MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Display of physical quantities

Bachelor's Thesis

ONDŘEJ KOSTÍK

Brno, Spring 2022

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Display of physical quantities

Bachelor's Thesis

ONDŘEJ KOSTÍK

Advisor: Ing. Jiří Čulen

Department of Computer Systems and Communications

Brno, Spring 2022


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Ondřej Kostík

Advisor: Ing. Jiří Čulen

Acknowledgements

I want to thank my family and my friends for supporting me during my studies. Special thanks go to my advisor Ing. Jiří Čulen, whose insight and advice have been an immense help while working on this thesis.

Abstract

This thesis aimed to analyze the possibilities of information transmission between the display device and the sensor and summarize the possibilities of information transmission used in embedded devices in general. Furthermore, this thesis aimed to design and implement a device based on the MSP432E411Y-BGAEVM microcontroller to read and display data from the MDG-04 sensor using a serial line and the ModBus-RTU protocol. The results of this thesis are an overview chapter describing the possibilities of information transmission and a functional prototype of the display device.

Keywords

Embedded Systems, SOBHA, VF, VF, a.s., Microcontroller, MCU, MDG-04, MSP432, MSP432E411Y, MSP432E4, Modbus

Contents

List of Tables

List of Figures

Introduction

Although initially designed for pocket calculators [1], microcontrollers have quickly found their use in many fields, and we encounter them practically daily. They have become invaluable for commercial and industrial use due to their versatility, small size, low power requirements, and expandability. A single microcontroller with a few sensors can work as a room thermometer or control a whole assembly line in a factory.

This thesis will focus on microcontrollers' expandability aspect, specifically the different ways a sensor can be connected to a microcontroller, their benefits, shortcomings, and situations to which a specific connection type is best suited.

The VF, a.s., a partner of this thesis, makes, among other things, a radiation monitoring technology, the simplest example of which can consist of a single microcontroller and a sensor. They already have an existing solution using a multi-chip microcontroller but are looking to reproduce the design using a single-chip unit.

Therefore, the goal of this thesis is twofold: to explore different connection options for sensors and microcontrollers and to implement a prototype radiation monitoring device using Modbus-RTU communication protocol on a single-chip microcontroller.

The hardware used to create the prototype device is the Texas Instrument's MSP432E4 microcontroller, Kentec's K350QVG display, and VF's MDG-04 gamma detector.

The first chapter introduces microcontrollers, their history, and their place in the world.

Chapter two summarizes the different information transmission methods available, both at the physical layer and in terms of communication protocols, focusing on the Modbus-RTU protocol.

The third chapter describes the hardware used to create the prototype radiation meter and the initial setup and physical connection between the different hardware components.

A critical part of this thesis, the implementation of the prototype radiation meter itself, is described in the fourth chapter, along with comments on essential segments of the device's source code design. The code itself is added as an electronic attachment to this thesis.

The fifth and final chapter summarizes the goals and results of this thesis and comments on the prototype device's use going forward.

1 Microcontrollers

Microcontrollers are small devices or chips that can provide the full functionality of a regular computer but on a vastly smaller scale. This chapter will introduce microcontrollers' history and some of the most common uses.

1.1 History

The first device, considered a microcontroller, was designed by Gary Boone and Michael Cochran in the early 1970s [2].

This Texas Instruments device, called TMS1802NC, was initially intended for pocket calculators and was quite revolutionary at the time. Compared to a microprocessor, which needed several peripherals to make up a working device, the microcontroller included all those parts on a single chip, making products utilizing these smaller and cheaper to manufacture.

Texas Instruments quickly realized that calculators were the least this invention could accomplish and envisioned embedding microcontrollers in most daily objects, essentially laying the groundwork for the entire embedded systems industry [3].

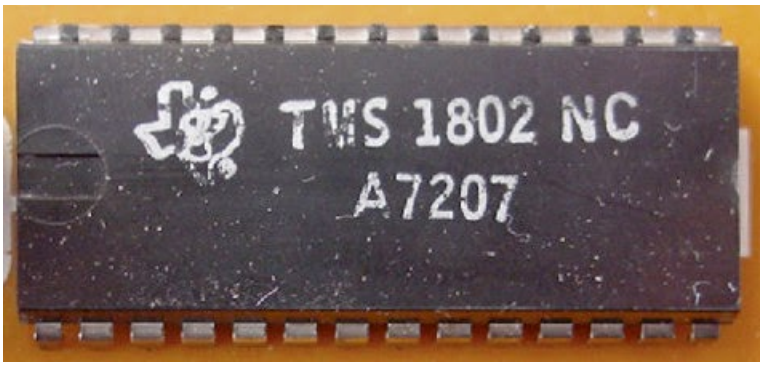

Figure 1.1: TMS1802NC, the first microcontroller [4]

1.2 Applications

Since the beginning, microcontrollers have been designed with embedded applications in mind, so their most significant characteristics are low power requirements and small physical size. This allows them to find their use in almost every industry, from pocket calculators, kitchen scales, or fuel dispensers at gas stations to televisions, smart home appliances, cars, or even clothes. Some people might not even realize that there are probably dozens of microcontrollers in things they own. For example, "it is estimated that today's well-equipped automobile uses more than 50 microcontroller units (MCUs)" [5], and that number is expected to rise with the growing market with electric vehicles and the development of autonomous systems.

However, consumer-oriented devices are not the only place where microcontrollers have found their home. Microcontrollers and programmable logic controllers or FPGAs are used to create distributed control systems that control and monitor every aspect of many industrial processes worldwide. Regulating valves, electrical junctions, or entire machines, as well as taking measurements even at potentially dangerous places, can be done safely and more efficiently from a control room that does not even have to be on site.

2 Transmission of information

Now that the concept of microcontrollers has been established, this thesis will focus on one of their essential characteristics: expandability.

In this context, expandability means the microcontrollers' ability to connect to different peripherals. The setup of such a connection might not always be as straightforward as it could seem. The communication between a microcontroller and a peripheral can be implemented in many ways, using many different physical connections and communication protocols.

This thesis will first look at some of the connections available at the physical layer, then at several protocols that work above the physical layer, and lastly at some standalone solutions that define their own physical layer as well as the protocol above it.

2.1 Lower layers

Providing a physical connection is the groundwork needed for communication and information transmission between two devices. There are several connections available to use on the physical layer when considering microcontrollers and their peripherals. Some of them are going to be mentioned in this section.

2.1.1 RS-422

Officially called ANSI TIA/EIA-422, RS-422 was released in 1975 as the next logical step after the successful RS-232 standard. While it still uses full-duplex mode, the difference over its predecessor is in the use of differential signaling, which enables it to work at higher speeds and over greater distances. The RS-422 also works in full-duplex mode [6].

This standard also has stricter limits considering the number of receivers on a bus, which is limited to ten. Somewhat curious is the situation considering maximum transmission distance. While many sources claim that the maximum transmission distance this standard can handle is about 1200 meters, an application report by Texas Instruments claims that "No restriction on maximum cable length is imposed by the RS-422 standard" [7]. However, with growing dis-

tance, transmission speeds are severely limited, with the transmission speed over distance graph very similar to the newer RS-485 standard (see Figure 2.2).

2.1.2 RS-485

Officially called ANSI TIA/EIA-485, this connection standard was first introduced by the Electronic Industries Alliance (EIA) in 1983 and is still a widely used standard several decades later [7, 8].

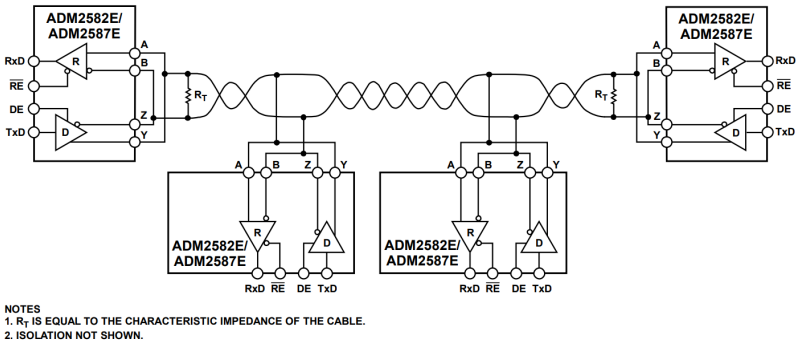

Figure 2.1: RS-485 wiring [9]

Unlike the RS-422, the RS-485 standard is not designed for full-duplex communication and can only use half-duplex mode, which uses three wires, two for the signal lines (see Figure 2.1) and one for the ground connection. The two signal lines usually run in a twisted pair cable, allowing low electrical noise sensitivity [10].

In terms of long-distance connections, the RS-485 standard allows for connection distances over 1200 meters. However, with significantly reduced speeds compared to a short-distance connection, as seen in Figure 2.2.

It also allows for 32 receivers on a single bus and theoretically up to 256 if using repeaters. Because of this, the allowance for long cable runs and low sensitivity to electrical noise, RS-485 is well-suited for large industrial systems that are easy and cheap to implement.

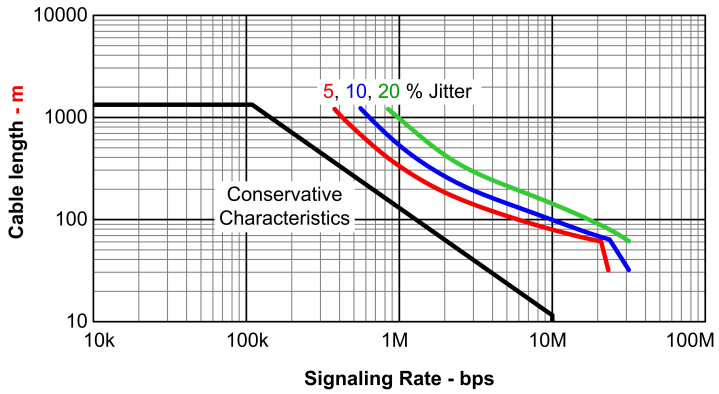

Figure 2.2: RS-485 speed over distance [11]

2.1.3 Ethernet

Ethernet was developed by Xerox engineers in the 1970s, and Figure 2.3 is a hand-drawn diagram by Robert Metcalfe used to present the idea of Ethernet to the National Computer Conference in 1976. In 1980 Xerox, Intel, and Digital Equipment Corporation formed a group called DIX, which was supposed to promote Ethernet as a standard. With the help of the IEEE organization, the first Ethernet standard was released in September 1980. It used a coaxial cable as a bus and achieved speeds up to 10 Mbit/s [12].

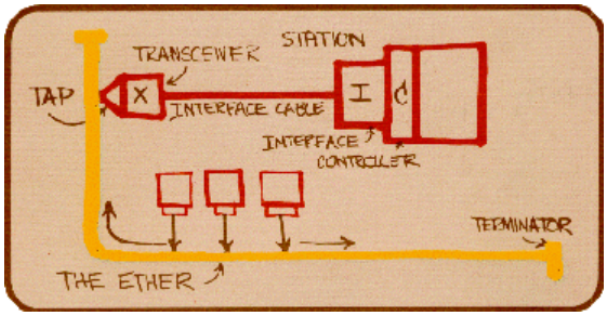

Figure 2.3: Early diagram of the Ethernet [13]

Initially, it was intended for several devices communicating using the Ethernet to connect on a single bus. This approach, however,

created signal collisions that had to be solved. A device connected to Ethernet had to check the bus for ongoing transmissions and transmit only when there was no other activity. However, that does not prevent collisions from happening. In case of a collision, the transmission is stopped, and a signal is sent to notify the other devices. All devices transmitting at that moment wait for a random amount of time before trying the whole process again.

Nowadays, the Ethernet standard is defined by the IEEE 802.3. Most Ethernet connections use a UTP cable, consisting of four unshielded twisted pairs of wires or a fibreoptic cable, and speeds of 1Gbit/s or more are becoming more common by the day. Average network infrastructure has also changed. The move from linear to star topologies that use a separate cable for each device mostly got rid of signal collisions.

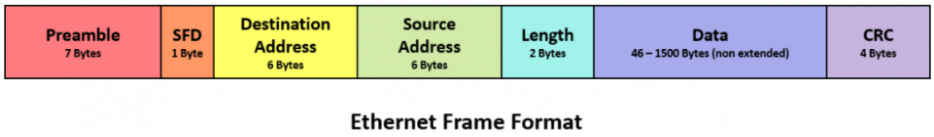

Figure 2.4: Ethernet frame [14]

Ethernet uses Ethernet frames as a structure to send data. Each frame consists of several parts, as seen in Figure 2.4. The preamble indicates a beginning of frame transmission, and SFD is used as a delimiter indicating the start of the frame. Destination and source addresses are MAC addresses of the respective devices. Length represents the length of the frame, while the data part is the actual data we are transmitting. Lastly, there is a CRC, which is used to detect data corruption during transmission.

2.1.4 Current loops

Current loops are simple electronic circuits capable of transferring a single analog value over long distances. A simple current loop (Figure 2.5) consists of only a handful of parts. The power supply, transmitter, receiver, and the wired connection itself.

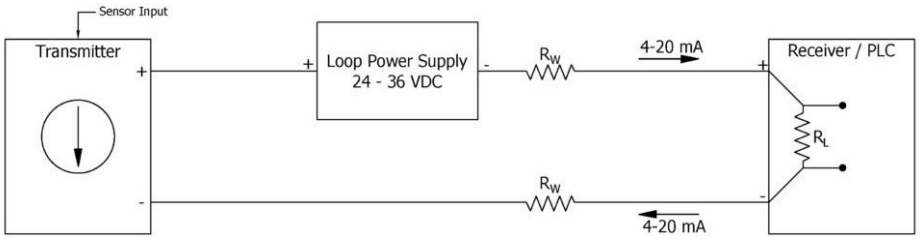

Figure 2.5: 2-wire current loop [15]

The power supply provides power to the loop, which can power its components, mainly the transmitter and the sensor. The transmitter then gets a value from the sensor, converts it to be represented by a value between 4 mA and 20 mA (the current boundaries may be different, but 4–20 mA has become an industry standard), and then regulates the current flow in the circuit. Finally, the receiver measures the current in the circuit and, knowing the sensor's measurement range, converts it back to the original value.

The receiver, in this case, is typically a programmable logic controller (PLC), which, while technically not a microcontroller, is a microcontrollers' alternative used for industrial applications. A microcontroller, however, can also use current loops if a peripheral measuring current in the loop is used.

Current loops might seem too simple to be the dominant standard in the industry still, but there are good reasons for that. The main advantages of current loops (and 4–20 mA loops specifically) include

- insensitiveness to most electromagnetic interference,
- using less wiring than other methods and therefore cheaper initial cost,
- superiority over long distances because the voltage dropoff is not that much of a concern and
- easy fault detection because 4 mA represents a value of 0, so in the event of failure of the power supply or some other critical component, there will not be any current running through the loop.

Although, current loops also have their disadvantages. A single loop cannot transfer data from more than one sensor, and with the number of connected sensors growing, the wiring and insulation costs become relatively high [16].

2.2 Communication protocols

Even if connected with the same technology on the physical layer, devices still might not work together correctly. That is where communication protocols come in. If all devices in a system need to work together correctly, they all have to support and use the same protocol.

2.2.1 Modbus

Modbus is an application layer communications protocol developed by Modicon in 1979, primarily for their PLCs. Nowadays, "Modbus" is a registered trademark of Schneider Electric USA, Inc. – current owner of Modicon brand – and the Modbus protocol specification is maintained by the Modbus.org organization [17].

Modbus was initially developed to be used on top of a serial connection with two main types, RTU (Remote Terminal Unit) and ASCII, with Modbus over TCP/IP, introduced later as the popularity of Ethernet in industrial use grew [17].

Protocol Data Unit (PDU)

Modbus communication revolves around messages sent between a master and a slave when using serial communication or a server and a client when using the Ethernet. The core of these messages is called the PDU. PDU consists only of the function code and the data segment (see Figure 2.6).

There are 255 function codes available, 0 is invalid, and 128—-255 is reserved for exception responses. Some function codes belong to the "Public Function Codes" category and are well defined by the Modbus protocol specification, while others are user-defined [18].

The data segment can contain arguments for the functions or some other data, depending on the specific function and the direction of the communication.

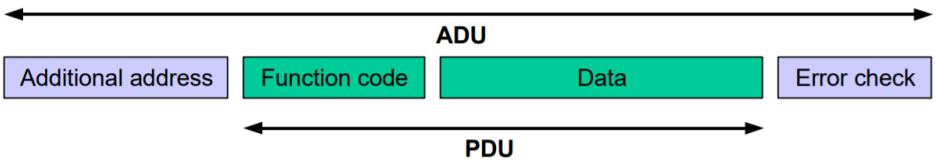

Figure 2.6: Modbus message structure [18]

An example of such communication, taken and translated from VF's Q-7P1-06 internal document summarizing Modbus protocol, can be seen in Figure 2.7, where Device A reads 30 registers from device B's RW register space beginning at address 0x0AB5 [19].

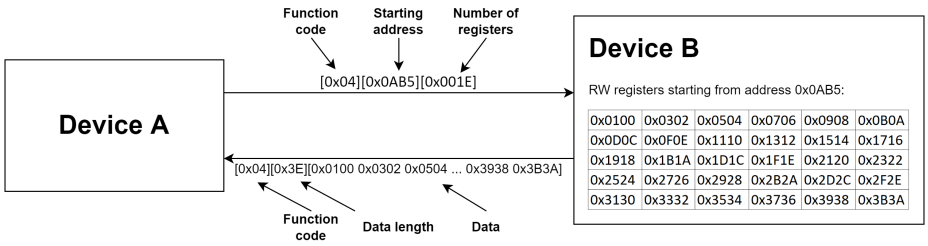

Figure 2.7: Example of Modbus message transaction [19]

Depending on the underlying transmission type, other data is added alongside the PDU to create the Application Data Unit (ADU), such as source and destination device addresses and CRC (also seen in Figure 2.6).

Modbus-RTU

Modbus-RTU sends data in bytes, where each byte contains two 4-bit hexadecimal characters. The format for each byte in RTU mode transmission is one start bit, 8 data bits, one parity bit, and one stop bit. As the specification requires, the default parity is even, but others may be implemented. If no parity is set, then another stop bit takes its place [20].

At the beginning of a Modbus-RTU message frame (Figure 2.8) is a slave address. This is used to declare which device the message is

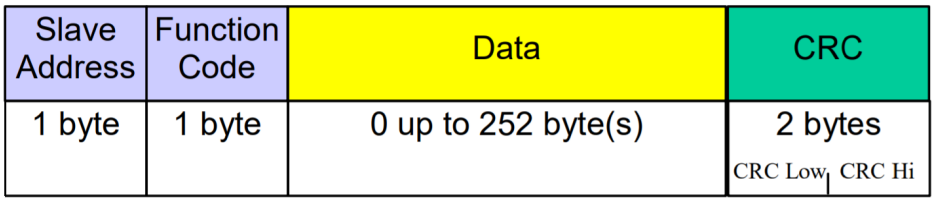

Figure 2.8: Modbus-RTU frame [20]

intended for on the bus. Modbus-RTU allows 247 slaves on a single bus, with address 0 reserved for broadcast and addresses 248–255 reserved for other purposes [20].

To mark the beginning and the end of a message (one frame/ADU), a silent period of at least 3.5 character times is used. A similar mechanism is used to detect any corruptions in the received data. If a silent period of longer than 1.5 character times is detected between two characters, the message is considered incomplete [20].

Modbus-RTU messages also include a CRC (cyclic redundancy check) as the last segment of the frame. It checks the contents of the entire message and is applied independently of parity settings [20].

Modbus-ASCII

Modbus-ASCII uses two ASCII characters for each byte of the message. This mode is intended for use when the communication link or the device does not support the timers needed for Modbus-RTU. A semicolon character is used to declare the start of a message frame and a CRLF to mark the end of a frame [20].

Modbus over TCP/IP

As time went by, a need emerged for Modbus over different than serial connections, which led to Modbus TCP/IP development. At its core still lies the same Modbus PDU, but an MBAP header is added, which consists of four parts:

- Transaction Identifier, which is used for transaction pairing.

- Protocol Identifier, used for intra-system multiplexing. Modbus protocol default identifier is 0.
- Length, which is a number of bytes following, including the PDU.
- Unit Identifier, which serves a similar function as a slave address in a Modbus ADU on a serial line, and is often used when Modbus TCP/IP message goes through an Ethernet gateway and is supposed to continue traveling on a serial line to the destination device [21].

2.2.2 DMX

The DMX512 standard was created in 1986 by the United States Institute for Theatre Technology to control lighting in theatres and other entertainment applications. It uses a master/slave model in which a DMX controller (a master) sends commands to the first DMX device on the bus (a slave). A daisy-chain configuration is used to connect more devices on a single cable run (a channel), where each DMX device acts as a master to the next one, with up to 512 devices on one channel. This helps significantly with wiring because each device had to be connected to the controller using a separate cable before DMX came around [22].

Each device has its own address (usually adjustable by physical switches). When a controller sends commands to a channel, it sends a data frame consisting of a series of bytes. The sequence number of the byte is the address of the target device, and its value is the data intended for this device. Typically, a DMX controller controls the devices, but DMX-USB interfaces are available for controlling via software [22].

DMX uses RS-485 (covered in Subsection 2.1.2) as the underlying connection type but specifies the connectors used. The only connectors permitted by the ANSI E1.11-2008 standard are 5-pin XLR and RJ45, with RJ45 only permitted for fixed installations only accessible to qualified users. Suppose a device manufacturer uses a different connector. In that case, it has to be in a situation where XLR or RJ45 connectors are impossible to use, and an adapter to a 5-bit XLR needs to be provided [23].

2.3 Standalone solutions

Some standards, created with specific applications in mind, define their own physical layer as well as the communication protocol above it. This section will present two examples of such.

2.3.1 CAN

A CAN bus is a communications standard introduced by Bosch in 1986. CAN bus was developed for use in the automotive industry to expand the functionality of the growing number of sensors and electronics embedded in cars and, as a byproduct, reduce the wiring complexity and manufacturing costs [24].

Physical layer

Before CAN bus came around, each sensor had its own bus – or rather a wire – which connected it to the dashboard. CAN bus makes this easier by connecting all the car's sensors using only four wires. These four wires are the power, ground, and the two signal lines, CANL and CANH, usually run in a twisted pair to prevent electrical noise interference. Each device, called a CAN node, connected to the bus has its unique ID, which affects its priority for communicating on the bus. Nodes with lower IDs have a priority if more are trying to transmit simultaneously. That is because CAN defines "dominant" and "recessive" bits, which stands for logical 0 and 1, respectively, with the bus default state being a recessive 1. So in a situation when nodes A and B start transmitting simultaneously if A's ID is lower, it will "overwrite" B's ID on the bus, causing B to detect an error and stop transmitting, while A will continue the transmission without any interruptions [25].

There are more versions of the CAN standard, but in the standard CAN (version 2.0A), each device uses an 11bit identifier, which would result in 2048 possible unique IDs. In reality, there are factors such as transmission speed and distance, limiting the number of nodes on a single bus, and usually, there are a few dozen nodes on a single bus at most. As for transmission speeds and long-distance connections, the CAN bus can transmit at 1 Mbit/s over approximately 40 meters.

However, as seen in the other connection types, transmission speed gets lower with higher distance, so at 500 meters, the transmission speed would be approximately 125 kbit/s [26, 27].

Above the physical layer, CAN messages are called "frames." There are four types of frames.

Data Frames

Data frames are the basic type of frames used to send data on the bus and consist of seven fields: start of frame, arbitration (identifier and RTR bit), control (two reserved bits and data length code), data, CRC, acknowledge (ACK) bit, and end of frame. The specific lengths of these fields differ in the standard (CAN 2.0A) and extended (CAN 2.0B) version of the protocol. The structure of data frames used by CAN 2.0A can be seen in Table 2.1 [28].

Table 2.1: A CAN 2.0A data frame structure

Field	Length (bits)	Description
Start of Frame (SOF)	1	Signals the start of the frame
Identifier	11	Node's unique identifier
Remote Transmission Request (RTR)	1	Dominant in data frames, reccessive in remote frames
Reserved	2	Must be dominant
Data length code	4	Length of data in bytes
Data field	0-8 bytes	Length determined by Data length code
Cyclic redundancy check	15	
CRC delimiter	1	Must be recessive
Acknowledge (ACK)	1	Must be recessive
ACK delimites	1	Must be recessive
End of frame (EOF)	7	Must be recessive

An essential part of data frames is error checking. Besides the CRC, bit-level checks, frame checks, and acknowledgment checks are performed. Errors in CRC or invalid bits in delimiters will set the ACK bit to notify the transmitter of failure [28].

Error Frames

If a node detects an error, it will immediately broadcast an error frame consisting of a six-bit error flag and an eight-bit delimiter. Each node counts the number of errors detected and is in one of three modes based on that number. For values in the range of 1–127, the node is considered fully functional, but at least one error has been detected.

For values in the range of 128–255, the node starts to transmit at a slower rate, and for values above 255, the node is turned off and considered inoperational. Additionally, if no error occurs until the end of a frame, the transmitter considers the transmission error-free, and corrupted messages get retransmitted as soon as the bus is idle [28].

Remote Frames

A node uses remote frames to request data from another node. Its structure is the same as a data frame without the data field [28].

Overload Frames

Overload frame is generated if a node receives messages faster than it can process them. Like an error frame, an overload frame has a six-bit overload flag and an eight-bit delimiter [28].

2.3.2 KNX

The KNX is an open home automation standard created and maintained by the KNX Association CVBA, founded in 1999 by members from three existing associations: European Installation Bus Association, European Home Systems Association, and BatiBUS Club International [29].

KNX devices can communicate over multiple physical connections such as a twisted pair, power-line (sending data over existing electrical wiring), Ethernet cables, or wireless, with twisted pair being the most commonly used. KNX supports up to 50000 devices in one system, with a defined topology [29].

The smallest unit of the KNX system is the device (e.g., a sensor). Devices are linked together on "lines," which usually accommodate up to 64 devices, but with line repeaters installed, up to 255 devices can be on a single line. The next logical unit is an area. Areas are segments of a KNX system consisting of up to 15 lines connected using line couplers. Up to 15 areas can be connected using area couplers to form a complete system [29].

Each device in the system has its unique Individual Address, which is three numbers separated by a dot. These three numbers represent the

area number, the line number, and the sequential number, indicating the device's position on the bus. Area and line couplers also count as devices, and their sequential numbers are always 0 [29].

KNX uses "telegrams" to communicate, but their structure is different based on the underlying physical connection. With twisted-pair being the dominant physical connection used today, the focus will be only on KNX TP (twisted-pair) telegrams [29].

KNX TP telegrams consist of four fields (see Figure 2.9):

- Control field – defines the priority of the telegram.
- Address field – specifies source device's address and destination device's/group's address.
- Data field – contains the telegram's payload.
- Checksum field – used for parity checks [29].

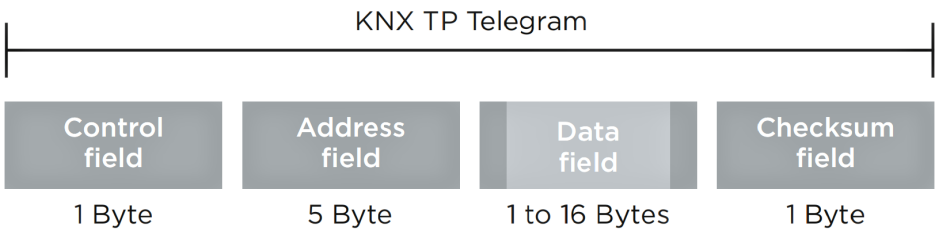

Figure 2.9: A KNX TP telegram structure [29]

Access to the bus is random and event-driven. Each device will start transmitting only if no other devices are, and in case of collisions, a device that transmits a telegram with lower priority will stop its transmission. If priorities are the same, a telegram with a lower source Individual Address is given priority [29].

KNX systems are managed using the KNX Engineering Tool Software, which, among other things, places devices into groups. These groups represent a logical set of devices that interact with each other (e.g., a switch and a relay controlling a light). One device can be in several groups, defining different behaviors based on different events [29].

2.3.3 DALI

DALI is an open standard used to control lighting systems. It was introduced in the late 1990s to replace older 0-10V lighting control systems. The DALI trademark currently belongs to the Digital Illumination Interface Alliance, and the standard is defined by the IEC 62386 by the International Electrotechnical Commission [30, 31].

DALI networks can implement any combination of star, daisy-chain, or tree topologies and contain up to 64 devices. Individual devices are connected on a bus of two wires, DA+ and DA- (both are often labeled just as "DA" because DALI devices are polarity-insensitive).

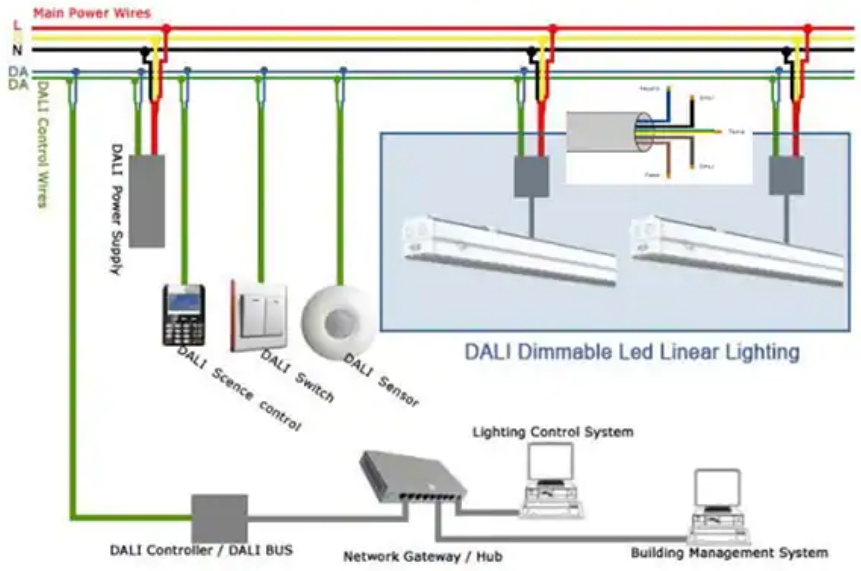

Figure 2.10: An example of DALI system topology [32]

The DALI bus is often run next to the power lines or even as part of the same multi-core cable. The bus provides both signal and bus power, but a DALI-certified power supply is required on each DALI network, where it provides up to 250mA, typically at 16V DC [30, 33].

The speed of the bus is fixed at 1200 bauds, which, combined with relatively high voltages used to transmit data (0 ± 4.5V for logical zero and 16 ± 6.5V for logical one), means the bus is very resistant to electrical interference. Individual bits are sent using the Manchester

encoding, which means that a bit representing a logical one is low for the first half and high for the second, with logical zero being the other way around. The data is sent in frames, which begin with a logical one followed by 8 to 32 bits of data in most-significant bit first order and an idle period of at least 2.45ms to mark the end of the frame [33, 34].

Each device on a DALI network has a unique address 0 to 63, which allows the controller to address an individual device. Devices can also be placed into groups, with up to 16 groups allowed on one network to be controlled simultaneously. Broadcasting is also available. Typical commands include turning lights on or off, dimming them, or setting the fade time, while other commands can request a device's status or place a device into a group [33, 35].

DALI standard is commonly used in larger buildings such as stadiums, airports, or office buildings. It is often connected to a building management system to be centrally controlled and allow easier automation [36].

3 Hardware used

This chapter will offer a short description and characteristics of the hardware used to create the prototype device.

3.1 TI MSP432E4 MCU

The microcontroller used for the device is the MSP432E411Y-BGAEVM made by Texas Instruments Inc. It features an ARM Cortex-M4F CPU, 1MB of flash memory, and 512 MB of SDRAM with a wide variety of peripherals such as Ethernet, USB, CAN, UART, and an integrated LCD controller [37]. Mainly, the LCD controller and the UART are of concern as those are the peripherals used for connecting the display and the sensor.

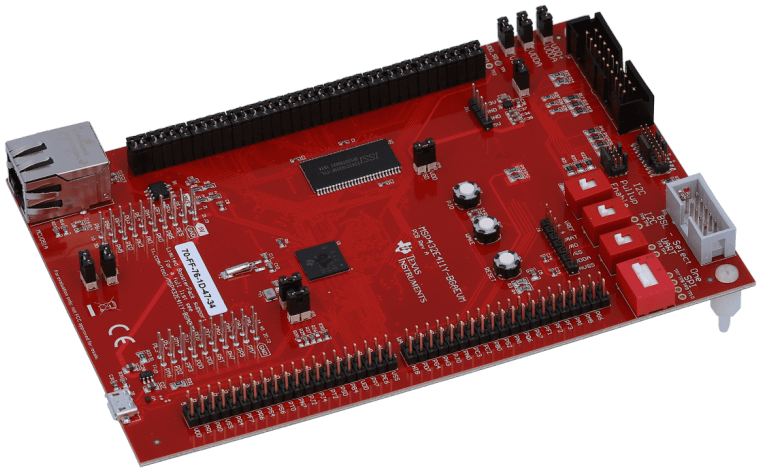

Figure 3.1: The MSP432E411Y-BGAEVM MCU [37]

3.2 Kentec K350QVG display

The display used for the device is the K350QVG-V2-F-04 made by Kentec Electronics Ltd. It is an LCD module with a 3.5" 262K col-

ors TFT display, LED backlight, and a touch panel [38]. The physical adapter used to connect the display to the microcontroller is the KDK350ADPTR-EVM made by Texas Instruments Inc.

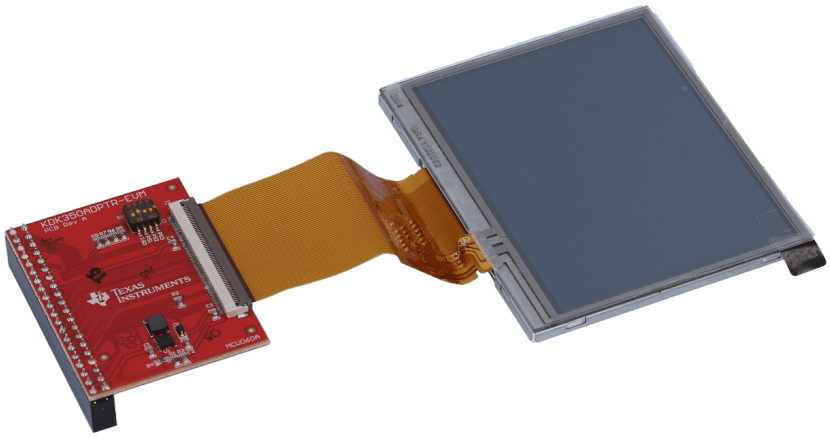

Figure 3.2: The KDK350 adapter with K350QVG display [39]

3.3 VF MDG-04 sensor

The sensor used is an MDG-04 Smart Dose Rate Meter made by VF a. s., designed to measure an air kerma rate. This sensor needs its own power supply and offers an RS-485 connection for communication.

Figure 3.3: The MDG-04 sensor

3.4 Setup and physical connection

The MDG-04 sensor is connected to the microcontroller using one of the microcontroller's UART peripherals in the assembled prototype device. Specifically, the UART6 which uses GPIO Port P pins PP0 and PP1 (found on the J1 header). However, since the sensor uses an RS-485 connection, which cannot be connected directly to a UART, an RS-485 to UART interface is needed (see Figure 3.4). This is because UART and RS-485 use different information transmission methods (differential signaling vs. single-ended signaling).

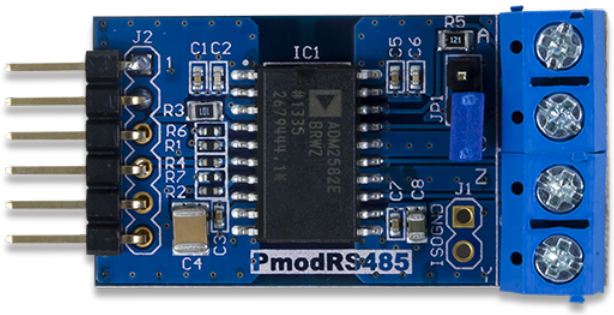

Figure 3.4: The RS-485 to UART interface [40]

The RS-485 to UART interface also requires one additional wire between itself and the microcontroller apart from the standard UART connection. That wire is used for DE/nRE (driver enable/not receive enable) signal, which controls whether the interface is transmitting or receiving on the RS-485 bus. This wire is connected to the GPIO Port P, pin PH4 on the microcontroller (found on the J1 header).

The other part of the assembled prototype device is the display connected to the LCD header on the microcontroller (J9 header) using the provided KDK350 display adapter.

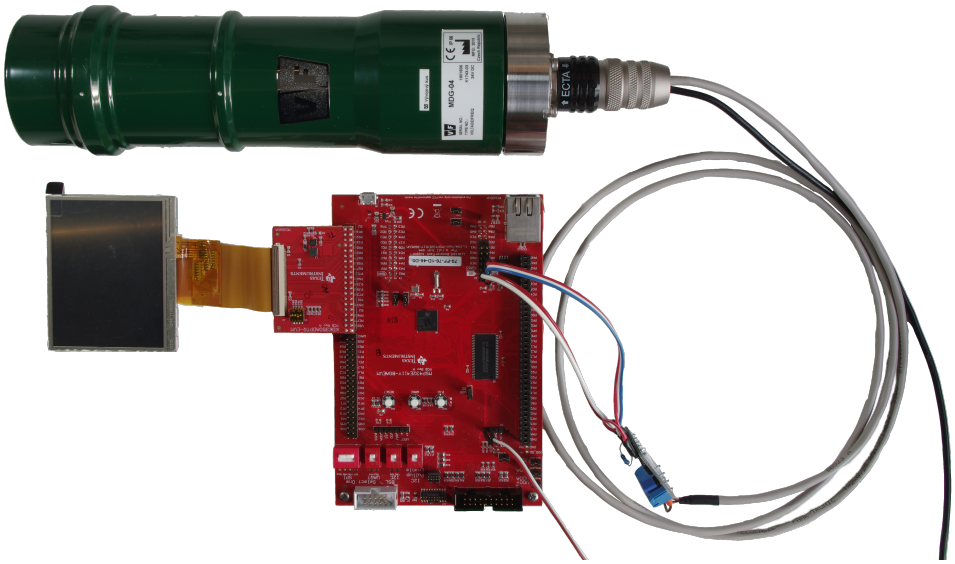

Figure 3.5: The fully assembled prototype device

Lastly, there are two power supplies, one providing 12V DC power to the sensor and another providing 5V DC power to the microcontroller. The fully assembled device (excluding the power supplies) can be seen in Figure 3.5.

4 Device implementation

The device's basic functionality can be separated into two modes: sensor lookup and data reading.

Sensor lookup is the first presented to the user when the device is turned on. It periodically sends a request on the serial line; if there is an answer, the user is prompted to use the found sensor or continue scanning; if there is no answer, the scanned address is incremented, and a new request is sent.

Data reading happens once the user selects a sensor from which to read the data. In the data reading mode, the device periodically updates the data from the sensor, while the GUI displays the data and handles the user interaction.

4.1 Main structure

The code is organized into several header and source files based on the functionality it provides. The pairs of header and source files will be referred to as modules. The main functionality of the device is implemented in the *application*, *communication* and *gui* modules, with a few others providing the implementation details. There are several structures and global variables used throughout the program. Specifically, the *channels_data* and *sensor_info* structures are used to store all the channel data and the sensor address with identification, respectively.

4.2 Communication module

Most of the communication related functionality is placed in the *communication* module and the other modules it includes: *vf_standard* and *modbus*. The communication module contains functions that manage the transmit and receive buffers used by the UART peripheral, defines several communication contexts and states, and functions responsible for creating and sending a request to the sensor and parsing the sensor's response.

The communication contexts defined are *SENSOR_LOOKUP*, *RESET_DOSE*, *FETCH_CH_VALUES* and *FETCH_CH_PARS*. The current context is stored in a global variable *comm_context* and controls which request is sent to the sensor, how often it is sent and how the sensor's response is handled.

There are four communication states defined: *WAIT_TO_SEND*, *SEND_MESSAGE*, *WAIT_TO_RECEIVE* and *MESSAGE_RECEIVED*. The current state is stored in the *comm_state* global variable and tracks the state of the current message transaction.

4.3 Modbus library

The communication module uses this module internally to create and parse Modbus ADUs. As stated earlier in subsection 2.2.1, Modbus has several defined function codes for different use cases. However, there was no use for most of those in this thesis. Therefore the implementation only provides the functionality of two Modbus functions, "Read Input Registers" (function code 4) and "Write Single Register" (function code 6). The library provides two functions for each of them, one that builds a corresponding ADU and another that parses the response ADU. It is essential to state that while the Modbus protocol is defined as using a big-endian byte ordering to transfer data, this library is designed to be used on devices with a little-endian architecture, meaning that it changes data endianness as required. The library also keeps track of the most recently generated ADU's target address and function code. This information is then used in the parsing function to check the validity of the response. In case of an error, a corresponding value is returned.

4.4 GUI module

This module implements the GUI functionality of the device. It is used by calling a single function *gui_update*() which based on the *gui_context*, *update_gui* and *clr_screen* global variables displays the required data on the display. There are several GUI contexts defined: *SENSOR_LOOKUP_GUI*, *MENU_GUI*, *MEASUREMENTS_GUI* and *ERROR_GUI*. The current context is stored in the *gui_context* global

variable and essentially controls which part of the GUI is currently being displayed or updated.

The resulting GUI presented to the user can be seen in the following figures. The sensor lookup GUI while scanning an address can be seen in Figure 4.1 and again in Figure 4.2 after discovering a sensor. Figure 4.3 shows the menu GUI, while Figure 4.4 shows the measurements GUI, and finally, the error GUI can be seen in Figure 4.5.

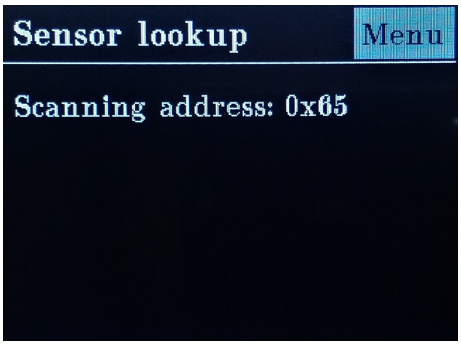

Figure 4.1: Sensor lookup GUI - scanning an address

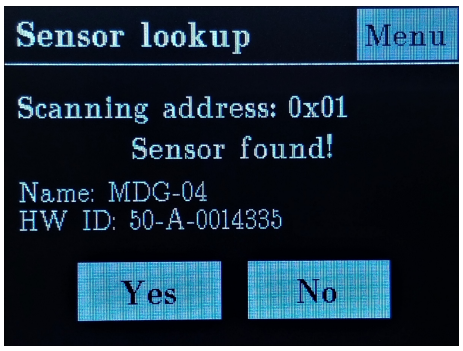

Figure 4.2: Sensor lookup GUI - sensor found

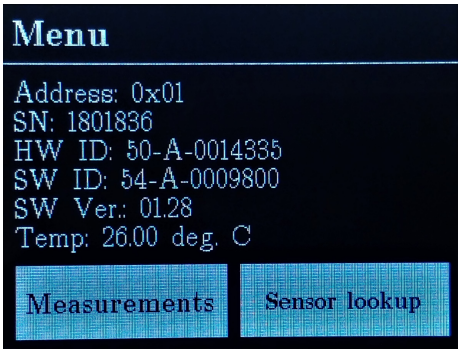

Figure 4.3: Menu GUI

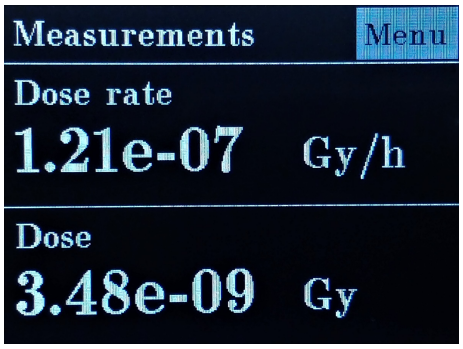

Figure 4.4: Measurements GUI

Figure 4.5: Error GUI

4.5 Application module

This module serves as an entry point to the program. It contains the *main*() function, defines the *channels_data* and *sensor_info* structures used throughout the program and also contains all the interrupt handlers used.

The *main*() function consists of two parts, setup and main control loop. During the setup, all the peripherals used by the program are set up, such as the UART module, GPIO headers and pins, LCD, touchscreen, and timers. Once this happens, the program enters an infinite loop – the main control loop – in which control flow is based on the global state variables. GUI and communication operations are handled independently based on each iteration's current contexts and states.

These global state variables are either set in the main control loop due to some event (e.g., a message was sent or received) or in one of the interrupt handlers.

4.5.1 Interrupts

Several interrupt handlers are defined for different interrupt sources: UART, timers, and touchscreen. There are two timers in use, each with its interrupt handler.

UART interrupt

The UART peripheral is set to raise an interrupt in two situations. When a character was just received, and at the end of transmission, after the last bit of the last character was sent. On both occasions, the same interrupt handler is called. The interrupt handler reacts differently based on the source of the interrupt. If it is handling the "character received" interrupt, and the "message received" timer has not run out yet, it saves the character into the *rx_buffer* and restarts the "message received" timer. If the interrupt source is the end of transmission, the direction of communication is changed. This behavior is handled using an interrupt because it is a time-critical operation as the sensor's response would not be received with the direction set wrong.

"Message received" timer

The "message received" timer is used to detect the end of a message that's being received. It is set to a timeout equivalent to the time it takes to transmit one and a half characters using the serial line as configured in the setup part of the program, conforming to the Modbus standard. This timer is started with the first character of a message received and is reset on each new received character. Once it runs out, the global variable *comm_state* is set to *MESSAGE_RECEIVED*. Cases in which the timeout represents an error in the communication and not an end of a valid message are handled later by the Modbus library's ADU parsing function.

"Send message" timer

This timer runs periodically and serves several purposes. Primarily, it sets the global variable *comm_state* to *SEND_MESSAGE*. Another functionality depends on the mode in which the device currently operates. In the case of sensor lookup, the occurrence of this timer's interrupt means that no response came back from the currently probed address, so this address is increased. In the data reading mode, if the *current_comm_state* is *WAIT_TO_RECEIVE*, the *comm_error_counter* is increased as such situation indicates that no response came from the currently used sensor.

Touchscreen interrupt

This interrupt is raised whenever the touchscreen registers a contact. However, the handler reacts only in the event of contact loss (i.e., the user lifts their finger from the touchscreen). Typically this handler checks where on the screen was the contact registered, and if it happened over a button that is currently drawn on the screen, the *gui_context* changes. In some instances, however, the handler also changes the *comm_context* or changes the timeout value of the "send message" timer. This happens when the user switches between the data reading and sensor lookup modes because these events also influence communication.

5 Results

This thesis aimed to explore the different ways of transmitting information and create a prototype device to read the radiation monitoring data from the MDG-04 sensor.

The information transmission methods were presented in the second chapter. Most of them were briefly explained, with an extra emphasis on the Modbus-RTU communication protocol as it was used later in the device's implementation.

Then, the implemented prototype device was introduced with an overview of the physical setup and the software functionality.

Therefore, the result of this thesis is mainly the second chapter summarizing the available information transmission methods and the source code used to create the prototype radiation monitoring device. The prototype device is perhaps the more important of the two as it is supposed to be used as a basis for a new product line that the VF, a.s. is planning.

Bibliography

1. *Microcontroller* [online]. Online: The Engineering and Technology History Wiki, 2017 [visited on 2021-10-23]. Available from: https://ethw.org/Microcontroller.
2. BOONE, Gary W.; COCHRAN, Michael J. *Variable function programmed calculator*. US. US Patent, 4074351. 1977.
3. *STANDARD CALCULATOR ON A CHIP ANNOUNCED BY TEXAS INSTRUMENTS* [online]. Online: Texas Instruments Incorporated, 1971 [visited on 2021-11-16]. Available from: https://web.archive.org/web/20060218021723/http://www.ti.com/corp/docs/company/history/calcchip.shtml.
4. SHIRRIFF, Ken. *The Texas Instruments TMX 1795: the (almost) first, forgotten microprocessor* [online]. Online, 2015 [visited on 2021-11-16]. Available from: http://www.righto.com/2015/05/the-texas-instruments-tmx-1795-first.html.
5. FLEMING, Bill. Microcontroller Units in Automobiles. *IEEE Vehicular Technology Magazine*. 2011, vol. 6, no. 3, pp. 4–8. Available from DOI: 10.1109/MVT.2011.941888.
6. *Introduction to RS-422: serial data standard* [online]. Online: Electronics Notes [visited on 2022-02-13]. Available from: https://www.electronics-notes.com/articles/connectivity/serial-data-communications/rs422-basics-tutorial.php.
7. SOLTERO, Manny; ZHANG, Jing; COCKRIL, Chris; ZHANG, Kevin; KINNAIRD, Clark; KUGELSTADT, Thomas. *RS-422 and RS-485 Standards Overview and System Configurations* [online]. Online: Texas Instruments Incorporated, 2010 [visited on 2021-11-14]. Available from: https://www.ti.com/lit/an/slla070d/slla070d.pdf.
8. KUGELSTADT, Thomas. *The RS-485 Design Guide* [online]. Online: Texas Instruments Incorporated, 2008 [visited on 2022-03-07]. Available from: https://www.ti.com/lit/an/slla272d/slla272d.pdf.

9. *Signal and Power Isolated RS-485 Transceiver with ±15 kV ESD Protection* [online]. Online: Analog Devices, Inc., 2018 [visited on 2022-02-23]. Available from: https://www.analog.com/media/en/technical-documentation/data-sheets/adm2582e_2587e.pdf.

10. KELLY, Jason. *RS-485 Serial Interface Explained* [online]. Online: CUI Devices [visited on 2021-11-14]. Available from: https://www.cuidevices.com/blog/rs-485-serial-interface-explained.

11. KUGELSTADT, Thomas [online]. Online: Texas Instruments Incorporated, 2012 [visited on 2021-11-14]. Available from: https://e2e.ti.com/support/interface-group/interface/f/interface-forum/207128/rs485-speed-vs-distance.

12. SPURGEON, Charles. *Ethernet: the definitive guide*. O'Reilly, 2000. ISBN 9781565926608.

13. METCALFE, Robert M. [online]. Online: IEEE, 1976 [visited on 2021-12-13]. Available from: https://www.ieee802.org/3/ethernet_diag.html.

14. *101 Series: Ethernet Back to Basics* [online]. Online: Fluke Corporation, 2019 [visited on 2021-12-13]. Available from: https://www.flukenetworks.com/blog/cabling-chronicles/101-series-ethernet-back-basics.

15. SEANBO. *Understanding current loop output sensors – Electronics World* [online]. Online: Fuentitech.com, 2021 [visited on 2021-10-31]. Available from: https://fuentitech.com/understanding-current-loop-output-sensors-electronics-world/26342/.

16. HERCEG, Edward. *4 to 20 mA current loops made easy* [online]. Online: WTWH Media LLC, 2020 [visited on 2021-10-31]. Available from: https://www.sensortips.com/featured/4-to-20-ma-current-loops-made-easy/.

17. WILLIAM L. MOSTIA Jr., P.E. *Introduction to Modbus* [online]. Online: Control Global, 2019 [visited on 2021-12-26]. Available from: https://www.controlglobal.com/articles/2019/introduction-to-modbus/.

18. *MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3* [online]. Online: Modbus Organization, 2012 [visited on 2021-12-26]. Available from: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.

19. *Komunikace zařízení RMS protokolem Modbus*. VF, a.s., 2017.

20. *MODBUS over serial line specification and implementation guide V1.02* [online]. Online: Modbus Organization, 2006 [visited on 2021-12-26]. Available from: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf.

21. *MODBUS Messaging on TCP/IP Implementation Guide V1.0b* [online]. Online: Modbus Organization, 2006 [visited on 2021-12-26]. Available from: https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf.

22. HAREENDRAN, T.K. *An Introduction to the Digital Multiplex Protocol* [online]. Online: ASPENCORE [visited on 2021-12-30]. Available from: https://www.electroschematics.com/an-introduction-to-the-digital-multiplex-protocol/.

23. *American national standard E1.11-2008: entertainment technology USITT DMX512-A, Asynchronous serial digital data transmission standard for controlling lighting equipment and accessories.* [Online]. Online: Entertainment Services, Technology Association, and American National Standards Institute, 2008 [visited on 2021-12-30]. Available from: https://tsp.esta.org/tsp/documents/docs/ANSI-ESTA_E1-11_2008R2018.pdf.

24. *History of CAN technology* [online]. Online: CAN in Automation [visited on 2021-12-12]. Available from: https://www.can-cia.org/can-knowledge/can/can-history/.

25. CORRIGAN, Steve. *Introduction to the Controller Area Network (CAN)* [online]. Online: Texas Instruments Incorporated, 2002 [visited on 2021-12-12]. Available from: https://www.ti.com/lit/an/sloa101b/sloa101b.pdf.

26. *CAN Physical Layers* [online]. Online: Kvaser AB [visited on 2021-12-12]. Available from: https://www.kvaser.com/about-can/the-can-protocol/can-physical-layers/.

27. *ISO1050 Isolated CAN Transceiver* [online]. Online: Texas Instruments Incorporated, 2009 [visited on 2021-12-12]. Available from: https://www.ti.com/lit/ds/symlink/iso1050.pdf.

28. FREUDENBERG, James S. Controller Area Network (CAN). In: 2008.

29. *KNX BASICS* [online]. Online: KNX Association cvba [visited on 2021-12-30]. Available from: https://www.knx.org/wAssets/docs/downloads/Marketing/Flyers/KNX-Basics/KNX-Basics_en.pdf.

30. *What is DALI ?* [Online]. Online: NVC International Holdings Limited [visited on 2022-02-23]. Available from: https://www.nvcuk.com/technical/what-is-dali-/460.htm.

31. *IEC 62386 - the international standard for DALI technology* [online]. Online: Digital Illumination Interface Alliance [visited on 2022-02-23]. Available from: https://www.dali-alliance.org/dali/standards.html.

32. SCHWEBER, Bill. *Understanding and Applying the New Standard Connectors for Indoor & Outdoor LED-Based Lighting* [online]. Online: Digi-Key Electronics, 2021 [visited on 2022-02-23]. Available from: https://www.digikey.com/en/articles/understanding-and-applying-the-new-standard-connectors.

33. *DALI Manual* [online]. Online: DALI AG, 2001 [visited on 2022-02-23]. Available from: https://web.archive.org/web/20130627012349/http://www.dali-ag.org/c/manual_gb.pdf.

34. *Decoding a Manchester Encoded Signal* [online]. Online: Digilent [visited on 2022-02-23]. Available from: https://digilent.com/reference/test-and-measurement/guides/manchester-encoding.

35. *Digitally Addressable Lighting Interface (DALI) Unit Using the MC68HC908KX8 Designer - Reference Manual* [online]. Online: Motorola, Inc., 2002 [visited on 2022-02-23]. Available from: https://www.nxp.com/files-static/microcontrollers/doc/ref_manual/DRM004.pdf.

36. *DMX vs. DALI Lighting Control: Which One to Choose?* [Online]. Online: RC Lighting Limited [visited on 2022-02-23]. Available from: https://rclite.com/blog/dmx-vs-dali-lighting-control/.

37. *MSP432E411Y-BGAEVM* [online]. Online: Texas Instruments Incorporated [visited on 2022-03-24]. Available from: https://www.ti.com/tool/MSP432E411Y-BGAEVM.

38. *K350QVG-V2-F* [online]. Online: Kentec Electronics Limited, 2012-05-28 [visited on 2022-03-24]. Available from: http://www.kentecdisplay.com/uploads/soft/Products_spec/K350QVG-V2-F-04.pdf.

39. *KDK350ADPTR-EVM* [online]. Online: Texas Instruments Incorporated [visited on 2022-03-24]. Available from: https://www.ti.com/tool/KDK350ADPTR-EVM.

40. *Pmod RS485 Reference Manual* [online]. Online: Digilent Incorporated [visited on 2022-03-28]. Available from: https://digilent.com/reference/pmod/pmodrs485/reference-manual.

A An appendix

As a result of this thesis, the source code used to run the prototype device is attached in the *source_code.zip* file. This file also contains a *setup.md* file containing instructions on the requirements and steps needed to import and use this project.