



TECHNISCHE
UNIVERSITÄT
WIEN

DIPLOMARBEIT

Low power wireless communication system

Development of an expandable wireless communication system based on
ATMEGA8L and nRF24L01+ for the deployment in laboratories

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Physikalische Energie- und Messtechnik

eingereicht von

Markus Kollarik, BSc

Matrikelnummer 01528951

ausgeführt am Institut für Angewandte Physik
der Fakultät für Physik der Technischen Universität Wien

Betreuung

Betreuer: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Gröschl Martin

Schleinbach, 05.02.2023

(Unterschrift Verfasser)

(Unterschrift Betreuer)

Abstract

The aim of this diploma thesis is to develop a programmable wireless transmission module for the application in laboratories. The module must satisfy certain selection criteria such as low price, availability, extensibility, interference resistance, transmission reliability, reliability and energy efficiency. In the first part of this thesis a literature study is carried out comparing different transmission standards and modules. The study led to the decision to develop the transmission module based on the ATmega8L microcontroller and the nRF24L01+ transceiver module.

The receive unit consists of a nRF24L01+ receiver module, an Arduino single-board microcontroller and a Raspberry Pi single-board computer. It is used to store incoming sensor data into a database. Furthermore, a webserver on the Raspberry Pi provides a web interface granting every user connected to the network access to the database data. The database data is displayed by the web interface in tabular and diagram format. In addition, it is possible to alert a specific user through email if a certain sensor value is out of a defined range.

The second part of this thesis describes the setup and implementation of this system. Finally, two sensor examples are carried out in order to demonstrate the technical functionality of the system.

Zusammenfassung

Zielsetzung dieser Diplomarbeit ist die Entwicklung eines programmierbaren drahtlosen Übertragungsmoduls für den Einsatz in Laboren. Das Modul muss bestimmten Auswahlkriterien gerecht werden. Die Auswahlkriterien sind geringe Kosten, Verfügbarkeit, Erweiterbarkeit, Störfestigkeit, Übertragungssicherheit, Zuverlässigkeit und Energieeffizienz. Der erste Teil der Arbeit umfasst eine Literaturstudie, welche unterschiedliche Übertragungsstandards und Übertragungsmodule miteinander vergleicht. Die Studie führte zu dem Ergebnis ein Übertragungsmodul auf Basis des ATmega8L Microcontrollers und des nRF24L01+ Transceiver Moduls aufzubauen.

Die Empfangseinheit umfasst ein nRF24L01+ Empfangsmodul sowie ein Arduino Einplatinencomputer und ein Raspberry Pi Einplatinencomputer. Sie wird dazu eingesetzt eintreffende Daten in einer Datenbank abzuspeichern. Des Weiteren wird von einem Webserver welcher auf dem Raspberry Pi läuft, ein Webinterface zur Verfügung gestellt, welches jedem Benutzer im gleichen Netzwerk Zugang zu den Daten in der Datenbank gewährt. Die Daten werden in Tabellen sowie in Diagrammen dargestellt. Ebenfalls ist es möglich einen definierten Benutzer mittels E-Mail zu benachrichtigen, falls ein bestimmter Sensorwert außerhalb eines definierten Bereichs liegt.

Im Zweiten Teil der Arbeit wird der Aufbau sowie die Umsetzung des Systems beschrieben. Abschließend werden zwei Sensor Beispiele durchgeführt um die technische Funktionstüchtigkeit des Systems zu zeigen.

Contents

1	Introduction	8
1.1	Background	8
1.2	Problem Statement	8
1.3	Purpose	8
1.4	Method	9
2	Automation Systems	10
2.1	Types of Control Systems	10
2.1.1	Centralized system	10
2.1.2	Decentralized system	11
2.1.3	Hybrid system	11
2.2	Topology	12
2.2.1	Star topology	12
2.2.2	Mesh topology	12
2.3	Connection	14
2.3.1	Wired Systems for Automation	14
2.3.1.1	KNX	14
2.3.1.2	LCN	14
2.3.1.3	Loxone	15
2.3.1.4	digitalSTROM	15
2.3.1.5	Discussion	15
2.3.2	Wireless Systems	16
2.3.2.1	Zigbee	16
2.3.2.2	EnOcean	16
2.3.2.3	Bluetooth	16
2.3.2.4	WLAN	17
2.3.2.5	Thread	17
2.3.2.6	nRF24L01+	18
2.3.2.7	Discussion	18
2.3.3	Wired Systems for on-board communication	21
2.3.3.1	1-Wire	21
2.3.3.2	UART	21
2.3.3.3	CAN	22
2.3.3.4	SPI	22
2.3.3.5	I2C	24
2.4	Tasks	25
2.4.1	Temperature Measurement	25
2.4.2	Air Pressure Measurement	26
2.4.3	ADC - Analog Digital Converter	26
2.4.4	Other Tasks	27
2.5	Conclusion	28
2.5.1	Wireless Transceiver - RS Components	28
2.5.2	Wireless Transceiver - Mouser Electronics	28
2.5.3	Wireless Transceiver - Amazon	29
2.5.4	Discussion	29

3	System Setup	30
3.1	System Structure	30
3.2	Module Description	32
3.2.1	Raspberry Pi	32
3.2.1.1	System Setup	33
3.2.2	Admin PC	35
3.2.2.1	System Setup	35
3.2.3	Arduino	37
3.2.4	Transceiver - nRF24L01+	38
3.2.4.1	Radio Control	39
3.2.4.2	Receiver path:	40
3.2.4.3	Transmitter path:	41
3.2.5	μ C ATmega8L	42
3.2.6	Programmer - Arceli TL866II Plus	45
4	System Implementation	48
4.1	Basic Operation Setup	48
4.1.1	Microcontroller - ATmega8L	50
4.1.1.1	Configure ATmega8(L) for power-save mode	51
4.1.1.2	Configure nRF24L01+	53
4.1.2	Prototype Board	55
4.1.2.1	Power supply	56
4.1.2.2	Asynchronous clock	56
4.1.2.3	Buffer capacitors	57
4.1.2.4	ADC	57
4.1.2.5	ATmega8L, nRF24L01+, Debug LED, BMP180	57
4.1.2.6	Calculate power consumption	58
4.1.3	Arduino	61
4.1.4	Raspberry Pi	63
4.1.4.1	Installations and settings	65
4.1.4.2	Database	69
4.1.4.3	Website	72
4.2	Step by step instruction for different sensors	78
4.2.1	Example: Temperature/Pressure Measurement	78
4.2.1.1	Temperature/Pressure Sensor	78
4.2.1.2	Microcontroller - ATmega8L	79
4.2.1.3	Arduino	81
4.2.1.4	Raspberry Pi	82
4.2.2	Example: ADC Measurement	84
4.2.2.1	ADC	84
4.2.2.2	Microcontroller - ATMEGA8L	85
4.2.2.3	Arduino	86
4.2.2.4	Raspberry Pi	86
5	Results	87
5.1	Sensor 1/2 example - nRF24L01BmpAdc.c	87
5.1.1	Settings	87
5.1.2	Results	87
5.1.2.1	ARD & ARC variation	88
6	Summary and Outlook	90

References	91
7 Appendix	94
7.1 Schematic	94
7.2 Code	95
7.2.1 ATmega8L	95
7.2.1.1 MAIN	95
7.2.1.2 AES	95
7.2.1.3 BMP085/180	95
7.2.1.4 I2C/TWI	95
7.2.1.5 nRF24L01	95
7.2.1.6 SPI	95
7.2.2 Arduino	96
7.2.2.1 MAIN	96
7.2.2.2 Library - RF24	96
7.2.3 RPi	96
7.2.3.1 Python	96
7.2.3.2 Webserver	96
7.3 Measurement Table	97

List of Figures

1	Star topology [6]	12
2	Mesh topology [6]	13
3	1-Wire lock [29]	21
4	SPI bus example [32]	23
5	TWI/I2C bus example [34]	24
6	MEMS temperature sensor [35]	25
7	MEMS piezoresistive pressure sensor [41]	26
8	Block diagram of the system structure	30
9	Picture of Raspberry Pi 4 Model B [43]	32
10	PuTTY configuration window	36
11	WinSCP login window	36
12	Arduino MEGA Breadboard Pins [44]	37
13	Arduino IDE Settings	37
14	Picture of Transceiver nRF24L01+	38
15	nRF24L01+ state diagram [27]	40
16	Atmega8L pins [46]	43
17	Arceli TL866II Plus [47]	45
18	Xgpro v9.00 main window	45
19	Xgpro v9.00 FUSE. Bits tab	46
20	ATmega8L CKSEL settings - frequency [45]	47
21	Basic system structure and used programming languages	48
22	ATmega8L code flowchart	50
23	Prototype board	55
24	Prototype board - schematic	56
25	Measurement current consumption prototype board	58
26	Measurement current consumption prototype board - zoomed in	59
27	Circuit design of the Arduino receiving unit	61
28	Wiring diagram of the Arduino unit	62
29	Arduino code flowchart	62
30	RPi system structure and used programming languages	63
31	Website requires username and password	67
32	<code>saveTest.py</code> flowchart	69
33	<code>checkdaily.py</code> flowchart	70
34	Startpage after logging in	72
35	Sidebar menu Clients	74
36	Error message Arduino	75
37	<i>ADC Sensor Data</i> - Python script started	75
38	Sidebar menu Logs - Table	76
39	Sidebar menu Logs - Graph	77
40	System structure of the Temperature/Pressure Measurement Example	78
41	Temperature measurement data	88
42	Pressure measurement data	88

List of Tables

1	Overview over the most common used wired automation systems	16
2	Overview over the most common used wireless automation systems	20
3	Four different SPI modes	23

4	Wireless Transceiver - RS Components	28
5	Wireless Transceiver - Mouser Electronics	28
6	Wireless Transceiver - Amazon	29
7	Comparison available transceivers regarding requirements	29
8	Different ARC and ARD values	89
9	Measurement data table part 1	97
10	Measurement data table part 2	98

1 Introduction

1.1 Background

Laboratory automation has its origins in the 1970s in the chemical industry. Its goal is to improve the reproducibility of certain processes and to reduce costs. Another benefit of automation processes is to reduce employees workload and therefore counter the lack of skilled labor as well as improved reliability [1]. Devices which are used include sensors (e.g., temperature, pressure, flow rate, ...) to keep stable environmental conditions during experimental measurements (**S**tandard **T**emperature and **P**ressure - STP) and actuators (e.g., pumps, valves, thermostat, stirrer, ...). The devices are connected to a computer (by wire or wireless) where the data is processed. Typical examples include storing sensor readings in a database and display the data in graphical form or send data back to an actuator to e.g., open a valve at a certain pressure level.

1.2 Problem Statement

For small laboratories and simpler tasks existing standards for data gathering and transmission are way too complex and costly. The objective of this thesis is to develop a cheap, wireless transceiver module which meets certain selection criteria as discussed in section 1.3.

1.3 Purpose

Although a wide variety of different automation technology standards had been developed over the course of the last decades, it is hard to keep up what the different standards offer and whether they will be still available in 30 years from now.

Goal of this master thesis is to conduct a literature review about the most common automation technology standards and select one of these standards to develop a programmable wireless transmission module which offers the user complete freedom of choice for security, extensibility and the type of sensor.

The chosen wireless standard must meet following selection criteria: price, availability, extensibility, interference resistance, transmission reliability, reliability and energy efficiency. The developed module offers connectivity to commonly used sensors and transmits their current status to a gateway to cause a reaction (e.g., send email to user if temperature in laboratory rises above 20 °C). The collected data can be displayed through a web interface in tabular and graphical view.

1.4 Method

The thesis is divided into three main parts. In the first part a literature review is conducted for the purpose of choosing the best suited automation technology standard for the project. The second part contains a technical description of the developed hard- and software. In the third part one can find a manual to replicate the project and a rough description of what has to be changed in hard- and software in order to work with different sensors/actuators.

It is not an all-encompassing manual, which would easily exceed the limits of this thesis by far. If the reader is looking for a deeper level of knowledge, they must read up on that subject, in particular the microcontroller manual and the manual of their chosen sensor.

2 Automation Systems

An automation system is the integration of sensors, controls and actuators to perform a certain task with reduced human intervention [2]. Early automation systems reach back to the Greeks and Arabs about 300 BC (float regulated water clock) but the real breakthrough was during the industrial revolution [3]. With the beginning of electrification more and more task got automated which led to the development of the first general home automation network technology X10 in 1975 [4].

The main purpose of an automation system is to reduce energy costs and work force. Other benefits include improved reliability, comfort and safety. Reliability and comfort are provided for example by devices controlling the ambient parameters like temperature in a laboratory and therefore guarantee same conditions with no human interaction. Energy savings can be achieved by smart energy management. Warning systems (e.g., rising CO₂ concentrations) address the safety aspect.

In a laboratory many tasks can be automated which include controlling lights, valves, thermostat, smoke detectors and cameras. These tasks require a power source on site and are not the main goal of this master thesis.

For a cheap, low power wireless automation system typical tasks are monitoring ambient temperature, pressure, humidity as well as motion detection. Therefore, the focus will be in finding the perfect fit for depicted tasks.

2.1 Types of Control Systems

When building an automation system, one has to choose between a centralized, a decentralized or a hybrid system. In the following section the three different control systems will be discussed.

2.1.1 Centralized system

In a centralized system one smart element controls all other elements. If it fails everything fails and no data will be transferred. It is essential for the systems because it handles incoming data from the sensors and controls the actuators [5].

An advantage of this systems is that only the smart element of the systems has to be intelligent and has to actively listen to all the incoming data traffic. This requires a much simpler internal structure for every sensor, which means that every sensor is cheaper and easier to maintain, it can be easily expanded. Another benefit is that low power energy systems can be realized with a centralized system.

The central component managing all the incoming data is called master. It can be differentiated between two different polling modes. First one is that the master requests in regular time intervals (programmable) data from the sensors (slaves). The second polling method is that every slave becomes a master and sends its data only when a value changes or in regular intervals to the central component which now is a slave. This system is called

a multi-master system and its advantage is that the sensors consume much less energy. The processing units of the sensors can sleep most of the time and have to be active only a fraction of the time. This expands battery life drastically.

2.1.2 Decentralized system

In a decentralized system every element is smart. If one element fails the remaining elements keep working. The actuators listen to the commands on the bus and react to specific commands [5].

While the initial building cost of a decentralized system maybe lower in comparison to a centralized system every expansion adds to the cost. It is therefore only recommended for relatively small systems or applications where its function has to be guaranteed even if one system fails. Another con is the complex installation, it is often necessary to call a professional installer [5].

2.1.3 Hybrid system

A hybrid system combines the centralized and decentralized system. If the central controller disconnects all essential elements remain functional. if one element disconnects the remaining elements keep working. It has therefore the highest reliability but it is also the most expensive one.

2.2 Topology

One of the first things you have to consider in building an automation system is to define how various components communicate in a network. A network topology defines the arrangement of various components in a network. Two of the most common architecture types for networking will be discussed in the following section.

2.2.1 Star topology

In a star topology all sensor nodes have a direct connection to a central point/hub/gateway (See figure 1 [6]).

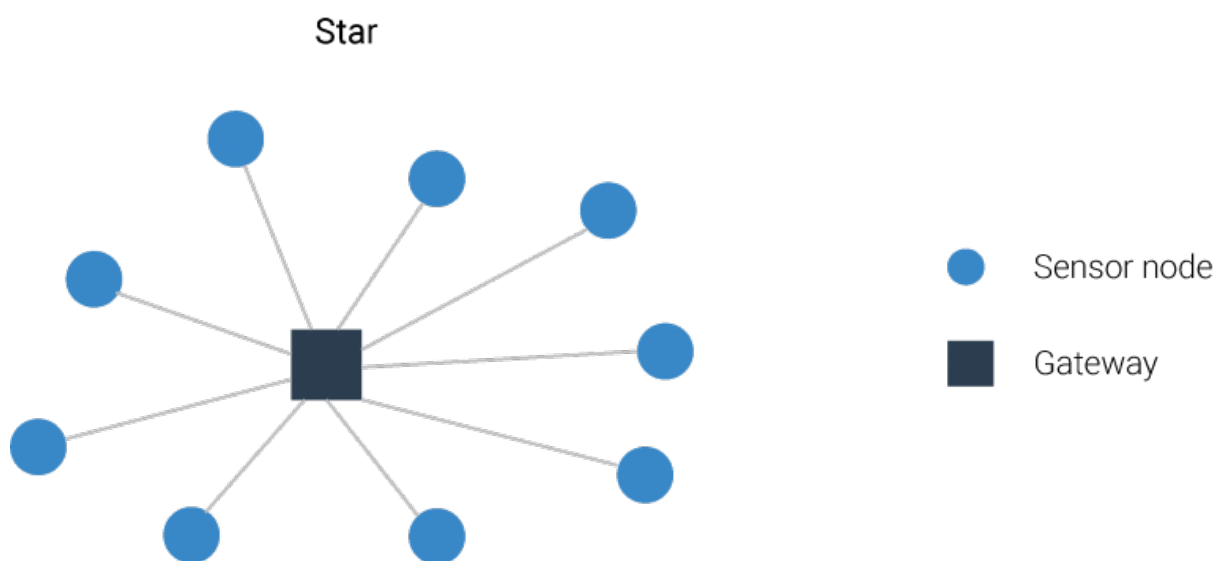


Figure 1: Star topology [6]

All traffic has to pass through the central point where typically the data is processed. It is an easy topology to design and implement and the simplicity of adding new nodes is one of its advantages. A major disadvantage is that the hub represents a bottleneck for incoming data and therefore has to provide high bandwidth, another con is that if the hub fails the system fails.

2.2.2 Mesh topology

In full mesh networks all nodes are interconnected (see figure 2 [6]). Messages hop from one node to another until they reach their destination (e.g., gateway). Every sensor node acts as a repeater that relays data from other nodes as well as transmitting its own data.

A partial mesh network is similar to a full mesh, but instead of every sensor node only some have a built-in relaying function. Mesh networks are commonly used for extending the coverage of short-range wireless technologies (e.g., Zigbee, Z-Wave) but on the other hand many nodes have to be installed to cover large buildings which makes mesh networks very expensive to install. An advantage of mesh networks is that if one node fails the data simply takes another path through the network. This improves reliability of the system. A

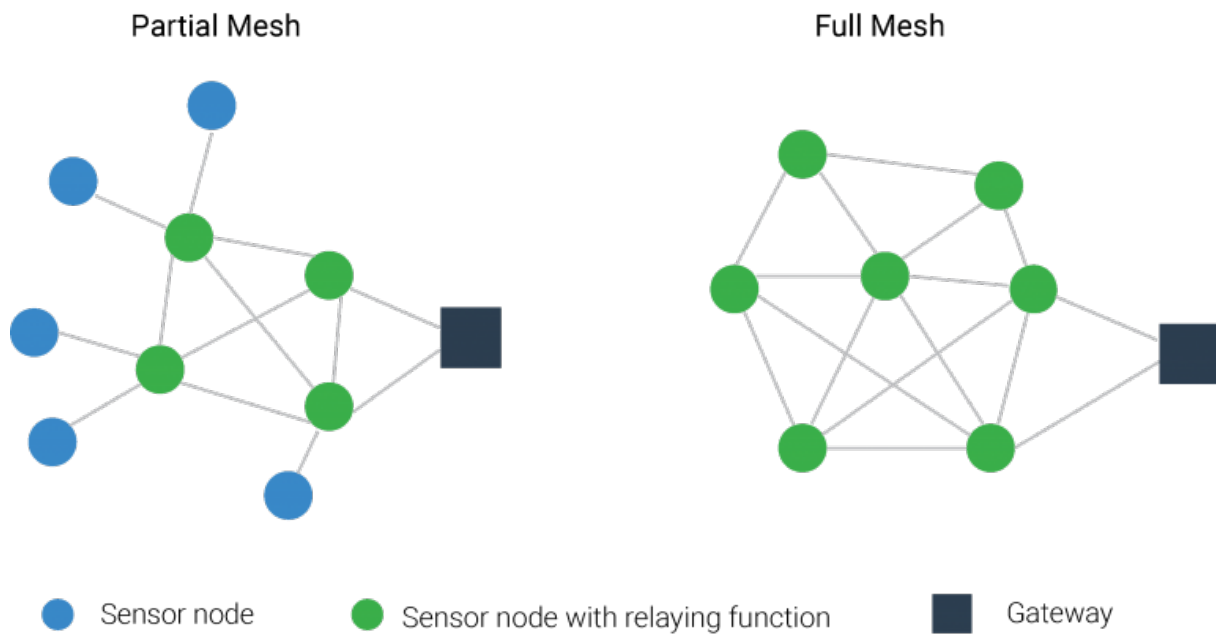


Figure 2: Mesh topology [6]

major disadvantage of mesh networks is the complicated network setup, management and maintenance. Another downside of mesh networks is its high power consumption. Each node has to be constantly “awake” and listen for incoming data to relay. High traffic will drastically decrease its battery life.

2.3 Connection

If two or more components in a (automation-)system need to communicate a connection needs to be established. The connection serves the purpose of transferring data and commands between different components. A distinction is made between *wired* and *wireless* systems. Each system has its benefits and drawbacks. In the following section a selection of the most common standards will be described.

2.3.1 Wired Systems for Automation

This section covers wired systems for automation purposes, not to be confused with wired systems commonly used for on-board communication (ref. section 2.3.3) which are suited for short distances and low bandwidth.

2.3.1.1 KNX

KONNEX (lat. connexio; engl. connection) is an open standard for commercial and domestic building automation. KNX is the successor of three earlier standards the EIB (**E**uropean **I**nstallation **B**us), BatiBus and EHS (**E**uropean **H**ome **S**ystems) [7]. Its goal was to ensure compatibility between different devices and systems from different manufactures.

Each device in a KNX network is connected to the power line and a control network. The wires can be laid separately or parallel. The control network uses twisted pairs (two wires) with differential signals and a voltage difference of 30 V. The bit rate is up to 9600 bit/s and the network is able to communicate up to a size of 10000 devices.

Many different devices can be integrated into a KNX network for example alarm system, heating, ventilation, lighting and sun protection. KNX devices can be programmed which adds flexibility to the system.

A disadvantage of KNX systems are high initial costs which are barely justified. Costs are partially very high, for example power supplies cost up to 370 € which is disproportionate for a transformer and few electronic components [8]. Another downside is that camera feeds and intercoms cannot be transmitted because of the slow bit rate. If you want to use such devices KNX has to be combined with other systems. When using KNX sensors and actuators one must calculate with a current consumption of 5 to 8 mA per device - for power saving KNX is not the recommended system.

2.3.1.2 LCN

Local Control Network is a proprietary building automation system released in 1993 by Issendorff KG. LCN is a decentralized organized system [9].

Each LCN module is identical and can be programmed to function as an actuator or a sensor. The modules are connected to the live “hot” wire, neutral, protection earth and a fourth wire for data transfer. The bit rate is equal to KNX networks (9600 bit/s) and up to 30000 modules can be controlled.

LCN is mainly used in commercial buildings but a home installation is possible. For the programming of the modules a qualified electrician should be contacted. The biggest downside of LCN is the proprietary hard- and software, for that reason alone it can be said that LCN should not be used for automation purposes. No one knows whether Issendorff will still exist in the next few years.

2.3.1.3 Loxone

Loxone is an Austrian technology company specialized in building automation. Founded in 2009 Loxone's goal is to offer a cheap and easy to use alternative to existing automation standards [10][11][12].

Loxone devices are connected to a 24 V power supply and a communication wire. Interfaces to automation systems from other companies like KNX, 1-Wire, EnOcean, RS485, RS232 and more are available. Advantages of Loxone devices are the easy to setup bus system, integration of different systems, security (usable without internet connection) and in comparison to KNX, its lower price. On the other hand, Loxone is a proprietary building automation system and therefore shares the downside of such a system with LCN.

2.3.1.4 digitalSTROM

DigitalSTROM differs from the above-mentioned systems that it uses the main power supply for data communication. It is not required to lay new wires which lowers the initial cost of digitalSTROM equipped automation systems [13].

DigitalSTROM uses like many other automation systems proprietary devices which are quite expensive but its high DIY-potential is its advantage in comparison to KNX, LCN and Loxone. A disadvantage refers to the security aspect. DigitalSTROM devices are not encrypted and therefore the system opens a gateway for hackers if it is connected to the internet.

2.3.1.5 Discussion

Each of the wired building automation systems which had been discussed have in common, that they are wired. This means that for the interconnection of every sensor and actuator in the system to the gateway a cable has to be laid. If it is desired that the cables fit into the wall an electrician is often required, especially in existing buildings. Electricians can be very expensive so it is worth considering a power-line type communication standard like digitalSTROM which uses the existing power line for communication. It should not be forgotten, that wired systems lack flexibility, consume more energy (ref. KNX section 2.3.1.1) and need proper cable management. In summary it can be concluded, that wired automation systems are hard to setup and extend but easy to sustain.

Benefits in comparison to wireless systems are improved reliability (works even in magnetic shielded environment), high security (no wireless hacking/sniffing, exception: digitalSTROM) and faster service (up to 1Gbps if Ethernet is used) [14]. Table 1 compares significant features of the presented wired automation systems.

Standard	KNX	LCN	Loxone	digitalSTROM
Open / proprietary system	open and int. standard	proprietary	proprietary	proprietary
Interfaces to other systems	vice versa	yes, e.g. EnOcean, BACnet	yes, e.g. KNX, EnOcean	yes, e.g., EnOcean
Programming effort	high	high	high	medium
Privacy & security	high	high	high	missing

Table 1: Overview over the most common used wired automation systems

2.3.2 Wireless Systems

In this section the most common wireless automation systems will be introduced. Section 2.3.2.6 presents the NRF24l01+, an alternative to the established systems which offers similar features for DIY-automation projects.

2.3.2.1 Zigbee

Zigbee is a wireless ad hoc network which was standardized in 2003. The name “Zigbee” refers to the zig-zag dance (waggle dance) of the honey bees.

Zigbee is based on the IEEE 802.15.4 standard for small, low-bandwidth, low-power and low-distance digital radios used for automation purposes. The specification operates in the ISM (Industrial, Scientific, Medical) radio band at 868 MHz and 2.4 GHz, a band which is internationally reserved for such purposes [15]. Data rates of 250 Kbps are provided as well as an AES-128 (Advanced Encryption Standard) encryption. Zigbee devices offer an 10-100 m light-of-sight transmission range (depends on environmental characteristics) but its mesh topology allows for networks to span over long distances. A huge security issue of the Zigbee standard is that Zigbee uses fallback keys for encryption negotiation which are known and cannot be changed. This makes the encryption highly vulnerable [16].

2.3.2.2 EnOcean

Standardized in 2012 EnOcean’s main goal is to offer ultra-low power wireless technologies for consumers. EnOcean achieves this goal by utilizing energy harvesting methods like slightest mechanical motion, indoor light or temperature differences [17]. Electromagnetic, solar cells and thermoelectric energy converters transform these energy fluctuations into usable electrical energy which can be further used for data transmission.

EnOcean operates in the ISM band at 868 MHz at a data rate of 125 Kbps. Its topology is mainly point to point or star and that offers a transmission range of up to 300 m in the open and 30 m in buildings. To reduce packet collisions and improve reliability EnOcean transmits three data packets in a pseudo random interval. AES-128 encryption is optional but included in every module shipped after April 2015.

EnOcean applications include occupancy sensors, light sensors, temperature sensors, humidity sensors, CO2 sensors and metering sensors.

2.3.2.3 Bluetooth

Bluetooth is a wireless technology standard released by Intel, Ericsson and Nokia in 1998. Its name derives from the 10th-century Danish king Harald “Bluetooth” Gormsson [18].

Bluetooth is implemented in every smartphone, wireless headphones, smartwatches and personal computers. The specification operates in the 2.4 GHz ISM band offering a data rate of up to 2 Mbps. In 2010 Bluetooth 4.0 was released with new power saving features. This so-called Bluetooth LE (**L**ow **E**nergy) opened the automation sector for the standard with a product range similar to Zigbee's. With Bluetooth 4.0 mesh topology found their way into the network standard [19].

Bluetooth uses E0 (stream cipher) to encrypt its data which is usually safe but as every wireless technology it can be attacked if enough effort is put into [20].

2.3.2.4 WLAN

WLAN stands for **W**ireless **L**ocal **A**rea **N**etwork a wireless network that links devices to form a LAN within a limited area. WLANs based on the IEEE-802.11 standards are called Wi-Fis.

The IEEE-802.11 standard was first introduced in 1997 and uses the 2.4 GHz ISM band. With Wi-Fi 5, which is the current standard (2022), the frequency range got extended to 5 GHz offering a data rate of up to 1.3 Gbps [21].

In comparison to the above introduced wireless network standards, Wi-Fi offers a much higher data rate (3-4 magnitudes higher) and the fact that it is used almost everywhere in the IT sector illustrate the high security standard. On the other hand, Wi-Fi devices have a very high power consumption (they have to be always on to receive incoming data) and therefore it is not possible to utilize WLAN for energy saving purposes. WLAN smart devices use 2.4 GHz and a lower transmission rate for a higher transmission range. However wireless Wi-Fi devices (barely available motion detectors) need frequent battery change and it is therefore not recommended to use it for low maintenance systems. Its high data rate on the other hand opens a possibility for audio or video streaming.

2.3.2.5 Thread

Thread is an open-source low-power mesh networking standard designed for Internet of things products. Members include Amazon, Apple, ARM, Bosch, Google, IKEA, LG, NXP Semiconductors, OSRAM, Samsung, Schneider Electric and Qualcomm [22][23][24].

This fairly new standard is based on the IEEE 802.15.4 standard operating at 2.4 GHz with a data rate of 250 Kbps (ref. Zigbee). An open-source implementation of Thread called OpenThread managed by Google is available. Thread is very similar to Zigbee such as it uses AES encryption. First Thread products were released in late 2020, first one was the HomePod mini [25].

Thread has got a huge advantage above all other presented wireless communication standards. The fact that many big tech companies are on board will probably lead to market domination in the next few years. The present situation (August 2022) is different because it's a fairly new standard and therefore not many devices are available.

For future DIY wireless projects, the nRF5340 designed by Nordic Semi Conductors is an interesting chip offering already a development kit released in the beginning of 2021 [26].

2.3.2.6 nRF24L01+

Unlike the previously featured wireless systems the nRF24L01+ is not a wireless standard that is used in smart devices. The nRF24L01+ is a single chip transceiver developed by Nordic Semi Conductors and offers wireless communication for little money. For 1.5 € per piece the chip offers an extremely cheap alternative to transceiver modules which are based on the Bluetooth, Zigbee and Thread standard. In the following key features of the nRF24L01+ are listed from the datasheet [27].

- Worldwide 2.4 GHz ISM band operation
- 250 Kbps, 1 Mbps and 2 Mbps on air data rates (annot. bps...bit per second)
- 11.3 mA TX at 0dBm output power (annot. TX...Transmitter; 0 dBm $\hat{=}$ 1 mW)
- 13.5 mA RX at 2 Mbps air data rate (annot. RX...Receiver)
- 900 nA in power down
- 26 μ A in standby-I
- 1.9 to 3.6 V supply range
- Enhanced ShockBurst™ (annot. special protocol for high power transmission)
- 6 data pipe MultiCeiver™ (annot. single module can listen up to 6 other modules at the same time)

Typical applications include ultra-low power sensor networks, home and commercial automation, asset tracking systems and advanced media center remote controls.

In combination with a power efficient microcontroller many wireless DIY projects can be implemented.

2.3.2.7 Discussion

Wireless systems are a suitable alternative to wired systems when upgrading existing buildings with “smart” features. Cheaper initial cost, less time spend in the setup process (No wires) and easy upgrades are few of its advantages.

On the other hand, wireless systems still need (although less) electrical power which is provided via wires, batteries or for many applications sufficient, energy harvesting (EnOcean). Another downside is that metal shielding (insulation, underfloor heating, plasterboard) blocks electromagnetic waves and extra care needs to be taken to the placement of wireless systems. Electric motors can cause interference and disturb communication. Furthermore, many applications are not wireless (e.g., security cameras, alarm systems,

entertainment systems) they need a constant power supply. In that case wired automation systems are more suitable.

Table 2 shows key features of the presented wireless automation systems.

Standard	NRF24	Zigbee	EnOcean	Bluetooth 5.0 (BLE)	Wifi	Thread
Network topology	star, tree	mesh	star, point to point	star, point to point, mesh	star	mesh
Range direct	long (>30m)	medium (10-15m)	medium (30m)	long (<100m)	long (50m)	medium (10-15m)
Max. transmission speed	2 Mbps	250 Kbps	125 Kbps	2 Mbps	>1Gbps	250 Kbps
Frequency	2.4 GHz	868 MHz 2.4 GHz	868 MHz	2.4 GHz	2.4 GHz 5 GHz	2.4 GHz
Security	depends on μ C	AES-128	AES-128 possible	E0 (stream cipher)	WPA2/WPA3	AES-128
Power usage	very low	low	very low	high	high	low
Availability	infinite	medium	low	low	high	low but \uparrow
Reliability	high	medium	high	high	low	high

Table 2: Overview over the most common used wireless automation systems

2.3.3 Wired Systems for on-board communication

Following section presents commonly used serial data busses.

2.3.3.1 1-Wire

1-Wire is a simple to use bus system developed by Dallas Semiconductor Corp. (today Analog Devices) [28]. It uses one data line (DQ) which acts as power supply, transmit and receive line. In practice 1-Wire needs a second line Ground (GND) for communication. The asynchronous data transmission (no clock signal) offers a data rate of 16.3 Kbps and therefore, a long transmission range of up to 100 m can be achieved. Each device (slave) which is connected to the master (μ C, PC) has a unique 64-bit identifier and an internal capacitor (800 pF) to store charge and power the device during the communication procedure.

1-Wire is used for small inexpensive devices like digital thermometers and weather instruments with extremely small power usage ($<5 \mu$ W). Another application includes door locks and the use of Dallas key or iButtons (see figure 3 [29]).



Figure 3: 1-Wire lock [29]

2.3.3.2 UART

Universal Asynchronous Receiver-Transmitter is a hardware device for establishing an asynchronous serial communication. UART is commonly used for the communication between computers and external devices. RS-232 (introduced in 1960), RS-485 and USB (Universal Serial Bus) are the most popular serial standards.

For a UART transmission to work there are several settings which need to be defined on both the transmitting and receiving side. The following enumeration shows the typical settings for a UART communication.

1. Baud Rate: 9600 bps
2. Data bit size: 8 bits
3. Parity: None
4. Stop bits: 1 bit
5. Flow Control: none

2.3.3.3 CAN

The **C**ontroller **A**rea **N**etwork is a vehicle bus system developed by Bosch in 1983 [30]. Its purpose was to reduce cable in cars to save copper and weight. Nowadays CAN has many applications beyond the initial one including building automation, 3D printers, elevators and medical instruments and equipment. The CAN bus system requires two logic signals in order to operate:

1. CAN-H: positive CAN signal (dominant high)
2. CAN-L: negative CAN signal (dominant low)

A third signal CAN-GND (Ground) is optional but often combined with a 5 V power supply.

CAN is a multi-master serial bus system using twisted pairs to reduce influence of external electromagnetic radiation and crosstalk. It offers protection against data collision (CSMA/CR) and data rates from 10 Kbps up to 1 Mbps. With 10 Kbps a transmission ranges up to 5 Km can be achieved, increasing the data rate to 1 Mbps reduces it to 20 m. CANs theoretical node limit is 128 units per bus, in practice 110.

2.3.3.4 SPI

Serial **P**eripheral **I**nterface is a serial synchronous communication bus system developed in 1987 used for short distance communication. It is primarily used for the communication between different ICs (**I**ntegrated **C**ircuits) and microcontrollers. The SPI bus system requires four logic signals in order to operate [31]:

1. SCLK/SCK: Serial Clock
2. MOSI: Master Out Slave In
3. MISO: Master In Slave Out
4. CS/SS: Chip/Slave Select

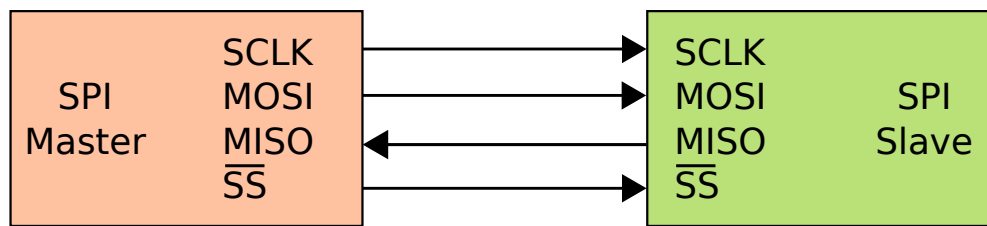


Figure 4: SPI bus example [32]

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Table 3: Four different SPI modes

A simple SPI bus example is shown in figure 4 [32].

Every chip needs a separate CS/SS signal for communication hence the number of slaves is limited to the number of output pins of a microcontroller. To solve this problem an inverse multiplexer (IMUX) or an GPIO expander chip can be used.

SPI devices communicate in full duplex mode this means that on every clock cycle master and slave transmit their respective data. If a master requests data from the slave two transmissions need to take place. The first transmission contains the command and in the second one a dummy byte is sent to give the slave an opportunity to send requested data to the master.

There are four different modes to set up polarity and phase of the clock. Table 3 illustrates the different settings.

CPOL (Clock Polarity)

- 0: clock idles low (0); leading edge is rising edge
- 1: clock idles high (1); leading edge is falling edge

CPHA (Clock Phase)

- 0: half cycle clock idle, followed by half cycle clock asserted
- 1: half cycle clock asserted, followed by half cycle clock idle

SPI is used for many low power applications e.g., the in section 2.3.2.6 presented transceiver (nRF24L01+) uses SPI to communicate with a microcontroller.

2.3.3.5 TWI/I2C

Two Wire Interface or Inter Integrated Circuit is a synchronous serial communication bus developed by Philips Semiconductors (known today as NXP) in 1982 [33]. It is commonly used for the communication between different ICs. The I²C bus requires two logic signals in order to operate:

1. SDA: Serial Data Line
2. SCL: Serial Clock Line

A common GND (Ground) connection is not required but recommended.

A simple TWI/I2C bus example is shown in figure 5 [34].

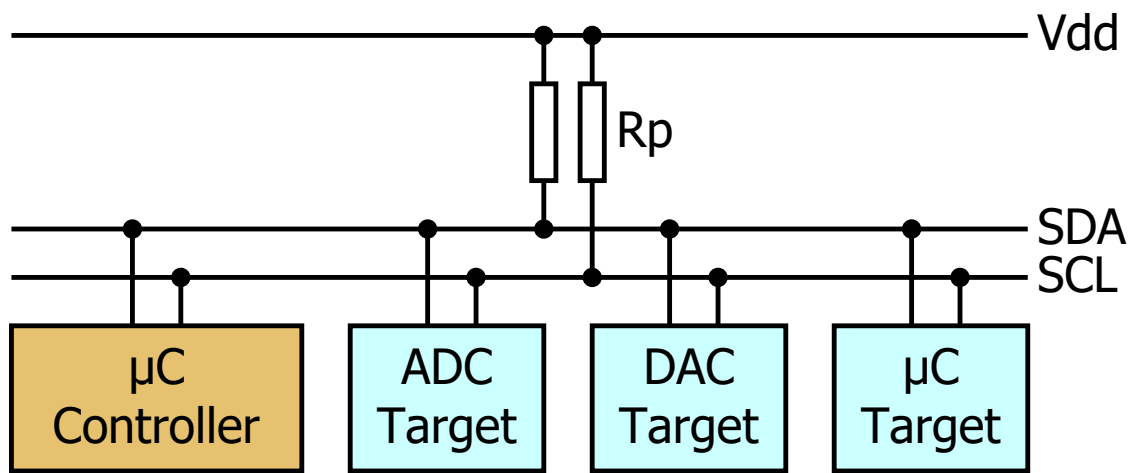


Figure 5: TWI/I2C bus example [34]

I2C uses open-collector or open-drain lines which need to be pulled up with external resistors (typical 4.7/10 K Ω ; R_p) to the supply voltage. One of I2C's features is its simplicity and low manufacturing cost but low data rate and higher power consumption in comparison to SPI. The number of nodes is limited by the address space (7 bits $\hat{=}$ 128) and its total bus capacitance of 400 pF which in practical equals to a few meters bus length. Due to the bus high impedance and therefore low noise immunity extra care needs to be taken in placing and connecting I2C components.

In an I2C bus system following procedure takes place during a transmission. The master pulls the SDA line to ground and thereby initiate the start of a data exchange (START). The master then lays a read or write bit followed by the slave address on the data line. After 8 bits the slave acknowledges the data with a low on the data line (ACK) otherwise the transmission stops. The transmission is followed by one or more data bytes which again, need to be acknowledged after each byte. A STOP signal (pulling SDA to high when SCL is high) terminates the transmission.

2.4 Tasks

There are many tasks in laboratories which can be automated. Some automation problems are more complex than others and need a power supply on site, some can be solved through a low energy wireless transmission system. In the following section a selection of purposes for low power wireless transmission systems will be discussed and how physics make these sensors possible. In section 2.4.4 more low-power tasks but also applications suited more for wired systems are listed - a detailed look on these would go beyond the scope of this theses.

2.4.1 Temperature Measurement

One of the most important measurements in laboratories is to measure the temperature. Many physical effects show strong temperature dependence so it is recommended to keep track of the temperature, measure its influence and keep stable conditions.

MEMS stands for **MicroElectroMechanical Systems** and describes microscopic devices. These devices combine micromechanical structures and logic elements to form microscopic structures measuring various physical values. Due to its size MEMS require tiny amounts of current and are very interesting for low power applications. MEMS are produced by applying thin films of substrate (1 to 100 μm) onto a surface, using lithography to alter the substrates properties and etching excess substrate from the surface [35][36][37].

MEMS temperature sensors consist of a meander shaped structure as shown in figure 6 [35]. The resistance of the metal is calculated as

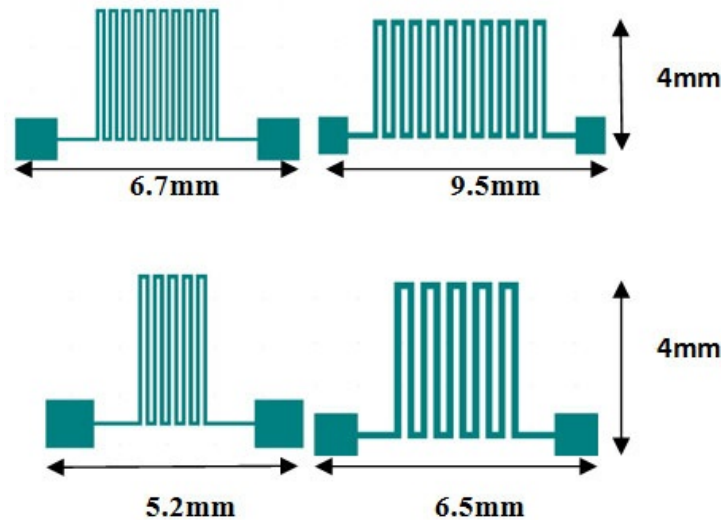


Figure 6: MEMS temperature sensor [35]

$$R = \rho L / A .$$

If the relationship between measured resistance and temperature is linear, equation

$$R_t = R_0 [1 + \alpha(t - t_0)]$$

describes the correlation.

Where,

R_t is the resistance at t °C

R_0 is the resistance at 0 °C

α is the temperature coefficient of resistance

t is the temperature to be measured

t_0 is the initial temperature

By using resistance measurements, the temperature can be calculated.

2.4.2 Air Pressure Measurement

In many laboratories a pressure difference between the external environment and the internal work place needs to be kept. The aim is for example to reduce dust from getting into a clean room (positive pressure) or prevent germs from leaving the work place (negative pressure).

Cheap low energy pressure sensors like the BMP180 from Bosch use the piezoresistive effect as its physical principle. The piezoresistive effect describes the change of resistivity of a material (conducting, semi-conducting) when exposed to a tension. It can be described as the inter-atomic spacing (due to strain) affects the bandgaps and alters the difficulty for electrons to be raised into the conduction band, which changes the material resistance [38][39][40].

MEMS piezoresistive pressure sensors consist of four meander shaped structures placed on silicon. For measurement purposes they are arranged to form a Wheatstone bridge as seen in figure 7.b) [41].

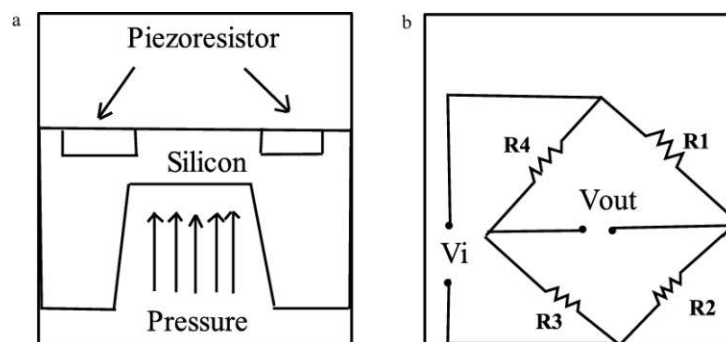


Figure 7: MEMS piezoresistive pressure sensor (a) Schematic cross section; (b) Wheatstone bridge [41]

2.4.3 ADC - Analog Digital Converter

An ADC is a system that converts an analog signal into a digital signal. Nowadays almost every microcontroller has a built-in ADC which converts an analog input voltage

to a digital number representing the magnitude of the voltage. This system is useful for measuring the output voltage of many analog sensors and trigger an event if a certain threshold is exceeded. Examples for typical ADC applications include analog temperature sensors, light sensors, pressure sensors, sound sensors and accelerometers.

A disadvantage of built-in ADCs in microcontrollers are the low resolution they offer. Most ADCs only have 8 or 10-bit resolution, which means that they are able to identify $2^8=256$ or $2^{10}=1024$ individual analog signals. In order to resolve this problem, there are many different ADCs on the market using a SPI or I2C interface with resolutions up to 24-bit ($\hat{=}$ 16 777 216 different levels).

The minimum voltage difference an ADC can discriminate (voltage resolution) can be calculated by dividing the voltage measurement range by the number of intervals (M..Resolution)

$$Q = \frac{\Delta V}{2^M} .$$

For 8-bit resolution and 3.3V voltage measurement range, the equation returns a voltage resolution of 12.9 mV.

2.4.4 Other Tasks

There are countless different sensors on the market measuring physical (temperature, humidity, pressure, heat, sound pressure, brightness, acceleration) and chemical (pH, ionic strength, electrochemical potential) properties [42]. Some examples are thermocouple, manometer, acceleration sensor, rain sensor, strain gauge, humidity sensor, force sensor, position sensor, CCD-sensor, filling level sensor, flow meter, hall sensor, gas sensor, particle sensor, motion sensor and smoke detectors all of which can be in automated to a certain degree.

There are two possibilities how sensors can provide data for further computation - analog or digital. In the field of automation systems digital sensors steadily displace analog sensors. Most sensors have an integrated ADC to digitize analog measurement data. If not included, the built-in ADC of a microcontroller or ADC-ICs with integrated data busses can be used. For digital sensors an I2C or SPI interface is sufficient to derive measurement data. If a sensor uses a bus protocol the microcontroller does not provide (for example CAN), compatible bus controller ICs can be used (e.g., MCP2515 by Microchip).

Apart from that many actuators (electric motors, servos, step motors, hydraulic cylinder, valves) can be automated too. A disadvantage of actuators in terms of wireless systems is the typically high power consumption. Wired automation systems would be a better alternative, especially considering parasitic radiation emanating from electro motors.

Finally, one of the most important features an automation system should fulfill is its capability to fetch and store measurement data. A data logger which can be easily accessed within a private network and its ability to alert the user if preset values are exceeded should be the root of every automation system.

2.5 Conclusion

After discussing the pro and cons of the different automation systems, it is time to check the transceiver modules that are available on the market. **Important note:** Individual transceiver chips are not taken into account, only fully operational boards that can be easily connected to a microcontroller. Designing and manufacturing individual boards would go beyond the scope of this thesis and would also make possible replicas very work-intensive. Therefore, the product range of three different suppliers were checked - RS Components, Mouser Electronics and Amazon (05.12.2021).

2.5.1 Wireless Transceiver - RS Components

Table 4 shows the transceiver that were listed on RS Components on given date. There

Product	Standard	Availability (Qty)	Price
MRF24J40MA-I/RM	Zigbee	no, (exp. 2023)	14,02 €
XBee 802.15.4 RF-Module	Zigbee	yes (12)	52,58 €
QFM-TRX1-24G	Wlan	yes (1)	7,02 €
nRF52833 DK	Bluetooth, Thread, Zigbee	yes (28)	67,20 €
nRF51-DK	Bluetooth	yes (4)	57,49 €

Table 4: Wireless Transceiver - RS Components

are four different sensors available, three of them cost over 50 € which is well above the limit for a cheap wireless transceiver. Nevertheless, the nRF52833 DK and to a certain extent the nRF51-DK are very interesting products offering a development board with debugging features for developing prototypes using the Bluetooth, Thread and Zigbee standard.

However, for this project only the transceiver QFM-TRX1-24G remains a possibility with the drawback that the transceiver uses the WLAN standard which does not offer the energy efficiency as similar products. Another downside is that only one piece is in stock and it is unknown when it will be increased (2021 - corona pandemic - supply shortage).

2.5.2 Wireless Transceiver - Mouser Electronics

Table 5 shows the transceiver that were listed on Mouser Electronics on given date. Mouser Electronics has a very sparse product range offering only one product. The

Product	Standard	Availability (Qty)	Price/piece
MRF24J40MAT-I/RM	Zigbee	yes (1)	14,35 €

Table 5: Wireless Transceiver - Mouser Electronics

MRF24J40MAT-I/RM which is not available at RS Components is in stock, but only one piece making the transceiver not a suitable candidate for this project.

2.5.3 Wireless Transceiver - Amazon

Table 6 shows the transceiver that were listed on Amazon on given date. In comparison to

Product	Standard	Availability	Price/piece
ESP8266	WLAN	yes	3,28 €
HC-05	Bluetooth	yes	5,79 €
RFM69HW 433Mhz	no, 433/868/915 MHz	yes	4,52 €
RFM12B HopeRF		yes	1,51 €
nRF24L01+		yes	1,51 €

Table 6: Wireless Transceiver - Amazon

the two above discussed companies Amazon offers a wide variety of different transceivers. Most of them are sold in sets of 3 to 10 for a very low price.

2.5.4 Discussion

To select the best fitted transceiver to be used in this thesis, the remaining transceivers will be compared in table 7 whether they meet desired criteria listed in section 1.3.

Product	ESP8266	HC-05	RFM69HW	nRF24L01+
Price	3,28 €	5,79 €	4,52 €	1,51 €
Standard	WLAN	Bluetooth	no, 433/868/915 MHz	no, 2,4 GHz
Availability	✓	✓	✓	✓
Extensibility	✓	✓	✓	✓
Security	✓	✓	✓	✓
Interference resistance	✓	✓	✓	✓
Transmission reliability	✓	✓	✓	✓
Reliability	✓	✓	✓	✓
Energy efficiency	~	+	+	+

Table 7: Comparison available transceivers regarding requirements

The comparison shows that although the four presented transceivers are based on different communication standards, each of them fulfills the requirements for this project and can be used without hesitation. Due to its very low price and good documentation the decision was made to use the nRF24L01+ in the course of the project.

3 System Setup

3.1 System Structure

The basic system structure for this project is shown in figure 8. A transceiver module

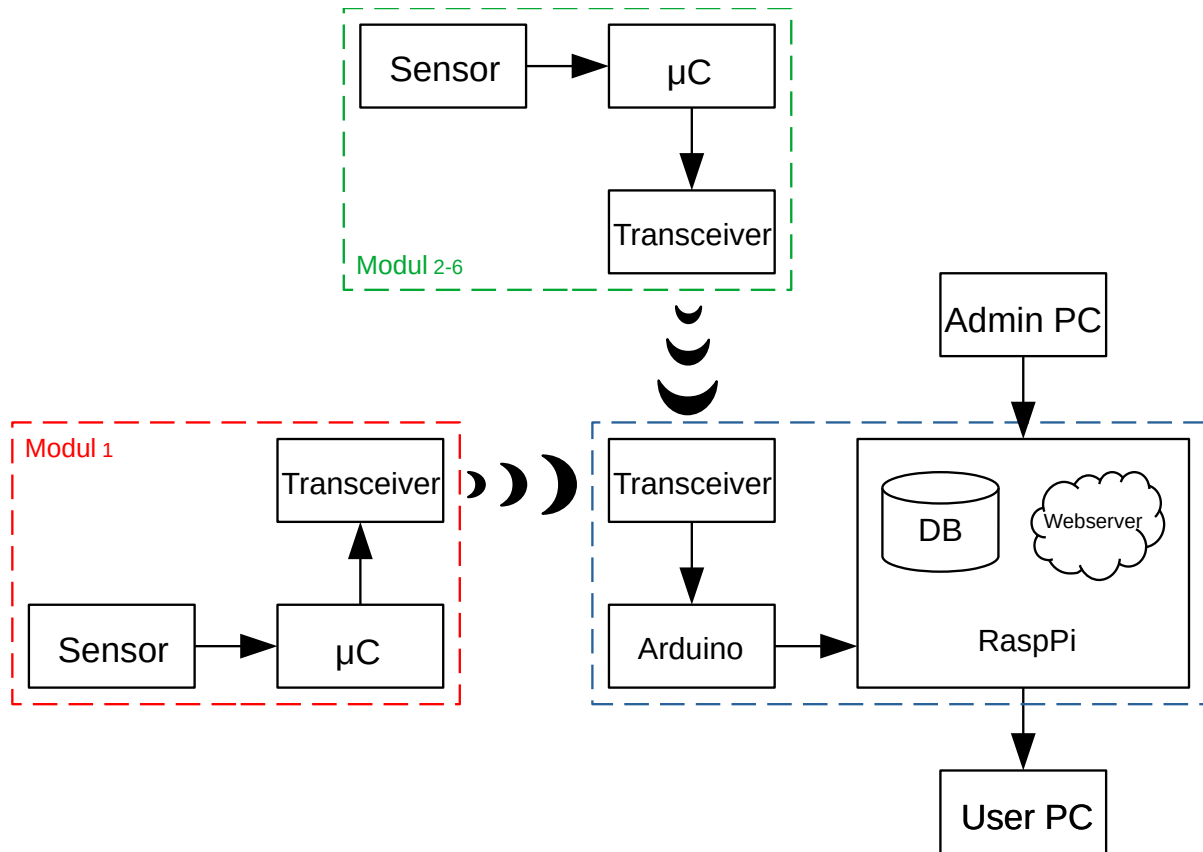


Figure 8: Block diagram of the system structure

consists of three different components (red dashed line). The transceiver, a μC (microcontroller) and a sensor. Up to six different modules (illustrated by the green dashed line block - Module 2-6) are able to transmit their data to the receiving transceiver which forwards the data to an Arduino which further forwards the data to a RaspberryPi where the data is stored in a database (blue dashed line).

Let's start by discussing the elements inside the red dashed line rectangle. The microcontroller which is used in this project is the ATmega8L which will be discussed in more detail in section 3.2.5. Every given time interval the microcontroller wakes up from its sleep state, reads data from the sensor, forwards it to the transceiver where it is transmitted to the receiving unit and then the microcontroller gets right back to the sleep state. This is being repeated every few seconds up to days customizable by the user. The sleep state is being used for energy saving purposes, to increase battery life. A calculation on the power consumption will be made in section 4.1.2.6.

For the sensor block almost every sensor using common bus protocols can be used. In section 4.2.1 and 4.2.2 two different types of sensors will be discussed. The first one is the

BMP180 a temperature and pressure sensor using the I2C/TWI bus for communication, for the second example the built-in ADC of the ATmega8L is being used to convert the analog output of a possible analog sensor, simulated by a potentiometer.

For the transceiver block the nRF24L01+ (refer section 2.3.2.6 and 3.2.4) will be used. The nRF24L01+ will be in power down mode most of the time, only when it receives a wake-up signal from the microcontroller, data will be sent and after that the nRF24L01+ will go back into the power down mode.

Inside the blue dashed line rectangle are three different elements. For the transceiver block the nRF24L01+ will be used again but in contrast to the transmitter the receiver will be constantly awake expecting incoming data. The Arduino controls the transceiver and forwards incoming data via UART to the RaspberryPi (RPi). UART (refer section 2.3.3.2) is a common interface on PCs and microcontrollers and is implemented by a USB interface connecting the Arduino to the RPi. The intermediate step to use an Arduino for data communication with the transceiver is not necessary but convenient. The nRF24L01+ could be directly connected to the RPi but to do this the casing of the RPi (which is optional but almost always used) has to be opened. The decision was therefore made to use and describe the Arduino as a GPIO (**G**eneral **P**urpose **I**nput/**O**utput) expander for the RPi.

The RPi serves two purposes. First one is to store incoming data into a database, second one is to grant every user in the same network access to the data. Therefore a webserver runs on the RPi serving requested data and diagrams of sensor data history to users (marked as the User PC block in figure 8).

3.2 Module Description

After introduction of the basic system structure and the purpose of the different blocks, this section describes the individual modules in more detail and the basic setup processes for programming. Every block shown in figure 8 will be discussed except the Sensor block. For that section 4.2.1 and 4.2.2 provide an introduction.

3.2.1 Raspberry Pi

Raspberry Pi is a small single board computer developed by the Raspberry Pi Foundation in 2012. It is widely used for many different purposes including Internet of Things based applications, robotics, image/video processing, weather station, web server, NAS (Network-attached storage) and game emulator. Due to its low price tag starting at 35 € and low power consumption (under 5 W) RPi gained popularity very fast. The internet is filled with countless different tutorials making the Raspberry Pi a perfect choice for this thesis.

Over the years many different series/generations have been released. The most recent one (09.08.2022) is the Raspberry Pi 4 Model B (2019) offering a 1.5 GHz 64-bit quad core ARM Cortex-A72 processor, onboard Wi-Fi, Bluetooth 5, gigabit Ethernet, two USB 2.0 ports, two USB 3.0 ports, 1–8 GB of RAM, and dual-monitor support via a pair of micro-HDMI (HDMI Type D) ports for up to 4K resolution (see figure 9 [43]). However,



Figure 9: Picture of Raspberry Pi 4 Model B [43]

it is insignificant what type of RPi is being used for this project due to its good downward compatibility and the limited features that are used (Webserver, Database, UART). A RPi of the first generation (1 Model B) was used in the course of this project which was absolutely sufficient, only loading times or update procedures took longer due to its single core 700 MHz processor and 512 MB RAM.

3.2.1.1 System Setup

In the following the initial steps for commissioning the RPi are illustrated. **Notice:** Not every step is described in detail but in general sufficiently detailed.

Step 1 - Setup and start Raspberry Pi

The main storage of a RPi is a SD card - newer versions use microSD. On it the operating system is among other things installed. The RPi Imager is an easy way to install an OS (**O**perating **S**ystem) on a SD card. The software can be downloaded from <https://www.raspberrypi.com/software/>. For the installation process a SD card reader is required connected to a PC. By following the instructions, the OS is written on the SD card, after that the SD card can be inserted into the matching slot on the RPi. After connecting mouse, keyboard and a monitor the RPi can be started by connecting a power supply.

Step 2 - First boot

After the first boot and first settings, which can take a while, are completed it is recommended to update and upgrade the OS. For this, following commands have to be entered in the terminal.

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Furthermore, the SSH access has to be activated. SSH stands for **S**ecure **S**hell and allows system administrators to securely access a computer over an unsecured network. By default, SSH is deactivated for obvious security reasons (default user, default password allows for unauthorized access). To activate SSH access following command has to be entered in the terminal.

```
$ sudo raspi-config
```

Following the menu to "Interfacing Options / SSH" and confirm the options activates SSH. Every authorized PC can now establish a SSH connection to the RPi (important for Admin PC).

Step 3 - Install web server and database

Next step is to install a web server, for this tutorial an Apache web server will be installed.

```
$ sudo apt install apache2
```

To check if the web server is running enter the IP address of the RPi into a web browser. If the Apache2 Debian Default Page pops up everything works fine.

Next up (if not already installed) PHP, SQLite3 and a PHP-extension for SQLite3 has to be installed.

```
$ sudo apt-get install php
$ sudo apt-get install sqlite3
```

```
$ sudo apt-get install php-sqlite3
```

PHP is a recursive acronym and stands for **PHP: Hypertext Preprocessor**. It is used to create dynamic websites or web applications. **SQLite** (**Structured Query Language Lite**) is the most popular database engine of the world. It is used for managing data held in a relational database management system. PHP is in combination with **SQLite** an essential part for displaying stored data in a database on a website.

3.2.2 Admin PC

The Admin PC is a very important part in the system. It is a crucial tool for coding and setting up the RPi. The operating system is not that important but many problems (ATmega8L programming) are easier to solve on Windows machines. The instructions which are presented in the following sections assume that a Windows machine is used but technically experienced users can easily adapt to different operating systems. An internet connection is recommended for downloading required software. For normal operation e.g., to retrieve data from the data logger (RPi) a closed home network is sufficient. This is also valid for User PCs and every internet-capable device accessing the web interface.

3.2.2.1 System Setup

There is various software that needs to be installed on the Admin PC. For communication with the RPi it is recommended to install PuTTY and WinSCP.

PuTTY

PuTTY is an open source SSH and telnet client written and maintained by Simon Tatham. It can be downloaded from <https://www.putty.org/>. Putty is used to establish a SSH connection to the RPi by connecting to its IP address. Figure 10 shows the PuTTY configuration window to connect to a RPi with an IP address of 10.0.0.2. By connecting to the RPi via SSH it is possible to control the RPi via terminal commands. This is useful for many things like debugging, updating, change permissions, installing additional libraries and creating new databases. It is recommended to acquire basic knowledge in Linux commands.

WinSCP

WinSCP (**W**indows **S**ecure **C**opy) is an open source **SSH File Transfer Protocol** (SFTP) and FTP client for Microsoft Windows. It can be downloaded from <https://winscp.net/eng/index.php>. WinSCP is used for quick and easy file copying between a local and a remote computer. In this case the example code provided in this thesis can be modified and transferred to the RPi via WinSCP. Figure 11 shows the WinSCP login window to connect to a RPi with an IP address of “10.0.0.2” and the username “pi”.

For programming the Arduino and the microcontroller (ATmega8L) Arduino IDE, Microchip studio and TL866II Plus Programmer Application Software needs to be installed.

Arduino IDE

The Arduino IDE **I**ntegrated **D**evelopment **E**nvironment is an open-source software that is used to write and upload code to Arduino (section 3.2.3) boards. It can be downloaded from <https://www.arduino.cc/en/software> where you can find a “Getting Started” page too. The Arduino is programmed in C/C++. A code example will be provided in section 4.1.3. For uploading code to the Arduino, the Arduino has to be connected to the PC and in the Arduino IDE under the menu item *Tools* the correct Board, Processor and Port has to be selected.

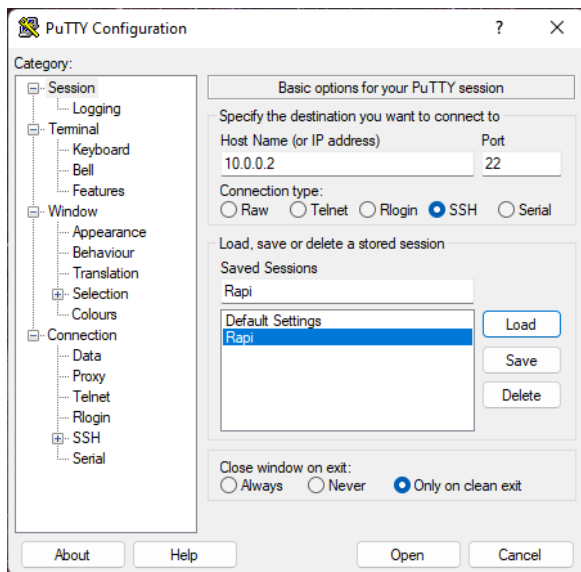


Figure 10: PuTTY configuration window

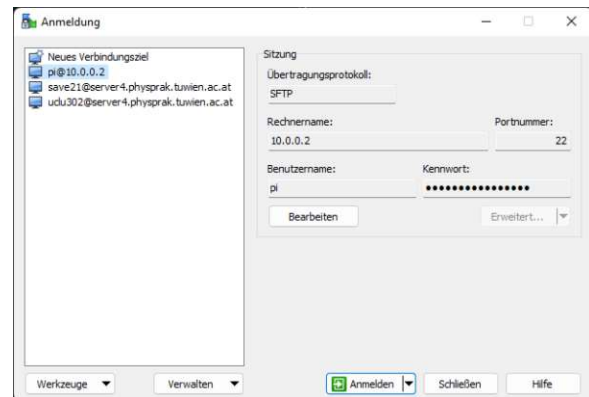


Figure 11: WinSCP login window

Microchip Studio

Microchip Studio is an IDE for developing and debugging AVR microcontroller applications. It was formerly known as AVR Studio or Atmel Studio. It can be downloaded from <https://www.microchip.com/en-us/tools-resources/develop/microchip-studio#Downloads>. Microchip Studio is used to develop, debug and compile C/C++ or Assembly code written for AVR. AVR is a family of Atmel/Microchip microcontrollers including the ATmega series and therefore also the ATmega8L (section 3.2.5).

After starting Microchip Studio, a *New Project* can be created by selecting a suitable programming language (C/C++), select *GCC C Executable Project*, naming and choosing a save location, “click” *OK* and choose the correct device (*ATmega8*).

TL866II Plus Programmer Application Software

The TL866II Plus Programmer Application Software is a software for the Arceli TL866II Plus (section 3.2.6). It can be downloaded from <http://www.xgecu.com/en/download.html>. The software is used to upload the compiled code to the microcontroller. The uploading procedure and necessary software settings will be discussed in section 3.2.6.

3.2.3 Arduino

Arduino is an open-source hardware and software company which offers microcontroller kits for building digital devices. Arduino boards use a variety of microprocessors (ATmega328P, ATmega2560, ARM Cortex) and can be easily programmed via the Universal Serial Bus (USB) interface. An Arduino board can be connected to a breadboard with the help of jumper cables. The simplicity of Arduino boards lowers the initial hurdle for inexperienced users to undertake their own micro controller projects and contributes to its popularity.

Typical model names are UNO, DUE, MEGA and can be bought from the official Arduino website. Due to its open-source platform electrical wiring diagram and layouts are freely available and can easily be used and modified. Therefore, countless replicas are on the market offering the same product for only a fraction.

In this project an Arduino MEGA 2560 was used but it can be easily replaced with an Arduino UNO. Figure 12 shows the Arduino MEGA Breadboard with all its accessible pins [44]. The UNO model offers similar features in comparison to the MEGA only the pin

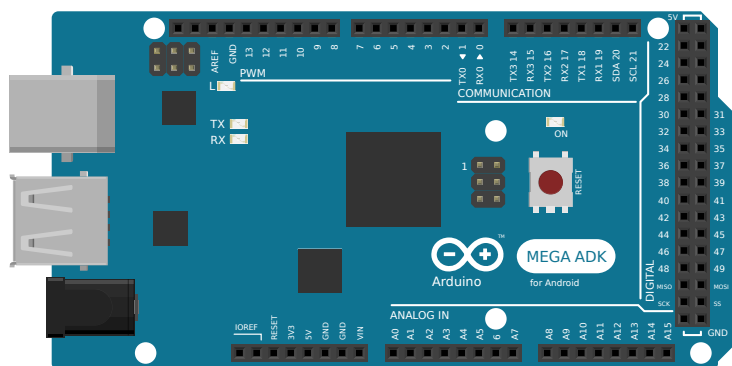


Figure 12: Arduino MEGA Breadboard Pins [44]

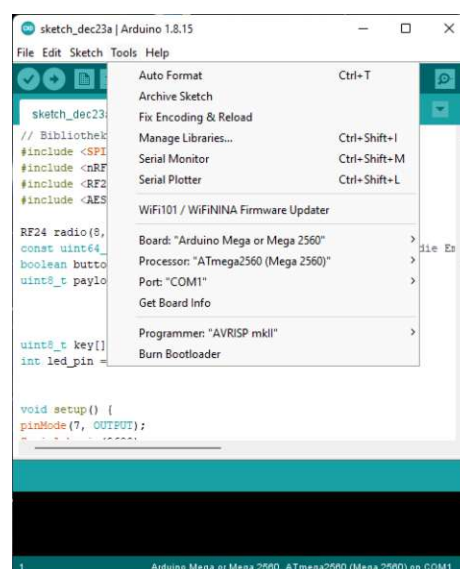


Figure 13: Arduino IDE Settings

count is reduced. Tutorials on how to connect the nRF24L01+ to the UNO are available on the internet, basically only the pins used for communication vary.

Figure 13 shows the Arduino IDE board, processor, port and programmer settings for an Arduino MEGA 2560. The port number may differ but can be easily checked by opening the *Windows Device Manager* and looking for Arduino in *Ports (COM & LPT)*.

3.2.4 Transceiver - nRF24L01+

An introduction of the nRF24L01+ and its key features had been already given in section 2.3.2.6. In this section the features will be described in more detail.

The small board dimensions of 33.1089 mm x 15.0622 mm allow the construction of small integrated devices. The nRF24L01+ application board is shown in figure 14.

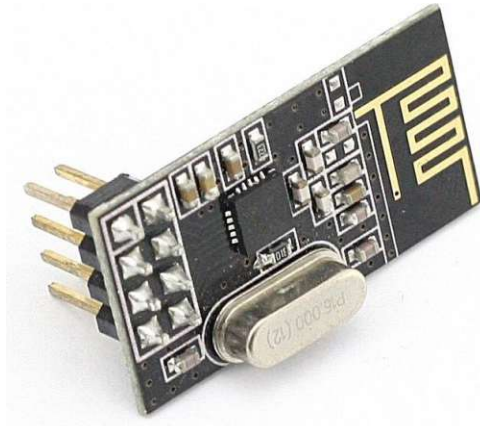


Figure 14: Picture of Transceiver nRF24L01+

To configure and operate the nRF24L01+ the Serial Peripheral Interface (SPI) is used. The register map contains all configuration registers in the nRF24L01+ and can be accessed via SPI. The data and control interface of the nRF24L01+ consists of the following six 5 V tolerant digital signals according to the datasheet [27]:

1. IRQ: this signal is active low and controlled by three maskable interrupt sources
2. CE: this signal is active high and used to activate the chip in RX or TX mode
3. CSN: **C**hip **S** **N**ot - SPI signal
4. SCK: **S**erial **C**lock - SPI signal
5. MOSI: **M**aster **O**utput, **S**lave **I**nput - SPI signal
6. MISO: **M**aster **I**nput, **S**lave **O**utput - SPI signal

Including the two supply pins VDD (1.9 to 3.6 V - typical 3.0 V) and VSS (0 V) there are in total 8 pins.

For data transmission between two nRF24L01+ there are a few important settings which have to be configured beforehand. These settings are listed in the following lines.

1. Auto Retransmit Count: 0-15 (after x retries data transmission fails)
2. Pipe Address: Standard Transmit Address equal to Receive Address Data Pipe 0: 0xE7E7E7E7E7
3. RF (**R**adio **F**requency) Channel Frequency: 2.400 GHz to 2.525 GHz in steps of 1 MHz → 125 Channels in total

Austria: allowed frequencies: 2.400 GHz to 2.4835 GHz (Frequenznutzungsverordnung §9 Absatz 1 Anlage 2)

4. Payload Size: 1 to 32 bytes

5. RF Data Rate:

250 Kbps

1 Mbps

2 Mbps

6. CRC Length: 1 to 2 bytes

7. RF Output Power:

-18 dBm $\hat{=}$ 15.8 μ W

-12 dBm $\hat{=}$ 63.1 μ W)

-6 dBm $\hat{=}$ 251 μ W

0 dBm $\hat{=}$ 1 mW

For standard operation a data rate setting of 250 Kbps is recommended, offering the highest transmission range.

CRC stands for **C**yclic **R**edundancy **C**heck and describes an error-detecting code which is used to detect accidental changes to digital data (e.g., noise). It works as follows: a short (1 to 2 bytes) check value is attached to the payload (calculated via a polynomial) and on the receiver side the same polynomial is applied to the data. The calculation yields zero if there are no detectable errors. It is possible that two or more errors lead to the same result but it is very unlikely. CRCs can be easily realized in hardware and therefore are preferably used for many different applications.

The RF Output Power (transmission power) is measured in dBm. dBm (**d**ecibel-**m**illiwatts) is used to indicate a power level in decibels with reference to one milliwatt. The mathematical relationship between x in dBm and the corresponding power P in mW is given as follows:

$$x = 10 \log_{10} \frac{P}{1mW} .$$

Vice versa:

$$P = 1mW * 10^{\frac{x}{10}} .$$

3.2.4.1 Radio Control

Figure 15 shows the state diagram of the nRF24L01+ [27]. The state diagram shows the operating modes and how they function. When connecting a voltage above 1.9 V (And lower than 3.6 V) to the VDD pin the nRF24L01+ enters the *Power on reset* state, where it stays for 100 ms to stabilize signals and afterwards enters the *Power Down* state. In *Power Down* state the nRF24L01+ draws a current of 900 nA which is perfect for



Figure 15: nRF24L01+ state diagram [27]

battery powered devices. By setting the PWR_UP bit in the CONFIG register to logic 1, the nRF24L01+ enters after 1.5 ms the *Standby-I* state, drawing a current of 26 μ A.

Proceeding from the *Standby-I* state two different paths can be taken. Differentiated in using the nRF24L01+ as a transmitter or a receiver.

3.2.4.2 Receiver path: By setting the PRIM_RX bit in the CONFIG register to logic 1 and draw the CE input pin to logic 1 the nRF24L01+ enters after stabilizing signals for 130 μ s (*RX Settling* state) the *RX Mode* state. In RX mode the nRF24L01+ demodulates signals from the RF channel, checks for errors (CRC) and presents the payload of the packet in a vacant slot in the RX FIFOs. The RX FIFO (**F**irst **I**n – **F**irst **O**ut) is a temporary storage which holds the received bits until they are read by the microcontroller.

The nRF24L01+ can leave the *RX Mode* state by drawing the **CE** pin to logic 0. It is also possible to remain in the *RX Mode* state and simply forward all incoming data to the RX FIFO. In our case the nRF24L01+ connected to the Arduino stays in the *RX Mode* state and forwards its data to the Arduino and finally to the RPi. More on that in section 4.1.3.

3.2.4.3 Transmitter path: By loading data into the TX FIFO (TX FIFO not empty), setting the **PRIM_RX** bit in the **CONFIG** register to logic 0 and drawing the **CE** input pin to logic 1 for more than 10 μs the nRF24L01+ enters after stabilizing signals for 130 μs (*TX Settling* state) the *TX Mode* state. The nRF24L01+ stays in *TX Mode* until it finishes transmitting a packet. If the TX FIFO is not empty the nRF24L01+ remains in *TX mode* and transmits the next packet. If the TX FIFO is empty and **CE=0** the nRF24L01+ enters the *Standby-I* state from where it can change to the power saving state *Power Down* by setting the **PWR_UP** bit in the **CONFIG** register to logic 0 or it is possible to initiate the next data transmission.

Detailed flow charts and code snippets will be given in section 4.1.3 for the receiver and section 4.1.1 for the transmitter.

3.2.5 μ C ATmega8L

The ATmega8L is a low-power 8-bit microcontroller belonging to the AVR microcontroller family developed in 1996 by Atmel, acquired by Microchip Technology in 2016. It is based on the RISC (**R**educed **I**nstruction **S**et **C**omputer) architecture which offers simpler instructions and higher clock frequency in comparison to the CISC (**C**omplex **I**nstruction **S**et **C**omputer) architecture. Due to AVR's simple architecture, easy programmability and free development software, AVR's are widely used in hobbyist and educational embedded systems (refer to section 3.2.3 - ATmega328P, ATmega2560).

In the following key features of the ATmega8L are listed from the datasheet [45].

- High-performance, Low-power Atmel®AVR® 8-bit Microcontroller
 - 130 Powerful Instructions – Most Single-clock Cycle Execution
 - 32×8 General Purpose Working Registers
 - Up to 16 MIPS Throughput at 16 MHz
- Advanced RISC Architecture
 - 8 Kbytes of In-System Self-programmable Flash program memory
 - 512 Bytes EEPROM
 - 1 Kbyte Internal SRAM
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescaler, one Compare Mode
 - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
 - Real Time Counter with Separate Oscillator
 - 6-channel ADC in PDIP package
Six Channels 10-bit Accuracy
 - Byte-oriented Two-wire Serial Interface
 - Programmable Serial USART
 - Master/Slave SPI Serial Interface
 - Programmable Watchdog Timer with Separate On-chip Oscillator
- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated RC Oscillator
 - External and Internal Interrupt Sources
 - Five Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, and Standby

- Operating Voltages / 2.7 V - 5.5 V (ATmega8L)
- Speed Grades / 0 - 8 MHz (ATmega8L)
- Power Consumption at 4 Mhz, 3 V, 25 °C
 - Active: 3.6 mA
 - Idle Mode: 1.0 mA
 - Power-down Mode: 0.5 μ A

The AVR family is classified into many different series offering variants for every purpose. The most common series are ATmega, ATtiny (slimmed down version in comparison to ATmega) and ATxmega (newest generation - enhanced version in comparison to ATmega). For this project a microcontroller of the ATmega series was chosen. The ATmega series offers great flexibility for prototyping and the fairly large program memory allows the programmer to not worry about program size.

Originally it was decided that an ATmega328P will be used as the microcontroller shown in figure 8. The ATmega328P offers low power consumption in power-save mode (0.75 μ A at 1.8 V, 25 °C) which is a quite significant power consumption save in comparison to the ATmega8L, which uses about 9 μ A at 3.0 V, 25 °C. In power-save mode the microcontroller can wake up from its sleep state every given time interval. A detailed description of this feature will be given in section 4.1.1. Unfortunately, the ATmega328P is out of stock and therefore the available ATmega8L had to be used. This is not a big issue because the power consumption is already very low and offers plenty of battery life (Refer calculation in section 4.1.2). However, future projects should be implemented with an up-to-date microcontroller such as ATmega328P.

For easy accessibility and prototyping the DIP Dual In-line Package variant was chosen. Figure 16 shows the DIP pin configuration of the ATmega8/L [46]. DIPs offer a

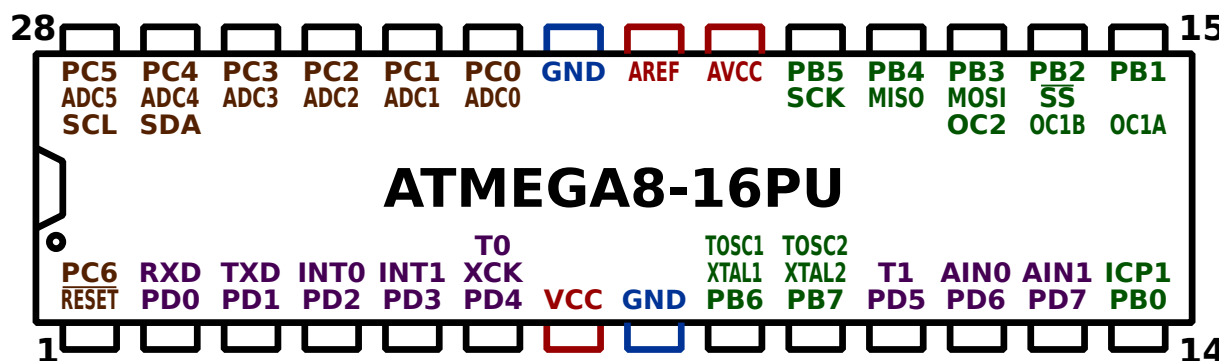


Figure 16: Atmega8L pins [46]

inter-lead spacing of 0.1 inches (2.54 mm) and rowing spacing of 0.3 inches (7.62 mm). A ZIF Zero Insertion Force socket is available for DIPs, which is shown in figure 17 and 23.

The ATmega8L code is written and compiled via Microchip Studio which was already presented in section 3.2.2. The datasheet is the main tool for coding, offering information

about register values and how to implement certain features. Section 3.2.6 describes the procedure to program the compiled machine code onto the microcontroller.

3.2.6 Programmer - Arceli TL866II Plus

The Arceli TL866II Plus is a universal programmer that can be used to program over 15000 different chips. In this project the TL866II is used to upload code to the ATmega8L. The TL866II is not an ideal choice for fast code debugging because it requires the processor to be removed from the test board for the programming procedure. There are better alternatives on the market but none of them can be used for such a wide variety of different chips. Figure 17 shows the Arceli TL866II Plus Programmer [47].



Figure 17: Arceli TL866II Plus [47]

The TL866II Plus Programmer Application Software Xgpro v9.00 main window after startup is shown in figure 18.

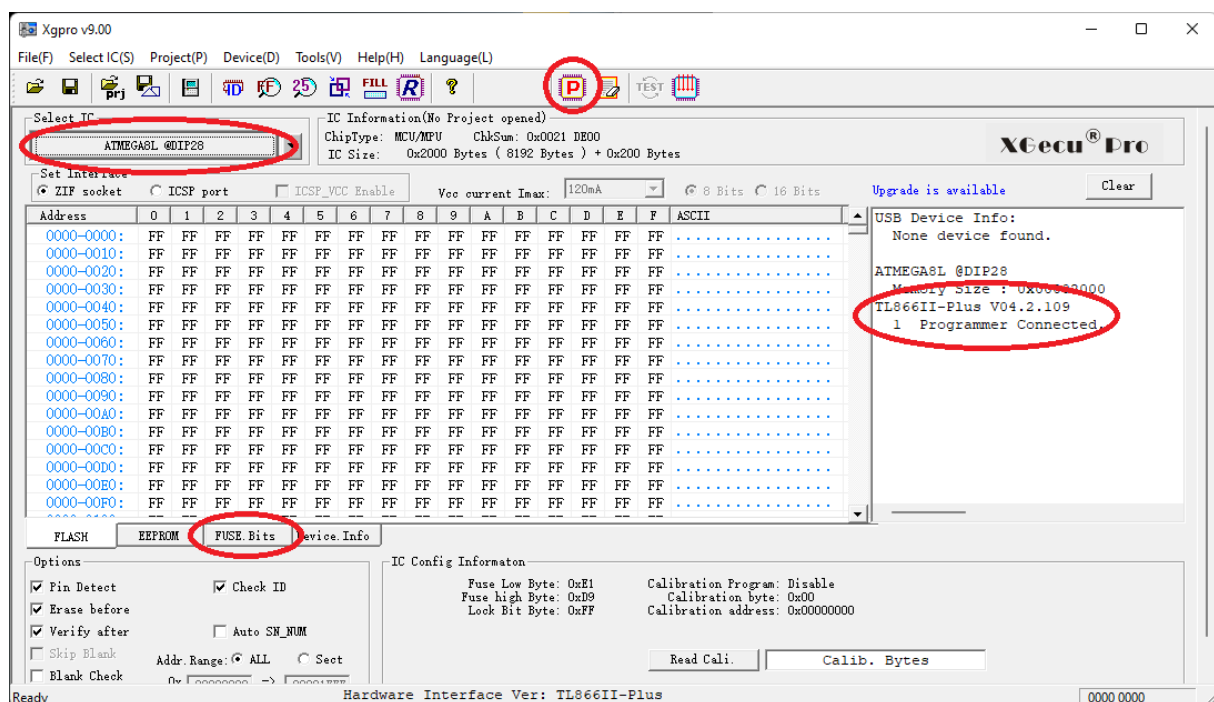


Figure 18: Xgpro v9.00 main window

The most important areas are highlighted with a red circle. On the right side of the main window, it is possible to check whether the programmer is correctly connected to the PC. The first step before programming a microcontroller is to select the correct IC (in this example the ATMEGA8L @DIP28 Dual In-line Package) via the *Select IC* button. In the second step the correct *.hex*-file has to be loaded. The *.hex*-file is provided by the Microchip compiler and can be found in the *Microchip projects* folder - subfolder Debug. After that it is recommended to check the settings of the FUSE bits which can be done by “clicking” on the *FUSE. Bits* tab. Figure 19 shows the open *FUSE. Bits* tab window.

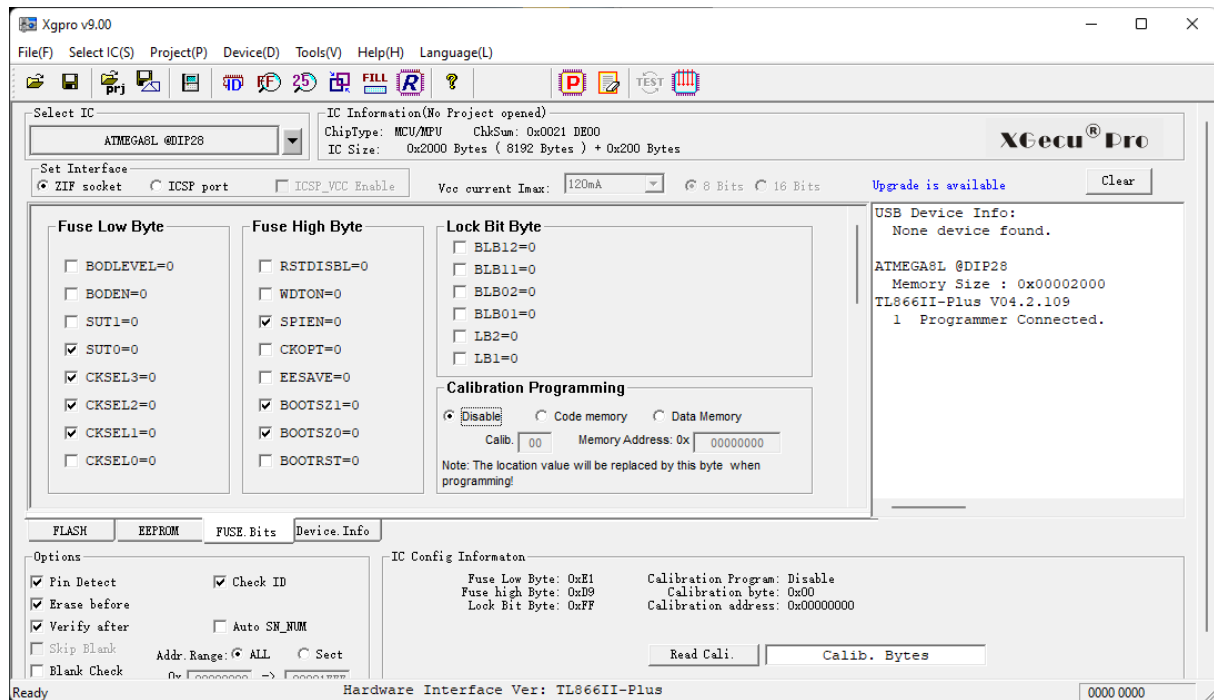


Figure 19: Xgpro v9.00 FUSE. Bits tab

Fuse bits are like little switches that can be turned on and off to enable and disable various features on AVR microcontroller. A detailed description of the features can be found in the datasheet [45]. To set up the processor speed the bits CKSEL0 to CKSEL3 and SUT0 and SUT1 are of special interest. In default settings the processor runs at 1 MHz which can be increased to 8 MHz by “checking” the CKSEL0=0 bit and “unchecking” the CKSEL2=0 bit. For different clock frequencies refer to the datasheet table shown in figure 20 [45].

To program an IC with the TL866II it is required to check if the IC is placed and secured correctly in the IC ZIF socket. A “click” on the highlighted **P** as shown in figure 18 opens a new window called *Chip Program*. Make sure to set the correct program range (Include FUSE.Bits), if it is desired to use a higher clock frequency as the default one and press *Program*.

Table 9. Internal Calibrated RC Oscillator Operating Modes

CKSEL3..0	Nominal Frequency (MHz)
0001 ⁽¹⁾	1.0
0010	2.0
0011	4.0
0100	8.0

Note: 1. The device is shipped with this option selected

Figure 20: ATmega8L CKSEL settings - frequency [45]

4 System Implementation

4.1 Basic Operation Setup

Figure 21 shows the basic system structure, used programming languages and communication flow between the different modules.

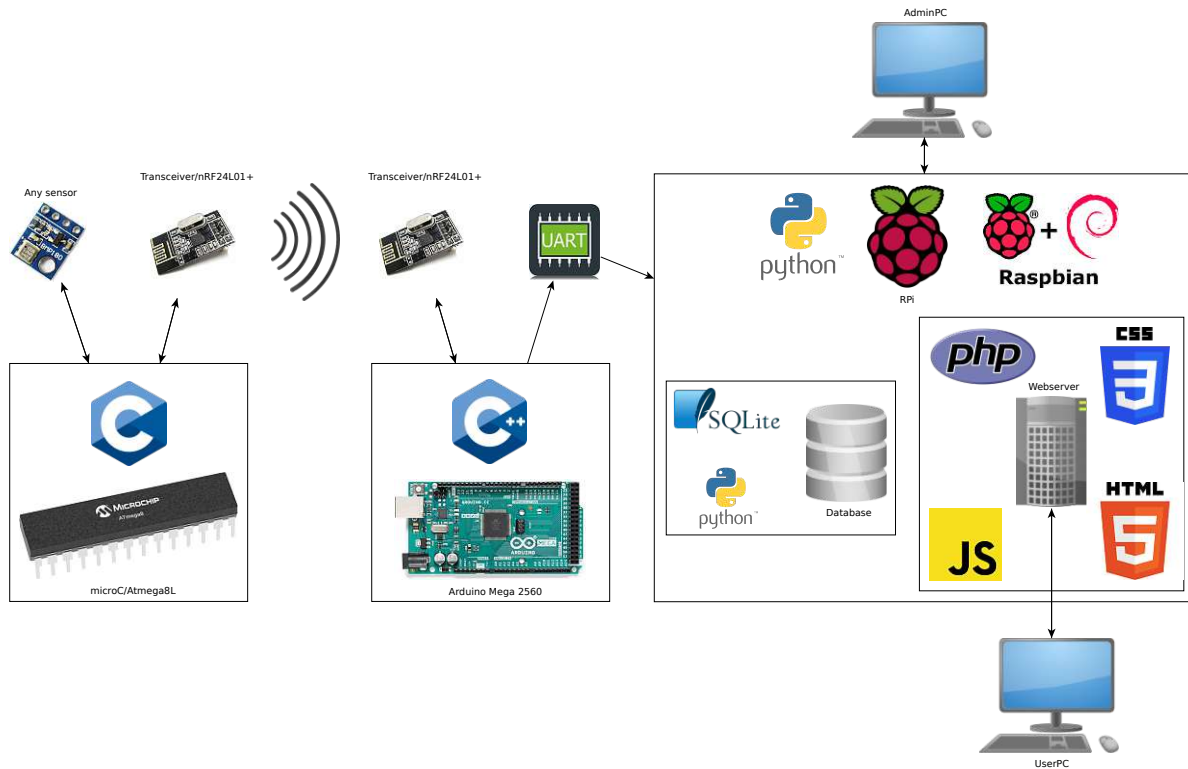


Figure 21: Basic system structure and used programming languages

Every data collection procedure starts on the left side of the figure. The microcontroller ATmega8L which is programmed in C (indicated by the C programming language logo) gathers the data every given interval a sensor (of any kind) provides. The data is then sent to a transmitter (nRF24L01+) and afterwards collected by a receiving nRF24L01+ connected to an Arduino Mega 2560. The Arduino is programmed in C++. Through an UART interface the data is then transmitted to the RPi where a Python script reads the incoming data and stores it by using the database engine SQLite local on the RPi in a beforehand created database file. Parallel to the storing Python script runs another script which periodically checks the data and sends an email to a specified address if a problem occurs. Furthermore, it sends an email daily, giving information about the current system status. The database is accessed through a webserver using a combination of PHP, CSS, JS, HTML programming languages to display stored data to any UserPC connecting to the webserver.

Every module and basic software functions will be discussed in the following sections starting with the ATmega8L microcontroller. Full commented code will be provided in the

appendix (7.2). This chapter gives an overview on the thought process how to implement such a system by providing flow charts and highlighting important code lines.

4.1.1 Microcontroller - ATmega8L

As previously in section 4.1 discussed the ATmega8L is programmed in C in Microchip Studio. Figure 22 shows the flowchart of a basic transmission with the ATmega8L. The

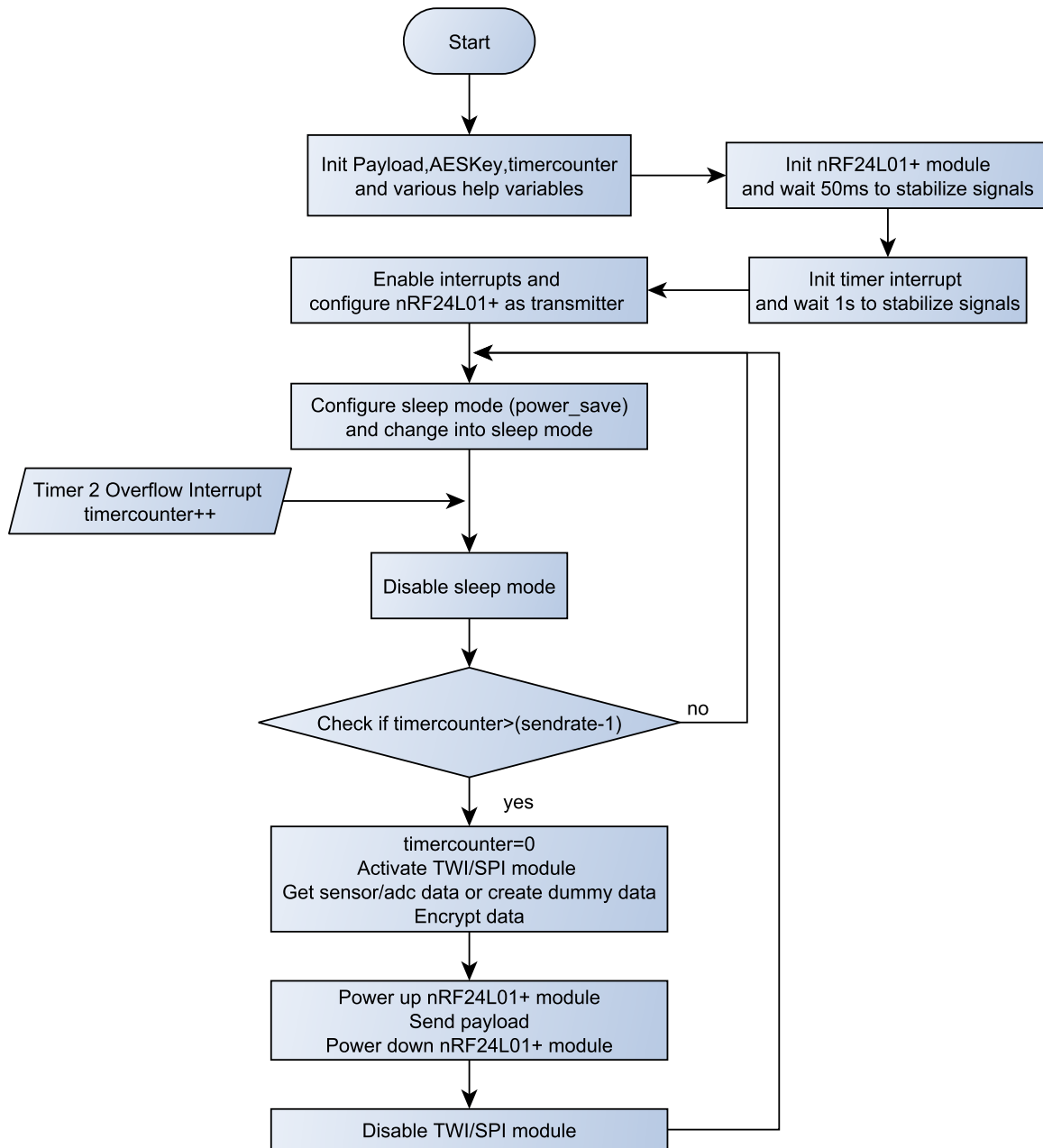


Figure 22: ATmega8L code flowchart

main code starts by defining various variables. The **Payload** variable is used to store sensor data which then will be AES-128 encrypted with the help of an **AESKey** before it is transmitted over the air with the nRF24L01+ module. In the next step the **nRF24L01+ module** will be initialized (more on that in section 4.1.1.2) followed by the initialization of the **timer interrupt** (see section 4.1.1.1). After globally **enabling interrupts** and

configuring the nRF24L01+ as transmitter the **sleep mode** will be configured and entered.

If a **Timer 2 overflow interrupt** occurs the ATmega8L will wake up from its sleep mode and **check if enough time passed** since the last transmission procedure. If that statement is true sensor data will be gathered and saved in the previously defined **Payload** variable, followed by encrypting the data with the previously defined **AESKey**.

After powering up the nRF24L01+ module, sending the data and powering down the module the ATmega8L returns the power-save mode where it remains until another **Timer 2 overflow interrupt** occurs.

Two of the most difficult challenges when writing an energy-efficient code for the ATmega8L are on the one hand to configure the microcontroller to use as little energy as possible and on the other hand to configure the nRF24L01+ to send acquired data and otherwise remain in power-down mode with minimal current draw. Sections 4.1.1.1 and 4.1.1.2 take a deeper look at these challenges. With the help of key code fragments, the required settings to achieve those tasks will be discussed.

4.1.1.1 Configure ATmega8(L) for power-save mode

A look at the ATmega8(L) datasheet - section *Power Management and Sleep Modes* shows that the ATmega8 offers five different sleep modes which can be selected by setting sleep mode corresponding bits in the MCU Control Register [45]. The sleep mode which is of most interest for this project is called *Power-save Mode* which is identical to Power-down, with one exception (see datasheet p.34) [45]: “If Timer/Counter2 is clocked asynchronously, that is, the AS2 bit in ASSR is set, Timer/Counter2 will run during sleep. The device can wake up from either Timer Overflow or Output Compare event from Timer/Counter2 if the corresponding Timer/Counter2 interrupt enable bits are set in TIMSK, and the global interrupt enable bit in SREG is set.”

That means it is possible to let the μC sleep most of the time and thereby minimizing the current draw. Every few seconds (if a Timer 2 overflow interrupt occurs) the μC wakes up from its sleep and decides if enough time has passed since the last transmission procedure. If so, new sensor data will be gathered and transmitted to the host (RPi). Otherwise, the μC will sleep for another time interval.

Lets see how that description from the datasheet translates into C code:

```
ASSR |= (1<<AS2); //enable asynchron mode – clock timer/counter2 from 1
    crystal oscillator connected to the TOSC1 pin
TCCR2 |= (1<<CS22)|(1<<CS21)|(1<<CS20); //set prescaler to 1024 2
while((ASSR & (1<<TCR2UB))); //wait for end of access 3
TIFR = (1<<TOV2); //clear interrupts (*) datasheet: "Alternatively, 4
    TOV2 is cleared by writing a logic one to the flag."
TIMSK |= (1 << TOIE2); //enable timer compare match interrupt – if 5
    overflow in timer/counter2 occurs → interrupt
TCNT2 = 0; //initialize the timer/counter2 with 0 6
```

To understand that code it is necessary to understand the fundamental concepts of bit manipulation. Individual bits can be set by using the bitwise *OR* operator (`|`) and reset by using the bitwise *AND* operator (`&`) in combination with the bitwise *NOT* operator (`~`). Bits can be flipped by using the bitwise *XOR* operator (`^`). A bit shift can be done by using the operators `<<` (left shift) and `>>` (right shift).

This knowledge applied to the above included C code shows that in the codes first line the AS2 bit from the ASSR **ASynchronous Status Register** is set. Note: The ATmega8 header file `#include <avr/io.h>` includes definitions for every pin and register bit. It is defined that AS2 equals 3 and therefore `1<<AS2` equals `1<<3` which means that the bit 0b00001000 is set which corresponds to the AS2 bit (see datasheet).

Now that AS2 is written to 1 Timer/Counter2 is clocked asynchronously from the crystal oscillator connected to the TOSC1/XTAL1/PB6 pin (refer section 4.1.2.2). The next line of the code sets all of the CS (**C**lock **S**elect) bits from the TCCR2 (**T**imer/**C**ounter **C**ontrol **R**egister 2). As a result of this the frequency provided by the crystal oscillator is divided by 1024 which reduces its design frequency from 32 768 Hz to 32 Hz. If another write is performed to any of the three Timer/Counter2 Registers while the update busy flag is set (TCR2UB from the ASSR gets set if TCCR2 is written), the value might get corrupted and an unintentional interrupt can occur. The while loop checks for the bit and continues only when the busy bit is cleared.

In the next line the Timer/Counter2 Overflow Flag (TOV2) from the TIFR (**T**imer/**C**ounter **I**nterrupt **F**lag **R**egister) is manually cleared by writing a logic one to the flag. This bit is set when an overflow occurs in Timer/Counter2. After that the Timer/Counter2 Overflow Interrupt can be enabled by setting the TOIE2 **T**imer/**C**ounter2 **O**verflow **I**nterrupt **E**nable bit of the TIMSK (**T**imer/**C**ounter **I**nterrupt **M**ask Register). The timer initialization concludes by setting the start number of the Timer/Counter Register 2. We want the μ C to sleep as long as possible therefore TCNT2 is initialized with 0. The TCNT2 is an 8-bit register offering a counting range from 0 to 255 which divides the crystal oscillator frequency once more to 32 Hz/256=1/8 Hz. To activate interrupts globally on the μ C one more line of code is necessary - `sei()`; (**S**et **E**nable **I**nterrupts). Now every 8 seconds an overflow occurs in Timer/Counter 2 which wakes the μ C (if the following settings are made inside the main loop).

```
OCR2 = 0; //dummy access 1
while((ASSR & (1<< OCR2UB))); //wait for end of access 2
3
set_sleep_mode(SLEEP_MODE_PWR_SAVE); //set sleep mode power-save 4
//sleep_enable(); //enable sleep mode 5
sleep_mode(); //change into sleep mode 6
//sleep_disable(); //first thing after waking from sleep: 7
```

The first two lines (in the beginning of the main loop) are required if it is not guaranteed that the interrupt and main loop take more time than one clock cycle of the

Timer/Counter 2 ($1/32.768 \text{ kHz} \approx 31\mu\text{s}$). Otherwise, the interrupt logic is deactivated and the μC stays in sleep mode forever. The two other lines setup the sleep mode (power-save) and switch into it. If a Timer/Counter 2 overflow interrupt occurs, the μC gets back to the main loop after processing the interrupt service routine, to the next code line after switching into sleep mode. There sensor data can be gathered, encrypted and send. Afterwards, all interfaces that are used have to be switched off (eg., SPI, TWI) before entering the power-save mode, otherwise the microcontroller draws unnecessary current the next time it wakes up.

Notice: It should not be forgotten to reactivate the modules before they are used again. Additionally, it is very important to avoid floating inputs as they increase current consumption. Therefore, it is recommended to enable the internal pull-ups on unused pins.

After processing the last lines of the main loop, it continues from the top where the μC enters power-save mode.

4.1.1.2 Configure nRF24L01+

First of all to establish a communication between the μC and the nRF24L01+ it is required to initialize the nRF24L01+ module (see flowchart figure 22 second block). During the initialization procedure it is defined which pins are used for the chip enable and chip select signals (`wl_module.h`), the external Interrupt 0 which is connected to the IRQ of the nRF24L01+ is initialized and activated (`wl_module.c`) and the spi module of the μC is initialized (`spi.c`).

The used libraries (`nRF24L01.h`, `wl_module.h`, `spi.h`) and C codes (`wl_module.c`, `spi.c`) are slight variations of the files offered by Ernst Buchmann, Stefan Engelke and Brennan Ball. Some changes had to be made in order to work properly. Most of it is related to the `power_down` and `power_up` procedures as well as the configuration of the maximum package transmission retries.

After initializing the nRF24L01+, the module has to be configured as transmitter. Calling the function `wl_module_tx_config(wl_module_TX_NR_0)`; configures the module i.a. with its default transmission address `0xE7E7E7E7E7` (see nRF24L01+ datasheet [27]). The function also does `power_up` the module and sets the correct RF channel (defined in `wl_module.h`) frequency (has to match which the receiving nRF24L01+ connected to the Arduino). Additionally, the RF output power (0dBm), a data rate of 250 Kbps and other interrupt IRQ, CRC and transmission retry related settings are configured. The corresponding register mapping table which is used for configuring the nRF24L01+ registers is located in the `nRF24L01.h` file.

A transmission procedure is initiated by calling the `wl_module_power_up()`; function which as the name suggests does `power_up` the nRF24L01+. **Important note:** It is required to draw the CE pin to logic high for the radio to enter *Standby-I* state, different to the procedure given in section 3.2.4.1 and stated in the datasheet [27]. Just setting the `PWR_UP` bit in the `CONFIG` register is not enough to `power_up`.

Calling the `wl_module_send(tmpOutput,wl_module_PAYLOAD);` function (`tmpOutput` holds the encrypted payload and `wl_module_PAYLOAD` holds the payload size) transmits the data to the receiving nRF24L01+ module. Incoming interrupts coming from the IRQ pin of the nRF24L01+ are handled by the `ISR` (interrupt service routine) bound to the `INT0_vect` interrupt 0 vector which is located in the `nRF24L01main.c` file.

The data transmission is concluded by calling the `wl_module_power_down();` function returning from the *Standby-I* state to the *Power Down* state.

For the AES encryption the functions `aes_expandKey();` and `aes_encryptWithExpandedKey();` were used. These functions are defined in the `aes.c`, `aes.h`, `aes_asm.S` files written by Berthold Van den Bergh (Karl Malbrain) and were left unchanged.

4.1.2 Prototype Board

The prototype board is used to test the code written for the ATmega8L. At first a breadboard was used to check the functionality of the software but unfortunately the reliability of a breadboard is quite bad and caused many headaches. After switching to a stripboard and connecting the electrical components through a wiring pencil and soldering, most of the problems were solved. Figure 23 shows the circuit design of the stripboard. The

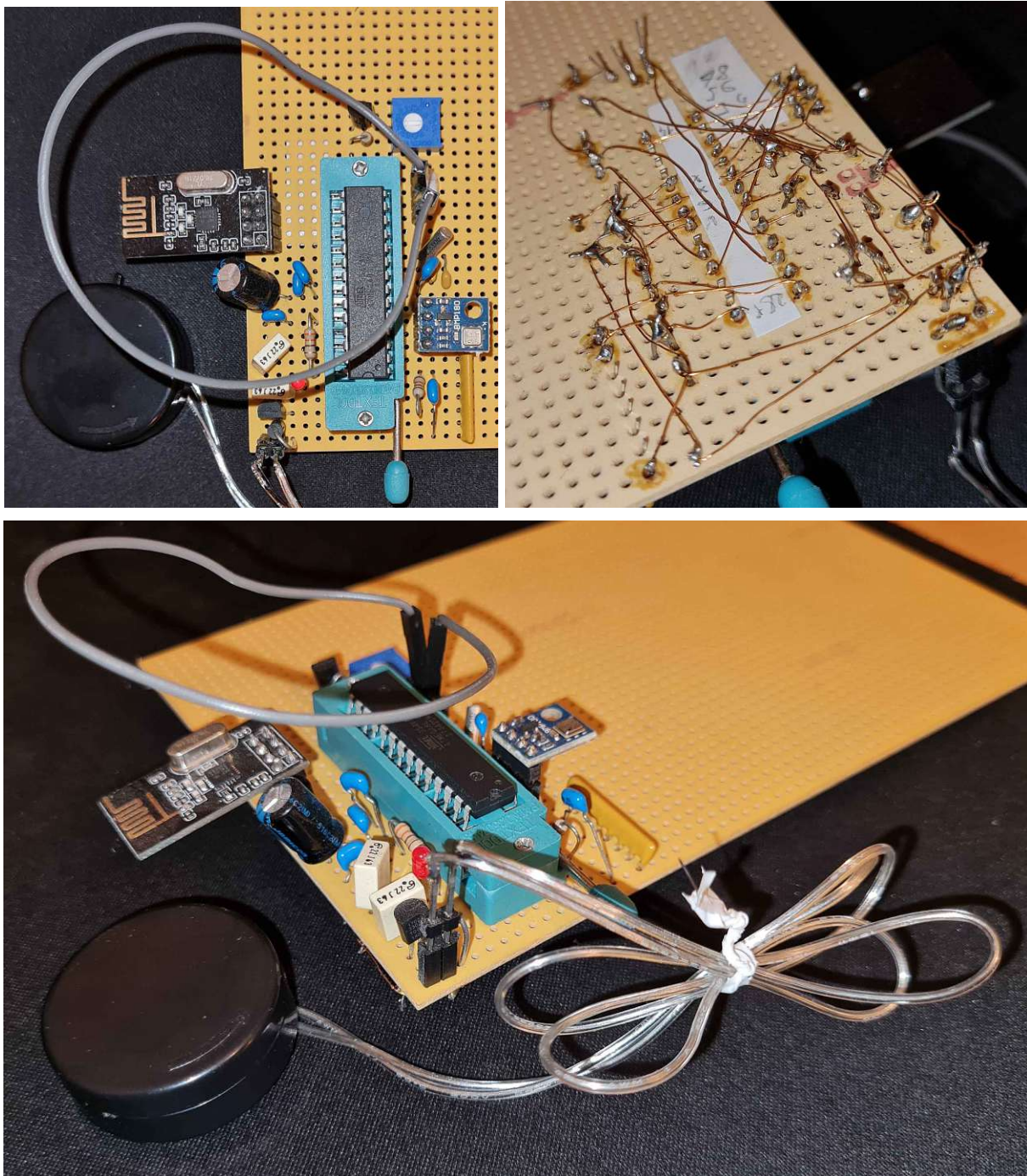


Figure 23: Prototype board

complete schematic is found in the appendix (7.1). The circuit design consists of several different function blocks which can be simplified and structured as shown in figure 24.

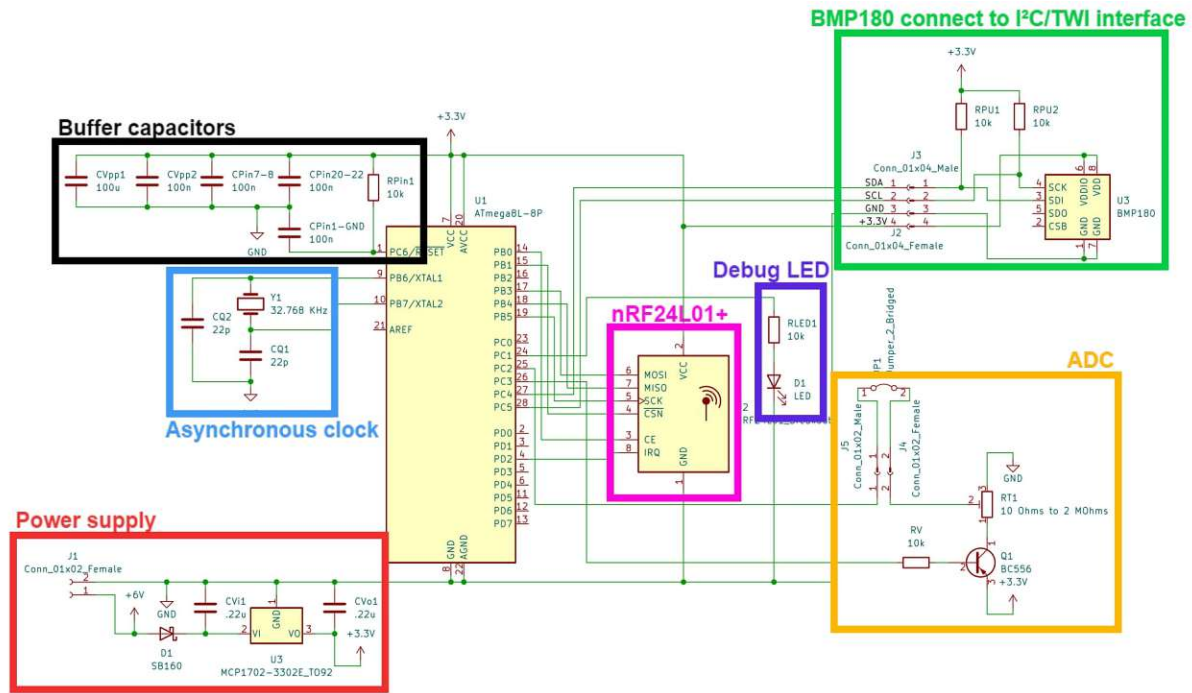


Figure 24: Prototype board - schematic

4.1.2.1 Power supply

Two 3 V button cells of the type CR2032 power the prototype board. A SB160 Schottky diode is used to offer reverse polarity protection for the subsequent circuit. According to the datasheet the SB160 has a forward voltage of 0.3 V at 25 °C junction temperature and 10 mA forward current [48]. This guarantees longer operational time in comparison to a silicon diode with a forward voltage of approximately 0.7 V.

Following the SB160 a MCP1702-3302E CMOS **Low DropOut** (LDO) voltage regulator is used to offer a constant power supply of 3.3 V. Two buffer capacitors stabilize the signals as shown in the application circuit offered in the datasheet [49]. The MCP1702 offers a low dropout voltage of under 1 V, a quiescent current (typical) of only 2 μ A and an input operating voltage range of up to 13.2 V. It is therefore possible to connect a power supply different to the 2xCR2032 as long as the voltage is under 13.2 V and above approximately 4.5 V.

4.1.2.2 Asynchronous clock

To reduce power consumption the ATmega8L remains in power-save mode (refer section 4.1.1) most of the time, where only its timer/counter 2 will run during sleep. An external watch crystal connected to the ATmega8L pins PB6/XTAL1 and PB7/XTAL2 offers the required clock pulses for operating timer/counter 2. Typical watch crystals offer a frequency of 32 768 Hz (allows for simple 2-bit calculations e.g., division by the 10-bit prescaler (1024 steps) leads to 32 Hz), small dimensions, high accuracy and low power consumption.

The watch crystal needs two additional load capacitors with a capacitance of about 12.5 pF in order to oscillate. In this regard microchip released informative applications

notes [50]. For the prototype board two 22 pF capacitors were used which performed well enough for given purpose.

To calculate the accuracy of the external oscillator it is important to know the accuracy of a watch crystal/RTC (**R**ea**T** **T**ime **C**lock). A typical RTC has an error of about 20 ppm ($2 * 10^{-5}$) which gives an error of $86400 * 2 * 10^{-5} = 1.73$ s over a day. In a month the time deviation adds up to $30 * 1.73 = 51$ seconds or about 1 minute a month which is completely fine for given purpose.

4.1.2.3 Buffer capacitors

It is recommended to place capacitors in short distance to the power supply of every chip. The capacitors supply the ICs if short peak current is drawn from the supply and the chemical reaction of the connected battery is too slow to follow the fast peak current draw. Typical values of such buffer capacitors are 100 nF and therefore they were included in the circuit design. In addition to the 100 nF capacitors a 100 μ F (x1000 capacity) electrolytic capacitor is placed in short distance to the nRF24L01+ offering enough reserves during the transmission procedure where the current draw can reach up to 20 mA.

4.1.2.4 ADC

To test the internal ADC (**A**nalog **D**igital **C**onverter) of the ATmega8L (for more details about the implementation refer to section 4.2.2) a switchable (on/off) voltage divider was implemented on the prototype board. A PNP transistor (BC556) is used to switch on the power supply for the voltage divider during every AD conversion.

Drawing the ATmega8L pin PC3 to logic “0” allows a basis current to flow through the basis resistor RV which causes the current to flow through the collector supplying the voltage divider trimmer RT1. A part of the voltage (depending on the position of the sliding contact of the trimmer (pin 1/2)) is guided to the pin PC2 (ADC2) followed by an AD conversion. The remaining voltage drops between pin 2/3. An AD conversion is concluded by drawing the pin PC3 to logic “1” to stop current from flowing through the transistor respectively trimmer and drawing unnecessary current.

4.1.2.5 ATmega8L, nRF24L01+, Debug LED, BMP180

Core element of the prototype board is the ATmega8L microcontroller. A ZIF **Z**ero **I**nsertion **F**orce socket holds the ATmega8L in place and allows for easy code testing and debugging.

The nRF24L01+ is directly connected to the power supply and the SPI interface of the microcontroller. The IRQ (**I**nterrupt **R**e**Q**uest) pin is connected to the ATmega8L pin PD2 (INT0), the CE (**C**hip **E**nable) to PB0 and CSN (**C**hip **S**elect **N**ot) to PB1. For easy plug-in functionality the nRF24L01+ is connected through a two row 4 pin (2x4) female pin header.

A debug LED is connected to pin PC1 for simple debugging purposes. It is not used in the final software but it can be used to indicate if the transmitter works as expected by

flashing the LED every few seconds/minutes. Note that every flashing draws unnecessary current and therefore reducing battery life.

The BMP180 is a simple digital pressure sensor from BOSCH which will be used in the programming example described in section 4.2.1. It offers an I2C interface for easy system integration with a microcontroller. As described in section 2.3.3.5 I2C uses open-collector lines which need to be pulled up with external resistors to the supply voltage. Therefore, the Data (SDI) and the Clock (SCK) lines are pulled up to the 3.3 V power supply through 10 K Ω resistors.

4.1.2.6 Calculate power consumption

To estimate the total power consumption of the prototype board, two different scenarios have to be measured. First one is the normal case, which refers to the consumption most of the time (ATmega8L in power-save - no data transmission). The current consumption during this mode was measured as 11 μ A.

Second one is the transmission case (the ATmega8L is active, reads sensor data and the nRF24L01+ transmits the collected data). To measure and therefore estimate the typical power consumption of the prototype board during data acquisition and transmission a 5.6 Ω shunt was placed in between the power supply (2xCR2032) and the power input of the board (see schematic 7.1 - J1 (bottom right)). The voltage drop through the resistor was measured with a **RIGOL DS1054** 50MHz oscilloscope. Figure 25 shows the measured signal on the shunt. One can see that over 20 voltage peaks extend over a span of 53.2 ms

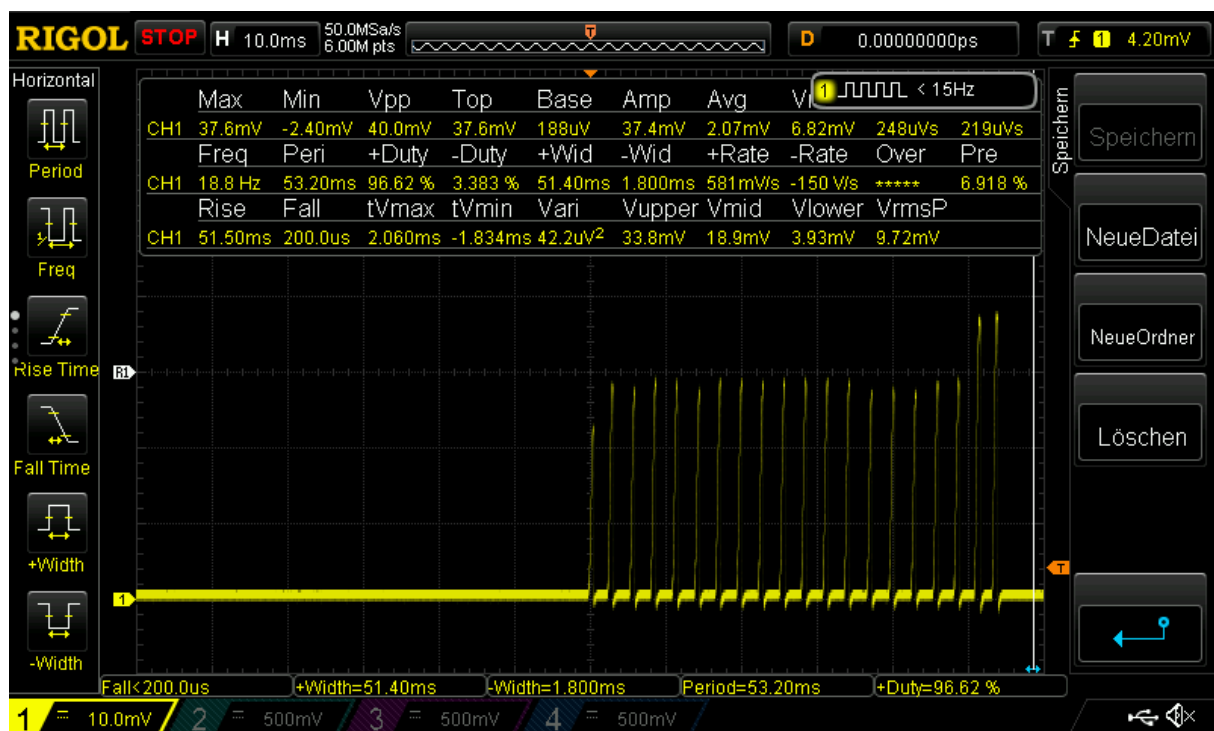


Figure 25: Measurement current consumption prototype board

with a maximum amplitude of 37.6 mV. A closer look into each peak offers figure 26 with a time resolution of 1 ms per division. Each peak has a width of approximately 500 μ s

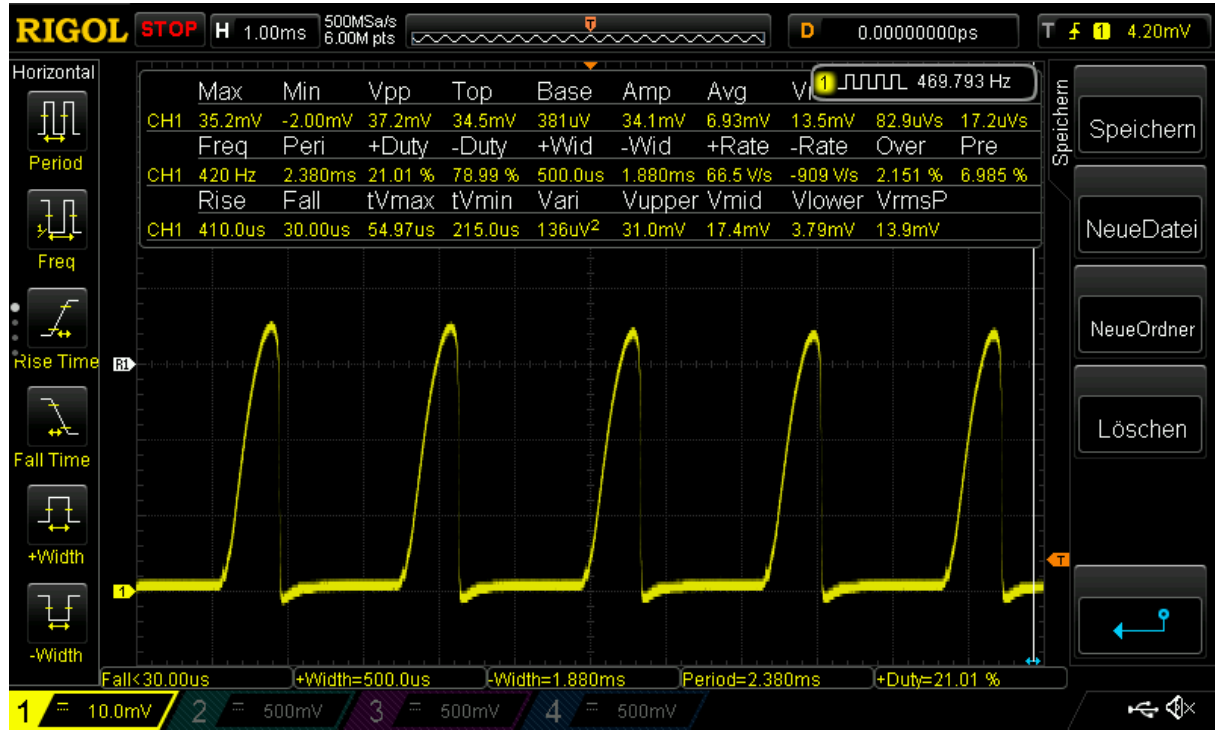


Figure 26: Measurement current consumption prototype board - zoomed in

and repeats every 2.38 ms. Its shape resembles the shape of a rectangular function passed through a low pass filter.

For a worst-case estimation of the total power consumption during transmission it is reasonable to make following simplifications:

- each peak is a rectangle
- each peak has a height of 37.6 mV
- the 5.6 Ω shunt has a real resistance of 5.6 Ω - 5% tolerance = 5.32 Ω

Due to Ohm's law $U=R \cdot I$ the peak current can be calculated as:

$$I = \frac{U}{R} = \frac{37.6mV}{5.32\Omega} = 7.07mA \approx 7mA.$$

Multiplied with the peak width and peak count results in an electric charge of:

$$Q_{\text{transmit}} = n \cdot I \cdot t = 20 \cdot 7mA \cdot 0.5ms = 70\mu As.$$

Comparison to the measured current consumption during power-save mode of 11 μA shows that transmission and standby consumption are of same magnitude if a transmission occurs every few seconds. If the transmission interval is increased to e.g., 15 min the influence of the transmission procedure decreases significantly to

$$100 \cdot \frac{Q_{\text{transmit}}}{Q_{\text{standby}}} = 100 \cdot \frac{70\mu As}{15 \cdot 60 \cdot 11\mu As} \approx 0.7\%,$$

and therefore, the battery life can be calculated by only considering the standby current consumption.

It is possible to power the prototype board with two CR2032 coin cells as specified in section 4.1.2.1. A typical CR2032 battery has a rated capacity of 210 to 240 mAh but it is possible that coin cells of “No name” vendors offer only about 50% of the rated capacity [51]. Let’s assume that the coin cell has a capacity Q_{bat} of 200 mAh. The current consumption over a day adds up to:

$$Q_{\text{day}} = 11\mu A * 24h = 264\mu Ah.$$

Therefore, the battery life can be calculated as follows:

$$t_x = \frac{Q_{\text{bat}}}{Q_{\text{day}}} = \frac{200mAh}{264\mu Ah} = 758d.$$

The self-discharge rate of coin cells is 1% of capacity loss per year at $20^\circ C$.

To sum up it can be said that a typical battery life of about 2 years can be expected using a similar setup as described in this thesis.

4.1.3 Arduino

On the receiving side the nRF24L01+ is connected to the Arduino via an adapter board as shown in figure 27 and 28. The adapter board is widely available on the internet and

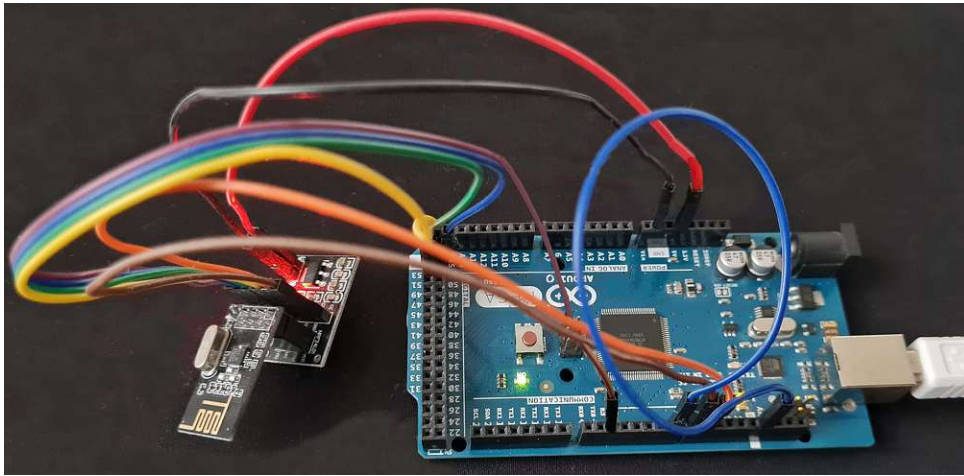


Figure 27: Circuit design of the Arduino receiving unit

offers easy connection of nRF24L01+ modules to breadboards e.g., Arduino, input voltage ranges from 4.8 V to 12 V and stable operation due to additional capacitors. However, it is possible to connect the transceiver module directly to Arduino breadboard by using the 3.3 V power supply instead (nRF24L01+ is only 1.8 to 3.3 V tolerable) but it is recommended to place an additional capacitor ($\approx 100\mu F$) parallel to the power supply. The SPI input pins of the nRF24L01+ are 5 V tolerable but a generally a safer approach is to use the adapter circuit.

Figure 28 shows the wiring diagram of the Arduino receiving unit connecting the Arduino to the adapter board. The nRF24L01+ is plugged into the neighboring 2x4 pin female header as shown in figure 27.

Figure 29 shows the flowchart of the Arduino code used for receiving incoming data by the transceiver module, decrypt its data and transmit it via UART (USB) to the processing unit - the RPi. The full Arduino main code including comments is provided in the appendix (7.2).

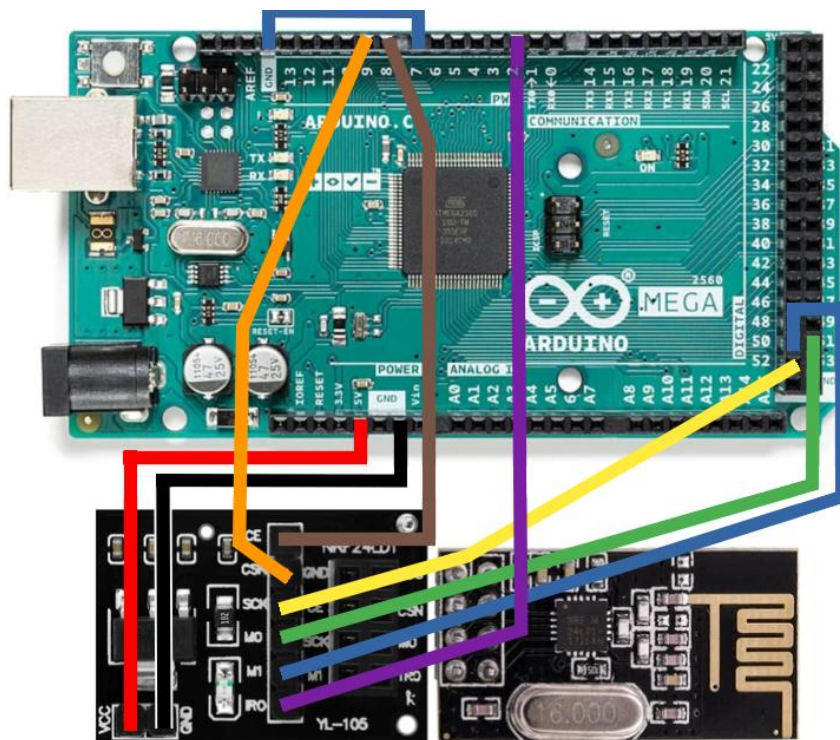


Figure 28: Wiring diagram of the Arduino unit

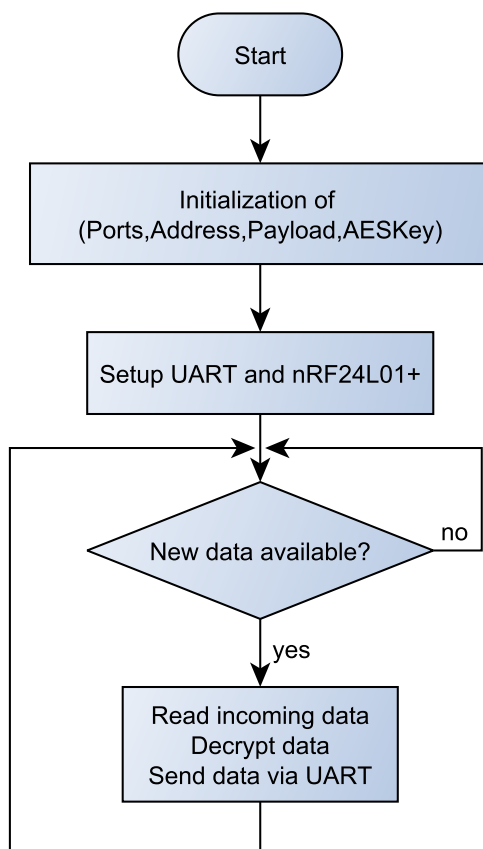


Figure 29: Arduino code flowchart

4.1.4 Raspberry Pi

The RPi serves many functions as shown in figure 30 and discussed in section 4.1.

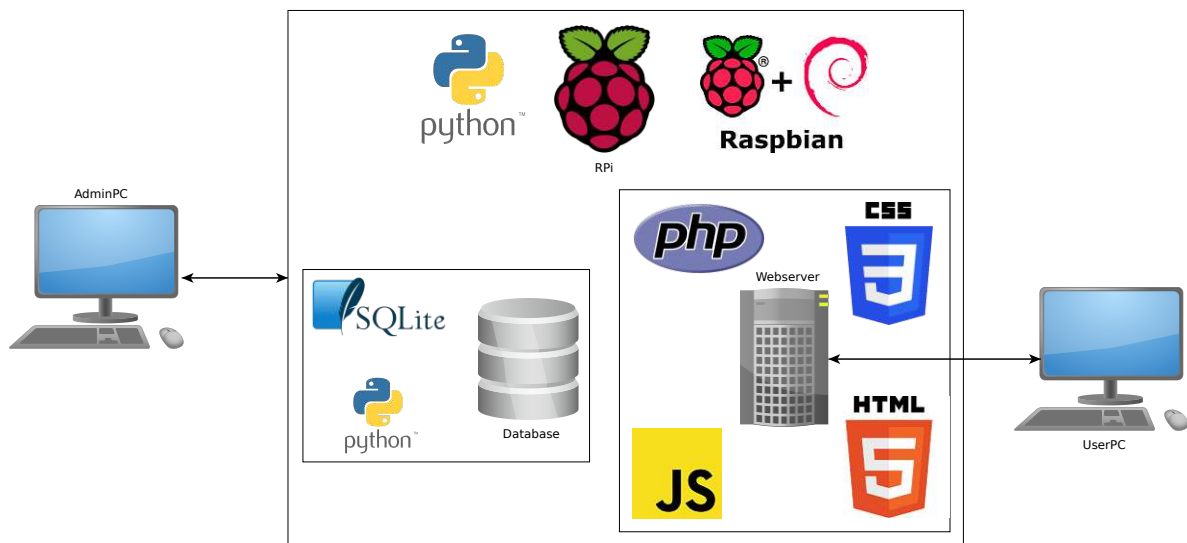


Figure 30: RPi system structure and used programming languages

Its scope can be divided into three different key tasks:

1. Store incoming data (from the Arduino) into a **database**
2. **Check** data and **alert** user if a problem occurred
3. Offer access to the **database** through a **web interface** provided by a **webserver**

To fulfill each of these tasks it is recommended to choose suitable programming languages. The chosen languages and their intended purpose will be described in the following.

Database

For reading and storing incoming data the script language **Python** was chosen. **Python** is a general-purpose, high-level programming language which is due to its simplicity and high popularity a great fit for given purpose. To store and manage the data the database engine **SQLite** was chosen. **SQLite** is the most popular database engine of the world which makes it an ideal choice for this project.

Check & alert

To check the incoming data for errors or problems and give the user updates about the automation system it makes sense to follow the same approach and let a **Python** script (in combination with **SQLite** code) check the data. To alert the user an email will be sent which uses a SMTP (**S**imple **M**ail **T**ransfer **P**rotocol) client called **msmtp**.

Web interface

The web interface is used to display the data stored in the database and con-

trol (start/stop) the **Python** scripts. The standard markup language for documents designed to be displayed in a web browser is **HTML** (**H**yper**T**ext **M**arkup **L**anguage). Other languages like **PHP** (stands for the recursive initialism **PHP**: **H**ypertext **P**reprocessor), **CSS** (**C**ascading **S**tyle **S**heets) and the scripting language **JS** (**J**ava**S**cript) are core technologies of the World Wide Web and were used in combination with **HTML** for the web interface.

PHP code fragments can be embedded in **HTML** by using the tag:

```
<?php
  // PHP code goes here
?> .
```

PHP is used in combination with **SQLite** to access the database and display its data, as well as controlling (start/stop) the **Python** scripts.

CSS is used to design the website, change its layout, color, fonts, table design, button design and much more. It is introduced in **HTML** by using the tag:

```
<style>
  <!-- CSS code goes here -->
</style>.
```

JS defines the webpage behavior on the client side. It is used for automatic refreshing of the webpage, dynamic coloring of the control buttons and displaying the database data through a graph. It is introduced in **HTML** by using the tag:

```
<script>
  <!-- JS code goes here -->
</script>.
```

The first steps for commissioning the RPi were already discussed in section 3.2.1.1, now it is important to install and setup required software (section 4.1.4.1), that the provided code in the appendix (7.2) can be run on the RPi.

4.1.4.1 Installations and settings

Install scheduler

Schedule is a **Python** library that is used for scheduling periodic jobs like updating the user via email every 24h about the current automation system status or alerting the user every 30min about an error. The package installer for **Python** - **pip** is used for installing the library.

```
$ sudo pip install schedule
```

Setup USB permissions

The **Python** script which is used for reading and storing incoming data is controlled by the web interface. It is therefore required to add the Apache user to the dialout group so that if the script which is run by Apache, can access the USB device and open a serial communication.

```
$ sudo usermod -a -G dialout www-data
```

After restarting the RPi the user **www-data** should be member of the dialout group.

Python permissions

To start **Python** script through **PHP** it is required to add following shebang line at the beginning of the **Python** script.

```
#!/usr/bin/python3.7
```

The shebang line determines the script's ability to be executed like a standalone executable. In addition, the file needs to have correct permissions i.e., the file needs to be executable.

```
$ sudo chmod +x myscript.py
```

Setup mail client **msmtp**

To send an email the SMTP client **msmtp** needs to be installed.

```
$ sudo apt install bsd-mailx msmtp msmtp-mta
```

This command should i.a. create a file called **msmtprc** in **/etc/**. If it does not exist it has to be created with following content:

```
#Set default values for all accounts.
defaults
auth on
tls on
tls_starttls on
tls_trust_file /etc/ssl/certs/ca-certificates.crt
logfile /var/log/msmtp.log
```

```
#Gmail settings
account gmail
host smtp.gmail.com
port 587
from name@gmail.com
user name
password XXXXXX
```

```
#Set a default account
account default : gmail
```

In this example the email provider Gmail was used, but it is possible to use a different email provider. **Notice:** Gmail requires an app password to log into the account which has to be generated in the Google Account Settings → Security → App passwords.

Next it is recommended to secure this file, as there is a clear text password in it.

```
$ sudo chown root:msmtp /etc/msmtpc
$ sudo chmod 640 /etc/msmtpc
```

Finally, a logfile has to be created otherwise **msmtp** throws an error message.

```
$ sudo touch /var/log/msmtp
$ sudo chown msmtp:msmtp /var/log/msmtp
$ sudo chmod 660 /var/log/msmtp
```

Password protect website

To save webpage access from improper usage it is recommended to password protect the website. A **.htaccess** file tells the web server what folder you want to protect and what username/password file to use. The webserver folders are generally located in **/var/www/html** so it is recommended to put the **.htaccess** file in there (**/var/www/html/.htaccess**). Following content should be written to the **.htaccess** file.

```
AuthUserFile /home/pi/Documents/.htpasswd
AuthType Basic
AuthName "My_restricted_Area"
Require valid-user
ErrorDocument 404 /404.html
```

The specification of an **ErrorDocument** is optional but it is recommended to catch further problems.

The line *AuthUserFile* specifies the location of the password file which should be saved locally on the RPi in a folder which is not fetchable through the webserver with a browser (e.g., **/home/pi/Documents/.htpasswd**). The password file can be created by typing following command:

```
$ htpasswd -c -B /home/pi/Documents/.htpasswd pi
```

which is followed by two prompts asking for the password you want to set.

```
$ New password:
```

```
$ Re-type new password:
```

Lastly it is required to change following line in `/etc/apache2/apache2.conf` from:

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

to:

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
```

and enable module rewrite. After restarting the server, the webpage access is limited to users who know the defined username and password (see figure 31).

```
$ sudo a2enmod rewrite
```

```
$ sudo service apache2 restart
```

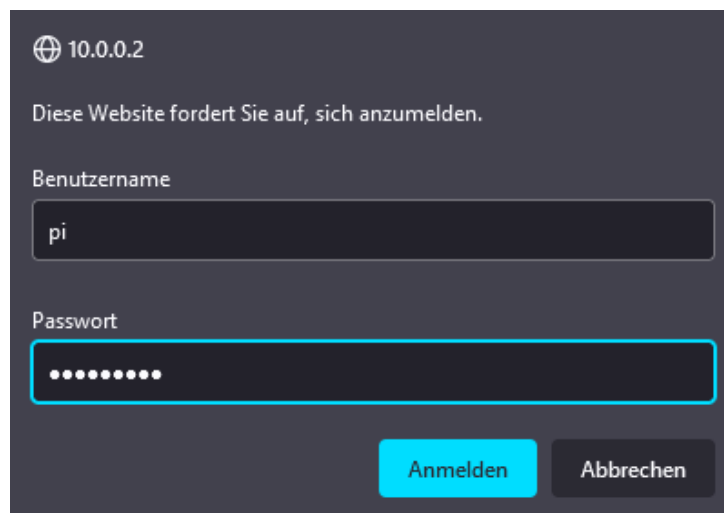


Figure 31: Website requires username and password

Setup shutdown trough web interface

If it is desired to shut down the RPi there is typical one approach. Connect to the RPi through **SSH** and type in:

```
$ sudo shutdown -h now .
```

This is quite cumbersome and slow. To make things easier a shutdown button is provided by the web interface (refer to the sidebar figure 34) which allows every user knowing the website password to shut down the RPi. To make sure that **PHP** and respectively the webserver is able to use the shutdown command it is required to set the SUID-Bit (also setuid, **Set User ID**) of the shutdown command. The shutdown command was located in `/usr/sbin/shutdown` for a long time but was moved during writing of this thesis to `/bin/systemctl`.

```
$ sudo chmod +s /bin/systemctl
```

Note: When the setuid or setgid attributes are set on an executable file, then any users able to execute the file will automatically execute the file with the privileges of the file's owner (commonly root) and/or the file's group, depending upon the flags set.

4.1.4.2 Database

Initialization

Before storing data into a database, the database and its table layout has to be initialized. The python script `initialize_DB_Tables.py` does just that by creating a database file called `IoT.db` containing the table `ATMEGA8_Data`. The table contains five columns: `id` (number of current entry), `Date` (date of entry), `Temperature` (Temperature readings of BMP180), `Pressure` (Pressure readings of BMP180), `ADCValue` (ADC2 input reading). The table entries correspond to data gathered and transferred by the ATmega8L and need to be customized for specific applications.

It is possible to create the database file on the admin PC (if a python interpreter is installed) and transfer it with winSCP to the RPi or simply enter in the SSH terminal following command (if file located in the same folder):

```
$ python3 initialize_DB_Tables.py
```

Read and store data

For reading incoming data and storing it in the before created database `IoT.db` the python script `saveTest.py` is being used. Figure 32 shows a simplified flowchart of `saveTest.py`. The reading and storing procedure will be described in detail in section 4.2.1.4, as the code is adapted to the examples given in section 4.2.1 and 4.2.2. The script starts by **defining and calling the function `get_lock()`**. This

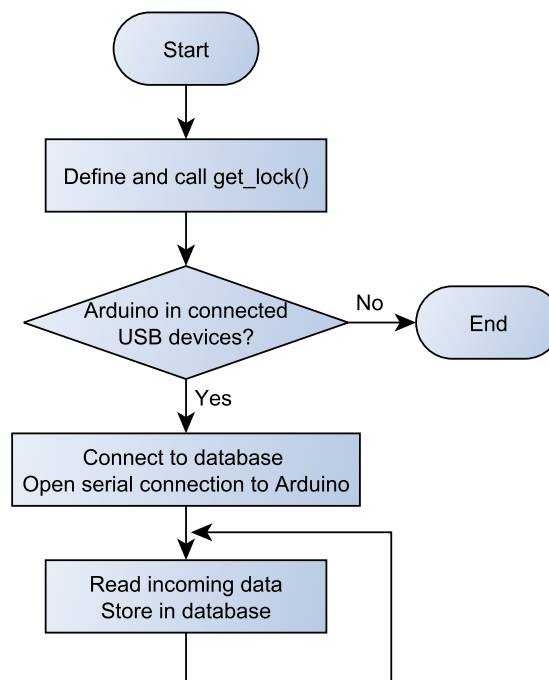


Figure 32: `saveTest.py` flowchart

function checks if an instance of the python script is already open and denies opening

a second one. Only one `saveTest.py` script can be active at a time and store data into the database. After that it is checked if the **Arduino is in the connected USB devices**. If not, an exception is raised and the script is terminated. Otherwise a **connection to the database** as well as a **serial connection to the Arduino** is established. An infinite while loop **reads incoming data** and **stores it** in the format provided by the `initialize_DB_Tables.py` file. The same condition applies to the code inside the while loop as for the table initialization file: The table entries correspond to data gathered and transferred by the ATmega8L and need to be customized for its application.

Unlike the initialization script, `saveTest.py` can be started through the web interface as shown in figure 37.

Check & alert

The python script `checkdaily.py` gives the user daily updates about the automation system and checks current database data for problems and alerts the user through email. Figure 33 shows a simplified flowchart of `checkdaily.py`. The functions `dai-`

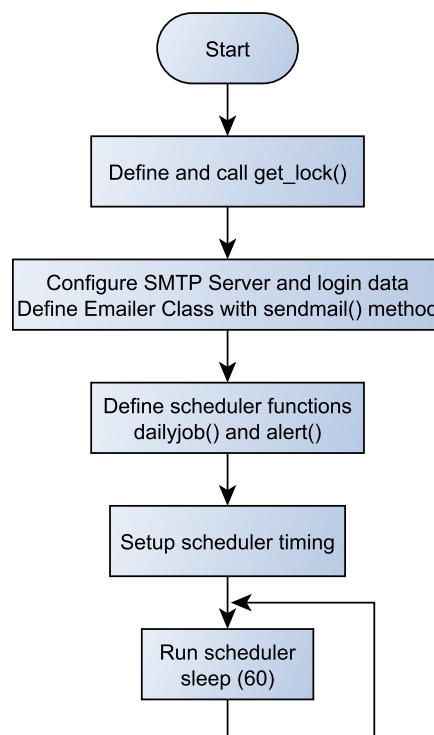


Figure 33: `checkdaily.py` flowchart

`lyjob()` and `alert()` can be customized. They are described in section 4.2.1.4 as the code is adapted to the examples given in section 4.2.1 and 4.2.2.

The script starts by calling the above mentioned `get_lock` function. After that the **SMTP server is configured** as well as the **login data** defined. The class **Emailer** holds the method `sendmail` which creates the email header, connects and log-ins to the mail server and sends the mail. Then the scheduler functions `dailyjob()` and

`alert()` are defined. The code is concluded by **setting up a scheduler timing** (how often are the functions called) and **running the scheduler** every **60 seconds**.

4.1.4.3 Website

The webserver folders are located in `/var/www/html`. Entering the IP address of the server in this case 10.0.0.2 into any browser within the local network opens the start page `/var/www/html/index.html` (see figure 34) after entering the correct username and password (see figure 31). The IP address of the server is the same as used when opening a SSH connection through PuTTY. It can be gathered by checking the routers list for the RPi's address or by entering in the RPi's terminal:

```
$ ifconfig
```

and reading the IP address right beside the label `inet`.

Home

The start page gives a short introduction about the different features the website offers (see figure 34). On the left side is a sidebar menu located which redirects the user to predefined websites after "clicking" on the corresponding name. **Home** redirects to the start page HTML file `/var/www/html/index.html`. Every HTML document



Figure 34: Startpage after logging in

follows the same structure.

1. Document declaration on top of the document specifying the **Document Type Definition** (DTD)
2. HTML-header which holds technical or other document related information - it contains not visible page content.
3. HTML-body which holds visible page content

A typical HTML document example is given in the following.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Website title</title>
    <meta content="text/html; charset=ISO-8859-1" http-equiv="content-
      type">
```



```

<!-- other header informations -->
<link rel="stylesheet" type="text/css" href="/style/dropdown.css">
</head>
<body>
<!-- Load an icon library -->
<!-- arrow -->
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
font-awesome/4.7.0/css/font-awesome.min.css">
<!-- The sidebar --
<?php include 'php/sidebar.php'; ?>
<p>Visible content</p>
</body>
<script src="js/dropdown.js"></script>
</html>

```

The `index.html` file is similar in structure, adding only a few body entries and a second script printing current time and date. The sidebar menu is outsourced to a PHP file as it is visible on every site. Outsourcing has the advantage that the outsourced code can be integrated to the website similar to a function call. The file needs to be located in `/var/www/html/php/sidebar.php` and is embedded by using the tag:

```
<?php include 'php/sidebar.php'; ?>
```

The corresponding CSS code needs to be placed in `/var/www/html/style/dropdown.css` and is integrated through the following line:

```
<link rel="stylesheet" type="text/css" href="/style/dropdown.css">
```

Lastly the corresponding JS code needs to be placed in `/var/www/html/js/dropdown.js` and is integrated through:

```
<script src="js/dropdown.js"></script>
```

In the `sidebar.php` file the location of the HTML documents to which the sidebar menu entries redirect, are specified.

Clients

By “clicking” on the sidebar menu **Clients** the HTML document `/var/www/html/clients.html` opens (see figure 35). There are two similar rows visible. One for



Figure 35: Sidebar menu **Clients**

controlling (start/stop) the **ATmega8L Data** gathering script and showing the latest data entry. The second one for controlling the **Status Updates** script.

ATmega8L Data

To start/stop a **Python** script through **HTML** the user needs an interface to interact with the website. For this problem **HTML** offers a form and buttons which can be integrated through following code fragment.

```
<form action=" " method="post">
  <input type="submit" name="start1" id="start1" value="start" />
  <input type="submit" name="stop1" id="stop1" value="stop" />
</form>
```

The form uses the method “post”. To collect form data after submitting an **HTML** form the variable `$_POST` in **PHP** is used. Following code, which is a fragment of the first few lines of the `/var/www/html/clients.html` code shows that the **PHP** script checks if the “start1” button was “clicked” and if that statement is true starts the **Python** script.

```
<?php
if(isset($_POST['start1']))
{
    $mystring = exec('/home/pi/code/saveTest.py >/dev/null 2>&1 &');
    sleep(5);
    exec('pgrep saveTest.py', $output);
    if($output)
    {
        echo '<script>alert("Successfully connected!")</script>';
    }
    else
    {
        echo '<script>alert("Can not connect to Arduino -> Please check connection")</script>';
    }
}
```

```

}
if (isset($_POST['stop1'])) {
    exec("sudo kill saveTest");
}
?>

```

The `exec()` function is used to execute an external program in this case the Python script that is located in `home/pi/code/saveTest.py`. On the right side of the file name are a few statements which are described in the following.

1. `>`: redirect output
2. `/dev/null`: special file used for disposing unwanted output streams
3. `2>&1`: merge output from stream 2 with stream 1 - stderr merge with stdout and dispose output
4. `&`: run Python script in background

After 5 seconds (function `sleep(5)`) it is checked whether the connection was successful. Therefore the command `exec('pgrep saveTest.py', $output)` checks if a python script with the name `saveTest.py` is running (returns a value if running - otherwise not). If the script is still running an alert pops up with the message "Successfully connected!" otherwise the alert shown in figure 36. The

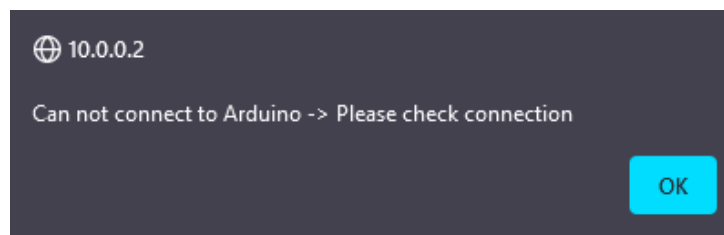


Figure 36: Error message Arduino

script can be stopped by "clicking" on the "stop1" button which executes the command '`sudo pkill saveTest`' killing the process with the name `saveTest`. For the button coloring based on "clicking" the start or stop button a JS script is used (`/var/www/html/clients.html` lines 180 to 226). The button "stop" is colored red if the script is stopped, the button "start" is colored green if the script is running (see figure 37). If the connection to the Arduino was not successful



Figure 37: ADC Sensor Data - Python script started

the user needs to "click" on the "stop" button otherwise the colors do not match the current status.

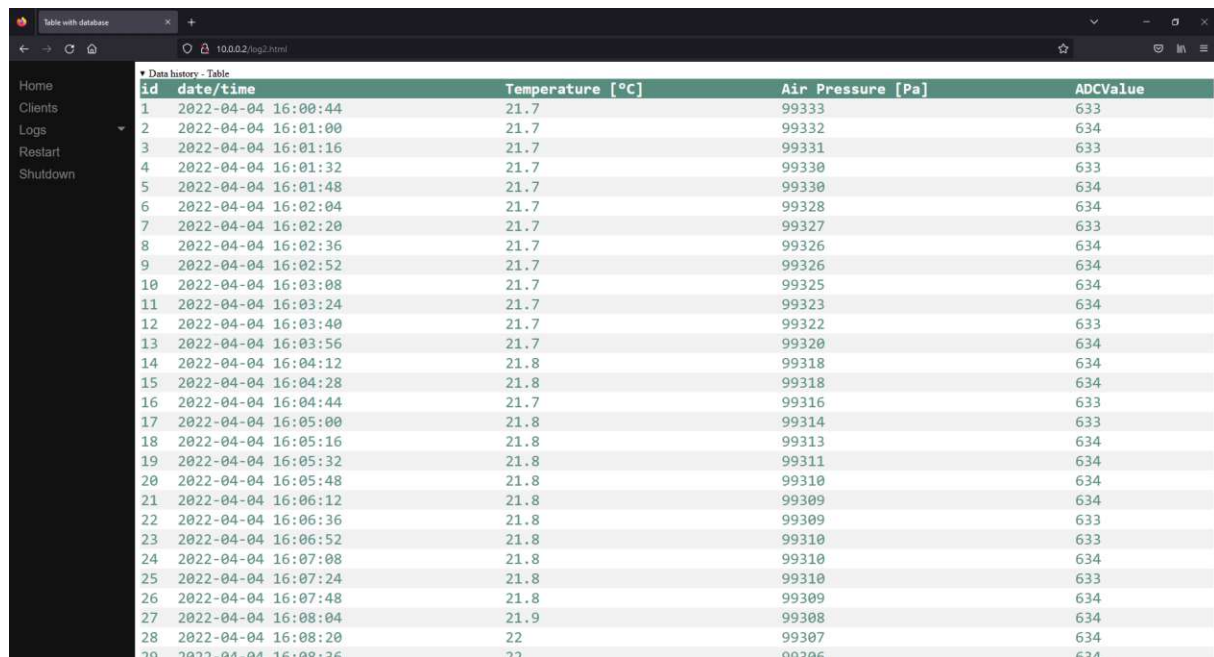
A PHP script is used to access the data from the SQLite3 database (`/var/www/html/clients.html` lines 137 to 156) and print the latest entry as table content.

Status Updates

The **Status Updates** row controls the script located in `home/pi/code/checkdaily.py` using the same approach as described section 4.1.4.3 **ATmega8L Data**.

Logs

“Clicking” on the sidebar menu **Logs** opens a dropdown menu with one entry **Log1** which redirects to the HTML document `/var/www/html/log1.html`. The dropdown menu is not required since there is only one table dataset to print. However, if more than one ATmega8L transmits data the website can be easily expanded by adding another entry in the division class container **dropdown-container** in the `/var/www/html/php/sidebar.php` file. Figure 38 shows the website opened by “clicking” **Log1** and **Data history - Table**. The HTML code is similar to the code used in

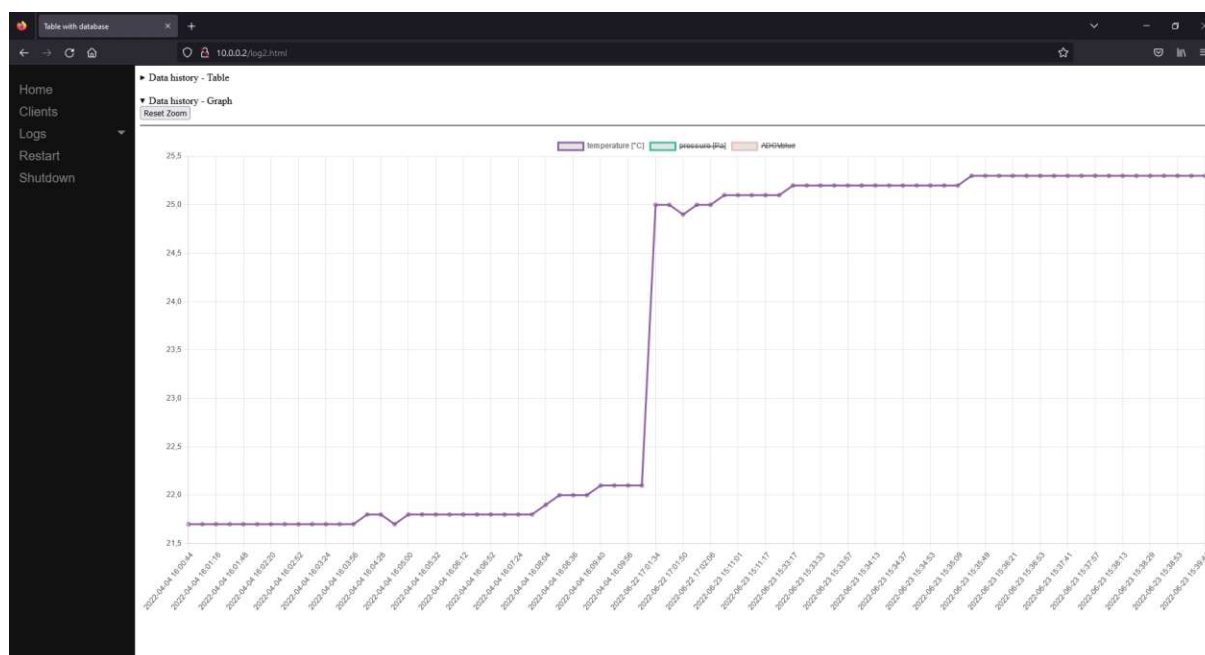


id	date/time	Temperature [°C]	Air Pressure [Pa]	ADCValue
1	2022-04-04 16:00:44	21.7	99333	633
2	2022-04-04 16:01:00	21.7	99332	634
3	2022-04-04 16:01:16	21.7	99331	633
4	2022-04-04 16:01:32	21.7	99330	633
5	2022-04-04 16:01:48	21.7	99330	634
6	2022-04-04 16:02:04	21.7	99328	634
7	2022-04-04 16:02:20	21.7	99327	633
8	2022-04-04 16:02:36	21.7	99326	634
9	2022-04-04 16:02:52	21.7	99326	634
10	2022-04-04 16:03:08	21.7	99325	634
11	2022-04-04 16:03:24	21.7	99323	634
12	2022-04-04 16:03:40	21.7	99322	633
13	2022-04-04 16:03:56	21.7	99320	634
14	2022-04-04 16:04:12	21.8	99318	634
15	2022-04-04 16:04:28	21.8	99318	634
16	2022-04-04 16:04:44	21.7	99316	633
17	2022-04-04 16:05:00	21.8	99314	633
18	2022-04-04 16:05:16	21.8	99313	634
19	2022-04-04 16:05:32	21.8	99311	634
20	2022-04-04 16:05:48	21.8	99310	634
21	2022-04-04 16:06:12	21.8	99309	634
22	2022-04-04 16:06:36	21.8	99309	633
23	2022-04-04 16:06:52	21.8	99310	633
24	2022-04-04 16:07:08	21.8	99310	634
25	2022-04-04 16:07:24	21.8	99310	633
26	2022-04-04 16:07:48	21.8	99309	634
27	2022-04-04 16:08:04	21.9	99308	634
28	2022-04-04 16:08:20	22	99307	634
29	2022-04-04 16:08:36	22	99306	634

Figure 38: Sidebar menu **Logs** - Table

`/var/www/html/clients.html` except that in this case the complete database entry is plotted. Another “click” on **Data history -Table** hides the table.

“Clicking” on **Data history - Graph** opens the graph as shown in figure 39. It is possible to zoom-in and -out the graph by aiming the mouse at the area of interest and using the mouse wheel. The zoom can be reset by “clicking” the button “Reset Zoom” which is located beneath **Data history - Graph**. Furthermore “clicking” on the names of the graph entries (temperature [°C], pressure [Pa], ADCValue) located above the line-chart lets the user hide/show entries in the graph. The diagram automatically adjusts its borders to fit the new value range.

Figure 39: Sidebar menu **Logs** - Graph

For the creation of this chart the JS library **Chart.js** is used (see `/var/www/html/log1.html` lines 97 to 99). To control the zoom the **Chart.js** plugin **chartjs-plugin-zoom** is used (see `/var/www/html/log1.html` lines 105 to 107). Refer to `/var/www/html/log1.html` - lines 109 to 157 for the JS code used to create the graph.

Restart

“Clicking” on the sidebar menu **Restart** redirects the user to the PHP script located in `/var/www/html/php/restart.php` where the script executes the command `exec("/sbin/shutdown -r now");` which restarts the RPi if the manual given in section 4.1.4.1 - **Setup shutdown trough web interface** has been followed. `/var/www/html/php/restart.html`

Shutdown

“Clicking” on the sidebar menu **Shutdown** redirects the user to the PHP script located in `/var/www/html/php/shutdown.php` where the script executes the command `exec("/sbin/shutdown -h now");` which shuts down the RPi if the manual given in section 4.1.4.1 - **Setup shutdown trough web interface** has been followed.

4.2 Step by step instruction for different sensors

4.2.1 Example: Temperature/Pressure Measurement

Figure 40 shows the system structure of the temperature/pressure measurement example. The setup is almost identical to the setup described in section 4.1 with the difference that

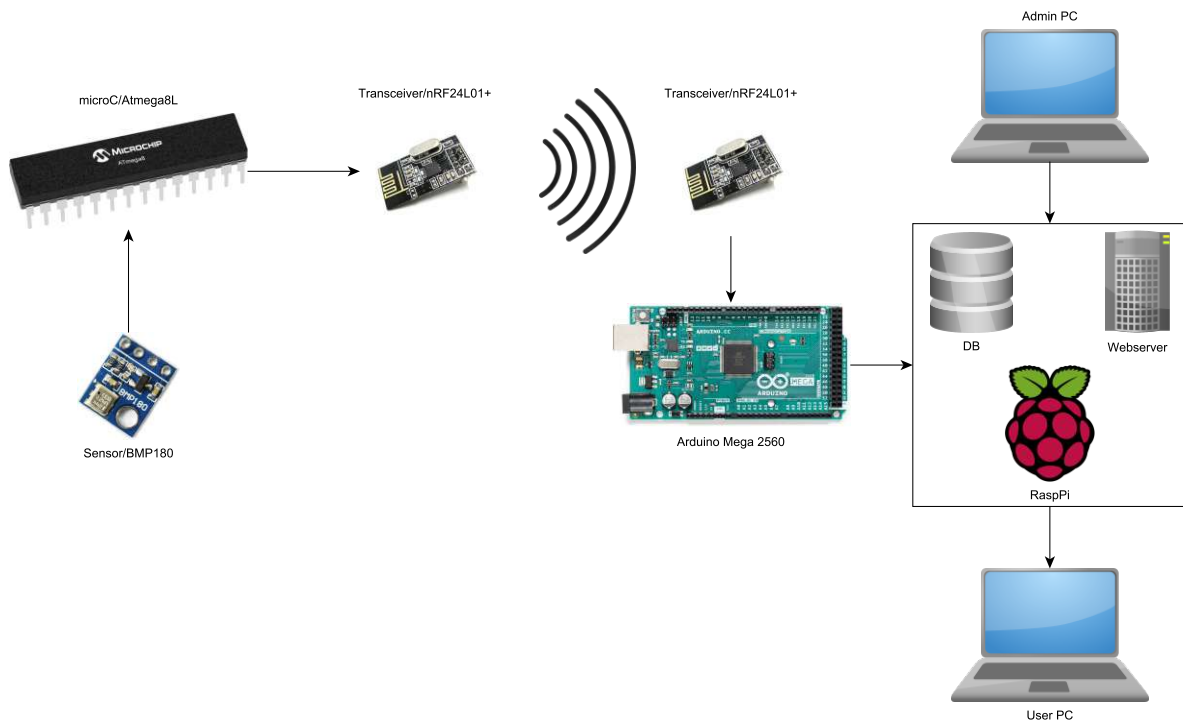


Figure 40: System structure of the Temperature/Pressure Measurement Example

the sensor used is specified (BMP180). In the following the implementation of the sensor will be discussed, what code changes are made to gather, transmit, store and print the sensor data. The hardware - how to connect the sensor to the prototype board is discussed in section 4.1.2 (see figure 24).

4.2.1.1 Temperature/Pressure Sensor

The BMP180 is an ultra-low power, low-voltage digital pressure sensor used for consumer applications. It is based on piezo- resistive technology for EMC (ElectroMagnetic Compatibility) robustness, high accuracy and linearity as well as long term stability. In the following key features of the BMP180 are listed from the datasheet.

- Pressure range: 300 ... 1100 hPa (+9000 m ... -500 m relating to sea level)
- Supply voltage: 1.8 ... 3.6 V (VDD)
- Package: LGA package with metal lid - 3.6 mm x 3.8 mm x 0.93 mm height
- Low power: 5 μ A at 1 sample / sec. in standard mode
- Low noise (ULPM): 0.06 hPa (0.5 m) in ultra low power mode
- Low noise (ARM): 0.02 hPa (0.17 m) in advanced resolution mode

- Temperature measurement included
- I2C interface
- Fully calibrated

Typical applications include enhancement of GPS navigation, in- and out-door navigation, weather forecast and vertical velocity indication.

4.2.1.2 Microcontroller - ATmega8L

The manual given in section 4.1.1 holds true for every sensor connected to the ATmega8L. The only difference is, that the point **Get sensor/adc data or create dummy data** of the flow chart as seen in figure 22 gets specified and will be discussed in the following (`nRF24L01BmpAdc.c` combines the codes used in example 1 and 2).

To communicate with the sensor the I2C/TWI protocol is used which uses the library provided by Peter Fleury `i2chw/twimaster.c`, `i2cmaster.h`. For pressure and temperature readings of the BMP180 the library `bmp085/bmp085.c`, `bmp085.h` provided by Davide Gironi is used. BMP085 is a function compatible predecessor of the BMP180. It is therefore possible to use code written for the BMP085 and use it to communicate with the BMP180.

The datasheet of the BMP180 by BOSCH specifies a flow chart (page 15) on how to calculate pressure and temperature for the BMP180. The functions given in `bmp085/bmp085.c` use exactly this flow chart to calculate the pressure and temperature which can be read by calling the functions `bmp085_gettemperature()`; (returns a **double** value) and `bmp085_getpressure()`; (returns a **long** value).

To store the data readings of the BMP180 on the ATmega8L for further processing (i.g. encryption, transmission), variables have to be declared. For this purpose, the special data type **union** is used that allows to store different data types in the same memory location. In C code this translates as shown in the following.

```

union Temp                                     1
{                                              2
    double dtemp;                             3
    uint8_t itemp[4];                         4
}temp;                                        5
                                              6

union Press                                   7
{                                              8
    long lpress;                               9
    uint8_t ipress[4];                       10
}press;                                       11

```

A **double** value for the temperature (**dtemp**) and a **long** value for the pressure (**lpress**) is used. The memory can also be accessed by using the `uint8_t` array datatype (`itemp[]`, `ipress[]`), which splits the respective memory locations in four bytes. For initializing the payload with the BMP180 readings this is the matching datatype as the payload is also

stored in 16 `uint8_t` bytes. Following code shows the function calls for temperature and pressure reading and the payload initialization procedure which are performed every time the ATmega8L wakes up.

```

temp.dtemp = bmp085_gettemperature(); //[degC] 1
press.lpress = bmp085_getpressure(); //[Pa] 2

for(uint8_t i=0; i<4; ++i) //payload data 1:4 represents temperature 1
    / 4=MSB : little-endian
{ 2
    payload[i+1]=temp.itemp[i]; 3
} 4
for(uint8_t i=0; i<4; ++i) //payload data 5:8 represents pressure / 5
    8=MSB : little-endian
{ 6
    payload[i+5]=press.ipress[i]; 7
} 8

```

The payload byte 0 is used to count how many packages were sent since start-up (**Notice:** Resets at 250, continues with 0). It is possible to use the use payload byte 0 to indicate which nRF24L01+ wants to transmit data (see section 4.2.1.3). The payload bytes 1 to 4 are used to store the temperature in the integer little-endian notation. Little-endian means that the **M**ost **S**ignificant **B**yte (MSB) is stored in the highest memory address (in this case `payload[4]`). Payloads bytes 5 to 8 are used to store the pressure in the same integer little-endian notation (MSB=`payload[8]`). On the receiving RPi the payload bytes have to be identified and reformatted to resemble the original values. This procedure will be described in section 4.2.1.4. Afterwards the 16 byte **payload** is encrypted with the `aes` library as seen in the flow chart figure 22 followed by the standard transmission procedure.

To use the BMP180 it is necessary to initialize/calibrate the sensor and the I2C interface. The function `bmp085_init()`; which is called before the main loop does just that by calling the `i2c_init()`; function and calibrating the sensor.

4.2.1.3 Arduino

The Arduino code (see section 4.1.3) does not change as the Arduino is only used as a gateway to collect, decrypt and redirect the payload to the RPi independent of its content.

Notice: If more than one nRF24L01+ module is used to gather and transmit data the code for the Arduino has to be adapted.

There are multiple possibilities on how to implement such a system, in the following two of the most common will be discussed.

Pipe address

The nRF24L01+ offers the so-called MultiCeiver™ feature as described in the nRF24L01+ datasheet on page 37 to 39 [27]. Up to six parallel data pipes with unique addresses can be received by one nRF24L01+ configured as a receiver on one frequency channel. Pipe 0 has a unique 5-byte address, pipe 1-5 share the four most significant address bytes. It is possible to change pipe 1 to a desired value, but the LSB must be unique for all six pipes.

In the examples given in this thesis only the pipe 0 is used with its default value of `0xE7E7E7E7E7`. If multiple nRF24L01+ transmit data on different pipes on the same frequency channel it is required to open more than one reading pipe on the Arduino by using multiple statements of the method `radio.openReadingPipe(X, value)`; where X represents the pipe (0 to 5) and value the pipe address. The method `radio.available(&pip)`; is used to check if a nRF24L01+ wants to transmit data where the arguments holds the pipe number of the active radio. A simple if statement checking for the pip number is sufficient to find out which nRF24L01+ in the system wants to send data.

Changing the pipe address on the ATmega8L is more complex. The function `wl_module_tx_config(wl_module_TX_NR_0)`; configures the module with its default (pipe 0) value `0xE7E7E7E7E7`. To change the pipe it is possible to alter the argument to another predefined value (see `wl_module.h`) or setup another TX pipe address (`tx_addr[]`) in the switch case statement in the `wl_module_tx_config(uint8_t tx_nr)`; function defined in `wl_module.c`.

Create individual header

A much simpler approach is to use the payload 0 byte to represent the transmitter. For example transmitter 1 has a fixed value of 42 stored in `payload[0]` and transmitter 2 the fixed value of 83 in `payload[0]`. The Arduino checks for the first payload byte it receives and processes the data accordingly (`Serial.println("Receive from sensor1")`; or `Serial.println("Receive from sensor2")`;) to inform the RPi which database table it should use.

4.2.1.4 Raspberry Pi

An introduction to the basic operation of the `python` script used for data storage (**Read and store data**) and email alerts **Check & alert** are given in section 4.1.4.2 (see flowchart figure 32 and 33). In the following it will be discussed how data storage works adjusted to the data provided by the ATmega8L (`saveTest.py`) and what the code example given by `checkdaily.py` does and how it can be customized. Additionally, it will be discussed how to adjust the website if different data is provided.

`saveTest.py`

As in section 4.2.1.2 discussed the data gathered from the BMP180 sensor is provided in the datatype **double** for the temperature and **long** for the pressure. The data is then split into individual bytes (**uint8_t**) and sent in little-endian notation to the RPi. On the receiving RPi the data is read (function `s.readline()`;) cast to the **integer** datatype and stored in individual lists (**temp**, **press**) as seen in the code below.

```
s.readline().rstrip() #first byte represents packetnumber      1
for x in range(4):                                             2
    temp.append(int(s.readline().strip(b'\r\n'))))              3
for x in range(4):                                             4
    press.append(int(s.readline().rstrip()))                    5
```

To identify the bytes stored in the lists as the data format which had been originally defined, the function `struct.unpack()`; from the library **struct** is used. The first argument defines the byte order and the datatype. For example, '`<f`' stands for little-endian float (4 byte) and '`<l`' stands for little-endian long (4 byte). Applied to the result of this function is the `map()` and afterwards the `join()` function. The `map()` function converts the resulting tuple into a **string** and the `join()` function joins the tuple into one string which can be stored in the database.

```
tuple = (datestamp,                                           1
        ''.join(map(str, (struct.unpack('<f', bytearray(temp))))), 2
        ''.join(map(str, (struct.unpack('<l', bytearray(press))))), 3
```

`checkdaily.py`

The `checkdaily.py` code runs the `alert()` schedule function every minute. The `alert()` function access the database and reads the last temperature entry. The temperature is compared to a predetermined value (in that case **21**) and if the last temperature table entry exceeds the predetermined value an email is sent to the specified email address. This is just an example on how to program such a scheduler and the function can be customized to specific preferences.

The `dailyjob()` schedule function runs once a day (in this case every day at **06:00**). This function sends an email to the specified email address containing the last temperature entry.

How often the scheduler should run can be set as seen in the code below.

```
schedule.every().day.at("06:00").do(dailyjob, 'It is 06:00') #change 1
time according to your preferences for daily updates
schedule.every(1).minutes.do(alert) #change time according to your 2
preferences for alerts
```

Website

To adjust the website for different database entries each file accessing the database has to be modified (`clients.html`, `log1.html`). The table entries of the website change as well as the PHP code used for accessing the database. If the changes are made according to the code example provided by this thesis, every similar database can be displayed.

4.2.2 Example: ADC Measurement

In this section the ADC of the ATmega8L is introduced by showing and explaining important code lines of the ADC implementation. An introduction on how the ADC is connected to the test circuit is given in section 4.1.2.4 (see figure 24).

4.2.2.1 ADC

As described in section 2.4.3 an ADC is used to convert analog signals into digital values which represent the magnitude of the voltage. These digital values are necessary for further data processing.

As the ATmega8L datasheet states the ATmega8L features a 10-bit ADC connected to an 8-channel analog multiplexer which allows for eight analog voltage inputs [45]. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. If the internal ADC voltage reference of 2.56 V is used, this translates to a voltage resolution of $2.56 \text{ V}/1024 = 2.5 \text{ mV}$. The ADC further contains a sample and hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A sample and hold is an indispensable part of an ADC to get an accurate conversion result.

The AD conversion is done through successive approximation. Successive approximation works as follows. The analog signal which is hold constant throughout the conversion through a sample and hold circuit, is compared to the voltage output an internal DAC (**D**igital **A**nalog **C**onverter) provides. The comparison starts with the MSB (**M**ost **S**ignificant **B**it) and ends on the LSB (**L**east **S**ignificant **B**it).

For example, the voltage on the ADC input is 1.33 V. This voltage is compared to the voltage the DAC supplies if only the MSB is set

$$V_{\text{DAC}} = \frac{V_{\text{REF}}}{1023} * 512 = 1.28 \text{ V}.$$

As the voltage is lower than the ADC input voltage, this bit can be set. Now the next bit (MSB-1) is set and the DAC voltage compared to the input voltage

$$V_{\text{DAC}} = \frac{V_{\text{REF}}}{1023} * (512 + 256) = 1.92 \text{ V}.$$

The DAC voltage is too high, so the (MSB-1) bit has to reset. If this procedure is completed until ending on the LSB, 1.33 V should be represented by the digital value of 531

$$N_{\text{digital}} = \frac{1.33 \text{ V} * 1023}{2.56 \text{ V}} = 531.$$

Another important value is given by the following statement given in the ATmega8L datasheet [45]. “By default, the successive approximation circuitry requires an input clock frequency between 50kHz and 200kHz to get maximum resolution.” This means that the ADC prescaler has to be set according to the internal clock frequency of the ATmega8L.

4.2.2.2 Microcontroller - ATMEGA8L

Following code lines are used to setup the ADC on the ATmega8L and setup the pins used for controlling the debug LED and the ADC test circuit power supply (see figure 24).

```
DDRC |= (1 << PC1)|(1 << PC3);           // Setup PC1 (LED) as output 1
      ; PC3 (ADC Switch on/off)

ADMUX = (1<<REFS1)|(1<<REFS0)|(1<<MUX1); //Internal 2.56V Voltage 1
      Reference with external capacitor at AREF pin and ADC2

ADCSRA = (1<<ADEN)|(1<<ADPS2); // Enable ADC, set prescaler to 16 2
      // Fadc=Fcpu/prescaler=1000000/16=62.5kHz 3
      // Fadc=Fcpu/prescaler=8000000/64=125kHz 4
      // Fadc should be between 50kHz and 200kHz 5
```

DDRC stands for **D**ata **D**irection **R**egister Port **C**. The ATmega8L has got 3 different ports Port B, C and D each containing up to 8 different I/O pins. The ADC multiplexer and the I2C pins are connected to the pins on port C therefore it makes sense to use adjacent pins for simple tasks like switching on/off the ADC test circuit power supply and controlling the debug LED to reduce wiring on the prototype board.

ADMUX stands for **A**DC **M**ultiple**X**er Selection Register. The ADMUX controls the ADC reference source REFS1:0 (**R**E**F**erence **S**election) bits and the ADC multiplexer MUX3:0 (Analog Channel Selection) bits. If both REFS bits are set to one the internal 2.56 V voltage reference is used. An external capacitor at the AREF (**A**nalog **R**E**F**erence pin for the A/D Converter) pin increases the voltage stability of the internal voltage reference and therefore is included in the prototype board schematic (see figure 24). By setting the MUX1 bit of the ADMUX register the ADC2 on pin PC2 is selected which is connected to the sliding contact of the trimmer (see section 4.1.2.4).

ADCSRA stands for **A**DC Control and **S**tatus **R**egister **A**. It is used to switch on the ADC (set the ADEN: **A**DC **E**Nable bit) and setup the ADC prescaler. ADPS2:0 are the **A**DC **P**rescaler **S**elect bits. If the ADPS2 bit is set, the prescaler is set to 16. As the ADC frequency should be between 50 KHz and 200 KHz (see section 4.2.2.1) this value is sufficient if the ATmega8L clock frequency is set to 1 MHz.

To initiate an AD conversion following code lines have to be implemented. In the code example given in the appendix (7.2 - *nRF24L01BmpAdc.c*) an AD conversion is carried out every time the ATmega8L wakes up from power save, so therefore these lines are placed within the while loop.

```
PORTC &= ~(1 << PC3); //switch on power supply for trimmer 1
ADCSRA |= (1<<ADEN)|(1<<ADSC); //start first conversion 2
while(ADCSRA &(1<<ADSC)); //wait until conversion is finished 3
//ACSR = 0x80; 4
ADCSRA &= ~(1<<ADEN); //switch off ADC 5
```

```

PORTC |= (1<<PC3); //switch off power supply for trimmer      6
                                                                    7
    adcddata.valadc16=ADC;                                       8

```

The first line is used to switch on the power supply for the circuit connected to the ADC by pulling the PNP transistor base to logic “0”. Then the AD conversion is started by enabling the ADC and setting the ADSC (**A**DC **S**tart **C**onversion) bit. The ADSC stays at logic one as long as a conversion is in progress. When it is complete it returns to zero. The while loop waits for the conversion to finish. After that the ADC and power supply are switched off.

The ADC data can be read by accessing the variable **ADC**. It is stored in the variable **valadc16**, element of the union structure **adcddata**.

4.2.2.3 Arduino

Refer to section 4.2.1.3.

4.2.2.4 Raspberry Pi

Refer to section 4.2.1.4.

5 Results

5.1 Sensor 1/2 example - nRF24L01BmpAdc.c

5.1.1 Settings

To test the code and ensure correct functionality a continuous measurement over 24 hours was performed. The code given in the appendix (See section 7.2) was left unchanged except in the `wl_module.c` file ARC (**A**uto **R**etransmit **C**ounter) was set to 1, ARD (**A**uto **R**etransmit **D**elay) was set to 750 μs and in the `nRF24L01BmpAdc.c` file the identifier *SENDERATE* was initialized with 75 (send every 10 minutes).

The prototype board with the transmitting nRF24L01+ was placed in approximately 4 m distance to the receiver - no walls in between and in direct sight. The antenna (onboard) orientation was not taken into consideration.

Notice: A wireless router using the 2.4 GHz band is located in about 1 m distance to the receiver, which could interfere the measurement.

5.1.2 Results

Table 9 and 10 show the database data entries added during the measurement and can be found in the appendix (See section 7.3).

As the trimmer position was held constant during the measurement, only fluctuations in the last digit of the ADC value occurred (629 to 631).

Moreover, the measurement showed, if the seconds of the table entries are compared (not shown in the tables presented in the appendix), that every 6 hours the data is sent 1 second earlier. Over a day this adds up to 4 seconds time deviation (46 ppm) which is approximately double the value presumed in section 4.1.2.2 (20 ppm).

Temperature and pressure curves are given by figure 41 and 42. These figures are screenshots from the webinterface graph similar to the graph presented in section 4.1.4.3 (Figure 39).

Furthermore, one can see that a few table entries are missing. There are no values transmitted at 05:15, 07:35, 9:30 and a few more. The missing entries add up to 8. From the 149 packets transmitted, a package loss percentage of 5.4 % can be calculated. This is quite surprising as ARC was set to 1, meaning that the package is re-transmitted if the preceding failed, which should reduce package loss.

The assumption was made, that the ARD value was set too low and should be increased, even if the datasheet states that: “ARD=500 μs is long enough for any ACK (ACKnowledge) payload length” (See datasheet [27] - page 32).

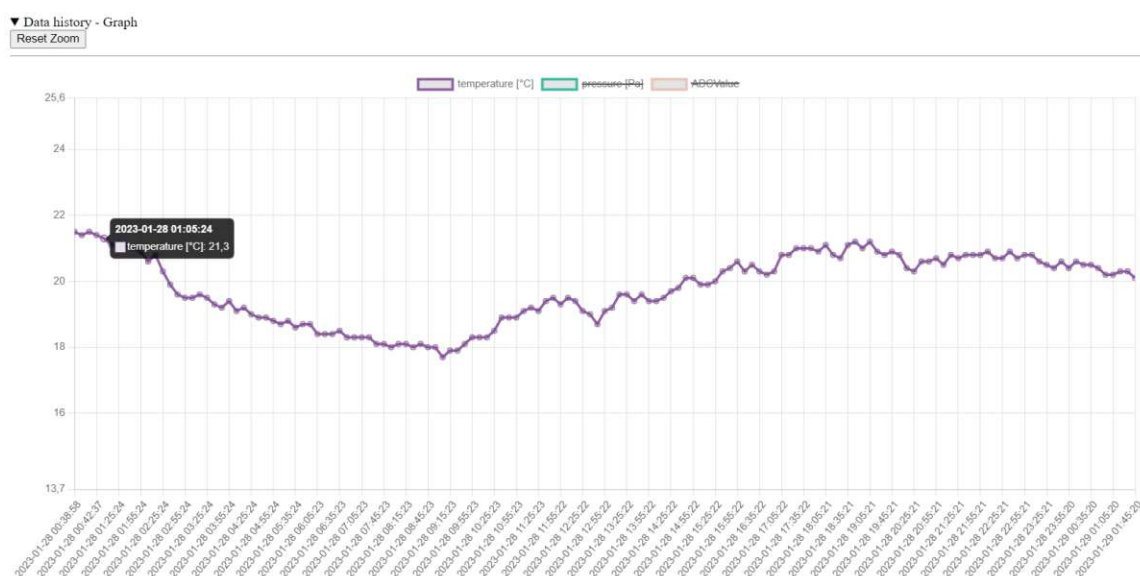


Figure 41: Temperature measurement data

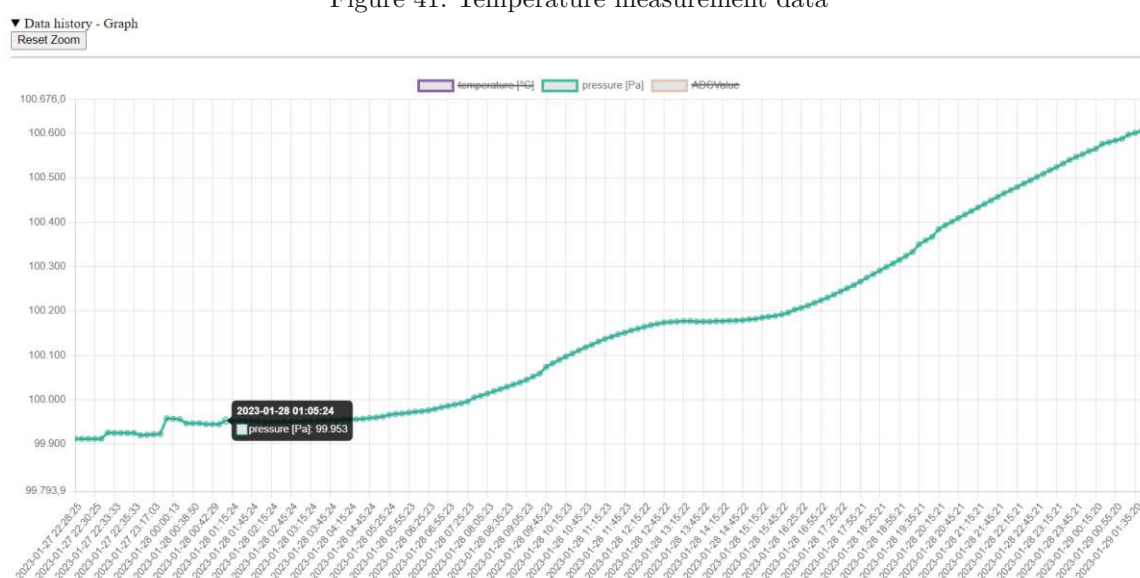


Figure 42: Pressure measurement data

5.1.2.1 ARD & ARC variation

Subsequently, several measurement series were performed with different distances, ARD and ARC values to determine the optimum value for ARD and ARC for reliable data transfers. Table 8 shows the measurement results where N_{sent} stands for the number of packages sent, N_{received} for the packages received and PL for the package loss.

One can see, that at constant held ARD ($750 \mu\text{s}$) variation of ARC has little to no impact on package loss (Table entries 1-5). Only when an ARC value of 5 is reached, the package loss is reduced to 0%.

In the following ARC was held constant at 1 and ARD was varied (Table entries 6-12). The measurement was terminated if a package loss was noticed. The results suggest that an ARD value of about $1750 \mu\text{s}$ or more is required to reduce package loss.

Entry	Distance [m]	ARC	ARD [μs]	N _{sent}	N _{received}	PL [%]
1	1	0	750	100	93	7
2	1	1	750	100	96	4
3	1	2	750	100	94	6
4	1	3	750	100	92	8
5	1	5	750	200	200	0
6	1	1	1500	25	22	12
7	1	1	3000	100	100	0
8	1	1	2750	100	100	0
9	1	1	2250	100	100	0
10	1	1	1750	100	100	0
11	1	1	1500	100	100	0
12	1	1	750	100	97	3
13	6*	1	1500	25	19	24
14	6*	1	2250	100	100	0
15	6*	1	1750	25	20	20
16	6*	1	2000	50	47	6
17	6*	1	2250	100	95	5
	* Through wall					

Table 8: Different ARC and ARD values

To confirm the result the distance between the transmitter and the receiver was increased (brick wall in between) and the ARD value varied (Table entries 13-17). The best result was achieved by setting ARD to 2250 μs .

However, a major drawback of the measurement method is, that to confirm the findings at least one package loss has to occur and it has to be reproducible. Unfortunately, many factors of influence cannot be controlled (e.g., interference radiation, person blocking the signal) so different results are common. Additionally, very important factors influencing the package loss are the data rate (250 Kbps, 1 Mbps, 2 Mbps), distance to the receiver, transmission power and sensor placement.

To achieve the highest reliability ARC and ARD can be set to their respective maximum values. However, this setting drastically increases power consumption, if it is placed in a noisy environment. In the code example provided in the appendix (7.2 - `wl_module.c`) ARD is set to 2250 μs (Maximum is 4000 μs) and ARC is set to its maximum value of 15. To find the best settings for individual purposes, the factors presented have to be considered. Moreover, there are nRF24L01+ modules on the market offering compatibility to external (SMA) antennas, increasing their transmission range. In conclusion it can be said that due to the large uncertainty of this measurement, it is recommended to do further research in this regard.

6 Summary and Outlook

This thesis should give an impression on how easy but yet so complicated it is to develop such an energy efficient, low-cost automation system.

During working on this project countless problems were faced, which needed creative solutions. For example, many parts which were selected beforehand were not available due to shortages (corona pandemic 2021) and therefore different parts had to be used.

Another issue were changes made in the operating system (see section 4.1.4.1) and changes done by the email provider **Gmail**, who changed how the account can be accessed through third party providers. It therefore cannot be excluded that future updates of the operating system or other changes (email provider, libraries) parts of this thesis do not work as described. Software is dynamic and it is not surprising that usually software needs continuous support to ensure functionality.

Furthermore, many problems were faced which were only fixed after researching for a long time. For example, solving the problem described in section 4.2.1.2 (see after **Notice:**) that the TWI register of the ATmega8L has to reset after waking up from power save mode did cost several hours to find and fix.

Particularly the sheer amount of programming languages which had to be learned where another challenge and therefore this thesis is only directed to people who have basic knowledge in coding and hardware development.

To sum up it was a great project combining many different aspects in hardware and software engineering and personally I learned a lot during the period of this thesis.

References

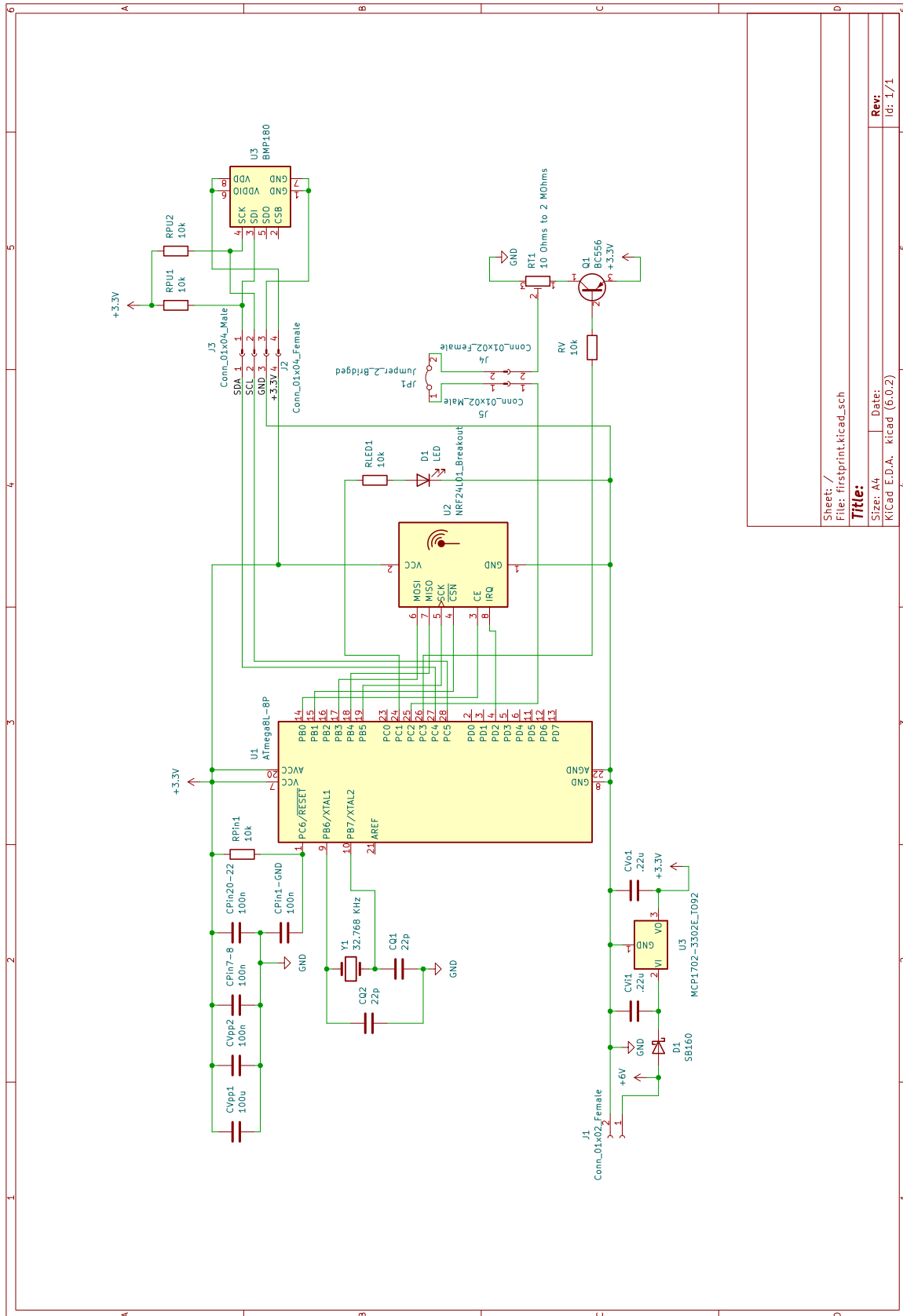
- [1] Whalen SA Markin RS. "Laboratory automation: trajectory, technology, and tactics." In: *Clinical chemistry*, 46(5), 764–771 (2000 May). DOI: <https://doi.org/10.1093/clinchem/46.5.764>.
- [2] Mikell P. Groover. *Fundamentals of Modern Manufacturing*. John Wiley & Sons, Inc., 2010. ISBN: 978-0470-467002.
- [3] Kamminga Johan Cotterell Brian. *Mechanics of pre-industrial technology: An introduction to the mechanics of ancient and traditional material culture*. Cambridge University Press, 1990. DOI: 10.1017/S0003598X00080431.
- [4] Jr. Edward B. Driscoll. *The history of X10*. URL: http://home.planet.nl/~lhendrix/x10_history.htm (visited on 09/04/2022).
- [5] *The difference between a centralized and a decentralized smart home system*. 2018. URL: <https://www.hestiamagazine.eu/the-difference-between-a-centralized-and-a-decentralized-smart-home-system#:~:text=In%20a%20decentralized%20system%20all,the%20remaining%20devices%20still%20function.> (visited on 09/04/2022).
- [6] BehrTech Blog. *Mesh vs. Star Topology – How to Find the Right Architecture for Your IoT Networks*. URL: <https://behrtech.com/blog/mesh-vs-star-topology/> (visited on 09/08/2022).
- [7] "KNX – der weltweit einzige offene Standard für die Haus- und Gebäudesystemtechnik." In: *KNX Deutschland im ZVEI – Zentralverband Elektrotechnik- und Elektronikindustrie e. V.* (2007 October).
- [8] Bernd Aschendorf. *Energiemanagement durch Gebäudeautomation*. Springer, 2014. DOI: <http://dx.doi.org/10.1007/978-3-8348-2032-7>.
- [9] *Local Control Network*. URL: https://de.wikipedia.org/wiki/Local_Control_Network (visited on 09/10/2022).
- [10] Ulrich Klein. *Loxone Smart Home im Test-Überblick – System für Häuser & Gewerbe*. 2020. URL: <https://www.homeandsmart.de/loxone-smart-home> (visited on 09/10/2022).
- [11] Ulrich Klein. *Smart Home Bussysteme im Vergleich und Überblick*. 2020. URL: <https://www.homeandsmart.de/smart-home-bussysteme-vergleich#:~:text=KNX%20ist%20aufgrund%20seiner%20Standardisierung,f%20C3%BCr%20private%20Wohnh%20C3%A4user%20genutzt%20wird.> (visited on 09/10/2022).
- [12] *Loxone oder KNX*. URL: <https://www.1home.io/de/blog/loxone-oder-knx/> (visited on 09/10/2022).
- [13] *digitalSTROM Smart Home – Funktionsweise, Architektur & Tests*. URL: <https://www.homeandsmart.de/digitalstrom-smart-home-hausautomation-intelligenter-strom> (visited on 09/11/2022).
- [14] *Wired vs. Wireless: Which Is Better for Home Automation?* URL: <https://theonetechstop.com/wired-vs-wireless-which-is-better-for-home-automation/> (visited on 09/15/2022).
- [15] Silvia Benetti. *ZigBee, Z-Wave oder WLAN: Der beste Funkstandard*. 2020. URL: <https://www.haus.de/smart-home/funkstandards-im-smart-home-29884> (visited on 09/20/2022).
- [16] Tobias Zillner. "ZIGBEE EXPLOITED - The good, the bad and the ugly." In: *cognosec* (2015 August).
- [17] *EnOcean*. URL: <https://en.wikipedia.org/wiki/EnOcean> (visited on 09/27/2022).

- [18] Ulrich Klein. *Bluetooth LE Smart Home Funkstandard – Wissen, Geräte, Bedeutung*. 2021. URL: <https://www.homeandsmart.de/bluetooth-low-energy-smart-home> (visited on 09/27/2022).
- [19] Florian Jung. *Funkstandards im Smart Home*. 2016. URL: <https://www.smart-home.one/funkstandards-im-smart-home-warum-nicht-wlan-und-bluetooth-201623> (visited on 09/27/2022).
- [20] Kasper Rasmussen Daniele Antonioli Nils Ole Tippenhauer. “BIAS: Bluetooth Impersonation AttackS.” In: *Research* (Sept. 27, 2022). DOI: 10.1109/SP40000.2020.00093. URL: <https://francozappa.github.io/about-bias/publication/antonioli-20-bias/antonioli-20-bias.pdf>.
- [21] Ulrich Klein. *WLAN Funkstandard – alles über WiFi im Smart Home*. 2021. URL: <https://www.homeandsmart.de/bluetooth-low-energy-smart-home> (visited on 10/02/2022).
- [22] *OpenThread*. URL: <https://openthread.io/> (visited on 10/05/2022).
- [23] *Smart Home mit Thread*. URL: <https://www.reichelt.de/magazin/ratgeber/smart-home-mit-thread/> (visited on 10/05/2022).
- [24] *Thread (network protocol)*. URL: [https://en.wikipedia.org/wiki/Thread_\(network_protocol\)](https://en.wikipedia.org/wiki/Thread_(network_protocol)) (visited on 10/05/2022).
- [25] *Apple introduces HomePod mini: A powerful smart speaker with amazing sound*. 2020. URL: <https://www.apple.com/ca/newsroom/2020/10/apple-introduces-homepod-mini-a-powerful-smart-speaker-with-amazing-sound/> (visited on 10/02/2022).
- [26] *nRF5340 DK*. URL: <https://www.nordicsemi.com/Products/Development-hardware/nrf5340-dk#> (visited on 10/05/2022).
- [27] *nRF24L01+ Single Chip 2.4GHz Transceiver - Preliminary Product Specification v1.0*. URL: https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf (visited on 10/12/2021).
- [28] *Informationen über den 1-Wire-Bus, Einsatzbereich, Nutzen, Installation*. URL: <https://shop.elabnet.de/info/1-wire> (visited on 10/07/2022).
- [29] Stan Zurek. *iButton (1-Wire) lock and key*. 2006. URL: https://commons.wikimedia.org/wiki/File:1-Wire_lock.jpg (visited on 10/07/2022).
- [30] *CAN bus*. URL: <https://www.can-cia.org/can-knowledge/can/can-history/> (visited on 10/07/2022).
- [31] *Serial Peripheral Interface*. URL: https://www.mikrocontroller.net/articles/Serial_Peripheral_Interface (visited on 10/08/2022).
- [32] en:User:Cburnett. *A single master and a single slave on a Serial Peripheral Interface (SPI) bus*. 2006. URL: https://commons.wikimedia.org/wiki/File:SPI_single_slave.svg (visited on 10/08/2022).
- [33] *I2C*. URL: <https://en.wikipedia.org/wiki/I%C2%B2C> (visited on 10/10/2022).
- [34] Tim Mathias. *An example I2C schematic with one controller (a microcontroller), three target nodes (an ADC, a DAC, and a microcontroller), and pull-up resistors Rp*. 2021. URL: https://commons.wikimedia.org/wiki/File:I2C_controller-target.svg (visited on 10/10/2022).
- [35] Sam Jebakumar J Veeramani P Vimala Juliet A and Jagadish R. “Design and Fabrication of Temperature Sensor for Weather Monitoring System using MEMS Technology.” In: *Oriental Journal of Chemistry* (8.10.2018). DOI: <http://dx.doi.org/10.13005/ojc/340537>.

- [36] *How Does MEMS Temperature Sensor Works*. URL: <https://econtroldevices.com/how-does-mems-temperature-sensor-works/#:~:text=The%20MEMS%20temperature%20sensors%20are,and%20no%20contact%20temperature%20sensors>. (visited on 10/11/2022).
- [37] *GY-68 BMP180 Barometrischer Sensor Luftdruck Modul für Arduino und Raspbbery Pi Datenblatt*. URL: https://cdn.shopify.com/s/files/1/1509/1638/files/GY-68_BMP180_Barometrischer_Sensor_Luftdruck_Modul_fur_Arduino_und_Raspberry_Pi_Datenblatt.pdf?15836792964504220844 (visited on 10/11/2022).
- [38] *MEMS pressure sensors*. URL: <https://www.avnet.com/wps/portal/abacus/solutions/technologies/sensors/pressure-sensors/core-technologies/mems/> (visited on 10/13/2022).
- [39] *So nutzen Sie die Vorteile der piezoresistiven Druckmesstechnik*. 2020. URL: <https://www.fluid.de/mechatronik/so-nutzen-sie-die-vorteile-der-piezoresistiven-druckmesstechnik-123.html> (visited on 10/13/2022).
- [40] *MEMS Capacitive vs Piezoresistive Pressure Sensors – What are their differences?* 2020. URL: <https://esenssys.com/capacitive-piezoresistive-pressure-sensors-differences/#:~:text=Piezoresistive%20technology%20measurement%20principle,physical%20pressure%20applied%20upon%20them>. (visited on 10/13/2022).
- [41] Rama Komaragiri Suja K J Kumar G S and Nisanth A. “Analysing the effects of temperature and doping concentration in silicon based MEMS piezoresistive pressure sensor.” In: *ScienceDirect* (6-8.09.2016). DOI: 10.1016/j.procs.2016.07.189.
- [42] Dr. Hans-Gerd Kürschner. *Sensors - Systems for the detection of object states and properties*. URL: <https://www.indutrax.net/en/technologien/sensorik/> (visited on 10/13/2022).
- [43] Michael H. („Laserlicht“) / Wikimedia Commons / CC BY-SA 4.0. *Raspberry Pi 4 Model B from the side*. 2019. URL: https://de.wikipedia.org/wiki/Datei:Raspberry_Pi_4_Model_B_-_Side.jpg (visited on 10/13/2022).
- [44] althaus. *Arduino Due (Rev2b)*. 2013. URL: https://paulvollmer.net/FritzingParts/parts/Arduino_DUE_V02b.html (visited on 10/13/2022).
- [45] *ATmega8 - ATmega8L*. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf (visited on 10/12/2021).
- [46] FDominec. *Pin names of the ATMEGA8 microcontroller. Drawn using information from the Atmel datasheet*. 2013. URL: https://de.m.wikibooks.org/wiki/Datei:Atmega8_pinout.svg (visited on 10/13/2022).
- [47] *ARCELI TL866II Plus USB Hochleistungs-EEPROM Flash BIOS Programmierer*. URL: <https://www.amazon.de/ARCELI-TL866II-Hochleistungs-EEPROM-Programmierer-ATMEGA/dp/B07CQQBGVK> (visited on 10/13/2022).
- [48] *SB120 - SB160: 1.0A SCHOTTKY BARRIER RECTIFIER*. URL: <https://www.diodes.com/assets/Datasheets/ds23022.pdf> (visited on 10/12/2022).
- [49] *MCP1702: 250 mA Low Quiescent Current LDO Regulator*. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/22008E.pdf> (visited on 01/16/2022).
- [50] *AVR4100: Selecting and testing 32kHz crystal oscillators for Atmel AVR microcontrollers*. URL: <http://ww1.microchip.com/downloads/en/appnotes/doc8333.pdf> (visited on 10/12/2022).
- [51] Mathias Jensen. *White Paper SWRA349 - Coin cells and peak current draw*. 2010. URL: <https://www.ti.com/lit/wp/swra349/swra349.pdf?ts=1663462237174> (visited on 01/28/2023).

7 Appendix

7.1 Schematic



7.2 Code

7.2.1 ATmega8L

7.2.1.1 MAIN

- nRF24L01main.c
- nRF24L01BmpAdc.c

7.2.1.2 AES

- aes.c
- aes.h
- aes_asm.S

7.2.1.3 BMP085/180

- bmp085.c
- bmp085.h

7.2.1.4 I2C/TWI

- i2cmaster.h
- twimaster.c

7.2.1.5 nRF24L01

- nRF24L01.h
- wl_module.c
- wl_module.h

7.2.1.6 SPI

- spi.c
- spi.h

7.2.2 Arduino

7.2.2.1 MAIN

- arduinocode.c

7.2.2.2 Library - RF24

- nRF24L01.h
- printf.h
- RF24.cpp
- RF24.h
- RF24_config.h

7.2.3 RPi

7.2.3.1 Python

- initialize_DB_Tables.py
- saveTest.py
- checkdaily.py

7.2.3.2 Webserver

- HTML
 - clients.html
 - index.html
 - log1.html
- CSS
 - style/dropdown.css
- PHP
 - php/restart.php
 - php/shutdown.php
 - php/sidebar.php
- JS
 - js/dropdown.js

7.3 Measurement Table

id	Date	T [°C]	P [Pa]	ADC	881	1/28/23 10:05	18.30	100090	630
830	1/28/23 1:05	21.30	99953	631	882	1/28/23 10:15	18.30	100097	630
831	1/28/23 1:15	21.10	99953	630	883	1/28/23 10:25	18.50	100104	630
832	1/28/23 1:25	21.10	99953	630	884	1/28/23 10:35	18.90	100111	630
833	1/28/23 1:35	21.10	99952	630	885	1/28/23 10:45	18.90	100118	629
834	1/28/23 1:45	21.00	99952	630	886	1/28/23 10:55	18.90	100124	629
835	1/28/23 1:55	20.90	99951	630	887	1/28/23 11:05	19.10	100131	630
836	1/28/23 2:05	20.60	99950	630	888	1/28/23 11:15	19.20	100137	630
837	1/28/23 2:15	20.80	99950	630	889	1/28/23 11:25	19.10	100142	629
838	1/28/23 2:25	20.30	99950	630	890	1/28/23 11:35	19.40	100147	630
839	1/28/23 2:35	19.90	99950	631	891	1/28/23 11:45	19.50	100151	629
840	1/28/23 2:45	19.60	99951	630	892	1/28/23 11:55	19.30	100156	630
841	1/28/23 2:55	19.50	99951	630	893	1/28/23 12:05	19.50	100160	630
842	1/28/23 3:05	19.50	99952	629	894	1/28/23 12:15	19.40	100164	630
843	1/28/23 3:15	19.60	99952	630	895	1/28/23 12:25	19.10	100168	629
844	1/28/23 3:25	19.50	99952	629	896	1/28/23 12:35	19.00	100171	630
845	1/28/23 3:35	19.30	99953	630	897	1/28/23 12:45	18.70	100174	630
846	1/28/23 3:45	19.20	99954	630	898	1/28/23 12:55	19.10	100175	630
847	1/28/23 3:55	19.40	99954	630	899	1/28/23 13:05	19.20	100176	629
848	1/28/23 4:05	19.10	99955	630	900	1/28/23 13:15	19.60	100177	630
849	1/28/23 4:15	19.20	99956	630	901	1/28/23 13:25	19.60	100177	630
850	1/28/23 4:25	19.00	99956	631	902	1/28/23 13:35	19.40	100176	630
851	1/28/23 4:35	18.90	99957	630	903	1/28/23 13:45	19.60	100176	630
852	1/28/23 4:45	18.90	99959	630	904	1/28/23 13:55	19.40	100176	630
853	1/28/23 4:55	18.80	99960	630	905	1/28/23 14:05	19.40	100177	629
854	1/28/23 5:05	18.70	99962	630	906	1/28/23 14:15	19.50	100177	630
855	1/28/23 5:25	18.80	99966	631	907	1/28/23 14:25	19.70	100178	630
856	1/28/23 5:35	18.60	99968	630	908	1/28/23 14:35	19.80	100178	630
857	1/28/23 5:45	18.70	99969	630	909	1/28/23 14:45	20.10	100179	630
858	1/28/23 5:55	18.70	99971	631	910	1/28/23 14:55	20.10	100181	629
859	1/28/23 6:05	18.40	99973	630	911	1/28/23 15:05	19.90	100182	630
860	1/28/23 6:15	18.40	99974	630	912	1/28/23 15:15	19.90	100185	630
861	1/28/23 6:25	18.40	99976	630	913	1/28/23 15:25	20.00	100187	630
862	1/28/23 6:35	18.50	99979	630	914	1/28/23 15:35	20.30	100189	630
863	1/28/23 6:45	18.30	99983	630	915	1/28/23 15:45	20.40	100192	630
864	1/28/23 6:55	18.30	99986	631	916	1/28/23 15:55	20.60	100196	630
865	1/28/23 7:05	18.30	99989	630	917	1/28/23 16:15	20.30	100203	630
866	1/28/23 7:15	18.30	99992	630	918	1/28/23 16:25	20.50	100207	630
867	1/28/23 7:25	18.10	99996	630	919	1/28/23 16:35	20.30	100212	629
868	1/28/23 7:45	18.10	100005	630	920	1/28/23 16:45	20.20	100218	630
869	1/28/23 7:55	18.00	100009	630	921	1/28/23 16:55	20.30	100224	630
870	1/28/23 8:05	18.10	100014	629	922	1/28/23 17:05	20.80	100230	631
871	1/28/23 8:15	18.10	100019	630	923	1/28/23 17:15	20.80	100237	630
872	1/28/23 8:25	18.00	100024	630	924	1/28/23 17:25	21.00	100244	630
873	1/28/23 8:35	18.10	100029	629	925	1/28/23 17:35	21.00	100251	631
874	1/28/23 8:45	18.00	100034	630	926	1/28/23 17:45	21.00	100258	630
875	1/28/23 8:55	18.00	100039	630	927	1/28/23 17:55	20.90	100266	630
876	1/28/23 9:05	17.70	100045	630	928	1/28/23 18:05	21.10	100275	631
877	1/28/23 9:15	17.90	100052	629	929	1/28/23 18:15	20.80	100283	630
878	1/28/23 9:25	17.90	100059	630	930	1/28/23 18:25	20.70	100291	630
879	1/28/23 9:45	18.10	100074	630	931	1/28/23 18:35	21.10	100299	630
880	1/28/23 9:55	18.30	100082	630	932	1/28/23 18:45	21.20	100307	630

Table 9: Measurement data table part 1

933	1/28/23 18:55	21.00	100315	630
934	1/28/23 19:05	21.20	100324	630
935	1/28/23 19:15	20.90	100333	631
936	1/28/23 19:35	20.80	100350	630
937	1/28/23 19:45	20.90	100359	630
938	1/28/23 19:55	20.80	100367	630
939	1/28/23 20:15	20.40	100384	630
940	1/28/23 20:25	20.30	100393	631
941	1/28/23 20:35	20.60	100401	630
942	1/28/23 20:45	20.60	100409	630
943	1/28/23 20:55	20.70	100417	630
944	1/28/23 21:05	20.50	100425	630
945	1/28/23 21:15	20.80	100433	630
946	1/28/23 21:25	20.70	100441	630
947	1/28/23 21:35	20.80	100449	630
948	1/28/23 21:45	20.80	100457	630
949	1/28/23 21:55	20.80	100465	630
950	1/28/23 22:05	20.90	100472	630
951	1/28/23 22:15	20.70	100479	630
952	1/28/23 22:25	20.70	100487	629
953	1/28/23 22:35	20.90	100494	630
954	1/28/23 22:45	20.70	100502	630
955	1/28/23 22:55	20.80	100509	630
956	1/28/23 23:05	20.80	100517	630
957	1/28/23 23:15	20.60	100524	630
958	1/28/23 23:25	20.50	100532	629
959	1/28/23 23:35	20.40	100540	631
960	1/28/23 23:45	20.60	100547	630
961	1/28/23 23:55	20.40	100553	630
962	1/29/23 0:05	20.60	100560	630
963	1/29/23 0:15	20.50	100565	630
964	1/29/23 0:35	20.50	100576	630
965	1/29/23 0:45	20.40	100580	630
966	1/29/23 0:55	20.20	100584	630
967	1/29/23 1:05	20.20	100588	630
968	1/29/23 1:25	20.30	100597	630
969	1/29/23 1:35	20.30	100601	631
970	1/29/23 1:45	20.10	100605	631

Table 10: Measurement data table part 2