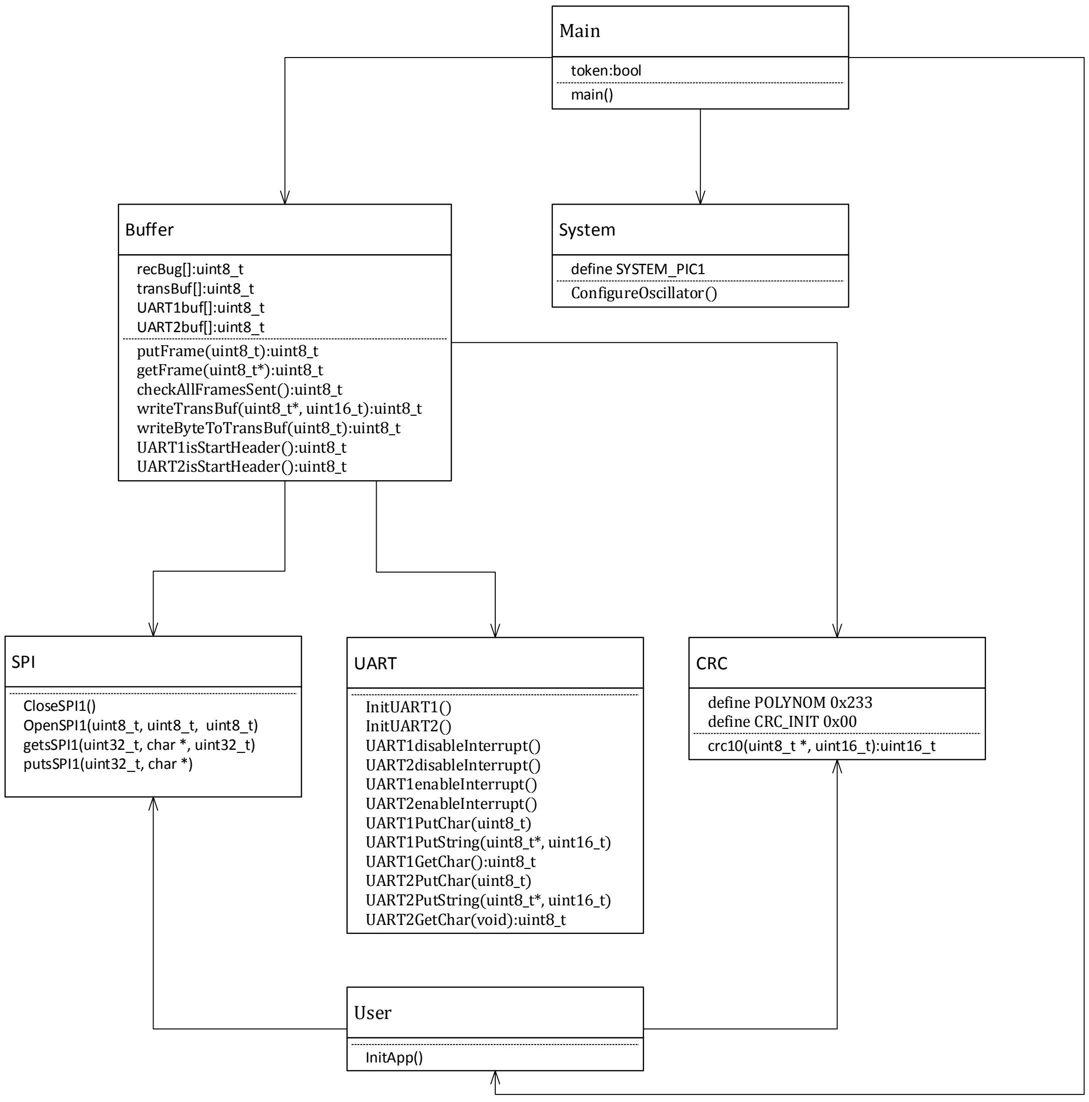

Main
token:bool
main()
Buffer
recBug[]:uint8_t
transBuf[]:uint8_t
UART1buf[]:uint8_t
UART2buf[]:uint8_t
putFrame(uint8_t):uint8_t
getFrame(uint8_t*):uint8_t
checkAllFramesSent():uint8_t
writeTransBuf(uint8_t*, uint16_t):uint8_t
writeByteToTransBuf(uint8_t):uint8_t
UART1isStartHeader():uint8_t
UART2isStartHeader():uint8_t
System
define SYSTEM_PIC1
ConfigureOscillator()
SPI
CloseSPI1()
OpenSPI1(uint8_t, uint8_t, uint8_t)
getsSPI1(uint32_t, char *, uint32_t)
putsSPI1(uint32_t, char *)
UART
InitUART1()
InitUART2()
UART1disableInterrupt()
UART2disableInterrupt()
UART1enableInterrupt()
UART2enableInterrupt()
UART1PutChar(uint8_t)
UART1PutString(uint8_t*, uint16_t)
UART1GetChar():uint8_t
UART2PutChar(uint8_t)
UART2PutString(uint8_t*, uint16_t)
UART2GetChar(void):uint8_t
CRC
define POLYNOM 0x233
define CRC_INIT 0x00
crc10(uint8_t *, uint16_t):uint16_t
User
InitApp()

Figure 4.8: Component overview

The described protocol is designed for being deployed to a PIC micro-controller 24HJ running at 40 MHz using the Microchip *MPLAB X IDE* [42]. The implementation can be split up in the following components (see Figure 4.8):

- The component *Main* contains the control procedure of the protocol.

- The component *UART* is responsible for the UART communication. Hence, it provides methods for the initialization as well as for sending and receiving bytes.
- The component *SPI* controls the access to the hardware SPI module of the micro-controller.
- The component *Buffer* handles all memory management for the send and receive buffers as well as the two UART buffers.
- *CRC* contains the implementation of the CRC algorithm.
- *System* and *User* are utilized for the initialization of all hardware components.

Protocol Description

A detailed flow chart of the main program is shown in Figure 4.9. The configuration of the oscillator, the initialization of the I/O ports and other peripherals like for example the UART interface and the interface to the Ethernet endpoint are done after the micro-controller startup. Furthermore, the micro-controller with the label *SYSTEM_PIC1*, which is monitoring the token, gets assigned the token.

The main procedure can be split into four different parts:

- The procedure of the function *EthernetEndpointRead()* for reading Ethernet frames via the interface between the micro-controller and the Ethernet endpoint and saving them into a buffer is called if the flag *readyForEthernetReceive* is set to true. SPI is used to exchange data exchange with the Ethernet endpoint for the proof-of-concept implementation.
- The UART receive procedure repeatedly checks the number of bytes being received via the two UART interfaces. These bytes are counted by the variables *UART1CharCounter* and *UART2CharCounter*.

  First, it is checked whether the token start header is received on one of the two UART interfaces by comparing *UART1CharCounter* and *UART2CharCounter* with the length of the token start header. If this is the case, it is evaluated, if the received bytes are the start header by checking for the token start header identifier (ten consecutive binary ones, compare Section 3.2, Table 3.1). It is sufficient just to check the ten consecutive binary ones at the start of the token start header, because otherwise it is a frame header, where this bit pattern is not possible (see Section 3.2, Table 3.2). If a token start header is received, it is read and processed by setting the number of frames in this token slot, the token status and the length of the currently transmitted Ethernet frame. Additionally, the acknowledgement of the frames being sent in the previous token slot is checked by reading the frame numbers and verifying which of them have been correctly received by the second micro-controller.

  Afterwards, it is check if a frame is received on UART 1 or UART 2 by comparing *UART1CharCounter* and *UART2CharCounter* with the sum of the frame length and the length of the frame header. The received frame is saved to a buffer using the function *put-Frame()*, which also evaluates the correctness of the CRC of the frame. A timer with the

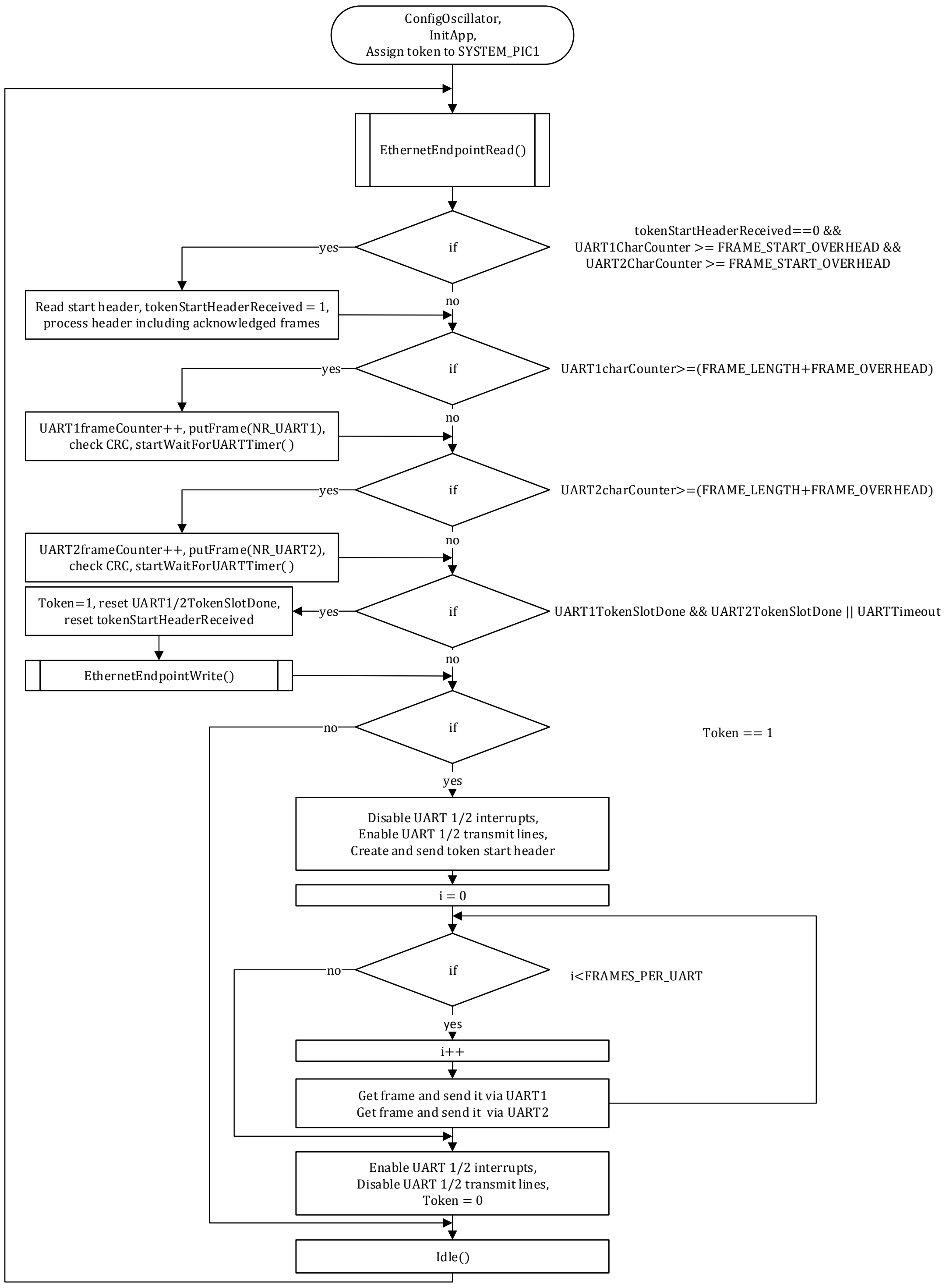

ConfigOscillator,
InitApp,
Assign token to SYSTEM_PIC1
EthernetEndpointRead()
if
tokenStartHeaderReceived==0 &&
UART1CharCounter >= FRAME_START_OVERHEAD &&
UART2CharCounter >= FRAME_START_OVERHEAD
yes
no
Read start header, tokenStartHeaderReceived = 1,
process header including acknowledged frames
if
UART1charCounter>=(FRAME_LENGTH+FRAME_OVERHEAD)
yes
no
UART1frameCounter++, putFrame(NR_UART1),
check CRC, startWaitForUARTTimer( )
if
UART2charCounter>=(FRAME_LENGTH+FRAME_OVERHEAD)
yes
no
UART2frameCounter++, putFrame(NR_UART2),
check CRC, startWaitForUARTTimer( )
if
UART1TokenSlotDone && UART2TokenSlotDone || UARTTimeout
yes
no
Token=1, reset UART1/2TokenSlotDone,
reset tokenStartHeaderReceived
EthernetEndpointWrite()
if
Token == 1
no
yes
Disable UART 1/2 interrupts,
Enable UART 1/2 transmit lines,
Create and send token start header
i = 0
if
i<FRAMES_PER_UART
no
yes
i++
Get frame and send it via UART1
Get frame and send it via UART2
Enable UART 1/2 interrupts,
Disable UART 1/2 transmit lines,
Token = 0
Idle()

Figure 4.9: Flowchart of the main loop

maximum timeout period for the successful reception of the frames on the other UART interface is started to detected problems on the TIA/EIA-485 lines, like for example, if two frames are received via the UART 1, then the specified time is waited for the reception of the two frames on UART 2. If the frames on UART 2 arrive in time, then the timer is stopped, otherwise the timer interrupt service routine is executed and it is assumed that there is a problem with the TIA/EIA-485 lines of the belonging UART interface. There is also a counter variable for receiving frames for each UART interface, which is increased by one after a successful frame reception.

  Finally, if all frames are received on both UART interfaces or a receive timeout occurred, then the token is assigned. In case a timeout occurred, an error handling is done and the receive buffers of the belonging UART are cleared. Additionally, it is checked, whether all frames of an Ethernet frame have been received, because then the write process to the MII can be started.

- After a complete Ethernet frame is received via UART, it will be written via an interface to the Ethernet endpoint with the function *EthernetEndpointWrite()*.

- The UART sending procedure is only started if the micro-controller holds the token, i.e., the token flag is set. In this case, the interrupts for the two UART interfaces are disabled due to the half-duplex communication, because otherwise it receives all sending data itself. The two TIA/EIA-485 transmit lines are enabled for being eligible to send data via TIA/EIA-485.

  Before the actual data can be transmitted, the token start header has to be created, like it is described in Section 3.2. Afterwards, the created header is sent via UART.

  Then a frame is read from the buffer with the function *getFrame()* and sent via UART 1. The same happens for UART 2. This is repeated until the number of frames per token slot is reached, i.e., four frames per token slot. The frame header, which is shown in Section 3.2, including the frame number and the calculated CRC is created by the function *getFrame()*. If there are no more frames to send, a frame with the highest possible frame number 63 is sent. These frames are used as heartbeat frames being able to detect problems of the TIA/EIA-485 interfaces even if no payload has to be transmitted.

  At the end of the sending procedure, the two TIA/EIA-485 transmit lines are disabled and the interrupts for the two UART interfaces are enabled. The token flag is set to false.

  If all frames of an Ethernet frame have been sent, then the transmit buffer is reset and the flag *readyForEthernetReceive* is set to signalize, that a new Ethernet frame can be read.

File Structure

The following files were created to implement the protocol:

- *main.c* contains the configuration of the oscillator, initializes the I/O ports and other peripherals like for example the UART interface. The flow chart of the main procedure of the micro-controller is shown in the last subsection. The UART and timer interrupt service routines are also located here.

- *buffer.c*: The variables for the receive and transmit buffer as well as the two UART buffers with the associated controlling variables are defined in this file. Therefore, all functions for writing and reading these buffers are implemented here.

- *buffer.h*: This header file is used to configure the buffer sizes and the token slot parameters, like for example, the frame length, the length of the token start header and the frame header and the number of frames per token slot. Additionally, the interfaces of the functions from *buffer.c* are declared here.

- *configuration_bits.c*: The functions for the hardware configuration of the PIC24HJ are defined here, like for example the settings for the boot segment, the code protection, the behavior at the power on reset, the watchdog settings and the programmer/debugger communication channel.

- *crc.c*: The algorithm for the 10-bit CRC (compare Section 2.2) is implemented here.

- *crc.h* defines the used polynomial and the initial value of the computation.

- *spi.c*: Routines for opening and closing the SPI (see Section 4.2), as well as the function for the SPI read/write operation are implemented in this file.

- *spi.h*: The pin mapping of the inputs and outputs of the SPI is done and the interfaces of *spi.c* are declared here.

- *system.c* contains the function for the oscillator configuration.

- *system.h*: The function of the micro-controller is defined here, e.g. *SYSTEM_PIC1* for token controlling micro-controller or *SYSTEM_PIC2* for the other one.

- *uart.c*: Functions for reading and writing data via the UART interface as well as for enabling and disabling the UART interrupts and the initialization of the two UART interfaces are implemented in this file.

- *uart.h* contains the pin mapping of the inputs and outputs of the two UART interfaces as well as the definition of the baud rate and the fast mode of the micro-controller. Additionally, the interfaces of *uart.c* are declared here.

- *user.c* initializes the digital GPIO pins and calls the functions for the initialization of the peripherals, like for example UART and SPI.

- *user.h* contains only the interface of the function *InitApp* from *user.c*.

4.5 Results

Performance

The results of the thesis are assessed with respect to the initial protocol. Therefore, a special test setup as shown in Figure 4.10, is used to analyze the performance of the initial protocol. There are two communication entities devoted as *System 1* and *System 2*, which are described in Section 4.1 in detail, and two Raspberry Pis (RPis), *RPi A* and *RPi B*, running a Linux operation system. The RPis are used to create Ethernet frames being sent via the network interface. Additionally, the UART interface of each RPi is connected to the debug interface of the hardware systems being able to monitor status messages via the terminal.

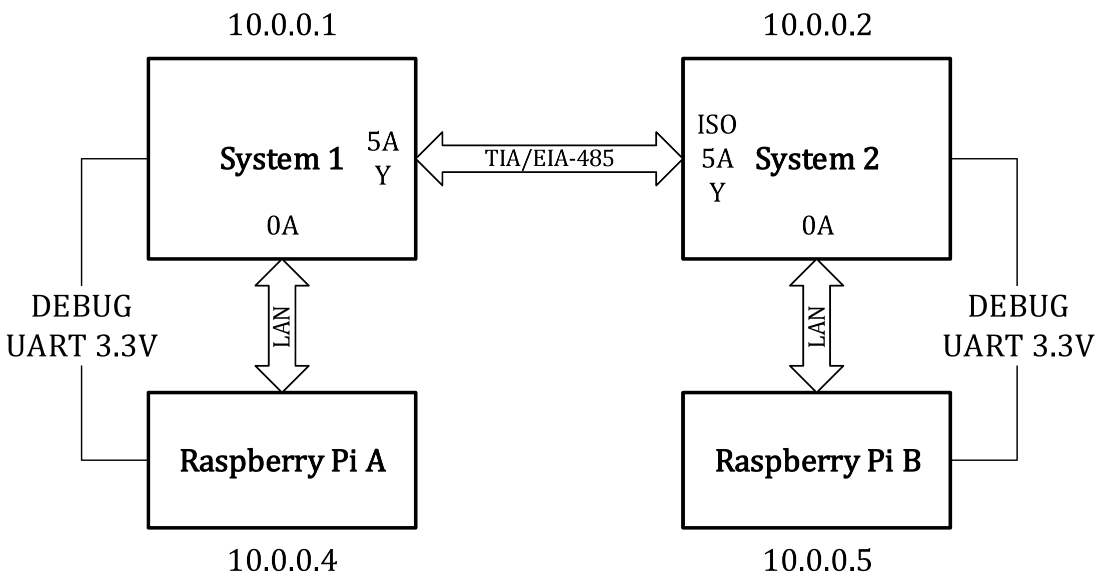


Figure 4.10: Test setup overview for the initial protocol

The tool *iperf* is used to obtain the performance of the initial protocol with bidirectional data transmission. *RPi A* is used as server and the following command is executed:

```
iperf -s -u -l 512 -i 1
```

The options of this command starts *iperf* in server mode (*-s*) using UDP (*-u*) rather than TCP with an Ethernet frame size of 512 bytes (*-l 512*). UDP uses a connectionless transmission model without any handshake mechanism. *iperf* establishes a host-to-host communication by using sockets before the performance measuring takes place. Hence, the measurement of the transmission time is not falsified. The option *-i* indicates a pause of 1 second between periodic bandwidth reports.

RPi B connects as client to the server with the address 10.0.0.4 via the following command:

```
iperf -c 10.0.0.4 -u -l 512 -i 1 -b 1200kbit -d
```

The option *-b* sets the target bandwidth for the sending data to 1200 kbit/sec. This command performs a bidirectional data transmission by using the option *-d*.

The client *RPi B* outputs the following for the bidirectional test (connection 3 from *RPi B* to *RPi A*, connection 4 from *RPi A* to *RPi B*):

```
------------------------------------------------------------
Client connecting to 10.0.0.4, UDP port 5001
Sending 512 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.5 port 51283 connected with 10.0.0.4 port 5001
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 512 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  4] local 10.0.0.5 port 5001 connected with 10.0.0.4 port 33680
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   141 KBytes  1.16 Mbits/sec
[  4]  0.0- 1.0 sec  28.8 KBytes   236 Kbits/sec   1.123 ms  123/ 180 (68%)
[  3]  1.0- 2.0 sec   145 KBytes  1.19 Mbits/sec
[  4]  1.0- 2.0 sec  28.2 KBytes   231 Kbits/sec   1.753 ms 235/ 292 (81%)
[  3]  2.0- 3.0 sec   143 KBytes  1.17 Mbits/sec
[  4]  2.0- 3.0 sec  28.2 KBytes   231 Kbits/sec   1.279 ms 232/ 290 (80%)
[  3]  3.0- 4.0 sec   144 KBytes  1.18 Mbits/sec
[  4]  3.0- 4.0 sec  28.0 KBytes   229 Kbits/sec   1.628 ms 231/ 287 (80%)
[  3]  4.0- 5.0 sec   144 KBytes  1.18 Mbits/sec
[  4]  4.0- 5.0 sec  28.3 KBytes   232 Kbits/sec   1.295 ms 235/ 291 (81%)
[  3]  5.0- 6.0 sec   145 KBytes  1.19 Mbits/sec
[  4]  5.0- 6.0 sec  28.1 KBytes   230 Kbits/sec   1.491 ms 234/ 290 (81%)
[  3]  6.0- 7.0 sec   144 KBytes  1.18 Mbits/sec
[  4]  6.0- 7.0 sec  28.3 KBytes   232 Kbits/sec   1.121 ms 234/ 290 (81%)
[  3]  7.0- 8.0 sec   143 KBytes  1.17 Mbits/sec
[  4]  7.0- 8.0 sec  27.9 KBytes   229 Kbits/sec   1.467 ms 233/ 289 (81%)
[  3]  8.0- 9.0 sec   145 KBytes  1.18 Mbits/sec
[  4]  8.0- 9.0 sec  28.4 KBytes   233 Kbits/sec   1.111 ms 232/ 289 (80%)
[  3]  9.0-10.0 sec   146 KBytes  1.19 Mbits/sec
[  3]  0.0-10.0 sec  1.41 MBytes  1.18 Mbits/sec
[  3] Sent 2882 datagrams
[  4]  9.0-10.0 sec  28.2 KBytes   231 Kbits/sec   1.400 ms 233/ 289 (81%)
[  4]  0.0-10.4 sec   293 KBytes   231 Kbits/sec   1.199 ms 2309/2895 (80%)
[  4]  0.0-10.4 sec  1 datagrams received out-of-order
[  3] Server Report:
[  3]  0.0-10.4 sec   293 KBytes   231 Kbits/sec   0.901 ms 2296/2881 (80%)
```

The server *RPi A* output shows the actual transmission speed (connection 5 from *RPi A* to *RPi B*, connection 3 from *RPi B* to *RPi A*):

```
------------------------------------------------------------
Client connecting to 10.0.0.5, UDP port 5001
Sending 512 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.4 port 33680 connected with 10.0.0.5 port 5001
[  3]  0.0- 1.0 sec  27.9 KBytes   229 Kbits/sec   0.240 ms  111/ 167 (67%)
[  5]  0.0- 1.0 sec   144 KBytes  1.18 Mbits/sec
[  3]  1.0- 2.0 sec  28.2 KBytes   231 Kbits/sec   0.934 ms 234/ 291 (81%)
[  5]  1.0- 2.0 sec   145 KBytes  1.19 Mbits/sec
[  3]  2.0- 3.0 sec  28.2 KBytes   231 Kbits/sec   0.811 ms 231/ 287 (80%)
[  5]  2.0- 3.0 sec   145 KBytes  1.19 Mbits/sec
[  3]  3.0- 4.0 sec  28.2 KBytes   231 Kbits/sec   0.752 ms 231/ 287 (80%)
[  5]  3.0- 4.0 sec   144 KBytes  1.18 Mbits/sec
[  3]  4.0- 5.0 sec  28.2 KBytes   231 Kbits/sec   1.700 ms 232/ 289 (80%)
[  5]  4.0- 5.0 sec   145 KBytes  1.19 Mbits/sec
[  3]  5.0- 6.0 sec  28.2 KBytes   231 Kbits/sec   0.744 ms 234/ 291 (81%)
[  5]  5.0- 6.0 sec   145 KBytes  1.19 Mbits/sec
[  3]  6.0- 7.0 sec  28.2 KBytes   231 Kbits/sec   0.675 ms 233/ 289 (80%)
[  5]  6.0- 7.0 sec   146 KBytes  1.19 Mbits/sec
[  3]  7.0- 8.0 sec  28.2 KBytes   231 Kbits/sec   0.846 ms 233/ 290 (81%)
[  5]  7.0- 8.0 sec   146 KBytes  1.19 Mbits/sec
[  3]  8.0- 9.0 sec  28.2 KBytes   231 Kbits/sec   0.167 ms 232/ 288 (80%)
[  5]  8.0- 9.0 sec   143 KBytes  1.17 Mbits/sec
[  3]  9.0-10.0 sec  28.2 KBytes   231 Kbits/sec   1.268 ms 233/ 289 (81%)
[  5]  9.0-10.0 sec   145 KBytes  1.19 Mbits/sec
[  5]  0.0-10.0 sec  1.41 MBytes  1.19 Mbits/sec
[  5] Sent 2896 datagrams
[  3]  0.0-10.4 sec   293 KBytes   231 Kbits/sec   0.902 ms 2296/2881 (80%)
[  3]  0.0-10.4 sec  1 datagrams received out-of-order
[  5] Server Report:
[  5]  0.0-10.4 sec   293 KBytes   231 Kbits/sec   1.199 ms 2309/2895 (80%)
```

The outputs of the client and the server show, that an average data rate of 231 Kbit/sec is achieved in both directions. The percentage of packet loss is high at 80 %, because the data rate with 1.2 Mbit/s is too high for the system due to the maximum achieved data rate of 231 Kbit/sec for the Ethernet frame size of 512 bytes. In the first second the packet loss is lower, because the buffer of the switch is empty at the start of the transmission. Hence, more datagrams are buffered in the first second. These performance tests are executed with the buffer sizes 64, 128, 256, 512, 1024 and 1500.

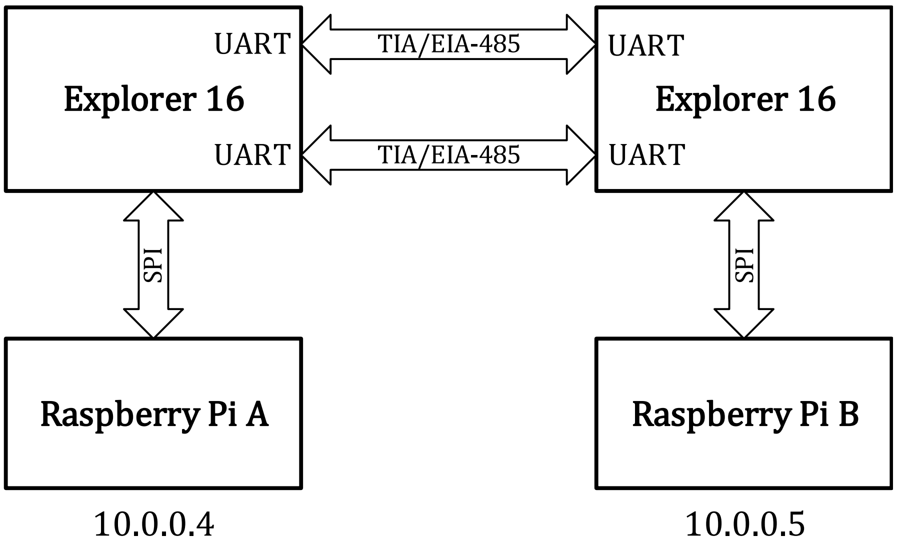


Figure 4.11: Test setup overview for new protocol

Another test setup is used for the new protocol to measure the performance. Two Microchip Explorer 16 development boards using the same micro-controller as before with additional TIA/EIA-485 drivers for the UART interfaces are used. Ethernet frames with the same size are sent from a RPi A via SPI to the micro-controller 1 of the first Explorer 16 board. The SPI operates at 10 MHz and is much faster than TIA/EIA-485, which remains the bottleneck of the system. Therefore, the Ethernet frame is read from the SPI and it is sent via both TIA/EIA-485 interfaces to the second micro-controller. After the complete Ethernet frame is received and buffered by the micro-controller 2, it is written via the SPI to RPi B. The same method is used to transfer data in the other direction from micro-controller 2 to micro-controller 1 to achieve a bidirectional transmission. After each sent frame, a random time x is waited until the next frame is sent to simulate jitter. A schematic drawing of the test sequence is shown in Figure 4.12.

The random variable X has a uniform distribution on the interval $[a, b] = [1, 1.2]$ ms. Its probability density function is:

$$f_X(x) = \begin{cases} \frac{1}{b-a} = \frac{1}{1.2-1} = \frac{1}{0.2}, & \text{if } x \in [1, 1.2]. \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

The expected value of x is $E[X] = \frac{a+b}{2}\frac{1+1.2}{2} = 1.1$ ms and the variance of x is $\sigma^2[X] = \frac{(b-a)^2}{12}\frac{(1.2-1)^2}{12} = 0,0033$.

The time for transmitting an Ethernet frame is measured using an oscilloscope. Afterwards, the transmission time is used to compute the transmission speed. Therefore, the micro-controller indicates the SPI read and write procedure toggling a GPIO pin for each these two operations. The GPIO pin indicating the SPI read procedure is measured with channel A of the oscilloscope on micro-controller 1. To measure an entire data transmission, the GPIO pin indicating the SPI write procedure is measured with channel B of the oscilloscope on micro-controller 2.

For example, a measurement with the oscilloscope of an Ethernet frame of size 64 byte is shown in Figure 4.13. The first rising edge of channel A (SPI read) is used as starting time and the second falling edge of channel B (SPI write) is used as end time for the measurement. The measured time interval $\triangle t$ is 2.96 milliseconds, which is shown in the top left corner. This time corresponds with an Ethernet frame size of 64 bytes to a data rate of 172.97 kbit/s.

These two types of measurements are executed with Ethernet frames of length 64, 128, 256, 512, 1024 and 1500 with bidirectional data transmission to evaluate the performance of both protocols. It is sound to compare the results of them, because both read data from an interface which is much faster than TIA/EIA-485 remaining the bottleneck of the system. Furthermore, both systems utilize the same PIC micro-controller running at a speed of 40 MHz and all the TIA/EIA-485 interfaces run at a speed of 1.2 MHz.

Table 4.3 shows the results of the performance test of the initial protocol with *iperf*. The data rate is very low at a frame size of 64 bytes, because there is a lot of overhead due to the handshake messages before starting the transmission. There is no additional overhead of *iperf* due to the connectionless transmission model of UDP.

Table 4.4 shows the results of the performance test of the new protocol. Hence, the time for receiving an Ethernet frame, sending it via TIA/EIA-485 and forwarding it to an Ethernet endpoint is measured. The data rate can easily be computed, because the frame size is known using the following formula:

$$\frac{Ethernet\ frame\ size\ [byte] * 8}{Transmission\ time\ [ms]} = Data\ rate\ [kbit/s]$$

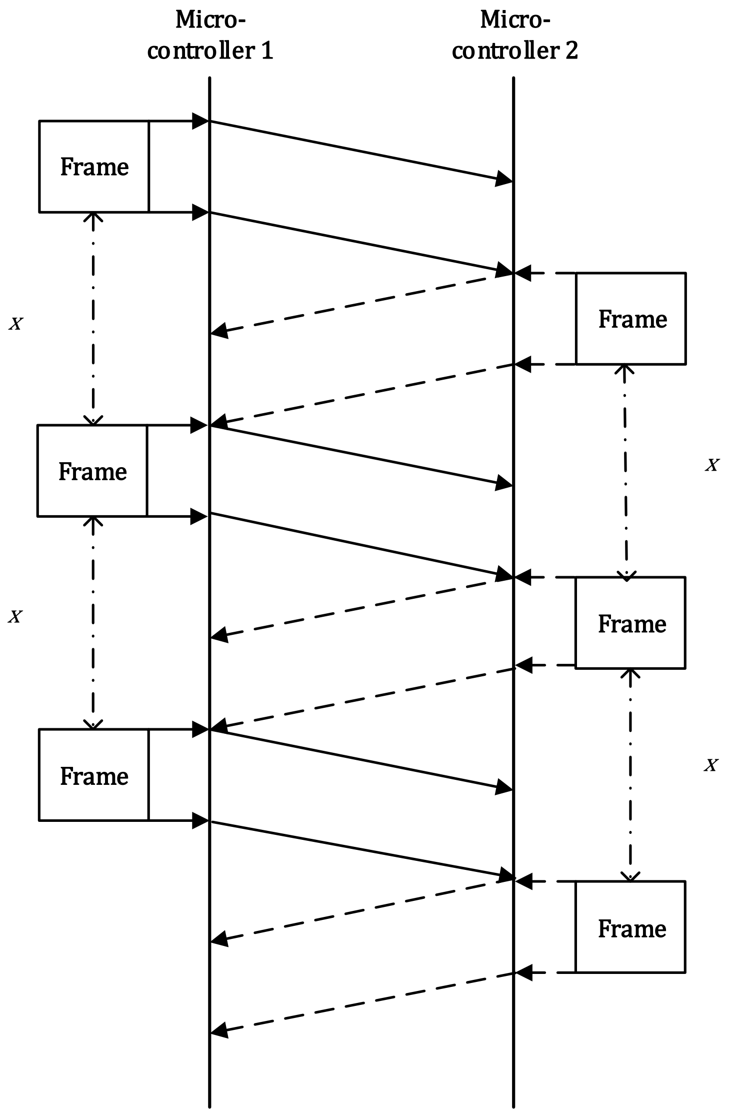


Figure 4.12: Test sequence for the new protocol

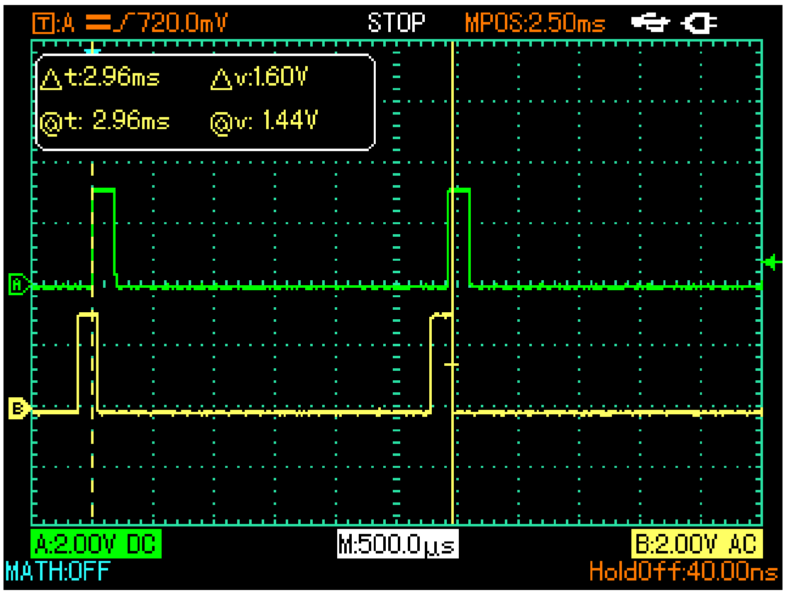


Figure 4.13: Time measurement of the transmission of an Ethernet frame (64 byte)

Ethernet frame size in byte	Send data rate in kbit/s	Data rate in kbit/s
64	1200	46.9
128	1200	80.4
256	1200	143.5
512	1200	231.0
1024	1200	336.3
1500	1200	340.9

Table 4.3: Performance of the initial protocol

Figure 4.14 illustrates the results of the two different performance tests by using a diagram. The Ethernet frame size and the data rate have been mapped to the two axes. The blue dashed line shows the performance of the initial protocol and the red solid line the performance of the new protocol.

The data rate is significantly improved compared to the initial protocol considering small Ethernet frame sizes up to about 800 bytes. This is achieved by exchanging the medium access scheme to the Token-Passing procedure. Hence, there are no handshake messages before starting transmission and no timeouts after a transmission of an Ethernet frame necessary, like they are used in the initial protocol. This leads to a much better utilization of the TIA/EIA-485 interfaces at small frame sizes. The bigger the Ethernet frames, the longer takes the buffering of the frames of the new protocol. Therefore, the initial protocol achieves higher data rates than the new one when transmitting Ethernet frame with a size of 800 bytes and above. However, in NCSs, packet

Ethernet frame size in byte	Transmission time in ms	Receive data rate in kbit/s
64	2.96	172.97
128	3.728	274.68
256	7.44	275.27
512	15.04	272.34
1024	30.08	272.34
1500	46	260.87

Table 4.4: Performance of the new protocol

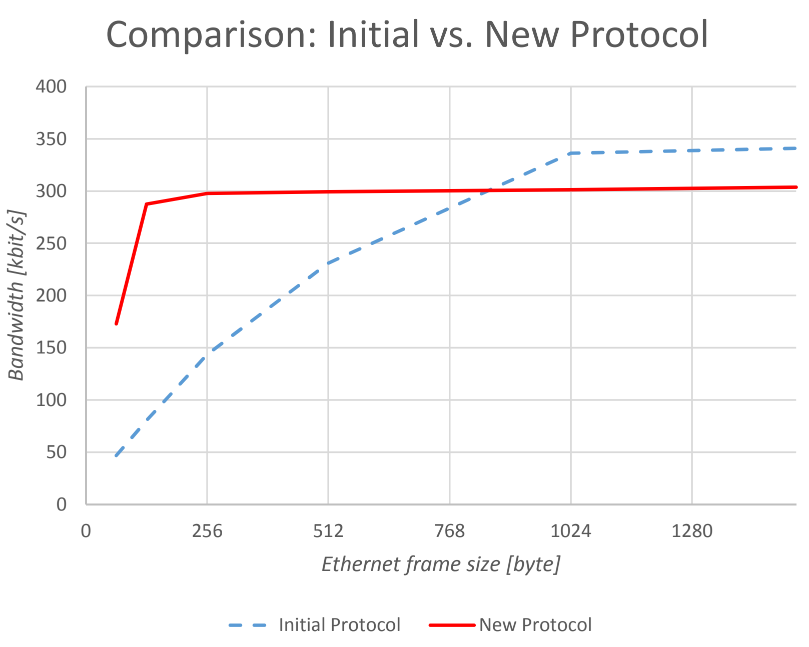


Figure 4.14: Performance comparison of initial and new protocol

sizes are typically low such that the performance gain is valuable.

Reliability

Another very important requirement of the system is the error handling. If only one character is interfered or lost during transmission, the entire Ethernet frame has to be sent again using the initial protocol. If a transmission error occurs in the new protocol, it is detected through the used 10-bit CRC or the parity bit of UART. In this case, only a part of the Ethernet frames has to be retransmitted. This is achieved by splitting the Ethernet frames into smaller frames during buffering, which are used to transfer the data via the TIA/EIA-485 interface.

CHAPTER 5

Conclusion

This thesis presents an approach for tunneling Ethernet traffic via TIA/EIA-485 in NCSs. For this purpose, an efficient protocol to carry Ethernet frames over redundant TIA/EIA-485 interfaces is developed to reach network lengths of up to 1200 meters. With this tunneling procedure, large NCSs relying on Ethernet can be accomplished.

First, the foundations for developing a new protocol are presented. The basics of the ISO-OSI layer model and the physical media are shown. Different existing medium access schemes are examined and checked regarding their suitability for reliable Ethernet tunneling via robust communication links. Furthermore, several data protection techniques and communication models are investigated. In order to get knowledge how the transferred Ethernet frames look like, a short summary of IEEE 802.3/Ethernet is given. Related scientific approaches are investigated.

Afterwards, the previously discussed concepts are used to develop a protocol fitting the given requirements. Hence, the discussed medium access schemes are compared and then the Token-Passing procedure is chosen, because it suits the application scenario best. Additionally, a fault model is proposed and methods for a reliable communication are elaborated. Based on this, an advanced concept for the new protocol is formulated, where details are specified, e.g. by using flow charts to show the procedure of the protocol.

Next, the hardware structure used for the initial protocol is described. The interfaces MII, SMI and SPI, used between the devices are summarized. Further, the initial protocol is investigated.

To show the feasibility and to analyze the performance of the protocol, a proof-of-concept implementation was established. Therefore, tests using bidirectional data exchange with different Ethernet frame sizes are executed. Based on this, the performance of the new protocol is examined. In a final step, the implementation of the newly created protocol is compared with the initial protocol to show the gain of the recently developed protocol.

The thesis spawned an efficient protocol to carry Ethernet frames over redundant TIA/EIA-485 interfaces being used to extend Ethernet networks in NCS up to a length of 1200 meters. Thereby, the data throughput and the reliability are significantly improved compared to the initial protocol. The new protocol performs much better at smaller Ethernet frame sizes up to about

800 bytes. This is achieved by exchanging the medium access scheme from a 2-way handshake procedure to the Token-Passing procedure. Hence, the timeouts during transmission are reduced significantly, because there are no handshake messages before starting a transmission and no timeouts after a transmission of an Ethernet frame necessary, like they are necessary in the initial protocol.

Additionally, if an error occurs during transmission, it is detected and only a part of the Ethernet frames has to be retransmitted. This is achieved by splitting the Ethernet frames into smaller frames. Such an error can be recognized through checking a CRC being added to every frame. The data rate of the TIA/EIA-485 interface is configurable depending on the network length.

The approach developed in this thesis is capable of dealing with physical redundant TIA/EIA-485 interfaces. Therefore, an interruption of one of the TIA/EIA-485 lines between two micro-controllers or an error of both TIA/EIA-485 connections is detected. After the problem on the communication channel is fixed, both channels can be utilized again for the transmission.

The presented method gives the basis for a reliable tunneling of Ethernet via TIA/EIA-485. As future work, it can be investigated how big Ethernet frames can be processed to increase the transmission speed.

Another open point is to use a variable number of frames per token slot. Therefore, the number of frames being sent in a token slot can be adapted to the size of the Ethernet frame. The token start header would allow such a strategy, because the number of frames is comprised in that header. Further also a changing frame size can be examined, but this creates more overhead as just changing the number of frames in a token.

List of Acronyms

ARCNET Attached Resource Computer NETwork.

AS-i Actuator Sensor Interface.

BI Beacon Interval.

BO Beacon Order.

CAN Controller Area Network.

CAP Contention Access Period.

CCITT Comité Consultatif International Téléphonique et Télégraphique.

CFP Contention Free Period.

CKE Clock Edge.

CKP Clock Polarity.

CPU Central Processing Unit.

CRC Cyclic Redundancy Check.

CSMA Carrier Sense Multiple Access.

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance.

CSMA/CD Carrier Sense Multiple Access with Collision Detect.

DCS Distributed Control System.

EDGE Enhanced Data rates for GSM Evolution.

FCS Frame Check Sequence.

FCST Field Control Station.

FDDI Fiber Distributed Data Interface.

GPRS General Packet Radio Service.

HD Hamming Distance.

IEEE Institute of Electrical and Electronics Engineers.

IP Internet Protocol.

IPC Industrial Personal Computer.

ISO International Organization for Standardization.

IT Information Technology.

LAN Local Area Network.

LLC Logic Link Control.

LON Local Operating Network.

MAC Medium Access Control.

MBP Manchester Bus Powered.

MDC Management Data Clock.

MDIO Management Data Input/Output.

MII Media Independent Interface.

MLT-3 Multi-Level Transmit with 3 Levels.

NCS Network Control System.

NRZ Non-return-to-zero.

OSI Open Systems Interconnection.

PDU Protocol Data Unit.

PHY Physical Layer.

RAM Random-Access Memory.

RPi Raspberry Pi.

RZ Return-to-zero.

SD Superframe Duration.

SDU Service Data Unit.

SMI Serial Management Interface.

SO Superframe Order.

SPI Serial Peripheral Interface.

SSI Synchronous Serial Interface.

TCP Transmission Control Protocol.

TIA/EIA Telecommunications Industry Association/Electronic Industries Alliance.

TTH Token Hold Time.

TTRT Target Token Rotation Time.

UART Universal Asynchronous Receiver Transmitter.

WLAN Wireless Local Area Network.

List of Figures

List of Tables

Bibliography

[1] IEEE Computer Society, *IEEE Std 802.3-2005*, 2005.

[2] Texas Instruments, *RS-422 and RS-485 Standards Overview and System Configurations*, 2010.

[3] Microchip Technology Inc., *Explorer 16 Development Board with 100-pin PIM*, http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=DM240001, Accessed: 2015-05-21.

[4] International Organization for Standardization (ISO), *ISO/IEC 7498-1*, 1994.

[5] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," in *Proc. of the IRE*, vol. 49, no. 1, January 1961, pp. 228–235.

[6] B. Reißenweber, *Feldbussysteme zur industriellen Kommunikation*, 3rd ed. München : Oldenbourg-Industrieverlag, 2009.

[7] C. M. Kozierok, *The TCP/IP-Guide: A Comprehensive, Illustrated Internet Protocols Reference*, 1st ed. No Starch Press, 2004.

[8] V. Cerf, Y. Dalal, and C. Sunshine, "Specification of internet transmission control program," Internet Requests for Comments (RFC), RFC 675, December 1974.

[9] F.-J. Kauffels, *Lokale Netze*, 16th ed. Heidelberg:mitp, 2008.

[10] IEEE Computer Society, *IEEE Std 802-2001*, 2001.

[11] Microsoft, *The OSI Model's Seven Layers Defined and Functions Explained*, https://support.microsoft.com/en-us/kb/103884, Accessed: 2015-05-21.

[12] J. Postel, "Internet protocol," Internet Requests for Comments (RFC), STD 5, September 1981.

[13] DIN, *Measurement and control; PROFIBUS; Process Field Bus; data transmission technic, medium access methods and transmission protocol, service interface to the application layer, management*, DIN 19245-1, April 1991.

[14] Robert Bosch GmbH., *CAN specification version 2.0*, 1991.

[15] A. Pantazi and T. Antonakopoulos, “Equilibrium point analysis of the binary exponential backoff algorithm,” *Computer Communications*, vol. 24, no. 18, pp. 1759–1768, December 2001.

[16] K. S. Kim and L. G. Kazovsky, “Design and performance evaluation of scheduling algorithms for unslotted CSMA/CA with backoff MAC protocol in multiple-access WDM ring networks,” *Information Sciences*, vol. 149, no. 1–3, pp. 135–149, January 2003.

[17] M. Short and M. Pont, “Fault-tolerant time-triggered communication using CAN,” *Industrial Informatics, IEEE Transactions on*, vol. 3, no. 2, pp. 131–142, May 2007.

[18] M. Castro, R. Sebastian, F. Yeves, J. Peire, J. Urrutia, and J. Quesada, “Well-known serial buses for distributed control of backup power plants. RS-485 versus controller area network (CAN) solutions,” in *Proc. of the 28th Annual Conference of the Industrial Electronics Society (IECON)*, vol. 3, November 2002, pp. 2381–2386.

[19] E. Ziouva and T. Antonakopoulos, “CSMA/CA performance under high traffic conditions: throughput and delay analysis,” *Computer Communications*, vol. 25, no. 3, pp. 313 – 321, 2002.

[20] P. Koopman and T. Chakravarty, “Cyclic redundancy code (CRC) polynomial selection for embedded networks,” in *Proc. of the International Conference on Dependable Systems and Networks (DSN)*, June 2004, pp. 145–154.

[21] T. Baicheva, “Determination of the best CRC codes with up to 10-bit redundancy,” *Communications, IEEE Transactions on*, vol. 56, no. 8, pp. 1214–1220, August 2008.

[22] P. Fischer, “Evaluation criteria for communication systems in building automation and control systems,” in *Proc. of the 5th International Conference on Federation of Automatic Control (IFAC)*, July 2003, pp. 17–22.

[23] K. Obermann and M. Horneffer, *Datennetztechnologien für Next Generation Networks*, 2nd ed. Springer Vieweg, 2009.

[24] TIA/EIA, *Commercial Building Telecommunications Cabling Standard*, TIA/EIA-568-B.1, April 2001.

[25] M. Felser and T. Sauter, “The fieldbus war: history or short break between battles?” in *4th IEEE International Workshop on Factory Communication Systems*, August 2002, pp. 73–80.

[26] M. Metter and R. Pigan, *PROFINET - Industrielle Kommunikation auf Basis von Industrial Ethernet Grundlagen*, 2nd ed. Publicis Corporate Publishing, 2007.

[27] T. N. C. Fernando and A. T. L. K. Samarasinghe, “Ethernet frame tunneling over GPRS/EDGE for universal network monitoring,” in *Proc. of the International Conference on Industrial and Information Systems (ICIIS)*, December 2009, pp. 55–61.

[28] J. Lee, J. Lee, Y. Yi, S. Chong, A. Proutiere, and M. Chiang, "Implementing utility-optimal CSMA," in *Proc. of the 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, September 2009, pp. 102–111.

[29] C. Ninagawa and T. Sato, "Transmission reserve table buffer for csma type control networks of embedded systems," in *Proc. of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, July 2009, pp. 140–143.

[30] M. Baz, P. Mitchell, and D. Pearce, "Improvements to csma-ca in ieee 802.15.4," in *Proc. of IEEE 14th International Conference on High Performance Computing and Communication 2012 & IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*, June 2012, pp. 1549–1554.

[31] M. Miskowicz, "Performance analysis of slotted-csma with geometric distribution," in *Proc. of IEEE International Workshop on the Factory Communication Systems (WFCS)*, May 2008, pp. 21–29.

[32] Y. Tay, K. Jamieson, and H. Balakrishnan, "Collision-minimizing csma and its applications to wireless sensor networks," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 6, pp. 1048–1057, August 2004.

[33] D. Miorandi and S. Vitturi, "Analysis of master-slave protocols for real-time-industrial communications over ieee802.11 wlans," in *Proc. of 2nd IEEE International Conference on Industrial Informatics (INDIN)*, June 2004, pp. 143–148.

[34] Y. Ren, Y. Wang, W. Ren, and K. Xia, "Design and realization on master slave distributed network monitoring and control system with high frequency vibrating screen," in *Proc. of International Conference on Test and Measurement (ICTM)*, vol. 1, December 2009, pp. 404–407.

[35] R. Gershon, D. Propp, and M. Propp, "A token passing network for powerline communications," *Consumer Electronics, IEEE Transactions on*, vol. 37, no. 2, pp. 129–134, May 1991.

[36] N. Malcolm and W. Zhao, "The timed-token protocol for real-time communications," *Computer*, vol. 27, no. 1, pp. 35–41, January 1994.

[37] S. Husein, M. Woodward, and S. Szumko, "A high performance token passing network for time critical applications," in *Proc. of IEEE Singapore International Conference on Networks & International Conference on Information Engineering 'Communications and Networks for the Year 2000'*, vol. 1, September 1993, pp. 3–7 vol.1.

[38] D.-W. Kim, H. S. Park, and W. H. Kwon, "The performance of a timer-controlled token passing mechanism with finite buffers in an industrial communication network," *Industrial Electronics, IEEE Transactions on*, vol. 40, no. 4, pp. 421–427, August 1993.

[39] Marvel, *Marvell-88E3016-Fast-Ethernet Data Sheet*, 2008.

[40] Microchip Technology Inc., *PIC24HJXXXGPX06/X08/X10 Data Sheet*, 2009.

[41] *WiringPi*, http://http://wiringpi.com/, Accessed: 2015-05-21.

[42] Microchip Technology Inc., *MPLAB X IDE*, http://www.microchip.com/pagehandler/en-us/family/mplabx/, Accessed: 2015-05-21.