Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Anders Vatland

Wireless M-Bus communication between equipment from different vendors

Trådløs M-Bus kommunikasjon mellom utstyr fra forskjellige leverandører

Master's thesis in Cybernetics and Robotics

Supervisor: Geir Mathisen

March 2020

NTNU
Norwegian University of Science and Technology

Anders Vatland

Wireless M-Bus communication between equipment from different vendors

Trådløs M-Bus kommunikasjon mellom utstyr fra forskjellige leverandører

Master's thesis in Cybernetics and Robotics
Supervisor: Geir Mathisen
March 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

MASTER THESIS DESCRIPTION

Candidate:	**Anders Vatland**
Course:	**TTK4900 Engineering Cybernetics**
Thesis title (Norwegian)	**Trådløs M-Bus kommunikasjon mellom utstyr fra forskjellige leverandører**
Thesis title (English):	**Wireless M-Bus communication between equipment from different vendors**

Thesis desctiption: Fresh tap water is regarded as a limited resource, which is delivered to the end users via a network of pipes, a critical infrastructure. This infrastructure has a very varying condition, with from time to time leakages, penetration of contamination and line breaks. In order to achieve information of the flow of water in the tubes, we will gather information from the end users water meters along the water tubes. The modern water meters are using wireless M-Bus (wM-Bus) for communication, and we intend to remotely read these meters in near real time.

The objective of this thesis will be to use a general wM-Bus kit for reading measurements from a commercial meter.

The tasks will be:

1. Conduct a litterary study concerning wM-Bus and remotely reading of meters.
2. Suggest and investigate test setups for remotely reading of commercial wM-Bus meters.
3. As far as time permits, implement and test out a system for remotely reading of commercial wM-Bus meters.

Start date: August 28th, 2019
Due date: Mars 16th, 2020

Thesis performed at: Department of Engineering Cybernetics
Supervisor: Professor Geir Mathisen, Dept. of Eng. Cybernetics

Abstract

Large amounts of water are lost due to pipe-breaks, contamination, and leakages in the water supply network. Modern utility meters may be used to keep track of the state of the network, and to predict equipment failure. These meters use Wireless M-Bus, a field proven standard, to communicate with data collectors. However, the standard has been left flexible in order to suit regulations across regions, which may introduce variations across solutions.

The following thesis investigates the use of equipment from different vendors, by implementing and testing a general WM-Bus kit in order to receive telegrams from a commercial meter. Several setups were implemented with the purpose of verifying that the equipment functioned as intended, and to test and experiment with different configurations.

The results show that the proposed system was unable to receive telegrams from a commercial utility meter, due to problem with the WM-Bus mode used. An explanation was found, showing that the software used was inoperable with the commercial meter.

The thesis concludes that the proposed system is not eligible for reception of telegrams from the commercial meter, but shows great promises of being able to with specific changes to the software stack. A proposal for a promising configuration of the system has been given, based on the experiences from the demonstration project.

Sammendrag

Rørbrudd, forurensing og lekkasje er årsaker til store tap av vann i vannforsyningsnettet. Feilene kan derimot bli oppdaget ved bruk av moderne målere som forutser feilene før de inntreffer. Disse målerne kan bruke Trådløs M-Bus, som er en hyppig brukt og testet standard for kommunikasjon med datainnsamlere. På grunn av at standarden skal passe til forskjellige land og regioner har den blitt laget med stor fleksibilitet. Denne fleksibiliteten kan medføre problematikk når utstyr skal brukes sammen.

Denne masteroppgaven undersøker sameksistens av utstyr fra forskjellige leverandører, ved å implementere og teste en generell WM-Bus datainnsamler for å motta telegrammer fra en kommersiell m. Flere oppsett ble implementert for å verifisere at utstyret fungerte som det skulle, og for å teste og eksperimentere med forskjellige konfigurasjoner.

Resultatene viser at det foreslåtte systemet ikke klarte å motta telegrammer fra måleren. Forklaringen p dette ble senere oppdaget, og skyldes at programvaren til demonstreringsprosjektet ikke er kompatibel med den kommersielle måleren.

Oppgaven konkluderer med at systemet ikke er i stand til å motta telegrammer fra måleren. Det antas derimot at det er gode muligheter for å klare det med spesifikke endringer i programvaren. Et forslag til hvordan dette kan gjennomføres er oppgitt i slutten av oppgaven, basert påerfaringer fra dette prosjektet.

Preface

This thesis is written for the Department of Engineering Cybernetics(ITK), a division of the Faculty of Information Technology and Electrical Engineering(IE) at the Norwegian University of Science and Technology (NTNU). The project is carried out with equipment and assistance from ITK.

This thesis has one supervisor, Prof. Geir Mathisen from ITK, NTNU. Geir has provided the author with direction and structure of the thesis, as well as technical insights and support for the demonstration project.

I would like to thank Geir for his support throughout the project, he has been extremely helpful. I would also like to thank Ellen Beate Hove in the IE Faculty Administration, and my parents Arild Vatland and Åse Elisabeth Vatland for their valuable support and guidance in difficult times. Lastly, I would like to thank my loving girlfriend, Nicoline Myhre Nedza, for her support and patience throughout my work on this thesis. I would not have been able to complete my thesis without her.

Table of Contents

List of Tables

List of Figures

Abbreviations

Symbol	=	definition
API	=	Application Programming Interface
APL	=	Application layer
AMR	=	Automatic Meter Reading
DLL	=	Data Link Layer
ELL	=	Extended Link Layer
PHY	=	Physical Layer
M-Bus	=	Meter Bus
MCU	=	Microcontroller Unit
EN	=	European Norm
WM-Bus	=	Wireless Meter-Bus
HAL	=	Hardware abstraction layer
RF	=	Transceiver

Chapter 1

Introduction

1.1 Background and motivation

Fresh tap water is considered a limited resource, which is shared between industry, government facilities, and private households. The water is delivered by the water supply network, a critical infrastructure with varying conditions across regions. Leakages, contamination, and pipe breaks contribute to a large loss of clean water, and large expenses in emergency repairs. In order to diminish the loss of water and increased costs, it is necessary to detect changes in flow and pressure and to be able to predict failures in equipment used in the water supply network. Such predictions will give the opportunity to respond to incidents before they happen. In order to remain updated on the state of the water supply network, and to be able to predict failures, it is necessary to collect data from a large amount of points spread across huge areas.

Trondheim municipality wishes to install an *Automatic Meter Reading (AMR)* solution to monitor the water usage of its inhabitants and businesses to save money on billing expenses. Such metering equipment may also include a pressure sensor, and the opportunity cost of installing them simultaneously is likely lower than installing the equipment separately. Modern metering equipment has the option of using **Wireless M-Bus (WM-Bus)** to communicate with the data collection infrastructure, and thus data collectors may be strategically placed to cover large areas of the water supply network, instead of adding data collection infrastructure to each single meter. This is an option to be considered if or when Trondheim municipality installs their AMR solution.

WM-Bus is a largely used and field proven standard for the remote reading of utility meters. As the standard has been modified to suit the regulations of regions and needs of customers, the standard has been left highly flexible. This flexibility introduces variations across solutions, and may cause problems when using equipment from different manufacturers. Based on this information, this thesis seeks to investigate the use of WM-Bus equipment from different manufacturers together, as it may be necessary or cheaper to

solve the above problem. In order to investigate interoperability of WM-Bus equipment in this thesis, a general WM-Bus kit and a commercial pressure meter has been acquired. The goal is to implement a demonstration project where the kit is used as a data collector, which will receive telegrams from the commercial meter and sends the data to a computer.

1.2 Limitations

The system presented in this thesis is limited to be a *proof-of-concept* for the use of 3^{rd} party equipment to receive telegrams from an existing set of deployed Kamstrup PressureSensor meters operating in WM-Bus mode C1. Thus it is limited to the use of an EZR32WG development kit as a data-collector using the STACKFORCE WM-Bus protocol stack mode C1. The implemented software may be portable to other ARM architectures, but this is not tested by the author. As the scope of this project is to successfully receive telegrams from a commercial meter, important factors to a real world system such as range, power consumption and security has been mentioned, but is not evaluated thoroughly. The stack limits the experimentation heavily as it mostly consists of binary/object code. The meters are limited to a Kamstrup PressureSensor and several ”dummy”-meters using the same hardware and software as the collector. Furthermore, the documentation on the PressureSensor delivered by Kamstrup is heavily limited, resulting in a lot of guesswork and trial-and-error.

1.3 Disposition

Chapter 2: Theory presents the Wireless M-bus specification and the STACKFORCE protocol stack.

Chapter 3: Literature review explores previous attemts at a similar project, and other uses of the Wireless M-Bus protcol

Chapter 4: Specification presents the system specifications and acceptance criteria.

Chapter 5: Design presents an overview of the system design. There are 4 different setups presented in this chapter, each of which will be described in detail.

Chapter 6: Implementation, testing and results explains how the system was implemented, the tests conducted with each setup is presented, and the results are presented.

Chapter 7: Discussion discusses the results of the tests from the different setups, and the system as a whole.

Chapter 8 and 9: Conclusion and Further work concludes the project, and presents options for what could be worked on in the future.

Chapter 2

Theory

2.1 Introduction

This chapter describes the necessary theory to better understand the development choices made by the author, and the functionality of the system. The WM-Bus protocol will be described in detail, and an overview of the software stack provided by STACKFORCE used in the demonstration project will be presented.

2.2 The Wireless M-Bus protocol

This section will give a brief overview of the WM-Bus protocol. The various modes will be described, and the telegram formats will be explained.

2.2.1 General

The Wireless M-Bus protocol is a wireless expansion of the wired M-Bus, which is a European Norm (EN), EN 13757 to be specific, that specifies the M-Bus standard. EN 13757 describes the M-Bus protocol in its entirety in seven different parts, shown in Table 2.1.

Part	Description
EN 13757 - 1	Data exchange
EN 13757 - 2	Wired M-Bus communication
EN 13757 - 3	Application protocols
EN 13757 - 4	Wireless M-Bus communication
EN 13757 - 5	Wireless M-Bus relaying
EN 13757 - 6	Local bus
EN 13757 - 7	Transport and security services

Table 2.1: The parts of EN 13757

Parts 1-3 describe the basis for creating an operational M-Bus application. Parts 4-7 describe additional features and functionality for specific usages. Relevant to this thesis is part 4, which specifies the radio expansion of M-Bus.

The Wireless M-Bus is specified in EN 13757 part 4, hereby referred to as EN 13757-4. EN 13757-4 gives a complete specification of a wireless application of the M-Bus standard. The wireless protocol is designed to operate on licence free ISM bands.

The M-Bus protocol stack implements a minimum of three abstraction layers, following the Open Systems Interconnection model (OSI model). The OSI model is an attempt at abstracting and standardizing different partitions of a computer system designed to communicate with other systems, into defined layers, by the International Organization for Standardization (ISO) [8]. The model presents seven layers of abstraction, divided as shown in Figure 2.1.

OSI model				
Layer			Protocol data unit (PDU)	Function[14]
Host layers	7	Application	Data	High-level APIs, including resource sharing, remote file access
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
	1	Physical	Symbol	Transmission and reception of raw bit streams over a physical medium

Figure 2.1: OSI Model abstraction layers. Figure taken from [8]

In the M-Bus protocol, networking is not used, and as a result, the layers 4, 5, and 6 are not implemented. The layers of M-bus can be compared to the OSI model in the following way:

OSI model	M-Bus
Application layer	Application layer
Presentation layer	
Session layer	
Transport layer	Used for additional security features
Network layer	
Data link layer	Data link layer
Physical layer	Physical layer

Table 2.2: M-bus standard related to the OSI model

The WM-Bus protocol replaces the Data Link Layer (DLL) and Physical Layer (PHY) layers. If wireless relaying (EN 13757-5) is used, it implements the Network layer.

2.2.2 WM-bus modes

WM-Mus supports a variety of modes, specified to operate on a variety of different frequencies and transfer rates. Each mode is assigned a combination of a letter and a number, where the number indicates whether the mode is used in uni- or bidirectional communication. The number 1 means the mode is unidirectional, and 2 is bidirectional. Below, in Figure 2.2 and Figure 2.3, is an overview of the most common modes used for WM-Bus.

Mode	Description
$S1, S2$	In the **S**tationary mode, the metering devices send their data several times a day. In this mode, the data collector may save power as the metering devices send a wakeup signal before transmitting their data.
$T1, T2$	In the Frequent **T**ransmit mode, the metering devices periodically send their data to collectors in range. The interval is configurable in terms of several seconds or minutes.
$C1, C2$	**C**ompact mode. This mode is similar to mode T but it allows for transmission of more data within the same energy budget and with the same duty cycle. It is suitable for walk-by and/or drive-by readout. The common reception of mode T and mode C frames with a single receiver is possible.
$N1(a-f), N2(a-f)$	**N**arrowband communication mode for long range transmissions.

Figure 2.2: The Wireless M-Bus Modes

Mode	Meter		Collector		Data Rate	Endocing	Modulation	Frequency [MHz]
	RX	TX	RX	TX				
$N1a, N2a$	✓	✓	✓	✓	4.8 kbit/s	NRZ	GFSK	169.406250
$N1b, N2b$	✓	✓	✓	✓	4.8 kbit/s	NRZ	GFSK	169.418750
$N1c, N2c$	✓	✓	✓	✓	2.4 kbit/s	NRZ	GFSK	169.431250
$N1d, N2d$	✓	✓	✓	✓	2.4 kbit/s	NRZ	GFSK	169.443750
$N1e, N2e$	✓	✓	✓	✓	4.8 kbit/s	NRZ	GFSK	169.456250
$N1f, N2f$	✓	✓	✓	✓	4.8 kbit/s	NRZ	GFSK	169.468750
$T2$	✓			✓	32.768 kcps	Manchester	FSK	868.30
$T1, T2$		✓	✓		100 kcps	3-out-of-6	FSK	868.95
$S1, S2$	✓	✓	✓	✓	32.768 kcps	Manchester	FSK	868.30
$C2$	✓			✓	50 kcps	NRZ	GFSK	869.525
$C2$				✓	32.768 kcps	Manchester	FSK	868.30
$C2$			✓		100 kcps	3-out-of-6	FSK	869.95
$C1, C2$		✓	✓		100 kcps	NRZ	FSK	868.95

Figure 2.3: Mode parameters

The S,T, and C modes are also defined at the 433MHz frequency to be used in markets where the 868MHz frequencies are not available or illegal.

As one can see from Figure 2.3, there is a trade-off between range and and transmission rate between the modes S and T. S gives longer range at the cost of fewer transmissions per day, whilst T mode allows more frequent transmits, but at the cost of a shorter range. Notably, the C mode, which is the newest addition to the modes, combines the best part of the two modes, allowing frequent transmissions at a long range. This is due to the support of NRZ encoding from newer RF chips, which is les power intensive, and thus allows more frequent transmissions within the same energy budget.

2.2.3 Telegrams

Frame formats

Wirless M-Bus supports two different frame formats, ”Frame format A” and ”Frame format B”. These are described by the figures 2.4 and 2.5 below.

First block							Second block			Optional block(s)	
Length	Ctrl	Manuf.	Adress	Version	Type	CRC	CI	Data field	CRC	Data	CRC
1 byte	1 byte	2 bytes	4 bytes	1 byte	1 byte	2 bytes	1 byte	Up to 15 byte	2 bytes	up to 16 byte	2 bytes
Data Link Layer Block							Application Layer Block(s)				

Figure 2.4: Frame format A. Adapted from [1]

As one can see from Figure 2.4, the telegrams consists of a minimum of two blocks. The first block contains information related to the DLL. The second block contains information related to the application protocol used and the selected layer. This will be described further in ”CI-Field” below.

First block						Second block			Optional block	
Length	Ctrl	Manuf.	Adress	Version	Type	CI	Data field	CRC	Data	CRC
1 byte	1 byte	2 bytes	4 bytes	1 byte	1 byte	1 byte	Up to 115 bytes	2 bytes	up to 126 bytes	2 bytes
Data Link Layer Block						Application Layer Block(s)				

Figure 2.5: Frame format B. Adapted from [1]

Frame format B is relatively similar to format A, the difference being that it does not contain a CRC field in the first block, and that it has larger data fields in subsequent blocks.

Length specifies the length of the telegram. It specifies the number of bytes in the following fields, excluding the CRC bytes.

Control (Ctrl) contains information on the type of frame used. Several ”function codes” are available for usage. The function code specifies whether the sender is a meter or collector, and what type of communication is ongoing, i.e alarms, ACK's[1], commands. The function code used in this project is 4_h: Send application data without request(Send/no reply). As the project only uses a unidirectional communication mode, this field is not relevant to explain further.

Manufacturer (Manuf.) specifies a three letter manufacturer code. The code is calculated based on the following formula:

$$\begin{aligned}\text{Manuf. id} =&[ASCII(1stletter) - 64] * 32 * 32 + \\ &[ASCII(2ndletter) - 64] * 32 + \\ &[ASCII(3rdletter) - 64]\end{aligned}$$

The code is formed from the 15 least significant bits. The MSB is used to specify whether the address of the unit is unique world wide (0), or unique within the transmission range of the device (1). Say the code is ”KAM”, the manufacturer id would be:

$$\begin{aligned}\text{Manuf. id} =&[ASCII(K) - 64] * 32 * 32 + \\ &[ASCII(A) - 64] * 32 + \\ &[ASCII(M) - 64] \\ =&[75 - 64] * 32 * 32 + \\ &[65 - 64] * 32 + \\ &[77 - 64] \\ =&11309 \\ =&2C2D_h \\ =&10110000101101\end{aligned}$$

Address is a number between 00000000 and 99999999. The number is either fixed or

[1] Acknowledgement messages, used in bi-directional communication

changeable. In a telegram the address is coded as 8 Binary Coded Decimal (BCD) packed digits. BCD simply means to represent each decimal with 4 bits, e.g

0 = 0000
1 = 0001
2 = 0010
3 = 0101

Version specifies the version of the meter. It is used by manufacturers to ensure that each address within a version number is unique. **Type** specifies what the meter measures. Some examples are electricity=02_h, gas = 03_h, pressure = 18_h.

The second block contains a CI-field and a data field. The **CI-field** specifies the protocol used, and thus the format and type of data following in the data field. The CI-field specifies the layer where the data is directed, which may be APL, TPL, or ELL. If ELL is used the CI field specifies the length and services of the additional layer.

Data field contains the data of the telegram. The data field of the telegram contains a **data header**, used to inform the receiver about the nature and encryption of the data. There are three kinds of headers, as described in Table 2.3. The header is defined by the mode used, according to EN 13757-4 [1]. Relevant to the demonstration project in this thesis is the *short header*, which is used in mode C1.

Header	Description
No header	If the CI field is 78_h, there is no data header.
Short header	Defines an access number, status byte and configuration word.
Long header	Contains all the fields from the short header, in addition to fields for ID, Manuf. ID, Type and Version.

Table 2.3: Data headers

Where the ”No header” indicates there are no information about the use of encryption, thus the data in unencrypted. The short header is structured as shown in Table 2.4.

Access	Status	Configuration
1 byte	1 byte	2 bytes

Table 2.4: Short data header

The long header is implemented as shown in Table 2.5.

ID	Manuf.	Version	Type	Access	Status	Configuration
4 bytes	2 bytes	1 byte	1 byte	1 byte	1 byte	2 bytes

Table 2.5: Long data header

ID, Manuf. version and type correspond to the values in the first block.
Access number is used for detecction of repeated frames. The number is incremented for each frame in order to ensure it is not already received.
Status contains data concerning the state of the meter. Used to alert the receiver about alarms or errors.
Configuration is used to set the encryption mode and the length of the encrypted data.

2.3 The STACKFORCE protocol stack

The software provided by STACKFORCE should be able to cover all region specific requirements across Europe, and is compliant with EN13757-3, EN13757-4, and the OMS specification [5]. It is available in binary/object code format and supports the ARM Cortex-M0+, M3 and M4 cores. The stack consists of three major parts: the Wireless M-Bus stack, RF driver, and HAL. These will be described in detail in order to give a better understanding of how telegrams are transmitted and received. Figure 2.6 displays an overview of the stack, which describes how the abstraction layers are connected to each other, to hardware, and what utility and security features the stack includes.

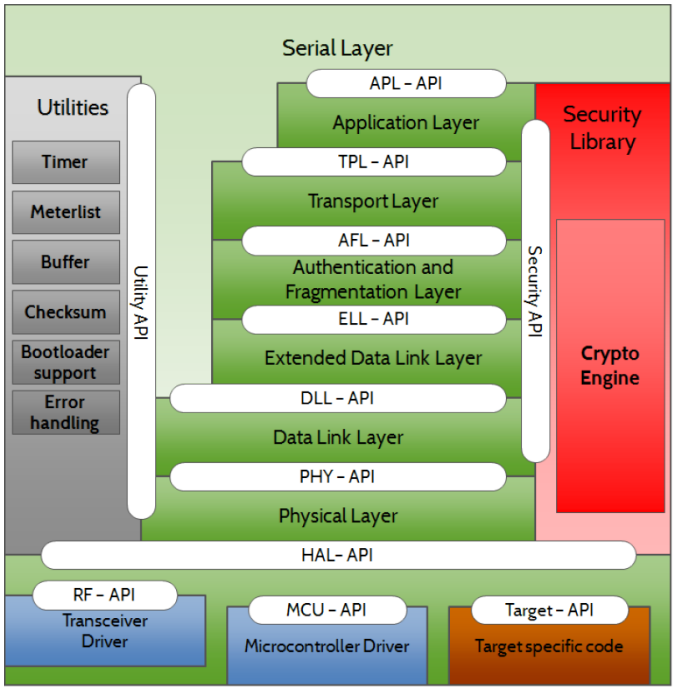

Figure 2.6: The STACKFORCE Wireless M-Bus Stack architecture. Taken from [7]

2.3.1 Wireless M-Bus stack

In this section, a presentation of the WM-Bus stack will be given.

WM-Bus modes

The stack is highly configurable in terms of WM-Bus modes. It supports the use of all modes described in Figure 2.2: S, T, C and N(a-f). Both meter- and collector devices can be configured in any one of these modes. Usually, the collector and meter are configured to use the same mode, but if for some reason another configuration is desired, STACKFORCE

has provided a table describing the compatibility of each mode, as can be seen in Figure 2.7 below:

	Meter S	Meter T	Meter C	Meter $N(a-f)$
Collector S	✓			
Collector T		✓		
Collector C		✓	✓	
Collector $N(a-f)$				✓

Figure 2.7: Compatibility matrix for the WM-Bus modes. Taken from [7]

As depicted in Figure 2.7, the modes implemented by the stack are mostly only compatible with devices using the same mode. The only exception is that a collector in C-mode is able to receive telegrams from a meter in T-mode, in addition to C-mode.

As described in Section 2.2.2, EN 13757-4 specifies the use of uni- and bidirectional communication. The stack implements these options for all modes. Figure 2.8 shows a simple unidirectional communication scheme where the meter transmits its data to the collector.

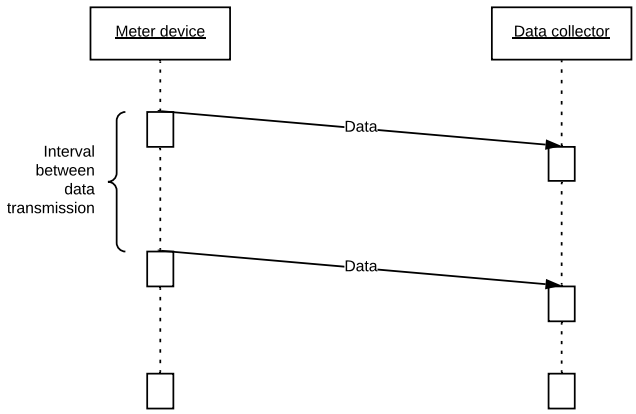

Figure 2.8: Unidirectional communication

In unidirectional communication the only sort communication is when a meter device transmits data or errors/alarms to the collector. In Figure 2.9, a typical bidirectional communication mode is described. The meter may measure several kinds of properties: ”User data” and ”User data 2”. It may only be sending periodical telegrams containing one of them (”User Data”) to the collector in order to save energy. Only sending User Data 2 when the collector requests it. It will repeat the response until the collector closes the communication.

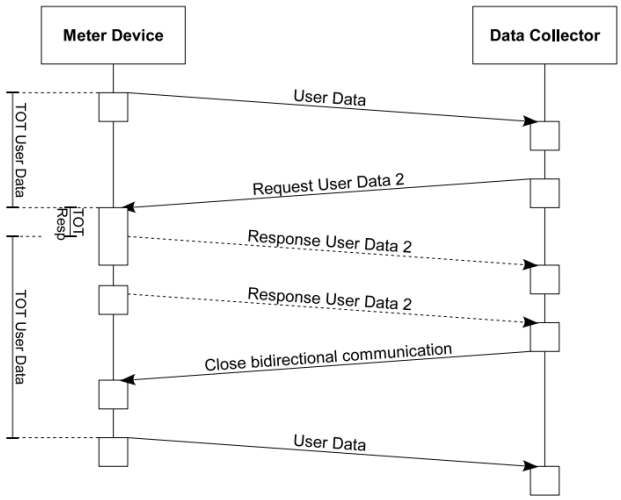

Figure 2.9: A typical flow of bidirectional communication. Taken from [7]

In bidirectional communications, messages are passed back and forth between the meter an collector. Several different forms of communications may occur such as sending of alarms, requesting of data, configuration of meter etc.

2.3.2 Abstraction layers

In addition to implementing the APL specified by EN 13757-3 and the PHY and DLL layers specified in EN 13757-4, the STACKFORCE protocol stack also implements an Extended Link Layer (ELL) and Transport layer (TPL) for implementation of security and privacy features.

APL

The application layer implements EN13757-3, as mentioned in Section 2.3.1. The APL is responsible for handling application layer data, which normally is the data of a telegram e.g pressure, energy etc. The application layer interface available to us is divided into six modules. The APL documentation [7] following the software stack lists everything available in this interface. The relevant parts for the demonstration project will be listed in this section.

Enumerations

Enumerations are used for setting contents of messages, return values of functions that add/remove meters, RF-adapters, and keys, and status for the application initialization function. These enumerations are not used or edited in the demonstration project.

Structures

Relevant **structures** to mention is the structure of a meter, and the structure of the meter list of the collector.

In order to be able to receive telegrams from any meter, the meter has to be added to the meter list of the collector. This is done by creating a meter entry and adding it to the list. A meter entry is a struct of the type s_apl_meterEntry_t which contains the address, mode, adapter adress (not used in this project), and encryption key of the meter. The data fields of a meter entry is as seen in Table 2.6.

Type	Tag	Description
s_wmbus_addr_t	s_meterAddr	The meter address
E_WMBUS_MODE_t	e_wmbusMode	WM-Bus mode of the meter
s_wmbus_addr_t	s_rfAdapter	RF adapter (not used)
uint8_t	pc_meterKey[0x10U]	Encryption key of the meter

Table 2.6: The structure of a meter entry

The meter list consists of the length of the list and a pointer to the first meter entry, as descibed in Table 2.7:

Type	Tag	Description
uint16_t	i_numberOfMeters	Number of meters in the meterlist
s_apl_meterEntry_t	ps_meterEntry	Pointer to the first meter entry

Table 2.7: The structure of a meter list

Event callbaks

There are four *event callbacks* ready to be used in the main file of the program. These are called whenever a specific event occurs. The four events are: whenever a telegram is received, whenever a telegram is sent, whenever a telegram is ready to be read, and whenever the stack receives a telegram with an unknown CI-field. The callbacks are listed in Table 2.8.

Event callback	Description
wmbus_apl_evt_rx()	Called whenever a telegram is succesfully received
wmbus_apl_evt_tx()	Called whenever a telegram is transmitted
wmbus_apl_evt_tlgAvailable()	Called whenever a telegram is available and ready to be read. This is the inteded method for extraction of data
wmbus_apl_evt_getCiHeader()	Called whenever a telegram with an unknown CI-field is received. Used to specify the type of header to use in case of unknown CI-fields

Table 2.8: APL Event callbacks

The stack differentiates between three different data headers: no header, short header, and long header. In the STACKFORCE WM-Bus C1 mode, the long header is used. The long header is the first block of the telegram, described in frame format A (2.4).

TPL

The transport layer handles data headers, which defines the encryption mode used. There are three different data headers, as described in Table 2.3.

ELL

The Extended Link Layer is not documented by SF. The only information is found in Figure 2.6, where we can see the layer connects the DLL and Authentication and Fragmentation layer. From EN 13757-4, the ELL provides additional control fields (instead of just one control field), which may include Synchronisation, Encryption, and Destination Address.

Data Link Layer

The Data Link Layer (DLL) is mainly used to interface the the PHY-layer with the ELL and TPL layers. It handles the first block of telegrams and sends the rest of the telegram to the next layer, which is either the TPL or ELL, based on the control field.

2.3.3 Hardware abstraction layer

The Hardware Abstraction Layer (HAL) is responsible for abstraction of all the hardware resources required by the stack and the RF driver. An overview of the HAL and how it connects to the rest of the stack is given in Figure 2.10.

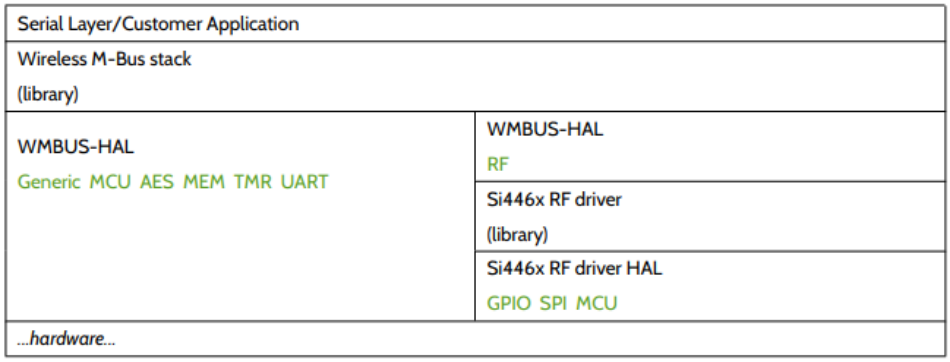

Figure 2.10: The role of the HAL. Taken from [7]

Table 2.9 further explains which interfaces between the stack and the harware that the HAL is responsible for.

Interface	Harware
WM-Bus	HW accelerated AES
WM-Bus	MCU core
WM-Bus	Non-volatile memory
WM-Bus	RF driver
WM-Bus	HW timer
WM-Bus	Serial
RF-driver	GPIO
RF-driver	SPI
RF-driver	MCU

Table 2.9: The interfaces of the HAL

As seen in Figure 2.10, the HAL is divided into two parts, one part for general hardware access, and one part for the transceiver (RF). The part which connects the stack to the transceiver is a little confusing, as it is connected in several layers. The WM-Bus RF HAL connects the WM-Bus stack to the Si446x RF library, which is separate from the WM-Bus stack. The Si446x has its own HAL, which connects it to the underlying hardware. If the stack should be used with another transceiver, the Si446x library has to be changed. Some notable functions of the RF HAL used in the demonstration project are listed in Listing 2.1 below.

```
1
2
3 bool_t  wmbus_hal_rf_rxInit (uint8_t *pc_quality, uint8_t c_len)
4   //Initiates the reception of data. As soon as the stack has been
      informed about the detection of a telegram, the stack will initialise
      the reception procedure by calling this function. More...
5
6 bool_t  wmbus_hal_rf_rxData (uint8_t *pc_data, uint16_t i_len)
```

```
7   //Retrieves a chunk of data from the RF driver. Just like the
      transmission, reception of data is done chunk-wise. For this, the
      function will be called as fast as possible to retrieve all data
      received by the transceiver. More...
8
9   bool_t  wmbus_hal_rf_rxFinish (E_HAL_RF_MODE_t e_mode)
10  //Finalises the reception of data. At least wmbus_rf_rxInit should be
      called before. More...
11
12  bool_t  wmbus_hal_rf_setSignalStrength (uint8_t c_signal)
13  //Sets the signal strength for transmission. Requests the RF driver to
      set the output power for the transceiver to the appropriate value.
      More...
```

Listing 2.1: A selection of some important functions of the WM-Bus RF driver

These functions are used in the fourth setup of the demonstration project. The implementation part of this thesis demonstrates how they are used.

Chapter 3

Literature Review

3.1 Introduction

A literature review has been conducted in order to collect useful insights on existing technology, the WM-Bus standard, and other similar projects. It showcases a previous master's thesis: *Demonstrering av konsept for innsamling og sammenstilling av data fra flere vannmlere ved bruk av trdls M-Bus* by Lier H. H. [4] and a security evaluation of the WM-Bus standard: *Wireless M-Bus Security Whitepaper* by Cyrill Brunschwiler [2], which includes some useful examples for reception and decryption of telegrams. The third literary piece is an attempt at creating a versatile data-collection system for IIoT in *Wireless M-Bus in Industrial IoT: Technology Overview and Prototype Implementation* by K. Zeman *et al.* [9], and the last piece is an overview of the various implementations of the WM-Bus standard by Sikora *et al.* in *Recent advances in en13757 based smart grid communication.*[6].

3.2 Previous masters thesis

An attempt at reading the Kamstrup PressureSensor was done by Lier, H. H. in *Demonstrering av konsept for innsamling og sammenstilling av data fra flere vannmlere ved bruk av trdls M-Bus* [4]. As the thesis is not published in English, this section will give a brief overview of the work.

The work done by Lier [4] was aiming at reading the Kamstrup meter by using the same equipment as presented in this thesis. Unfortunately, the project was unsuccessful in reading the meter, thus the main focus of the thesis was redirected at creating a cloud service for analysis of meter data, and remote configuration of the collector. The project used several dummy-meters to simulate a real water supply network, and the Amazon Web Services (AWS) for remote analysis of data and configuration of the collector. Lier states that further work should focus on reading actual water meters, by troubleshooting and further analysis of the WM-Bus standard, which is what this thesis will focus on.

3.3 Reading raw data from a kamstrup electricity meter

This section covers the work done by Compass security AG [2] on the security of the M-Bus standard. It includes relevant examples of how they received meter telegrams and decrypted them. These experiments are highly applicable to the demonstration project in this thesis, as we are trying to receive telegrams from a commmercial meter. The paper concludes thath there are several issues with the confidenciality, integrity, authenticity, and non-repudiation of the WM-Bus protocol. It is stated that the issues range from inadequate key lenght, manipulation of encrypted telegrams, to full exposure of key material. However, the important part related to this thesis is the reception of telegrams and the decryption and parsing of those.

In the tests conducted in order test the security of the WM-Bus protocol, Compass Security AG went through the raw data of a captured telegram from a Kamstrup electricity meter step by step. The security issues of the analysis is outside the scope of this thesis, and will not be discussed. However, it would be interesting to investigate the method for reception and decryption of telegrams.

The reception of telegrams is done with a ”Wireless M-Bus Analyser 1.0” from AMBER wireless GmbH. The analyzer is USB stick which allows reception of raw WM-Bus telegrams, to be displayed on a PC. For decryption of the telegrams, the CrypTool 1.4.3x., an open-source Windows program for cryptography and cryptanalysis, was used. In writing time, a new version of the M-bus analyzer is available, V3.0, which has integrated decryption, eliminating the need for an external crypto tool. The analyzer and cryptography tool was used to receive and decrypt telegrams from a known set of meters. The raw data of one of the telegrams received from a Kamstrup meter is presented in Table 3.1.

1E	44	2D	2C	07	71	94	15	01	02	7A	B3	00	10	85	BF
5C	93	72	04	76	59	50	24	16	93	27	D3	03	58	C8	

Table 3.1: Raw data frame from a Kamstrup electricity meter. Captured by [2]

To clarify what the contents of Table 3.1 means, an overview is presented in Figure 3.1, adapted from the results of Brunschwiler [2]. The table presents the raw bytes of the telegram and the parsed[1] data. Both the unencrypted and decrypted data is presented. The contents of the telegram can be summarized as follows: The telegram is sent from a Kamstrup electricity meter, it is a unidirectional telegram requiring no reply, and the data is encrypted requiring a separately forwarded key. The encrypted data protects the reading of the meter, which is 341kWh.

The work of Compass Security show that it is possible to receive telegrams from a commercial meter using third party equipment, and to decrypt the telegrams. The method of decrypting data may be useful if for some reason we are only able to receive the raw data.

[1] Translated from machine readable content to human readable content

Figure 3.1 is included in this literature review in order to present a complete picture of what a telegram looks like, an what it is composed of. It is very useful to keep as a reference when the work of this thesis is presented. Although some telegrams may include different or additional data fields, it is always useful to see a real world example of things.

First block			
Raw data	**Data field**	**Parsed data**	Data Link Layer
1E	Lenght	Telegram length: 30	
44	Control	Send, no reply	
2D	Manufacturer ID	KAM (Kamstrup)	
2C			
07	Device ID	Meter ID: 15947107	
71			
94			
15			
01	Version	Version 1	
02	Device type	Electricity meter	
Second block			Transport Layer
Raw data	**Data field**	**Parsed data**	
7A	CI	EN 13757-3 as application layer, relying on a short transport layer	
B3	Acces number	Current access number is 179	
00	Status	Meter initiated, no alarms or errors	
10	Configuration	Encryption mode 5, with AES in CBC mode	
85			
Second block (encrypted data)			
Raw data	**Data field**	**Decrypted and parsed**	
BF	Decryption verification		
5C	Decryption verification		
93	DIF	04: Instantaneous readout value, no extension fields	Application Layer
72	VIF	83: Primary VIF, Unit Energy 10Wh,has extension (VIFE0)	
04	VIFE0	3B: Forward flow contribution only	
76	Data	08 34 05 00: The value is coded LSB first, and represents a value of 341000. Together with VIF this leads to the reading of: 341kWh	
59	Data		
50	Data		
24	Data		
16	Filler		
94	Filler		
27	Filler		
D3	Filler		
03	Filler		
58	Filler		
C8	Filler		

Figure 3.1: An overview of the raw data of a telegram, and what it means

3.4 Wireless M-Bus in industial IoT

In recent literature regarding implementation of general Wireless M-Bus systems that are able to receive telegrams from various vendors, there is little to be found. There are, however, several papers evaluating the performance of WM-Bus in different scenarios. These include demonstration projects which implement a collector able to receive from one, or a few selected meters. There have been attempts at expanding the protocol to the scene of Industrial Internet of Things (IIoT), such as in *Wireless M-Bus in Industrial IoT: Technology Overview and Prototype Implementation* by K. Zeman *et al*. [9], which implements a data collector on a Raspberry Pi3 for use in the IIoT landscape. The collector is able to receive telegrams from several sensors/meters from different manufacturers, but it is stated that the data from each of the devices has to be *sniffed* and analyzed separately because the implementation of data packets is not identical.

There is a trend among academic research of having to analyze the raw data of the telegrams in order to implement a solution that can receive them. This seems to be a dig downside with the Wireless M-Bus protocol, as the time and work of making sure third party equipment are able to receive telegrams from only a selected few manufacturers may be slowing down development and research. It seems like there is a need for a data collection unit able to receive telegrams from all kinds of meters, or at least easily configurable to suit the required meters. The variety of different configrations can be classified in *dialects*, which are seperate classes of adaptations of the EN 13757-4 standard.

3.4.1 WM-Bus dialects

In *Recent Advances in EN13757 Based Smart Grid Communication* by Sikora *et al* [6], some of the dialects that has emerged is discussed, and categorized into three different kinds:

The First class of dialects are adaptions or restrictions of the APL-layer. Restrictions are done by the Open Metering System (OMS)-group and Dutch Smart Metering Recommendations (DSMR). These are standards built on top of[2] EN 13757 to further specify the application layer. The OMS standard has restricted and defined the use of the telegram fields, while DSMR has added additional features to the APL and application data block (the second data block). A third example of this class of dialects is the standard produced by Device Language Message specification (DLMS). DLMS has created a completely new and ”standalone” version of the application layer.

The second class of dialects are country-specific dialects. Mainly the adaptations of the standard in Italy and France. These dialects cover a wide range of the features and specifications in EN 13757[3].

In the **third class of dialects** the focus is on security features. There are a wide field

[2]They are compliant to the standard
[3]Especially in Part 4

of different security layers implemented. Some examples of security protocols used are Secure Sockets Layer (SSL), Transport Layer Security (TLS), and Advanced Encryption Standard (AES).

3.5 Summary

As seen in this literature review, there is a large amount of different adaptations of the Wireless M-Bus standard, and there is a need for a general data collection system, in order to be able to receive telegrams from various meters. The stack developed by STACK-FORCE combined with the WM-Bus kit is a proposed solution to this challenge, and this thesis seeks to use it in order to implement a system for monitoring the state of the water supply network.

Chapter 4

Specification

4.1 Introduction

This chapter describes the specifications of the project, as well as the accompanying acceptance criteria. The specification describes crucial functionality and modules. The acceptance criteria describes the necessary conditions for the specifications to be fulfilled.

4.2 Technical specification

1. The collector and dummy-meter should be implemeted on a EZR32WG development kit.
2. The collector and dummy-meter should use the STACKFORCE protocol stack.
3. The commercial meter should be a Kamstrup PressureSensor.
4. The interface between the collector and PC should be over USB, where the collector acts as USB Device, and the meter acts as USB Host.

4.3 Functional specification

1. The collector, commercial meter and dummy-meter shall operate in WM-Bus C1 mode.
2. The collector should have a list of meters to receive telegrams from.
3. The collector should be able to receive telegrams from a commercial meter, specifically a Kamstrup PressureSensor.
4. The collector should be able to send received telegrams to a stationary computer over USB.

4.4 Acceptance criteria

The acceptance criteria are the measurements in witch a corresponding functional specification statement should be evaluated.

1. The collector is able to successfully receive telegrams from the dummy-meter.
2. The collector is able to send received telegrams to a PC.
3. The collector is able to successfully receive telegrams from the Kamstrup meter.

4.5 Method

The following list presents how the work is carried out, and in which order.

1. Base the experiment on a test-setup implemented with the STACKFORCE protocol stack.
2. Verify the the test-setup works as intended, verify the Kamstrup meter works as intended.
3. Try to receive telegrams from the Kamstrup meter.
4. If unable to receive telegrams from the Kamstrup meter, create a setup combining the test-setup and the Kamstrup meter
   (a) Enter different layers in the stack and try to see where the Kamstrup telegrams stops.
   (b) Compare with dummy-meter
   (c) Test and experiment with results

Chapter 5

Design

5.1 Introduction

In this chapter, the design of the various setups will be presented, and the hardware and software choices will be explained.

With limited documentation on the Kamstrup meter, and limited options to make alterations to the software stack, there has been a need to test varying setups and configurations of the system. Therefore, this chapter will cover the design of the varying setups implemented by the author. The design of the setups will be presented in the order of which they were implemented, as seen in Table 5.1.

Setup	**Purpose**
Collector and dummy-meter	Implement a functional collector, verify it works by reading of telegrams from a dummy-meter.
Kamstrup Meter Reader and Kamstrup meter	Verify the Kamstrup meter is transmitting telegrams
Collector and Kamstrup meter	Try to receive telegrams from a commercial meter
Collector, Kamstrup meter and dummy-meter	Investigate reception of telegrams from the Kamstrup meter, experiment with telegrams from dummy-meter

Table 5.1: The different setups and their purpose to the project

The first and second setup seek to establish a basis for reading commercial meters. This is done by ensuring that the collector works with a meter implemented with the same stack, and by testing the Kamstrup meter with Kamstrup equipment. The third setup seeks to receive telegrams from the Kamstrup meter with the collector implemented with the

STACKFORCE stack. If the third setup fails, the fourth setup seeks to combine the first and third setup in order to investigate and experiment with the STACKFORCE protocol stack.

5.2 Collector and dummy-meter

The first setup presented consists of a collector and a dummy-meter, both implemented with the STACKFORCE protocol stack on a EZR32WG330 MCU mounted on a Wireless Gecko(WG) development kit. In this setup, telegrams are transmitted from the meter to the collector to the PC, as depicted in Figure 5.1.

The EZR32WG330 MCU is a Silicon Labs product designed for use in home automation, metering, alarm systems, etc. where low-power, sub-GHz[1] wireless communications is needed. The board is equipped with a transceiver module and a SMA[2] connector for the 868MHz radio band. The WG extends the functionality of the radio board with an on-board J-Link debugger, a virtual COM port, USB and Ethernet connectivity, LED's, push-buttons, and EXP headers for I2C, USART, and SPI peripherals.

Both the meter and collector should operate in *C1-mode*, thus the meter will broadcast telegrams with no dedicated receiver. As the meter operates in C-mode, frequent transmission of telegrams is possible, allowing faster collection of data. In this setup, the transmission rate of the dummy-meter will be configured to $6s$. The collector will be listening on the radio frequency specified by C1-mode[3], and receiving all telegrams transmitted by meters nearby. However, only the dummy-meter will be added to its meter-list, thus it will only "process" telegrams from that meter. All other telegrams will be denied in the DLL. The collector is connected to a stationary PC with USB, where the collector acts as a USB device and the PC a USB host.

The USB Host side should be able to receive messages and data from the collector. The traffic received from the collector should be displayed in a terminal for analysis, and preferably written to a file or sent to a cloud service. Although it would be useful to implement sending of commands from the PC to the collector, e.g for adding a new meter to the collector's meter list or request specific data, this will only be implemented if time allows.

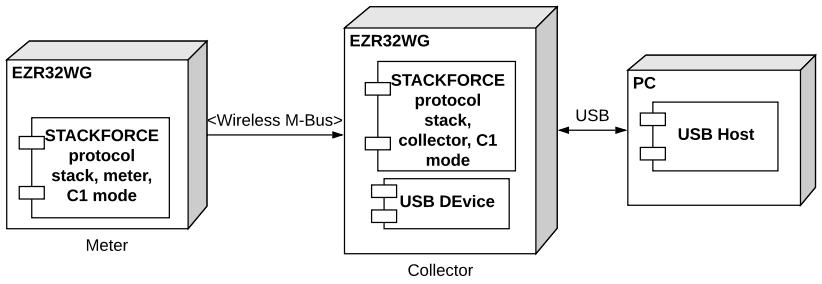

Figure 5.1: Collector and dummy meter

[1] Radio frequencies below $1GHz$
[2] Connector for RF coaxial connections
[3] $868.95Hz$

5.3 Kamstrup meter and Kamstrup meter-reader

In order to verify the Kamstrup meter is sending telegrams as it should, a Kamstrup USB meter reader was introduced to the project. The USB meter reader is a tool that enables reception of telegrams from Kamstrup meters. The device is plugged into a computer via USB, and is used together with the Kamstrup ”Metertool” software, which they refer to as ”PC Base”.

With the USB reader connected to a Windows PC, a meter list can be initiated in the PC Base and transferred to the USB reader. The USB reader can then start to receive telegrams from nearby meters, either by leaving it connected to the PC or a laptop, or by connecting it to a power supply and carrying it to the desired area. In this setup the USB reader will remain connected to a stationary Windows laptop, as there is no need to carry it around. When the USB reader has received telegrams from all meters in its meter list, the data is transferred to the PC Base, ready to be exported for analysis.

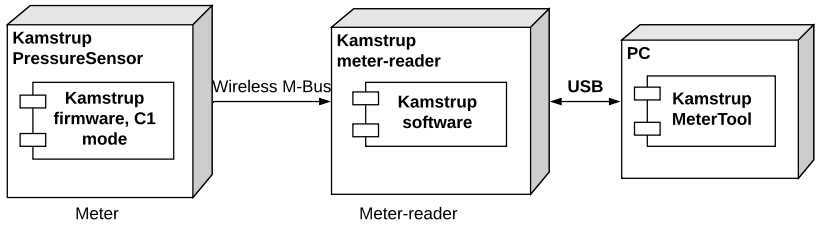

Figure 5.2: Kamstrup meter and Kamstrup meter-reader

5.4 Collector and Kamstrup meter

In this setup the collector will be configured the same way as in the previous setup in Section 5.2, the only difference being a change in the meter list, where the dummy-meter is removed and the Kamstrup PressureSensor is added.

The Kamstrup PressureSensor is a sensor designed to be installed directly connected to a pipe network. In this project, the sensor will be used to measure the atmospheric pressure, as the content of the transmitted telegrams are not important, but the telegram itself is. The meter operates with WM-Bus C1-mode, and transmits telegrams once every 96 seconds. Kamstrup follows the OMS-standard

The sensor used to measure pressure is a Kamstrup PressureSensor. The PressureSensor is designed to be installed directly connected to the pipe-network desired to monitor. It collects samples at a rate of $10Hz$, i.e ten times per second, which allows detection of bursts of changes in pressure. The sensor transmits its recordings once every 96 seconds over WM-Bus in mode C1, which allows 3^{rd} party equipment to receive its telegrams, given it has the ID and encryption key of the meter.

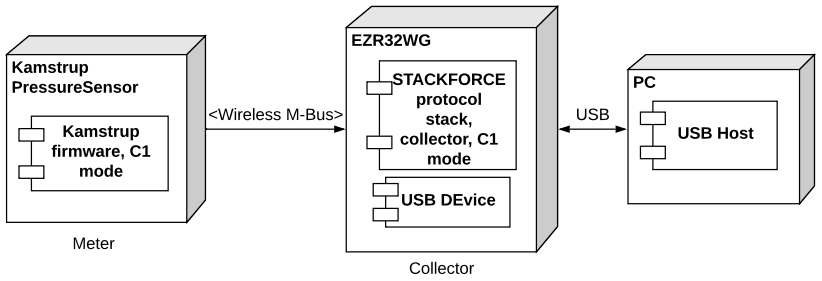

Figure 5.3: Collector and Kamstrup meter

5.5 Collector, Kamstrup meter and dummy-meter

The fourth and last setup in this project is a combination of the setups in Section 5.2 and 5.4. In this setup, both the dummy-meter and Kamstrup meter is added to the meter list of the collector. Some changes will be made to the addresses of these, in order to test how the stack handles unknown addresses. Additionally, an attempt at tracking incoming telegrams through the layers of the stack will be made.

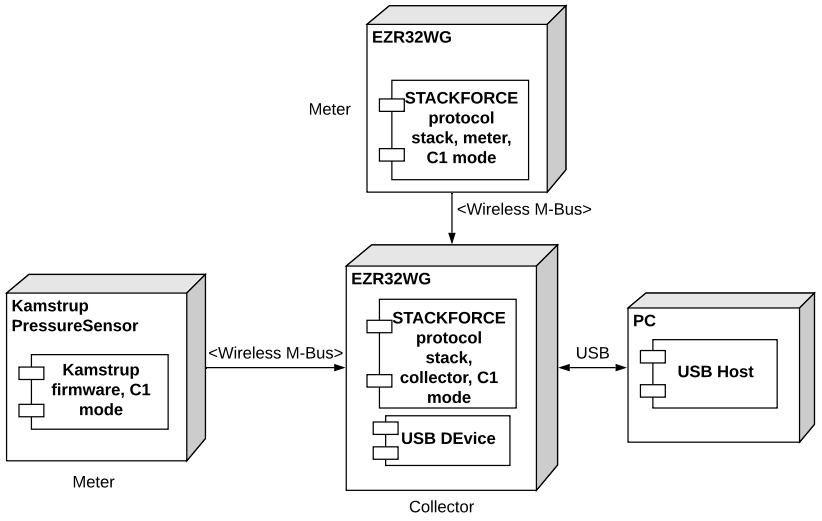

Figure 5.4: Collector, Kamstrup meter and dummy meter

Chapter 6

Implementation, Testing, and Results

6.1 Introduction

This chapter combines the implementation, testing, and results of the project. These topics have been combined, as there are four different setups presented, each of them having their own separate implementation, tests, and results. The author found it suitable to structure the chapter in this manner in order to ease readability. For each setup, the implementation of the hardware and software will be explained, the tests conducted will be described, and finally, the results of the tests will be presented.

6.2 Development enviroment

The software for the collector and meter devices is developed on the Windows version of IAR Embedded Workbench. This IDE was chosen by the author as it supports the Cortex M4 processor and has the development tools as needed: compiler, analysis tools, and debugger. Additionally, the author had access to a licence to the complete IDE.

The software developed for the interface between collector and an external computer was developed with Microsoft Visual Studio, by personal preference.

6.3 First setup: collector and dummy-meter

This section covers the implementation, testing, and results of the design presented in Section 5.2. Some of the content may seem unnecessary to include, however, it has been deemed important for reproducability. Figure 6.1 shows the first setup, which consists of a collector and dummy-meter.

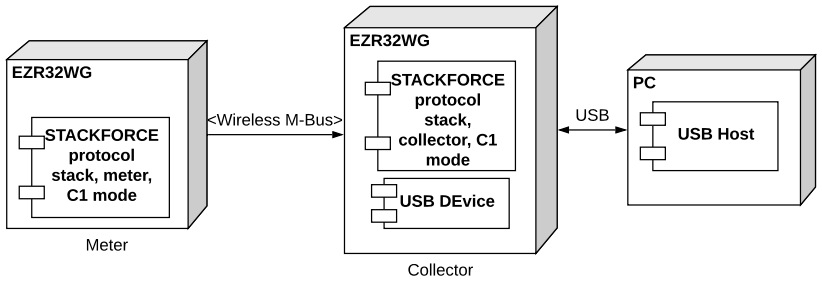

Figure 6.1: Collector and dummy meter

6.3.1 Implementation of the collector

As described in Section 2.3, a third party software stack delivered by STACKFORCE is used as a basis for the development of the firmware used in the collector and dummy-meter devices. The stack was downloaded from Silicon Labs's website [7]. After downloading the stack, the project can be opened in IAR Embedded Workbench. The project workspace can be found under /ide/iar/, where there are four demo projects present. The one used in this thesis was Demo_SLWSTK220A. After opening the workspace in IAR, the correct device type and mode was be selected. In IAR, this was done by following these steps:

1. Navigate to *project* → *options*

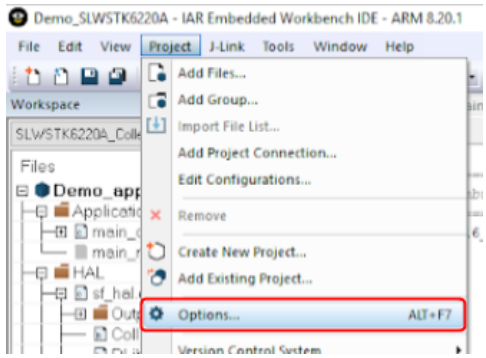

Figure 6.2: Selecting device type

2. Navigate to general options and select the *target* tab. Under *target*, select the relevant core architecture and device type. In this project the Cortex-M4 and Silicon Labs EZR32WG330F265Rxx.icf is used.

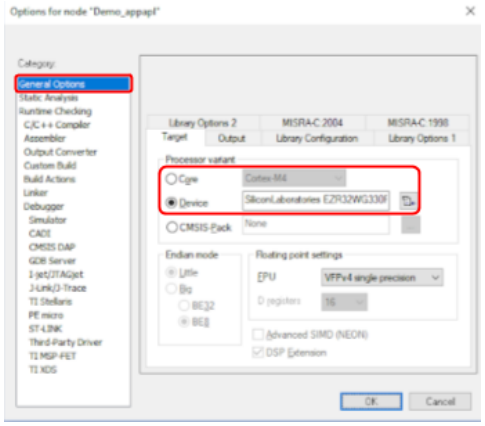

Figure 6.3: Selecting device type

In order to configure the WM-Bus mode, the preinclude header had to be changed. This was done by following these steps:

1. Navigate to *project* → *options*
2. Within *options*, select *C/C++ Compiler* under *Category*.
3. Select the *preprocessor* tab and choose the desired configuration file. In this project the ”collector_C1.inf”

This set the correct configurations for the device to act as a collector in mode C1.

In order for the collector to receive telegrams from a meter, the meter has to be added to the meterlist of the collector. In the project, the meter was initialized according to the structure in Table 2.6, with the following values:

```
s_apl_meterEntry_t gs_meterEntryOrginal = {
                // WMBus meter address
                {{0xce,0x9a},           /* Manufacturer (here STZ) */
                {0x80,0x00,0x00,0x01},  /* ident number            */
                0x23,                   /* version                 */
                WMBUS_DEV_TYPE_WATER},  /* type, here water        */

                // WMBus mode of the meter
                E_WMBUS_MODE_UNKNOWN,

                // WMBus RF adapter address of the meter (unused)
                {{0x0,0x0},{0x0,0x0,0x0,0x0},0x0,0x0},

                // WMBus meter key
```

```
15                  {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,
16                   0x88,0x99,0xAA,0xBB,0xCC,0xDD,0xEE,0xFF}};
```

As the dummy-meter does not use a RF adpter, this field was not used. A meter list was initialized with the length of 1, and with the meter entry created.

```
1 s_apl_meterList_t gs_meterList = {0x0001U, &gs_meterEntryOriginal};
```

To enable the collector to send messages to the PC via USB, the initialization of the USBD-library was added to the main program of the collector. To enable sending of received telegrams to the PC, the event callback (Table 2.8) for new available telegrams was used. The data could be read into a local buffer, which could be used to extract the data. The USBD_write was then used to transmit the data of the local buffer to the USB Host[1].

6.3.2 Implementation of the dummy-meter

The implementation of the meter was done in almost the same way as the implementation of the collector in Section 6.3.1. Firstly, in order to simplify development of the different configurations of the stack, a new *build configuration* was added to the project. Build configurations are separate build versions of the same project[2], often used to keep one version for debug and one for release, or to use the same project on different target devices. To add a new build configuration, the following steps were followed:

1. Navigate to *project* → *edit configurations...*, and click *new*.
2. Give the new configuration a name, and choose a tool chain.
3. Lastly there is an option to base the new configuration on another project configuration.

The meter configuration was created based on the the collector configuration. The only changes to setting up the stack configured as a meter was to change the preinclude file for the configuration. The preinclude file ”meter_C1.inf” was chosen, as the desired mode of the meter is C1.

In order for the collector to receive telegrams from the meter, the meter was given the same address and key as in the meter added to the meterlist of the collector in Section 6.3.1. WM-Bus mode is set by the stack, and there is no RF adapter.

6.3.3 Implementation of USB interface

Collector side (USB Device)

The USB interface between the collector and the PC was implemented as a USB host-to-device configuration, where the collector was configured to act as a UDB-device, and the PC a USB-Host. This was done by adding the Silicon Labs USBD-library [3] to the stack. The library is included with the stack when downloaded, but is not included in

[1] In this project the Host is a PC, but it could be any form of USB Host.

[2] For future development: Build configs can also be used to create configuratins for different modes

the project unless done manually. The library contains a lot of functionality, enabling communcation both ways. However, the only function used in this demonstration project was the ”USBD_write”-function, which writes data to a buffer on the MCU. This buffer is called a Device Endpoit, and is read by the USB Host. The device endpoints are defined as follows:

```
#define EP_IN  0x81
#define EP_OUT 0x01
```

The direction of the endpoints are defined based on the host, thus EP_IN is the buffer that goes in the direction from the device to the host, and is the buffer USBD_write writes to. EP_OUT would be the buffer to read from if commands are to be sent from a USB Host.

PC side (USB Host)

In the library downloaded from Silicon Labs [3], the USB host code is available. The Library is ready for plug-and-play in Microsoft Visual Studio (MSVS). The project was imported to MSVS, ready to be used. In order to communicate with the USB Device[3], the correct device vendor and id had to be defined. These are available in the USBD library of the collector, and in this project they are as follows: Device vendor: 0x10C4, and product ID: 0x0003. In addition, the device enpoint(s)[4] has to be defined. These are defined as EP_IN = 0x81 and EP_OUT = 0x01. With these defines in the MSVS project, the program is ready to run. The USB Host program includes a basic ”pinging-loop” from the host to the device, in order to see that the setup works as intended. After verifying that the USB communication was up and running, some alterations were made to the ”receiveMessage” function and the main loop, in order to properly receive messages and display them. Additionally, some quality-of-life changes were made to create an easier to read presentation of the messages and data.

6.3.4 Testing of first setup

After implementing the collector and dummy-meter according to Section 5.2, the setup was ready to be tested. This setup seeked to prove the communication between the collector and dummy-meter.

Receiving dummy-meter telegrams with the collector

To test that the collector was able to receive telegrams from the dummy-meter, the software was downloaded to each of the devices, and the devices were started. Both devices were placed in the same room, to ensure that range and obstacles would not interfere with the test.

[3] The collector.

[4] Buffers on the device the Host can write to and read from

Testing varying telegram length

In order to verify that the collector could receive telegrams of varying length, not just the length specified in the demo application, additional data fields were inserted into the telegram of the dummy-meter. This was important to establish, as the length of telegrams from the Kamstrup meter is unknown.

Adding additional meters

It is useful to know whether the collector can keep several meters in the meter list and receive from them. To investigate this, a second meter was implemented and added to the meter list of the collector. This meter was given a new vendor id, address, type, and version, in order to verify that those worked as well.

6.3.5 Results of tests

This section covers the results from the tests conducted on the first setup.

Receiving dummy-meter telegrams with the collector

In the first test, the collector was able to receive the telegrams from a dummy-meter and send them to the PC. The following data was displayed in the Windows console:

0 1 39 59 11 2 −123 4 1 19 12 0 1 2 15
33 −82 109 6 0 −91 27 0 32 −92 0

This data corresponds to the data that the dummy-meter is supposed to send, and thus confirmed that the collector and dummy-meter worked as intended.

Testing varying telegram length

By adding additional data fields to the telegrams that the meter sent, we were able to receive those as well, thus confirming that the stack can handle telegrams of varying length.

Adding additional meters

The collector handled reception of telegrams from two meters without any trouble, confirming that the collector was able to handle a meter list of two meters. A completely new address was also given to the meter, which the collector accepted.

6.4 Second setup: Kamstrup meter and Kamstrup Meter-Reader

This section covers the implementation, testing, and results of the design presented in Section 5.3.

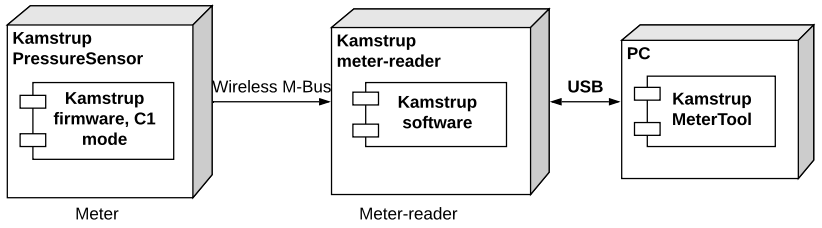

Figure 6.4: Kamstrup PressureSensor and Kamstrup Meter Reader

6.4.1 Implementation of second setup

In the third setup, a test intended to verify that the Kamstrup meter was transmitting telegrams was conducted. This was done by using a Kamstrup meter reader.

Kamstrup meter

The meter was placed in the same conditions as in Section 6.5.1.

Kamstrup meter-reader

To verify that the meter worked as intended, a Kamstrup Meter Reader and metertool software was used. The meter tool is distributed by Kamstrup, and has to be delivered by them. After installing the metertool program, the USB Reader can be inserted into the PC. This prompts the user with a dialog box, where you can name and save the USB Read. After the USB Reader is ready, a *group*[5] of meters can be created[6]. In this case, the meterlist consists of the Kamstrup meter. Firstly, the meter has to be added to the MeterTool software. The meter can be added manually if it is unencrypted, however, if it is encrypted, additional information has to be imported first. This can either be done by importing a file[7] forwarded by Kamstrup, or by logging in to their customer portal. In this project, a file was forwarded by Kamstrup. After importing the meter information, a group was created, and the meter was added to the group. The group was then added to the USB reader by selecting the connected reader, and clicking add group.

[5] A meterlist
[6] The order doesn't matter
[7] A .KEM file

6.4.2 Testing of second setup

With the meter added to the meter reader, everything left to do was to start reading with the meter reader. When finished reading, the results were presented in the metertool software.

6.4.3 Results of test

After leaving the meter reader to read telegrams for two minutes, the results were ready. A telegram was received, with values corresponding to the values displayed on the display of the meter. As the meter operates in C1-mode, the meter reader could not interfere with the meter, thus no changes or interferences with the meter had been made. ***Based on this, we could hereby conclude that the Kamstrup meter sent telegrams correctly.***

6.5 Third setup: Kamstrup meter and own collector

This section covers the implementation, testing, and results of the design presented in Section 5.4.

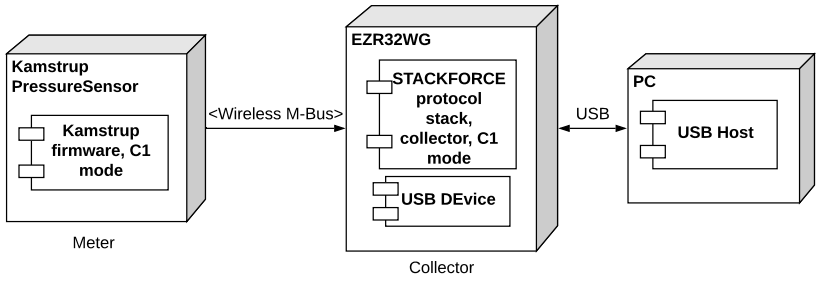

Figure 6.5: Collector and dummy meter

6.5.1 Implementation of third setup

In the second setup we sought to successfully receive a telegram from a Kamstrup PressureSensor with the collector implemented in Section 6.3.1. The only change implemented in the collector was adding a ”ping” to the computer whenever a telegram with a short header was available, as the demo only implemented functionality for extraction of telegrams with long headers. The ping was simply a message to the PC, saying the stack had reached a specific point in the code. Additionally, the Kamstrup meter was added to the meter list, as depicted in Listing 6.1 below.

```
s_apl_meterEntry_t gs_meterEntry = {
                  // WMBus meter address
                  {{0x2C,0x2D},           /* Manufacturer (here KAM) */
                  {0x30,0x34,0x32,0x34},  /* ident number            */
                  0x01,                   /* version                 */
                  WMBUS_DEV_TYPE_WATER},  /* type, here water        */

                  // WMBus mode of the meter
                  0x02, // C-mode

                  // WMBus RF adapter address of the meter (unused)
                  {{0x0,0x0},{0x0,0x0,0x0,0x0},0x0,0x0},

                  // WMBus meter key
                  {0x31,0x16,0xa7,0x35,0xb1,0x77,0x20,0x71,
                   0xda,0x78,0xa5,0xa4,0x51,0x6c,0xec,0x2c}};
```

Listing 6.1: Kamstrup meter entry

The address, mode, and key provided by Kamstrup was added to the meter, and the meter was added to the meterlist.

The Kamstrup PressureSensor was delivered ready-for-use, and was not possible to change

without tools outside the scope of this thesis. The meter operates in WM-Bus mode C1, and transmits is measurements once every 96 seconds. To ensure that the range between the two devices would not impact the tests, the meter was placed within a range of five meters from the collector.

6.5.2 Testing of third setup

The first test of this setup had the intention of trying to receive telegrams from the Kamstrup meter without making any changes to the software described in the implementation of this setup. This was done by downloading the stack to the collector device and running it.

The second test included the use of the callback function wmbus_apl_evt_getCiHeader, which is called whenever a telegram with an unknown CI-field is received. If the CI-field is unknown, we can investigate potential CI-fields and add them to the stack. To test wheter the function was called, a simple ping over USB was inserted at the start of the callback function.

Testing different WM-bus dialects was done by configuring the stack for different specifications, namely the OMS specification and the DSMR specification. This was done by setting the global variables of OMS and DSMR, which enables/disables the standards in wmbus_global.h to true.

6.5.3 Results of tests, third setup

First test

After running the devices with the collector configured as described in Section 6.5.1, there was no signs of received telegrams.

Second test

The second test tried to determine whether the stack did not know the CI-field of the telegrams from the Kamstrup meter. The event callback for unknown CI-fields was never called, indicating that the problem lied somewhere else in the stack.

Third test

The third test tried to determine whether the Kamstrup meter was using a WM-Bus addon. The setup was tested for three WM-Bus dialects[8]: OMS, DSMR v405, and DSMR v22. *None of these standards gave any results*.

[8] As described in **??**

6.6 Fourth setup: Collector, Kamstrup meter and dummy-meter

This section covers the implementation, testing and results of the design preseted in Section 5.5:

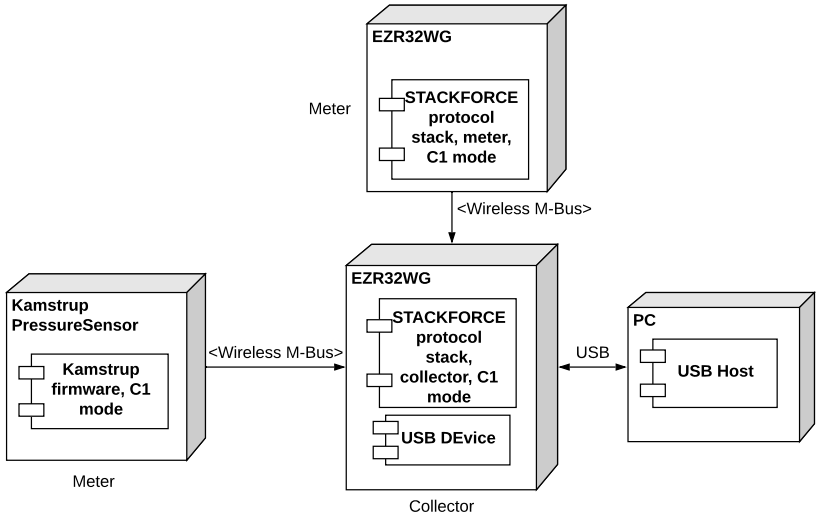

Figure 6.6: Collector, Kamstrup meter and dummy meter

The goal of this setup is to further test why the collector is unable to successfully receive telegrams from the Kamstrup meter. This will be done by trying to follow the flow of the telegrams from both meters throughout the stack, and to try to figure out where it gets rejected/accepted, and why. The telegrams from the dummy-meter are used as a reference for accepted telegrams.

6.6.1 Implementation of fourth setup

Collector

The collector was mainly implemented in the same was as in Section 6.3.1. In addition, the Kamstrup meter was added to the meterlist. This was done by calling the ”wmbus_apl_col_meterAdd” function after the initialization steps in the main program of the collector. During the tests of this setup, changes will be made to the collector in order to extract information on where the telegrams gets accepted/rejected.

Kamstrup meter

The Kamstrup meter was added to the system the same way as in Section 6.5.1 and 6.4.1.

Dummy-meter

The dummy-meter was implemented the same way as in Section 6.3.2. Some changes to the meter address were implemented during testing.

6.6.2 Testing of fourth setup

In the fourth setup the goal was to try to track the received telegrams of both telegrams.

Investigating calls to the RF driver

As most of the stack is only available in object code, there are limitations on where we are able to insert own code. In the WM-Bus hardware abstraction layer, the RF driver (which abstracts the ”real” RF driver to the stack) calls methods in the Si446x RF driver. We do not have the source file for the Si446x library, as it is delivered in object code, but we do for the WM-bus RF driver. However, the possibilities are still limited, as most of the functions simply call a corresponding function of the RF driver. For example:

```
1 void wmbus_hal_rf_powerOn(void)
2 {
3   sf_rf_powerOn();
4 }
```

Listing 6.2: A call from the WM-Bus HAL tor the RF driver

We are not able to explore what happens inside sf_rf_powerOn(), or edit the code, but we are able to insert own code in the wmbus_hal_rf_powerOn() function. This lets us get some insight of how the RF driver handles incoming telegrams. We are able to monitor all calls from the stack to the RF-driver by adding a USB write of the function name, as depicted in the following listing:

```
1 void wmbus_hal_rf_powerOn(void)
2 {
3   USBD_Write(EP_IN, "powerOn ", sizeof("powerOn "), NULL);
4   sf_rf_powerOn();
5 }
```

Listing 6.3: Adding code to monitor calls to RF driver

By adding a similar write to all the calls to the RF driver we are able to see which functions are called and when.

Reading of telegram content before decryption

Some of the calls from the stack to RF driver passes on arguments to the RF driver, some of them are parameters used to configure the operation of the tranceiver:

```
1 bool_t wmbus_hal_rf_setSignalStrength(uint8_t c_signal)
2 {
3   return sf_rf_setSignalStrength(c_signal);
4 }
```

Listing 6.4: RF operation

Others are buffers to read from or write to:

```
bool_t wmbus_hal_rf_rxData(uint8_t *pc_data, uint16_t i_len)
{
  return sf_rf_rxData(pc_data, i_len);
}
```

Listing 6.5: RF operation

The call to receive data to the RF driver passes on a pointer (*pc_data) to a buffer where the data retrieved from the transceiver is written. sf_rf_rxData reads the received data and stores it in the buffer, then returns TRUE or FALSE based on whether the operation went successfully. By creating a variable to store this boolean value, it is possible to initiate a read of the content in the buffer before it is deleted or overwritten.

```
bool_t wmbus_hal_rf_rxData(uint8_t *pc_data, uint16_t i_len)
{

  // Create temp to enable extraction of buffer data
  bool temp = sf_rf_rxData(pc_data, i_len);

  uint16_t i;

  for (i = 0 ;   i <= i_len;   i++)
  {
    USBD_Write(EP_IN, &pc_data[i], sizeof(&pc_data[i]), NULL);
  }
  return temp;

} /* wmbus_hal_rf_rxData() */
```

Listing 6.6: Sending rx data to the PC

By reading the data stored in the buffer pc_data points to, we are able to send the data retrieved from the tranceiver to a PC, where we can study the contents of the raw data. Notably to add, most of this data is encrypted, but the header (the first block) is unencrypted.

6.6.3 Results of tests

Studying receive patterns

As mentioned in Section 6.6.2, the sf_hal_rf.c file was configured to transmit function calls of RF driver to the USB host (PC). By analyzing these calls, a pattern could be observed. Whenever the dummy-meter transmitted telegrams[9], the following calls, described in Table 6.1, were called by the collector WM-Bus stack to the RF-driver.

[9] Observed by toggeling a LED on transmission

Function call	Order
wmbus_hal_rf_rxInit	1
wmbus_hal_rf_rxData	2
wmbus_hal_rf_rxData	3
wmbus_hal_rf_rxData	4
wmbus_hal_rf_rxData	5
wmbus_hal_rf_setPowerMode	6
wmbus_hal_rf_rxFinish	7

Table 6.1: Calls to RF driver when receiving a dummy-meter telegram

In addition, the following pattern, as described in Table 6.2, was observed during the test, occurring in a seemingly random fashion:

Function call	Order
wmbus_hal_rf_rxInit	1
wmbus_hal_rf_rxData	2
wmbus_hal_rf_rxFinish	3

Table 6.2: Calls to RF driver during testing

For now this is considered noise from other devices, as the stack seems to ignore them in the layers above HAL. This is probably due to the fact that the telegrams may be sent from a meter not added to the meter list, if not, they may be noise from other instruments not operating with the wireless M-bus protocol.

Analyzing ”noise” patterns

The kamstrup PressureSensor is trnsmits telegrams every 96 seconds. By analyzing patterns in the ”noise” received, a recurring signal was received every 96 seconds. In order to verify that it is sent from the Kamstrup meter, the ID has to be verified.

Removing the dummy-meter from the meter list

To observe how the stack handles telegrams from a simulated meter not inculded in the meterlist, the identification number (ID) of the meter was changed. Keeping the old ID in the meterlist of the collector. By observing the data received, the observation that the collector now receives the telegrams from the simulated meter, but neglects the data after checking it’s ID. Similar to the noise described in Section 6.6.3. Worth mentioning, is the fact that the new ID was the only data received and transmitted to the PC.

Reading of telegram content before decryption

Unfortunately, the extracted data did only contain the ID of the meter, not the whole telegram. Limiting the usefulness of the extraction, as the goal was to extract data similar to

what was done by Cyrill Brunschwiler in *Wireless m-bus security whitepaper* [2]. However, the data received on the PC-side contained some useful information. Firstly, we were able to see the corresponding address sent from the dummy-meter, confirming that the extraction was done correctly (the format of the data was transmitted to the PC correctly). Secondly, the signal received every 96 seconds contained the meter address [24 04 00 77], which is not the one provided by Kamstrup (30 34 32 34). By digging in the files provided by Kamstrup in order to add the meter to the MeterTool software, the confirmation that the meter address is 77000424 was found in an .xml file. This discovery gave us two results:

- The collector receives telegrams from the Kamstrup meter, but declines them due to some unknown fact
- The address used in the third setup was wrong.

In light of the knowledge that the collector in the third setup contained the wrong meter address, the tests of that setup had to be redone. The tests was conducted again, but with no success. There were still no signs of reception of telegrams in the APL layer.

Chapter 7

Discussion

7.1 Introduction

The goal of this thesis was to investigate the possibilities of using third party equipment to read utility meter data over Wireless M-Bus, in order to collect pressure and flow data from end users of the water supply network. The means used to investigate challenges with the coexistence of equipment from different manufacturers was through a literary study on similar attempts and adaptations of the Wireless M-Bus standard, and an implementation of a demonstration project. Several test-setups were implemented and tested in order to verify the equipment was functioning as intended, and to experiment with various setups. The results of these tests demonstrated the implications and difficulties that might arise when using third party equipment; *the goal of successfully reading a telegram from a Kamstrup meter was not accomplished*. However, the results provided some useful insights for future attempts at similar implementations.

7.2 Major findings

During finalizing the writing of this thesis, a major flaw in the demonstration project was detected by the author. This section describes the flaw, and seeks to present a guide/-solution for future attempts at a similar project. As the problem was detected after the demonstration project was finished, the author found it suitable to address the problem in the discussion part of the thesis.

In the ”release notes” file for the documentation, located under WMBUS Tools/Documentation/release_notes_1.0.6.txt in the project folder downloaded from Silicon Labs’s website [7], the release note 2518 under Change log 2.3.2, says ”The demo application for the APL don’t works with Mode C devices”. This means that the APL demo project is incompatible with the Kamstrup meter, as it operates with WM-Bus mode C1. There are no other mentions of this fact in other parts of the documentation, and this incompatibility is not

discussed or explained further. On Silicon Labs's websites it is specifically stated that the hardware and software used in this project should be able to communicate with mode C devices. Additionally, the collector and dummy-meter implemented in the first setup was using mode C1, and thus, the author of this thesis was working under the assumptions that the collector would work with commercial mode C devices as well. *This problem demonstrates the importance of reading documentation on third party software and equipment.*

As the release note specifically mentions that the *APL* demonstration does not work with mode C devices, there is reason to believe that the TPL demonstration might work with mode C devices, and thus the Kamstrup meter, based on the fact that no incompatability is mentioned for the TPL demo. Based on this information, an attempt at explaining how a similar demonstration project can be implemented with the TPL as the highest layer will be given in Section 7.2.1.

7.2.1 TPL demonstration project

This guide represents a suggestion for further work to achieve the main goal of this thesis, which was to demonstrate the interoperability between 3^{rd} party equipment and commercial meters. It is based on the author's experiences in the scope of this thesis, and is not tested in practise. The guide is intended for the use of the STACKFORCE protocol stack on an EZR32WG330 MCU mounted on a Wireless Gecko starter kit. It is intended to be used as a collector in mode C1, in order to achieve successful reception of telegrams from a Kamstrup PressureSensor.

As mentioned in Section 7.2, there is reason to believe that the TPL version of the protocol stack might work, as the stack is advertised as fully compliant to S, T, C and N(a-f) modes, but only the APL version is mentioned as not working in the release notes. In order to implement the collector with the TPL as the highest layer the following steps are recommended.

Download the stack, and import the project in IAR Embedded Workbench. The TPL demonstration project is located in "Wireless M-BUS stack/WMBUS Tools/Firmware/TPL firmware/ide/iar/Demo_SLWSTK6220A" named "Demo_apptpl.ewp".
The main file of the TPL demonstration application is located under Wireless M-BUS stack/WMBUS Tools/Firmware/TPL firmware/src/stack/src/apps/demos/tpl" named main_collector.c.

Configure the mode and target device as done in Section 6.3.1. The TPL demonstration project should then be ready for implementation.

Add a communications interface between the collector and a PC. The TPL demo should be able to use the same USBD library as used in the APL demo of this thesis.

Implement changes to the stack in order to receive and read telegrams from the Kamstrup meter. There are a few differences in the TPL application demo compared to the APL demo. The most notable ones are that the demo does not have an event callback function for reception of telegrams and no event callback for reading of received telegrams. The

TPL API does not contain an event callback for received telegrams[1], but it contains an event callback which triggers when a telegram is available, as seen in Table 7.1.

APL API callbacks	TPL API callbacks
wmbus_apl_evt_rx()	Not listed
wmbus_apl_evt_tlgAvailable()	wmbus_tpl_evt_tlgAvailable()

Table 7.1: APL vs. TPL callbacks

In the TPL demo, this callback is used to count telegrams received, and destroys the telegrams afterwards. This callback might be expanded to read the attributes of the telegram before destroying them. The TPL API lists a function named wmbus_tpl_getTlgAttr(), which might be used to read the telegram data and to write it to memory or send it to a PC before destroying the telegram. If possible, then a similar method as in the APL demo of extracting the telegram data may be used. Using the wmbus_apl_evt_tlgAvailable() function as an inspiration or guideline seems like a good choice.

Further implementation will rely on the results of this approach, and has to be evaluated thereafter. If this approach does not work, the thesis author would advise against using the stack for this purpose, as it would be very challenging, if not impossible, to fix the problem. This is due to the the fact that the stack is delivered in object code, and thus it might not be possible to implement a working fix. However, as some troubleshooting is expected to get the TPL version to work (if possible), then the following findings from the setups of the demonstration project might also be useful.

7.2.2 The setups

Although the goal of the project was not reached, the different setups and tests provided a lot of useful insights and results for future research on the development of a third party system for reading of measurements from a commercial meter. The findings from each setup will be presented in this section.

Findings from the first setup: General collector and dummy-meter

The first setup demonstrated a useful basis for testing and development of Wireless M-Bus solutions. The stack provided by STACKFORCE includes a lot of flexibility regarding choice of WM-Bus modes and the configuration of these, device architectures, interfaces to external devices, and API's. The system was only tested in a unidirectional communication scheme, with mode C1. Although other modes were not tested, the author is confident that other modes and configurations would be relatively simple to implement with the equipment used in this demonstration project.

It would be interesting to investigate how the demonstration project would handle equipment using other modes, especially modes C2 and the N modes. C2 would be interesting

[1] Like the one used to toggle LED's

as it is a bidirectional communication mode, which could enable more advanced functionality, not just meter readout. The N-modes would be interesting to test due to the long range they offer. S and T modes are considered inferior to C mode, as C mode enables either longer range or more frequent transmits as it uses a more energy efficient encoding. The use of C mode is also supported by the OMS group, which specifically states that modes S and T are deprecated/not recommended in their standard [5].

Findings from the second setup: Kamstrup meter and Kamstrup Meter Reader

The second setup was mostly set up for verification purposes, which was very useful. It was important to know that the problem with receiving telegrams from the Kamstrup meter did not originate from the meter itself. Additionally, when receiving telegrams from an unknown meter, detected in the HAL, the equipment used for this setup was crucial for being able to determine where the telegrams came from. The additional information from the meter installation files provided by Kamstrup allowed for the confirmation that the wrong address was being used for the Kamstrup meter. Unfortunately, the new information did not yield any progress in reading Kamstrup meter telegrams.

A useful addition to this setup would be to introduce a similar WM-bus analyzer to the one used by Cyrill Brunschwiler in the work on evaluating the security of the WM-Bus protocol [2]. Such an analyzer would not only confirm the meter is sending, but it would also give a lot of information on the format of the telegram. It would enable detection of what frame formats, layers, data headers and encryption the Kamstrup meter uses, which would be very valuable in order to narrow down the search of where the stack rejects the telegrams. It might also be possible to emulate such an analyzer with the software stack used in this project, if it is possible to extract the complete raw data of the received telegrams. Such a feature may be useful in the future if the proposed system works, because new meters may include additional layers or unknown fields. Sniffing the raw data from the telegrams of new meters may help in figuring out by which format they are transmitted, and may simplify the implementations of systems consisting of equipment from different vendors.

Findings from the third setup: Collector and Kamstrup meter

The third setup provided us with useful information regarding which layer the telegram *did not* reach. It never appeared in the Application layer, which enabled us to limit the search for where the stack rejected the telegrams to lower-level layers. The fact that the setup was unable to provide us with the CI information of the telegrams is a huge weakness of this demonstration project, as the CI field contains important information regarding how the telegram should be handled. Especially the fact that the stack was unable to alert about an *unknown* CI field was very unfortunate, as the information of knowing that the stack did not recognize the CI field would allow more dedicated investigation of which CI message the Kamstrup meter is using. The need for investigating the CI field further demonstrates the importance of the fourth setup, where the focus was on figuring out where the stack rejected the telegrams, by trying to extract and analyze the contents of the raw data. As the attempts in the fourth setup were unsuccessful, the need for a device such as the analyzer

discussed in the second setup is strengthened.

As mentioned in Section 2.2.2, C mode is the newest addition to the modes of the standard, which may introduce some complications before the manufacturers are able to implement the mode to work as intended, especially between vendors. This might be the reason as to why the stack is not compatible with C-mode devices, as advertised. A weakness of this setup is that it was only tested with one kind of meter. Even though the goal of the project was to be able to read telegrams from a deployed set of Kamstrup meters, it would be interesting to test for communication with commercial meters operating with different modes, to investigate the use of these.

Findings from the fourth setup: Collector, Kamstrup meter and dummy-meter

The fourth setup allowed us to determine that the collector was receiving telegrams from the Kamstrup meter, and that the telegrams were rejected somewhere in the stack, even though the meter was added to the meter list correctly. This was achieved by reading the content of the unencrypted telegrams after they were retrieved from the transceiver by the RF driver, and comparing the extracted data with data of telegrams received from the dummy-meter. Unfortunately, the setup was unable to extract the whole unencrypted telegram, but only the ID part of the address. This is a weakness of the method, because if the whole unencrypted telegram was available, there might be a way to configure the stack to be able to handle the structure of the mode C telegrams.

Aside from implementing the stack with the TPL version as described in Section 7.2.1, future implementation should focus on creating a functional ”sniffer” mode, to allow print-out of the full set of raw data from received telegrams. The results of this setup suggests that it might be possible to implement such a mode, as it demonstrates a partially functioning one. The need for such a ”sniffer” mode is supported by the work of Zeman *et al.* in *Wireless m-bus in industrial iot: Technology overview and prototype implementation.* [9], where they emphasise the need for a data sniffer in order to configure the collector to receive telegrams from different manufacturers. If the limitations of not having access to the source code prevents the development of a sniffer mode, a WM-Bus analyzer may be used instead, as suggested in the discussion of the second and third setup.

7.3 Limitations

The demonstration project presented in this thesis presents a system implemented with a EZR32WG330 MCU mounted on a Wireless Gecko development kit, using the STACK-FORCE WM-Bus protocol in order to try to receive telegrams from a Kamstrup Pressure-Sensor. It is not tested on other equipment, but the stack should be portable to other Silicon Labs wireless MCU’s with ARM based architectures. The system is only tested in mode C1, but the stack should be able to handle both uni- and bidirectional communications using S, T and N modes.

Chapter 8

Conclusion

This thesis investigated the use of third party equipment to read commercial water meters using Wireless M-Bus, based on the need for a general data collection system in the water supply network. This was done by implementing a general data collector and testing various setups in order to successfully receive a telegram from a commercial Kamstrup meter. Unfortunately, no telegrams were successfully received by the implemented data collector, as the software stack configuration used to implement the collector was incomplete, and thus it can be concluded that it is incompatible with the commercial meter.

The setups used to attempt reception, and later investigate problems with reception, included; (i) a collector and dummy-meter in mode C1, to ensure that the collector was able to receive telegrams in mode C1 and send the data to a connected PC, (ii) a verification of the operability of the Kamstrup meter with Kamstrup equipment, (iii) an attempt at reading the Kamstrup meter by configuring the collector according to EN 13757-4, an attempt which was unsuccessful, and (iv) an implementation in order to investigate possible explanations to why the collector was unable to receive telegrams from the Kamstrup meter, which was done by printing out the raw data of received telegrams and comparing them to the dummy-meter. Unfortunately, no valid cause was found. The explanation was later found in the release notes of the stack, stating that the configuration that had been used was unable to receive telegrams from meters using mode C. The specific cause of why the stack is unable to receive telegrams from mode C devices, is not given.

Although the goal of the demonstration project was not reached, the setups demonstrated promising functionality. The STACKFORCE stack includes a lot of flexibility, and works as intended when using dummy-meters. A proposal for another configuration of the stack has been provided in this thesis, which uses the transport layer as the highest layer. If the proposed configuration is unable to receive telegrams, the setups presented in this thesis may give inspiration on how to troubleshoot.

Chapter 9

Further work

The following list presents the recommendations on further work on the system presented in this thesis. Specific details are discussed in the Chapter 7, specifically Section 7.2.1, which proposes a TPL version of the demonstration project.

- Implement the TPL version of the demonstration project as proposed in Section 7.2.1.
- Implement a complete ”sniffer mode”, in order to be able to view the complete set of raw data of telegrams received, as discussed in the discussion of the fourth setup in Section 7.2.2.
- An alternative to the sniffer mode is to acquire a WM-Bus analyzer, as described in the discussion of the second setup in Section 7.2.2.
- If the TPL version of the demonstration project is able to receive telegrams from the Kamstrup meter, create an interface to a cloud service in order to remotely access data and configure the collector. The work by Lier, H. H., may be used as a reference [4].
- If the proposed system works, and an interface to a cloud service or something similar has been implemented, the system may be tested on actual meters connected to the water supply network.

Bibliography

[1] CENELEC, Aug. 2013. Communication systems for meters and remote reading of meters - part 4: Wireless meter readout(radio meter reading for operation in srd bands).

[2] Cyrill Brunschwiler, C. S. A., Jun. 2013. Wireless m-bus security whitepaper.

[3] Labs, S., Nov. 2017. Efm32 as usb device. https://www.silabs.com/documents/public/example-code/an0065-efm32-usb-device.zip, Visted at 2019-10.

[4] Lier, H. H., Jun. 2018. Demonstrering av konsept for innsamling og sammenstilling av data fra flere vannmlere ved bruk av trdls m-bus.

[5] OMS, Jan. 2014. Primary communication. Standard, Open Metering Systems Group, Marienburger Strae 15, 50968 Cologne, Germany.

[6] Sikora, A., Lehmann, P., Anantalapochai, N., Dold, M., Rahusen, D., Rohleder, A., Sep. 2014. Recent advances in en13757 based smart grid communication. Journal of Communications 9, 658–664.

[7] STACKFORCE, Mar. 2015. Stack reference manual. https://pages.silabs.com/wireless-m-bus.html?utm_source=web, Visted at 2019-08.

[8] Wikipedia.org, 2020. Osi model. https://en.wikipedia.org/wiki/OSI_model, Visted at 2020-01-16.

[9] Zeman, K., Masek, P., Krej, J., Ometov, A., Hosek, J., Andreev, S., Krpfl, F., May. 2017. Wireless m-bus in industrial iot: Technology overview and prototype implementation.

NTNU

Norwegian University of
Science and Technology