# Autonomous Temperature Sensor for Smart Agriculture

## Smart Measurement and Control

M.D. Huiskes

M. Miao

TU Delft Delft University of Technology

June 2020

# Autonomous Temperature Sensor for Smart Agriculture

## Smart Measurement and Control

by

**M.D. Huiskes, 4701291**

**M. Miao, 4684206**

Delft University of Technology

Faculty of Electrical Engineering,
Mathematics and Computer Science

In partial fulfilment of the requirements for the degree of

**Bachelor of Science**
In Electrical Engineering

To be defended in public on 30th June 2020 at 10:30 AM

Supervisors:

| | |
|---|---|
| Dr. Q. Fan | TU Delft |
| Ing. R.M.A. van Puffelen | TU Delft |
| MSc. L. Pakula | TU Delft |

# Abstract

Fruit-frost during spring is one of the main causes of damage to the harvest of fruit in orchards. Various systems using different methods of preventing spring-frost are available on the market. To determine when these systems should be activated, the temperature in the orchard needs to be determined.

In this project, a self-sustaining autonomous temperature sensor network is designed, which is capable of making a 3D temperature map of the orchard. The system is used to warn farmers when the threat of fruit-frost occurs and to gather data on the spatial variation of the temperature in the orchard. In this thesis, the focus is put on designing the smart measurement and control system. This includes choosing an appropriate control unit and temperature sensors. Also, the software for the control unit is designed, which allows for a smart measurement scheme that balances energy usage and measurement frequency. Finally, an estimation of the energy usage of the subsystem is given based on theoretical analysis.

# Preface

First of all, we would like to express our gratitude towards our supervisors Ing. R.M.A. van Puffelen, MSc. L. Pakula, and Dr. Q. Fan for their guidance and advice during the project. Also, we would like to thank everyone involved in the organisation of the bachelor graduation project for their flexibility during the COVID-19 pandemic. Last but not least, we would like to thank our group members Arnoud Bleeker, Martijn Hubers, Robin van der Sande and Ruben Wijnands for their continuous support, pleasant collaboration and tremendous effort during the project.

For the past eight weeks, we have been working on the design for the autonomous temperature sensor for smart agriculture project. We chose this project for its real-life applications and societal benefits. Furthermore, we believe that the diverse and multi-disciplinary nature of the project has greatly contributed to our personal development as electrical engineers. Even though the practical aspect of the project was compromised due to the circumstances, it was still very educational and interesting. We hope our efforts translate to a pleasant and informative reading experience.

*Martijn Huiskes*

*Michael Miao*

*June 2020*

# List of abbreviations

| | |
|---|---|
| **ABP** | Activation by personalisation |
| **ADC** | Analog to digital converter |
| **Apps** | Application session |
| **ASIC** | Application-specific integrated circuit |
| **Async** | Asynchronous |
| **CAN** | Controller area network |
| **CAT5** | Category 5 |
| **CISC** | Complex instruction set computer |
| **CLK** | Clock |
| **CPU** | Central processing unit |
| **CS** | Chip select |
| **EM** | Electro-magnetic |
| **FM** | Fast mode |
| **G...** | Group ... |
| **GPIO** | General purpose input/output |
| **I/O** | Input/output |
| **I2C** | Inter-IC-bus |
| **IC** | Integrated circuit |
| **KNMI** | Koninklijk Nederlands meteorologisch instituut |
| **Lkg** | Leakage |
| **LoRa** | Long range |
| **LP** | Low power |
| **LPRun** | Low power run |
| **LSE** | Low speed external |
| **MCU** | Microcontroller |
| **Meas** | Measurement |
| **MISO** | Master in slave out |
| **MOSI** | Master out slave in |
| **MSI** | Multi-speed internal |
| **Nwks** | Network session |
| **Ppm** | Parts per million |
| **PSM** | Power saving mode |
| **RISC** | Reduced instruction set computer |
| **ROI** | Range of interest |
| **RS** | Recommended standard |
| **RTC** | Real Time Clock |
| **RX** | Receive |
| **SCL** | Serial clock |
| **SDA** | Serial data |
| **SF** | Spreading factor |
| **SMAART** | Smart Universal Asynchronous Receiver-Transmitter |
| **SMBus** | System management bus |
| **SPI** | Serial Peripheral Interface |
| **SS** | Slave select |
| **Sync** | Synchronous/synchronisation |
| **Trans** | Transmission |
| **TX** | Transmit |
| **UART** | Universal Asynchronous Receiver-Transmitter |
| **USART** | Universal Synchronous/Asynchronous Receiver-Transmitter |
| **Vcc** | Voltage collector-collector |
| **Vdd** | Voltage drain-drain |
| **Vss** | Voltage source-source |
| **WDT** | Watchdog timer |

# Contents

# Chapter 1

# Introduction

In the Netherlands fruit production is a large segment of the economy. There are over 2600 fruit production companies which together use over 20,000 hectares of ground [1]. Most of these Dutch companies focus on the production of pears and apples. The fruit production in the Netherlands, but effectively all around the world, faces a large problem related to spring frosts in fruit trees. If temperatures drop below the critical temperature, open and blooming flowers of the fruit trees can be damaged causing less fruit to grow. The frost damages are mainly caused by the formation of ice, intracellular ice formation breaks the blossom's tissue structure and causes a cell death [2]. Freeze injury is nowadays the biggest problem of fruit production, which causes a loss far greater than any other type of natural hazard encountered with the production. As a result, the yield of production and distribution of fruits are restricted.

A lot of research has been done in order to obtain a reduction in the losses caused by spring frosts. Two of the main solutions that are provided are frost protection with sprinkler irrigation and frost protection with wind machines. The frost protection with sprinkler irrigation works using extra-cellular ice formation to prevent intracellular ice formation. Sprinkling water onto the tree's flowers and buds causes ice nucleation on the outer surface. This then causes the freezing of the water transporting vessels which protects the flowers due to gradual dehydration [3]. Frost protection with wind machines aims to prevent intracellular ice formation. By using a large wind machine or rotating fan a light wind ($1.5\ m/s$) is created which causes an instantaneous increase in temperature [4]. Because of the fan, a temperature increase is obtained of up to $1\ °C$ at a $15\ m$ distance from the wind machine. Due to this small increase, the flowers are protected from the frost.

## 1.1 The project

So now there are two effective solutions used to prevent spring frosts in fruit trees. Both of these solutions have a requirement to know the temperature of the air surrounding the trees, and their control systems make decisions based on these measurements. But in order to implement these solutions in large scale fruit production companies, an accurate temperature measurement is required over the whole field, rather than one single temperature measurement. Therefore this project focuses on the acquisition of the 3D temperature profile of a fruit orchard in the temperature range near the critical temperature.

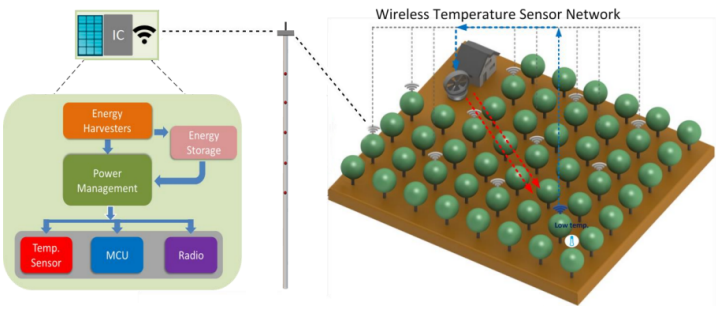


*Figure 1.1: A general overview of the system as a whole.*

*Figure 1.2: This figure gives an impression of how the system is installed in an orchard.*

The great advantage of the acquisition of the temperature profile in contrast to a single temperature measurement is that it provides the ability to perform local frost protection rather than frost protection over the whole field. This in turn provides a reduction in the use of resources such as water and electric energy. In addition, the acquisition of the temperature profile provides a valuable resource for further research on the effectiveness of the frost protection methods.

In Figure 1.1 an overview of the wireless sensor network is shown. This sensor network consists of multiple sensor nodes. A sensor node consists of an energy harvesting module, a control unit, a wireless communication module, and five temperature sensors, which are all integrated onto a pole as shown in Figure 1.2. The poles are positioned among the trees of the fruit orchard, and together they form a smart wireless temperature sensor network in the fruit orchard, as shown in Figure 1.1. In Appendix A.1 the choice of this implementation is explained. The design is split into three subgroups, which each focus on a different part of the system.

## 1.2 Subgroup division

The energy harvesting and control group is responsible for harvesting and storage of energy from ambient sources, and the distribution of energy to all components in the system. The wireless communication subgroup is responsible for communication between the end-devices and the base station. Lastly, the smart measurement and control group is responsible for reading sensor data and controlling the other subsystems, and is the focus of this thesis.

The data acquired from the sensors should be collected and converted such that it is transferable by the wireless communication module and interpretable by the base station. The subsystem is also responsible for managing the power consumption by controlling when certain parts are made inactive to conserve energy. Furthermore, it adapts the frequency of the measurements done based on how close it is to the critical temperature and the available energy in the system.

## 1.3 State of the Art

So to prevent fruit crop losses due to frost, the goal is to design a smart wireless temperature sensor network that is able to warn the farmer when parts of the fruit orchard reach a threshold temperature. Such a network can be classified as an Internet of Things network (IoT). IoT is based on devices that can analyse sensed information and transmit information to the user. These so-called 'smart' devices can be used in various applications such as smart cities, smart metering, security and emergency, industrial control, home automation, healthcare and in this area of research: smart agriculture [5]. Mahmoud et al. [6] state that the IoT is also called the third industrial revolution, which combines multiple sciences and technologies with each other, such as data acquisition, power consumption, and wireless sensor networks. Further they state that it is approximated that near the end of 2020, over 50 billion IoT devices will be connected over the internet. An IoT device consists of four main building blocks, which are the main control Units, sensors, communication modules and power sources [6]. Since this project requires a solution which makes use

of these four main components and is used in agriculture, the device which is designed can be classified under IoT for smart agriculture.

Currently, the use of labour is often needed to measure and monitor the temperature on a farm, and frost protection equipment is activated manually when a critical temperature is measured [7]. Research conducted by Ghaemi et al. [3] shows a successful implementation of automated irrigation systems for protecting blossom on peach and orange trees, where a few thermistors were connected by wires to the tree on a height of $1.5\ m$ above ground level to measure temperature. Pierce et al. [7] implemented an on-farm wireless sensor network, which provides real-time monitoring of the temperature with an accuracy of $0.1\ °C$ where an alarm is triggered if the temperature reaches a predetermined threshold, and is powered by a battery and optionally a solar panel. In the work of Sushanth et al. [8], an IoT solution is proposed which consists of a solar-powered Arduino board, where its temperature, humidity, and soil moisture sensors are connected by wire to this Arduino. If action is required, a message is sent to an Android application.

In the above-described solutions of Ghaemi et al. [3] and Sushanth et al. [8], the sensors are connected by wire to some form of a base station. When many sensors are distributed over a large orchard, in case of an orchard 3D temperature profile, this will result in a lot of wiring. Also, possible maintenance costs will increase when wires get disconnected. Therefore, a wireless sensor network solution is proposed. Furthermore, a solution is proposed which focuses on energy harvesting instead of battery-operated only, since battery-operated solutions go together with maintenance costs, for example, periodic replacement of batteries, and have a limited lifetime. Furthermore, the individual nodes of the above-proposed solutions are not able to work together smartly to form a smart system. These requirements finally lead to the design of a wireless, self-sustainable smart sensor network.

## 1.4 Structure of thesis

First of all, the program of requirements is given in Chapter 2. After that, the choices of hardware for the subsystem are elaborated in Chapter 3. Based on the selection of hardware, suitable software is developed which is explained in Chapter 4. Following this, a breakdown of the power consumed by the subsystem is discussed in Chapter 5. Finally, the thesis is concluded in Chapter 6.

# Chapter 2

# Program of requirements

The goal of the project is to develop an autonomous and self-sustaining system that makes a 3D temperature profile of an orchard and gives a warning when the threat of fruit-frost occurs. To get a clear view of what functionalities the system should have, a list of requirements is made. The requirements are divided into functional and non-functional requirements, which are further subdivided into mandatory and trade-off requirements.

## 2.1 Requirements of the entire system

### 2.1.1 System functional requirements

**Mandatory requirements:**

1. The system must create a 3D temperature map.
2. The system must harvest its energy from ambient sources.
3. The system must communicate wirelessly to a base station.
4. The system must alert the end-user when a programmed critical temperature is reached.
5. The system must store the measured temperature data.
6. The system must be scalable in terms of the number of nodes it can support.
7. The system must be scalable in terms of the number of functions it can perform.

**Trade-off requirements:**

8. The system should preferably support over-the-air firmware updates.
9. The system should preferably have self-diagnosis.
10. The system should preferably have energy-level monitoring.
11. The system should preferably have bi-directional communication.

### 2.1.2 System non-functional requirements

**Mandatory requirements:**

12. The system must be able to function in a 10-hectare field.
13. The system must be able to measure temperatures between $-10$ °$C$ and $10$ °$C$.
14. The system must measure with an accuracy of $\pm 0.5$ °$C$.
15. The system must have nodes with five temperature sensors at different heights.
16. The system must support a maximum of 2.5 meters between sensor and control unit.
17. The system must support one node per $100m^2$.
18. The system must have a lifespan of at least 20 years in normal operating conditions.
19. The system must operate during spring.
20. The system should measure the temperature at each node in the orchard at least once per hour.
21. The component cost of an end-device must be in the order of €100 or lower.

**Trade-off requirements:**

22. The system should preferably have an adaptive temperature measurement frequency.
23. The system should preferably measure the temperature at each node in the orchard at least once per half-hour.

## 2.2 Requirements of the measurement and control subsystem

### 2.2.1 Subsystem functional requirements

**Mandatory requirements:**

24. The subsystem must communicate with the wireless communication module via UART.
25. The subsystem must be able to read an analogue input.
26. The subsystem must be able to write a digital output.
27. The subsystem must support five temperature sensors. (see Appendix A.1)
28. The subsystem must have a configurable alarming-temperature.
29. The subsystem must only transmit during its assigned time-slot [9].

**Trade-off requirements:**

30. The subsystem should preferably have an adaptive temperature measurement frequency based on available energy and temperature.
31. The subsystem should preferably alert the base station within 5 minutes of reaching the user-configured alarming-temperature.

# Chapter 3

# Component selection

To fulfil its task, the Measurement and Control subsystem consists of two main parts. Firstly, temperature sensors are required to gather information from the environment of the node. Five of these sensors are required to gather enough information for an accurate 3D temperature map. The information gathered by these sensors needs to be transmitted to a base station via the wireless communication module, which is a requirement of the system. However, the sensor is unable to properly interface with the wireless communication module, and therefore an intermediary control unit is needed. This control unit is also used to adapt the frequency of the measurements[1] according to the measured temperature and the available energy. The control unit should, therefore, also be able to read out the available energy in the storage system. In this chapter, the hardware to perform these tasks is chosen systematically. First, the choice of the temperature sensor is considered, followed by the choice of the control unit. The chapter is concluded with a selection of optimal components for the project.

## 3.1 Temperature sensor

There are many different temperature sensors available on the market with different specifications. The most important specifications to consider are listed below.

#### Digital or analogue

Both digital or analogue temperature sensors can be used. However, digital sensors have the advantage that they have an ADC built-in and that the output is already a numeric value of a temperature value that is interpretable by the control unit. The output signal of an analogue sensor would first have to be converted into a digital value using an ADC and subsequently into a temperature using a model of the sensor behaviour. Since there is a distance of up to 2 meters that the signal has to travel over, as specified by Requirement 16, noise can be introduced in the signal. A digital implementation is less susceptible to noise and is therefore preferred over an analogue implementation.

#### Temperature measurement range

The temperature sensor should at least be able to measure the temperature in the range of interest (i.e. -10°C to 10°C), as is specified by Requirement 13. However, a wider measurement range is advantageous as it increases the applicability of the system in another climate or other weather situations.

#### Energy consumption

A low power implementation is required as the node needs to be self-sustaining. Therefore, power usage should be as low as possible. The power consumption of the sensor consists of two main parts: the running mode power consumption and standby power consumption. During standby, the sensor consumes less power as it is not measuring the temperature, and various internal components are turned off. The negative temperature change in the Netherlands over the last twenty years was $8.3°C/hour$ at maximum, which is found in Appendix B. For this design, the maximum measurement frequency is chosen to be once every 5 minutes. This ensures that a temperature change of 1 degree can be detected. Consequently, the sensor can spend most of its time in standby mode, thereby limiting its power consumption. This is quantified in Chapter 5. For this reason, the focus is on sleep-mode power consumption rather than

[1] In this chapter, and the remainder of the thesis, one measurement is defined as a collection of sensor readouts of the 5 sensors of a node at a certain time.

running-mode power consumption. An estimation of the power consumption at a given measurement frequency can be made using the standby current, active current and conversion time at a certain operating voltage. This is explained in further detail in Section 3.3.

#### Accuracy

The measured temperature should not deviate more than 0.5°C from the actual temperature, as specified by Requirement 14. The accuracy of the measurement consists of two factors: the round off error and the uncertainty of the measurement. The total deviation can be characterised by Equation 3.1.

$$Accuracy = E_{round-off} + E_{uncertainty} = \frac{resolution}{2} + uncertainty \leq 0.5°C \tag{3.1}$$

Here $E$ denotes the error in °C. The resolution is dependent on the temperature range of the sensor and the number of bits that it uses to represent the data. This accuracy is only required in the temperature range of interest (-10°C to 10°C), as specified by Requirement 13.

#### Error checking

To ensure reliable transfer of temperature measurements from the sensors, error checking is desirable. Especially since the design requires relatively long wires, which may introduce significant amounts of noise. Also, interference from EM radiation from for example the wireless communication module, power supply or external sources might deteriorate the signal quality. The most basic implementation of error checking is through the use of a parity bit, which is built into some serial communication protocols. Other more reliable techniques exist such as checksum computations and cyclic redundancy checks (CRC). Some sensors have these error checking techniques built-in. Other forms of error checking can also be implemented by for example looking at the transmitted data. If the data deviates too much from the expected values, a new measurement can be issued.

#### Cost

The cost of the components for the subsystem should be minimised such that Requirement 21 is satisfied.

#### Compatibility / Communication protocol

In case a digital sensor is used, the communication protocol that the sensor uses should ideally be supported by the control unit so no additional conversion has to take place. Off-the-shelf digital sensors usually interface with data transfer protocols such as UART (Universal Asynchronous Receiver-Transmitter), SPI (Serial Peripheral Interface), I²C (Inter-IC) or SMBus (System Management Bus). The latter is based on and generally compatible with I²C but supports higher data-rates. Other protocols exist, such as OneWire and CAN (Controller Area Network), but these are generally not used by off-the-shelf temperature sensors. Most temperature sensors make use of UART, I²C or SPI. These three communication protocols are discussed in more detail.

## 3.2 Serial communication protocols

A comparison of the properties and limitations of the three most common communication protocols can be found in Table 3.1. The importance of the specifications is considered, and an optimal communication protocol is chosen.

*Table 3.1: Comparison of different serial communication techniques [10] [11] [12] [13] [14].*

| | **UART/USART** | **I²C** | **SPI** |
|---|---|---|---|
| **Type** | Async/Sync | Async/Sync | Sync |
| **Typical data rate** | 9.6kb/s - 256kb/s | 100kb/s - 400kb/s (FM I²C) | 5 Mb/s - 20 Mb/s[2] |
| **Maximum slaves** | 1 | 1008 | Limited to number of pins |
| **Multiple masters** | No | Yes | No |
| **Error checking** | Yes (parity bit) | No | No |
| **Data frame size** | 5-8 bits | 8 bits | N/A |
| **Number of wires** | $2n$ | 2 | $3+n$ |
| **Duplexity** | Full-duplex | Half-duplex | Full-duplex |
| **Ack. scheme** | No | Yes | No |
| **Topology** | Star | Bus | Star |

Here (a)sync means (a)synchronous and FM I²C means Fast-Mode I²C, which runs at 400kb/s instead of the 100kb/s in normal mode. The maximum number of slave devices on an I²C bus depends on the number of bits used for addressing (7 or 10 bits). For the calculation of the required number of wires, $n$ is the number of sensors in the design. Half-duplex means that there can be bidirectional communication, but information can only be sent in one direction at a given time. Full duplex means that there can be simultaneous bidirectional communication.

For this design, some parameters are not of importance. The type, number of masters, data frame size and duplexity are therefore not included in the decision matrix. The focus of the design choice is on the topology and power consumption of the different communication techniques.

### 3.2.1 Topology

The topology of the different communication techniques determines the number of wires and interfaces that are needed between the control unit and temperature sensors. A bus topology is preferred over a star topology in terms of practicality, as fewer cables and connectors are needed. UART uses separate data lines for each of the sensors, and for each sensor, there is a receive and a transmit line. This means that 10 lines need to be drawn to support the 5 sensors. I²C uses 1 data-line which is used for both transmission and reception (SDA) along with a clock line (SCL) controlled by the master device. These are shared among the 5 sensors, meaning that only 2 wires are needed in total. SPI uses 1 data bus for communication from master-to-slave (MOSI) and a separate bus for slave-to-master communication (MISO). Furthermore, SPI requires a single clock signal line (clk) and multiple slave/chip select (SS) lines. The different topologies are illustrated in Figure 3.1, where Rx and Tx stand for receiver and transmitter respectively, SDA for serial data and SCL serial clock. MOSI and MISO stand for master-out-slave-in and master-in-slave-out, and SS for slave-select.

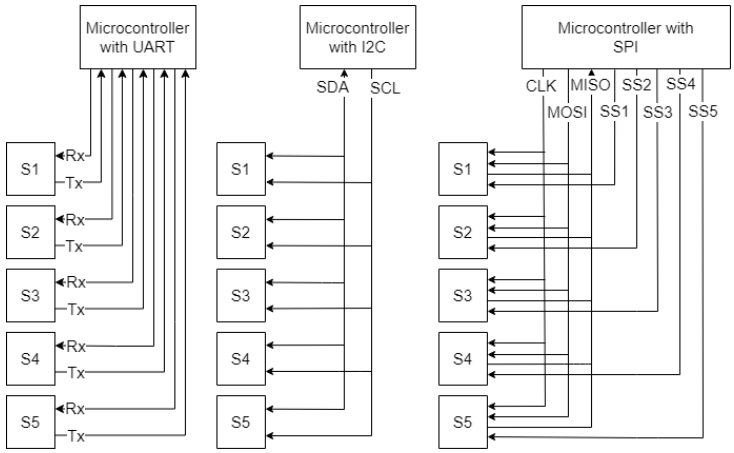


*Figure 3.1: Comparison of topologies of different communication protocols. Only data paths are shown.*

[2] In theory the speed of SPI can go higher than the specified value, but for the SPI temperature sensors no values higher than 20 Mb/s were found.

### 3.2.2 Energy consumption estimation

For the energy consumption estimates done in this section, the wire capacitance of a CAT5 cable can be used, which is typically around $56\ pF/m$ [15]. Also, 4-way ribbon cables could be used, which have a lower typical capacitance of $50\ pF/m$ [10], but are more susceptible to noise. As a reference, the capacitance of a CAT5 cable is therefore used to make the energy consumption estimation for the different serial protocols. The largest distance that needs to be covered by the cable is between the control unit and the lowest positioned sensor, which is 2.5 meters as indicated by Requirement 16. Temperature sensors and control units typically have between $5\ pF$ and $10\ pF$[3] capacitance on their input/output pins (I/O pins). The dynamic energy consumption of different protocols per transmission per sensor for different payload sizes are shown in Table 3.2. What should be kept in mind is that the static energy consumption will be different for each protocol depending on the control unit and temperature sensors, which will be considered when choosing the components.

#### UART

UART operates using 10 or 11-bit data frames containing 1 byte of data, a start-bit, stop-bit, and an optional parity bit. For the calculations, the configuration with parity bit is chosen. The length of the cables varies per sensor, ranging from 50 cm to 250 cm, depending on at what height the sensor is placed. This leads to an average cable length of 1.5 meters, which has a capacitance of $C_{cable} = 84\ pF$.
The average dynamic energy consumed $E_{dyn,UART}$ by 1 transmission to and from 1 sensor can be estimated using Equation 3.2.

$$E_{dyn,UART} = (C_{cable} + C_{MCU} + C_{sensor}) * V_{dd}^2 * n_{trans} \tag{3.2}$$

Here $C_{MCU}$ is the capacitance of the MCU serial interface pin, $C_{sensor}$ is the capacitance of the sensor serial interface, $V_{dd}$ the supply voltage and $n_{trans}$ is the number of signal digital logic transitions on the line.

The value of $n_{trans}$ depends on the contents of the payload and the size. For a power estimation, a scenario is considered where every consecutive bit has a 50% chance of being a transition, meaning that the number of transitions equals half the number of bits transmitted.

#### SPI

SPI does not work with data frames, but with continuous data transmission, which has the advantage that no stop and start bits are needed. It does however use a clock line, which continuously makes transitions between 0 and 1. For the power calculation, it is assumed that every consecutive bit has a 50% chance of being a transition. This means that the data line has on average 4 transitions per data byte. The clock line has 8 transitions per byte. Furthermore, the slave select has 1 transition per message. Equation 3.2 is used to calculate the dynamic energy consumption. Here the total sensor capacitance $C_{sensor}$ should be 5 times the capacitance of an individual sensor because multiple sensors are connected to the same bus.

#### I²C

I²C uses an open-drain configuration, which means that a pull-up resistor is needed. The value of this resistor depends on the rise time specified for I²C, the bus capacitance of the cable that is driven and the voltage parameters of the sensor/MCU. The minimum value of the resistor can be calculated with Equation 3.3, where $V_{dd}$ stands for the supply voltage, $V_{ol}$ for the voltage for which a sensor interprets the voltage as a logic 0, and $I_{ol}$ for the current at low-level [16].

$$R_{min} = \frac{V_{dd} - V_{ol}}{I_{ol}} \tag{3.3}$$

The values for $V_{ol}$ and $I_{ol}$ vary per sensor and MCU, and are found to be typically around $0.4\ V$ and $3\ mA$ respectively for a value[4] of $1.8\ V$ for $V_{dd}$. This observation is made by looking at the datasheets of sensors and control units discussed in Section 3.2.3 and Section 3.6. This results in a minimum resistor value of $467\ \Omega$.

[3] This estimation for the I/O capacitance has been found by looking at the datasheets of the components that are selected in Section 3.5 and Section 3.6.

The maximum resistor value $R_{max}$ can be calculated using Equation 3.4 [16].

$$R_{max} = \frac{t_r}{0.8473 * C_{bus}} \tag{3.4}$$

Here $t_r = 1\mu s$ is the rise time defined for I²C running in normal mode, which supports clock rates up to $100\ kHz$ [16]. This results in a maximum resistor value of $5901\Omega$.

The power consumed by the resistor when the line is pulled low can be found using Equation 3.5.

$$P = \frac{V_{dd}^2}{R} \tag{3.5}$$

This consumption is sustained over the period where the line is transmitting a logic 0. This time $t$ the line is draining current through the pull-up resistor is determined by dividing the number of 0-transmissions ($n_0$) by the data rate. Here it is assumed that half of the transmitted bits is a 0. The energy consumed per transmission can then be calculated by Equation 3.6.

$$E = P * t = P * \frac{n_0}{f} \tag{3.6}$$

As no real-life measurement is performed and different pull-up resistor values cannot be tested physically, $R_{min} = 467\ \Omega$ is chosen as pull-up resistor value when using I²C. This is done as the smallest resistor value yields the most stable signal. This is also the implementation with the highest power consumption, and in the power calculation therefore a worst-case scenario is assumed.

#### Overview

An overview of the energy consumed per transmission by each of the protocols is provided in Table 3.2. The power consumed by I²C greatly depends on what resistor value is chosen. Ideally, to minimise the power consumption, a large resistor is desirable. However, this leads to a less stable and less reliable signal. In practice, a value between the minimum and maximum should be chosen, which minimises the power consumption while maintaining stable communication.

*Table 3.2: Dynamic energy consumption of different serial communication protocols.*

| **Energy consumption for:** | **UART** | **I²C** | | **SPI** |
|---|---|---|---|---|
| | | $R_{max}$ | $R_{min}$ | |
| **1 data byte [nJ]** | 2.0 | 120.8 | 1527.4 | 8.4 |
| **10 data bytes [nJ]** | 20.2 | 614.9 | 7776.0 | 78.4 |
| **100 data bytes [nJ]** | 202.2 | 5556.4 | 70262.6 | 778.2 |

### 3.2.3 Choice of communication protocol

Some parameters are not important for this design and are therefore omitted from the decision matrix. Data rate affects the amount of time the transmission takes. During this time the control unit and the sensor should be active, meaning that at lower data rates both the control unit and sensor are active for a longer period of time, and therefore consume more energy. However, the frequency of operation of the control unit is expected to be below the maximum data rates of the listed protocols. Therefore, the maximum data rate is not important as the data rate is limited by the operating frequency of the control unit. Error checking is not considered, as it is something that would be beneficial if implemented already, but it is not required. The UART protocol implements optional error checking with a parity bit, which — while it could be useful — is far from ideal. Also duplexity is not considered, as simultaneous bi-directional communication is not required.
In Table 3.3 the decision matrix is shown comparing the features of the different communication protocols. The criteria are given a weight from 1 to 10 depending on their importance. The different serial communication protocols are then given a score from 1 to 10 for each of the criteria. The final weighted score then gives an indication of what option is preferred, where a high score is better.

[4]The $V_{dd} = 1.8\ V$ is a typical minimum value for the operating voltage of temperature sensors

*Table 3.3: Decision matrix for the communication protocol*

| Criterion | Weight | UART | | I²C | | SPI | |
|---|---|---|---|---|---|---|---|
| | | Score | Weighted | Score | Weighted | Score | Weighted |
| **Dynamic power consumption** | 8 | 8 | 64 | 5 | 40 | 7 | 56 |
| **Topology** | 6 | 5 | 30 | 8 | 48 | 6 | 36 |
| | | | 94 | | 88 | | 92 |

Dynamic power consumption has been given a high weight because it contributes to the total power consumption of the system. As the system should be self-sustaining the power consumption should be as low as possible. The topology has been given a lower weight as it mostly influences the practicality of the system, but is less essential than power consumption. As can be seen, the final scores are close to each other, and, therefore, sensors with UART, I²C and SPI are all considered.

## 3.3 Choice of temperature sensor

For this design, only digital temperature sensors are considered as they are less susceptible to noise. Another constraint is that it should operate with an accuracy of $\pm 0.5°\ C$ within the temperature range of interest specified by Requirement 13. As a low-power implementation is crucial for this system, the focus is on low power temperature sensors. From the available options, a decision is made based on power consumption, accuracy, interfaces, error checking capabilities and cost. In Table 3.4, multiple low power options are compared.

In this table, the average current is calculated in the situation where the sensor does 1 measurement every 5 minutes. Here the assumption is made that the average current consumed by the sensor linearly scales with the number of measurements done within a certain time frame. The value for 1 measurement per 5 minutes is found by linearly interpolating between the standby current and the average current at a certain frequency listed in the datasheet (varies per sensor), as is illustrated in Figure 3.2. Here 'A' stands for the number of conversions per second when 1 measurement is done per 5 minutes, and 'B' stands for the conversion frequency for which the average current consumption is listed in the datasheet. Here conversion is defined as the process of acquiring analogue data and converting it to a digital output. This calculation is necessary as not all datasheets mention the power consumption for the same frequency of measurements.

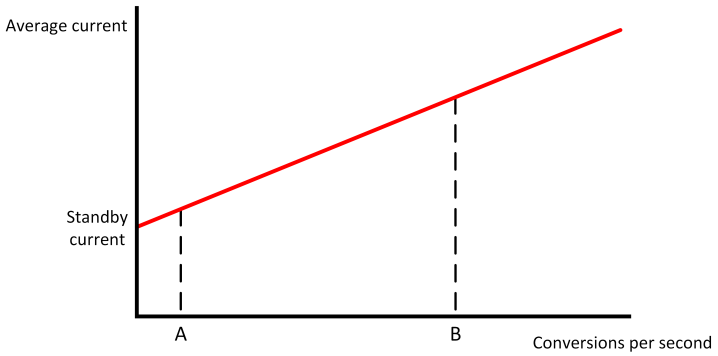


*Figure 3.2: Illustrative graph showing the way the average current for a temperature sensor is found for one measurement every 5 minutes. (Not on scale)*

The number between brackets in the interface fields indicate the number of I²C addresses that can be set. On an I²C bus, each sensor should have a unique address, so here the amount of addresses determines the number of required interfaces on the control unit. Furthermore, ROI means the range of interest. The prices are based on the largest purchase quantity ($\leq$ 10000) listed on Farnell (STM, TI and Microchip Inc.), Digi-Key (AMS and Max) and Mouser (Tyco Electronics). It should be noted that out of the selection of sensors that meet the requirements of the system, both UART and SPI are only supported by 1 single sensor family.

*Table 3.4: Specifications of selected temperature sensors.*

| **Model** | **STTS22HTR** [17] | **AS6212** [18] | **TMP102AIDRLT** [19] |
|---|---|---|---|
| **Manufacturer** | STM | AMS | TI |
| **Temperature range [°C]** | -40 to 125 | -40 to 125 | -40 to 125 |
| **Resolution** | 16 bits | 16 bits | 12 bits |
| **Accuracy in ROI typical [°C]** | 0.25 | $\leq$ 0.2 [5] | 0.5 |
| **Accuracy in ROI max [°C]** | 0.5 | 0.2 | 2.0 |
| **Error checking** | N/A | N/A | N/A |
| **Interface** | I²C (2) | I²C (8) | I²C (4) |
| **Voltage range [V]** | 1.5 - 3.6 | 1.71 - 3.6 | 1.4 - 3.6 |
| **Standby current (max) [µA]** | 0.50 | 0.10 | 0.50 |
| **Active current (max) [µA]** | 180 | - [6] | - [7] |
| **Average current [µA]** | 0.51 | 0.10 | 0.52 |
| **Bulk price per unit** | €0.588 | €1.125 | €0.717 |

| **Model** | **TMP107BIDR** [20] | **TSYS02S** [21] | **MAX31723** [22] |
|---|---|---|---|
| **Manufacturer** | TI | Tyco Electronics | Maxim integrated |
| **Temperature range [°C]** | -40 to 125 | -40 to 125 | -55 to 125 |
| **Resolution** | 14 bits | 16 bits | 12 bits |
| **Accuracy in ROI typical [°C]** | 0.125 | 0.5 | 0.3 |
| **Accuracy in ROI max [°C]** | 0.4 | 0.5 | 2 |
| **Error checking** | N/A | CRC | N/A |
| **Interface** | SMAART/UART[8] | I²C (1) | SPI |
| **Voltage range [V]** | 1.7 - 5.5 | 1.5 - 3.6 | 1.7 - 3.7 |
| **Standby current (max) [µA]** | 3.8 | 0.14 | 2.00 |
| **Active current (max) [µA]** | 300 | 420 | 1150 |
| **Average current [µA]** | 3.84 | 0.20 | 2.77 [9] |
| **Bulk price per unit** | €1.41 | €1.03 | €1.80 |

The resolution of the sensors in terms of the temperature of a sensor with an n-bit resolution can be calculated by Equation 3.7.

$$Resolution = \frac{1}{2^{n-8}} \tag{3.7}$$

Here 8 is subtracted from n because 8 bits are needed to represent the integer range from -40 to 125, and the remaining bits are used for the numbers after the decimal point. This results in a resolution of 0.0625 °C and 0.00391 °C for 12 and 16 bits respectively. Some sensors also provide the option to decrease the resolution to conserve energy.

To choose the most suitable temperature sensor, a decision matrix is made in which the specifications are evaluated and weighted. Here the same scoring system is used as before in Table 3.3. The total weighted score that results for each of the sensors then indicates which sensor should be used for the design.

[5] The typical accuracy was not listed in the datasheet, but this should at least be better than the maximum-confidence accuracy.
[6] The active current was not listed in the datasheet and therefore no comparison can be made.
[7] See Footnote 6
[8] SMAART is a proprietary technology by Texas Instruments that is compatible with the UART interface.
[9] This value is calculated using the active current and conversion time.

*Table 3.5: Decision matrix for the temperature sensor*

| Criterion | Weight | **STTS22HTR** ST-Microelectronics | | **AS6212** AMS | | **TMP102AIDRLT** Texas Instruments | |
|---|---|---|---|---|---|---|---|
| | | **Score** | **Weighted** | **Score** | **Weighted** | **Score** | **Weighted** |
| **Typical accuracy in ROI** | 5 | 8 | 40 | 8 | 40 | 6 | 30 |
| **Error checking** | 7 | 6 | 42 | 6 | 42 | 6 | 42 |
| **Interface** [10] | 5 | 6 | 30 | 8 | 40 | 6 | 30 |
| **Average current** [11] | 9 | 6 | 54 | 9 | 81 | 6 | 54 |
| **Bulk price per unit** | 5 | 8 | 40 | 7 | 35 | 8 | 40 |
| | | | 206 | | 238 | | 196 |

| Criterion | Weight | **TMP107BIDR** Texas Instruments | | **TSYS02S** Tyco Electronics | | **MAX31723** Maxim Integrated | |
|---|---|---|---|---|---|---|---|
| | | **Score** | **Weighted** | **Score** | **Weighted** | **Score** | **Weighted** |
| **Typical accuracy in ROI** | 5 | 9 | 45 | 6 | 30 | 7 | 35 |
| **Error checking** | 7 | 6 | 42 | 9 | 63 | 6 | 42 |
| **Interface** [12] | 5 | 8 | 40 | 6 | 30 | 8 | 40 |
| **Average current** [13] | 9 | 4 | 36 | 8 | 72 | 4 | 36 |
| **Bulk price per unit** | 5 | 6 | 30 | 7 | 35 | 6 | 30 |
| | | | 193 | | 230 | | 183 |

The typical accuracy in the temperature range of interest has been given a weight of 5 as a hard constraint was already put on this value, meaning that all sensors have a typical accuracy of $\pm 0.5$ °$C$ in the temperature range of interest. Any additional accuracy is preferred, but not necessary. Error checking is given a weight of 7 as it improves the reliability of the data transfer between the sensors and control unit, which is desirable in case noise is present. The interface has been a low weight as there is no clear preference in the type of interface (as seen in Table 3.3). However, some I²C sensors support less than 5 separate addresses on one bus, meaning that multiple busses have to be used. This means that the advantage of the simple topology offered by I²C is diminished. Average current is the largest contributor in the score as limiting power consumption is important due to the self-sustaining nature of the system.

Based on the total scores presented in Table 3.5, the AS6212 comes out as the preferred choice, with the TSYS02S as a good alternative.

## 3.4 Controller type

Controllers can be implemented on computers, commercially available off-the-shelf microcontrollers or custom-fabricated chips. Implementation of a computer-based system can easily be dismissed for the following reasons. Computers are designed for general use purposes, and therefore not optimised for a limited number of application-specific operations, which leaves a lot of unnecessary overhead. This generally comes at the cost of increased power usage, speed, physical dimensions, and price. Application-specific integrated circuit (ASIC) chips are specifically designed for the task at hand, but require significant financial investment and add more complexity to the design. The cost is only justified for large scale production of products due to the fixed cost that comes with production [23]. Therefore custom fabricated chips are not suitable for the design of a prototype and low-volume products.

The commercially available microcontrollers have a low power consumption relative to computers and are cheaper. They can often provide the necessary computing power for many embedded applications under low power conditions. Compared to custom-fabricated chips, microcontrollers do not require a huge investment and are more readily available. Therefore microcontrollers are the most compelling option, especially for a prototype design. For large scale production, a custom fabricated chip might be a more interesting option. However, for this research, a microcontroller implementation will be explored.

[10] Here the scores are based on the addresses supported by the I²C interfaces of the sensors.
[11] Average current usage for 1 measurement per 5 minutes.
[12] See Footnote 10.
[13] See Footnote 11.

## 3.5 Microcontroller specifications

Many off-the-shelf microcontrollers are available, and an appropriate microcontroller has to be chosen which fits the requirements of this project. The most important aspects to consider when comparing different microcontrollers are listed below.

#### Number of I/O ports

There should be at least 2 GPIO pins present on the chosen microcontroller to interface with the energy-level monitoring circuit.

#### Power usage & management

When quantifying the power usage of a microcontroller, different power categories need to be taken into account.

First and foremost, standby power is the power used while the microcontroller is waiting for an internal or external event to wake up the CPU to process data, make decisions and communicate with other system components [24]. For many low power sensing applications, the microcontroller spends the majority of its time in this state [25], often making it the largest factor in energy consumption. This is verified in Section 3.6.

The device can be periodically woken-up from the low-power operating mode using a Real-time clock (RTC) timer, which keeps track of the date and time on the device. It is therefore vital that the RTC is kept active during this low-power mode. Secondly, the peripheral power is the power used by peripherals such as an ADC, comparator and digital interfaces of the microcontroller with other components. Finally, active power is the power that is consumed while the microprocessor is actively performing tasks.

#### Clock frequency

A higher clock frequency results in a faster and more responsive system, as the clock frequency determines how many instructions are executed in a certain amount of time. A high clock speed is preferred and can be crucial in high-speed applications. However, the contribution of the clock speed to the on-time of the microcontroller is marginal since that the microcontroller software is relatively simple and few operations are done. Therefore clock speed is not a priority for this design. As high clock speed and power are inherently related, the choice of high clock speed should not come at a detriment to the power consumption.

#### Operation voltage

To minimise power consumption, a low operation voltage is desirable. Furthermore, the operation voltage of the microcontroller should ideally match with that of the other components attached to it, so that no voltage conversions are needed to supply the different components. It should be noted that lowering the operation voltage puts a constraint on the maximum clock speed of the microcontroller.

#### Memory size

The microcontroller has memory modules for instructions and data. The instruction memory should be large enough for the program to fit on the microcontroller. The size of the data memory module mainly depends on the accuracy of the measurements, and the amount of data that needs to be sent every transmission. MCUs with the same specifications can often be bought with various memory configurations, which only affects the rest of the MCU in terms of price.

#### Ports and Interfaces

To properly interface with external components like the sensors and communication module, the chosen microcontroller must have enough serial communication interfaces. At least one I²C interface is needed for the communication with the temperature sensors, and one UART interface is needed for communication with the wireless communication module. It should be noted that some microcontrollers have pins that can be programmed to interface using either I²C or UART, but not both at the same time.

**Cost**

The cost of the components must be minimised such that the total cost of the system does not exceed the requirement specified by Requirement 21.

## 3.6 Choice of microcontroller

For this design, microcontrollers from low-power families of several well-established brands are compared. Furthermore, the microcontroller should have at least one I²C interface, one UART interface and an RTC. In Table 3.6 the specifications of 6 microcontrollers are compared.

*Table 3.6: Specifications of selected microcontrollers*

| **Model** | **SI1060-A-GM** [26] | **EFM32ZG** [27] | **ATmega328PB** [28] |
|---|---|---|---|
| **Manufacturer** | Silicon Laboratories | Silicon Laboratories | Microchip Technology Inc. |
| **Family** | Si106x | EFM32 | ATmega32 |
| **Core size [bits]** | 32 | 32 | 8 |
| **Architecture** | RISC | RISC | AVR® enhanced RISC |
| **Max clock speed** | 25 MHz | 24 MHz | 20 MHz |
| **Operating voltage [V]** | 1.8-3.6 | 1.98-3.8 | 1.8-5.5 |
| **Running current [µA/MHz]** | 160 | 114 | - [16] |
| **Low power current [µA]** [14] | 0.6 | 0.9 | 1.3 |
| **Bulk price per unit [€]** | 3.51 | 0.905 | 1.04 |

| **Model** | **STM8L152K8Y6TR** [29] | **STM32L431KBU6TR** [30] | **STM32L151CBT6TR** [31] |
|---|---|---|---|
| **Manufacturer** | ST-Microelectronics | ST-Microelectronics | ST-Microelectronics |
| **Family** | STM8L | STM32L4 | STM32L1 |
| **Core size [bits]** | 8 | 32 | 32 |
| **Architecture** | CISC | RISC | RISC |
| **Max clock speed** | 16 MHz | 80 MHz | 32MHz |
| **Operating voltage [V]** | 1.8-3.6 | 1.71-3.6 | 1.65-3.6 |
| **Running current [µA/MHz]** | 200 | 84 | 214 |
| **Low power current [µA]** [15] | 1.4 | 1.4 | 1.2 |
| **Bulk price per unit [€]** | 0.97 | 1.85 | 1.81 |

A few of these factors are unimportant to the design. The clock speed is left out because speed is not crucial for this application, and the microcontroller spends most of its time in a low-power mode. The smallest ratio between running current and low power current of the selected MCUs is 60 for the STM32L431. At a clock frequency of 1 MHz and a measurement frequency of once every 5 minutes, this MCU would have to be active for 5 seconds per measurement to have the same active energy consumption as standby energy consumption. This translates to 5 million clock cycles per measurement for the active current to have the same contribution to energy consumption as the low power current. This is assumed to be far beyond the number of operations needed for this design. Furthermore, the running current is not considered due to its minimal contribution to the total power consumption. All other factors are weighed and scored in Table 3.7 in a similar way as with the other two decision matrices in Table 3.3 and Table 3.5.

[14] See Footnote 16.

[15] The running current was not listed in µA/MHz. The running current at 1 MHz is 240 µA

[16] The current of the lowest power mode that supports the RTC and that continues code execution after wake up.

*Table 3.7: Decision matrix for the microcontroller*

| Criterion | Weight | **SI1060-A-GM** Silicon Labs | | **EFM32ZG** Silicon Labs | | **ATmega328PB** Microchip Technology Inc. | |
|---|---|---|---|---|---|---|---|
| | | **Score** | **Weighted** | **Score** | **Weighted** | **Score** | **Weighted** |
| **Minimum operating voltage** | 8 | 6 | 48 | 5 | 40 | 6 | 48 |
| **Low power current** | 8 | 9 | 72 | 7 | 56 | 5 | 40 |
| **Bulk price per unit** | 5 | 4 | 20 | 7 | 35 | 7 | 35 |
| | | | 140 | | 131 | | 123 |

| Criterion | Weight | **STM8L152K8Y6TR** ST-Microelectronics | | **STM32L431KBU6TR** ST-Microelectronics | | **STM32L151CBT6TR** ST-Microelectronics | |
|---|---|---|---|---|---|---|---|
| | | **Score** | **Weighted** | **Score** | **Weighted** | **Score** | **Weighted** |
| **Minimum operating voltage** | 8 | 6 | 48 | 7 | 56 | 8 | 64 |
| **Low power current** | 8 | 5 | 40 | 5 | 40 | 6 | 48 |
| **Bulk price per unit** | 5 | 7 | 35 | 6 | 30 | 6 | 30 |
| | | | 123 | | 126 | | 142 |

Both the minimum operating voltage and the low power current are given a high weight of 8, as they both contribute to the power consumption of the system. Cost is given a lower weight as it is less important than power consumption.

From this decision matrix, it can be concluded that the STM32L151CBT6TR is the best-suited option for this design, with the SI1060-A-GM and EFM32ZG as good alternatives.

## 3.7 General remarks for chosen components

**Microcontroller - STM32L151CBT6TR**

The chosen STM32L151CBT6TR microcontroller features multiple low power modes [31, Chapter 3.1]. The lowest-power running mode is the low-power-run-mode (LPRun), with the multi-speed internal RC oscillator (MSI) set to $65.536\ kHz$ as clock source. To minimise the power consumption, this clock source is pre-scaled by 0.5, such that $f_{HCLK} = 32.768\ kHz$, which is the clock speed used by the MCU core and memory. This clock frequency is also from which the peripheral clock is derived. Furthermore, by default, the 16 MHz HSI (High-Speed Internal) clock source is used for the ADC, and this cannot be changed.

For the RTC the low-speed external (LSE) oscillator is used, which makes use of an external crystal for higher accuracy. This is done because of the timing requirements set by the wireless communication time slot scheme. All nodes are assigned a time slot in which they are allowed to send, as mentioned in Requirement 3. The maximum allowable time drift is 356.4 ms at a transmission rate of once every 5 minutes. A higher drift might lead to collisions in wireless transmissions.

For the external oscillator, the ABS07-166-32.768kHz-T crystal [32] is chosen as an external crystal with a frequency of $32.768\ kHz$, which is the frequency required for the LSE oscillator. The chosen oscillator has a load capacitance of $C_L = 7\ pF$ which is the maximum recommended value specified by the MCU datasheet [31]. For a crystal, the accuracy of the frequency produced by the crystal is strongly dependent on fabrication tolerances and the temperature. The chosen oscillator features a maximum fabrication tolerance of $\pm 10\ ppm$[17]. The crystals on the market with better fabrication tolerances have load capacitances higher than $7\ pF$, and are therefore not compatible with the MCU. The relation for the accuracy in ppm as a function of the temperature for the chosen crystal is given in Equation 3.8.

$$\Delta f(T) = \beta(T - T_0)^2 + f_0 \tag{3.8}$$

Here $\Delta f$ is the frequency deviation in ppm as a function of the temperature ($T$), $\beta$ the frequency deviation constant in $ppm/°C^2$, $T_0$ the turn-over temperature and $f_0$ the frequency deviation in ppm at the turn-over temperature. Furthermore, the drift can further increase due to the ageing of the crystal. According to the datasheet of the oscillator, ageing is $\pm 3\ ppm$ in the first year. For crystals in general, most ageing happens in the first year of operation, after which the contribution of ageing asymptotically decreases. During the

[17] $ppm$ means parts per million, where $1\% = 10000 ppm$

first 10 years, a worst-case drift of $\pm 10\ ppm$ is typical, but in practice, the drift is usually smaller [33]. To account for ageing effects of the crystal, $13\ ppm$ is added to the frequency tolerance, resulting in a total tolerance of $\pm 23\ ppm$, which is an estimation for the expected time drift in the 20-year lifetime of the product.

The frequency drift of the oscillator as a function of temperature is plotted in Figure 3.3. Here a tolerance worst-case tolerance of $\pm 23\ ppm$ is taken, a turnover temperature of $T_0 = 25\ °C$ and a frequency deviation constant of $\beta = -0.036\ ppm/°C$ as is specified by the datasheet.

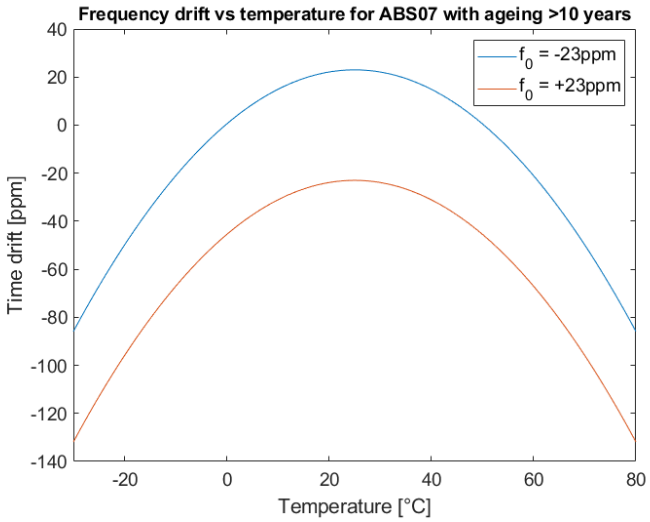


*Figure 3.3: Frequency deviation vs temperature for the ABS07 crystal oscillator*

To compensate for the frequency drift of the oscillator, periodic time synchronisation with the base station is needed. The full reasoning and explanation for this can be found in [9]. A complete diagram of the clocks and pre-scalers used can be found in Appendix C.

#### Temperature sensor - AS6212

The chosen AS6212 temperature sensor features a single-shot mode, in which the temperature sensor can do a single measurement when asked to, and then go to sleep till the next request. The clock frequency of the peripherals is set at $32.768\ kHz$ on the MCU, and therefore the I²C clock speed is also set to $32.768\ kHz$, so no extra clock source is needed. Furthermore, the datasheet of the sensor states that a $10\ nF$ decoupling capacitor is needed on the $Vdd$ line.

#### LoRa module - RN2483A

For the wireless communication module, the RN2483A is used. The reasoning for the choice of the wireless communication module can be found in [9]. The RN2483 uses a UART interface to communicate with the MCU and accepts a set of ASCII commands as input. The set of commands can be found in the command reference guide [34].

On the UART interface of the STM32L151CBT6TR, the oversampling rate is set to 8 by default and cannot be lowered. As the clock frequency of the peripherals is set at $32.768\ kHz$ on the MCU, the UART interface should operate at a Baud rate of $32.768\ kHz/8 = 4096\ bits/s$.

## 3.8 Summary hardware

In this design, the AS6212 is chosen as the temperature sensor. For each node, 5 of these sensors are connected to an STM32L151CBT6TR microcontroller using the I²C protocol. The microcontroller is connected to a wireless communication module with a UART interface. Furthermore, two GPIO pins are connected to the monitoring circuit of the energy storage system. An overview of the whole system can be found in Appendix D.

# Chapter 4

# Software description

In this project, the microcontroller is used to regulate the number of temperature measurements and is used to control the different components and data traffic of the system. Furthermore, to conserve energy, the frequency of the measurements should be adaptable based on the available amount of energy and the current temperature. An algorithm is devised to make a well-considered trade-off between measurement frequency and energy usage. In this chapter, the algorithm and its implementation in code are discussed, as well as tests on the different functions within it. The code itself can be found in a GitHub repository [35].

For this project, the Arduino compatible Nucleo L432KC development board from STM is used as a prototyping board. This development board is equipped with an STM32L432KC microcontroller which, like the chosen microcontroller, is based on the STM32 series. The development board features similar capabilities as the chosen microcontroller and is therefore suitable for testing code.

Different programming frameworks can be used to program the Nucleo L432KC, such as Arduino, Mbed and STM32Cube. Mbed and STM32Cube provide more features than the Arduino framework but are more complicated and harder to debug. Arduino is very well documented, provides a simple and clear programming environment and has a great amount of pre-made libraries that can be used. The STM32duino library is used to control the low-power modes and the RTC of the Nucleo board. Due to the limited amount of time, the Arduino framework is, therefore, deemed most suitable to develop a proof of concept.

A draw-back that comes with choosing the Arduino framework is that fewer low-power modes are available in the STM32duino library than are present on the microcontroller. Therefore, the ideal power modes have to be substituted by their closest counterpart. This limits the ability to test the behaviour of the chip in the desired power modes. The higher power usage is not an issue as the Nucleo board, in this case, is mainly used to test software behaviour. When the actual product is implemented, the STM32L151CBT6TR microcontroller would be used with the STM32Cube or Mbed framework instead of the development board with the Arduino framework.

In the code that is presented in this chapter, the deep-sleep-mode serves as a substitute for the stop-mode that would be used in an actual implementation without development board. Furthermore, the low-power-run-mode is replaced by the regular run mode.

## 4.1 Core functionality

In essence, the software has three main tasks: setting up the wireless communication module, reading out sensor data and controlling the frequency of temperature sensors and transmissions. After boot up, the system first enters a setup state in which all settings are configured, a connection with the LoRa network is established, and a time-slot is assigned. After this, the system enters one of two main modes of operation: normal operation and off-season operation. The two modes differ in what they focus on achieving. In normal operation, the system focuses on providing as many measurements as its energy level allows and how many are necessary. In off-season mode[18], the system focuses on maintaining enough power to transmit consistently with a lower measurement frequency, as during a part of the off-season period less energy can be harvested due to shorter days. The farmer can adjust what period of the year the system should be in what state via the base station.

[18] Here off-season is defined as the period of the year where no fruit frost occurs. The user can put the system in off-season mode from the base station.

However, since transmitting measurements and receiving information is energy consuming, it is important to make sure that there is enough energy in the system to do the transmission. Before measuring the temperature, the system first reads out the state of charge of the energy harvesting module. If not enough energy is present, the system should wait for a certain amount of time before attempting another measurement/transmission. Checking the energy level and determining what operation mode is used, is done in the power saving state. An overview of the functionality of the software is provided in Figure 4.1. Each of the states is elaborated in more detail in the following sections.

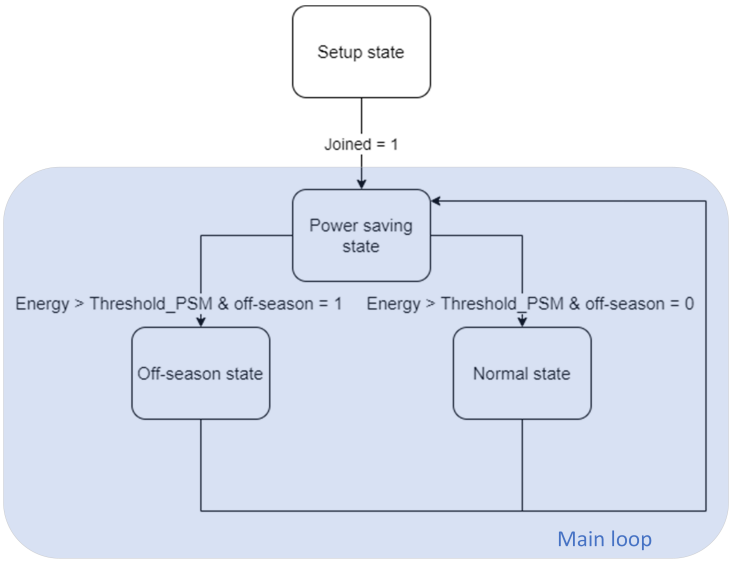


*Figure 4.1: A top level overview of the software. Here threshold_PSM is one of the configurable parameters of the system, which are explained in Table E.1*

### 4.1.1 Setup state

In the setup state, the software first configures the settings of the MCU, which include settings like the RTC time format, ADC resolution and wake-up sources. Following that, the system attempts to join the wireless network. Since this process makes use of wireless transmissions, the energy level is checked beforehand (further explained in Section 4.1.2). If enough energy is present, the system continues with the join procedure. If there is too little energy, the MCU enters sleep mode for 5 minutes before attempting again. If the attempt to join is unsuccessful, it means that there was a collision in the transmission. The system then waits a random time between 1 second and 5 minutes before attempting again in order to avoid another collision. Once the system has successfully joined the network, the program continues to the power saving state.

#### Join procedure

The join procedure involves setting up the UART baud rate of the wireless communication module, joining the network and obtaining a time-slot. To set the UART baud rate, a break condition is sent, after which the automatic baud-rate detection of the wireless communication module matches its baud-rate to that of the MCU. Through the established UART connection, several commands are sent to the wireless communication module. These include setting the pre-programmed unique device ID, the spreading factor, the network session key and the application session key, which are necessary for activation-by-personalisation (ABP), which is the join procedure chosen by the wireless communication group [9]. Finally, the system attempts to join the network via ABP. If the UART communication is successful, the wireless communication module replies with "ok". If the wireless transmission also completes successfully, the module returns "accepted" if the node has been accepted to the network. The MCU then requests a time slot from the base station. This is done by sending a request message to the base station. The base station then returns the current time, a free time slot, the critical temperature, and whether the off-season functionality should be turned on or off. This concludes the join procedure. This mode is not used anymore unless the system restarts after a shut-off, or an error occurs in the network.

### 4.1.2 Power saving state

After the join procedure, the system enters the power saving state. Here, it measures whether there is enough energy for a measurement and a transmission. If there is enough energy, this state is immediately exited. Otherwise, the system enters the low power mode for 5 minutes, after which the energy level is measured again. The energy level is then compared to the last level, to get an indication of how much energy is harvested during the last 5 minutes. Based on this energy harvesting rate, a prediction can be made on how long the system has to be in a low power mode to have enough energy for a transmission. This estimated duration is limited to a minimum of 1 minute and a maximum of 15 minutes. This minimum of 1 minute is set to make sure the system does not measure too often as it approaches the energy threshold for transmissions, thereby draining unnecessary amounts of energy. The maximum of 15 minutes makes sure the system does not sleep for too long, as the temperature slope calculation is only valid for short periods of time. After the system wakes up after the calculated duration, the energy is checked once again, and if enough energy is present, the time to the next available transmit window is calculated, and a wake-up alarm is set for that time. A flowchart of this function can be found in Figure H.1.

#### Check energy

The function `checkEnergy` calculates how many transmissions can be done in 12 hours with the current energy level and the worst-case energy consumption. The value of 12 hours is chosen as in the budding season nights can be up to 12 hours long [36]. During the night no energy can be harvested, meaning that the current energy level has to last through the whole night. The system has to be able to calculate how many transmissions can be done throughout one night to determine at what frequency measurements should be done. This is done by measuring the currently available energy, and quiescent energy consumption of the whole system. Based on this, an estimation can be made on how much energy is left for transmissions, and how many transmissions can be done in 12 hours.

The energy stored in the super-capacitor of the energy harvesting system can be found using the voltage over the supercapacitor. The energy harvesting module provides a voltage between $0\ V$ and $1.7\ V$, which is a scaled-down version of the actual super-capacitor voltage that ranges from $0\ V$ to $5.3\ V$. To read this voltage, the monitoring circuit first needs to be activated, which is done by providing a digital-high output to the diode of the circuit. The system then has to wait for $3\ ms$ before the voltage has settled. The voltage is then read with a GPIO pin that is configured as ADC, and the value can be used to calculate the energy stored via Equation 4.1. Here the capacitance is equal to the super-capacitor capacitance $C = 15\ F$. Also, it should be noted that the energy in the voltage range between $0\ V$ and $3.3\ V$ is not accessible.

$$E_{total} = \frac{1}{2} * C * (V_{adc} * \frac{5.3}{1.7})^2 - \frac{1}{2} * C * 3.3^2 \tag{4.1}$$

The quiescent energy consumption of the whole system (MCU, sensors, serial communication, wireless communication and power harvesting) in a worst-case scenario $34.7\ \mu W$, which is calculated from the values given in the energy harvesting report [37]. Additionally, there is a worst-case leakage current of the super-capacitor itself of $85\ \mu A$. These values are used to calculate the power usage with Equation 4.2.

$$P = P_{quiescent} + I_{lkg,c} * V * \frac{5.3}{1.7} \tag{4.2}$$

If this power is sustained over the period of 12 hours, the remaining energy left for transmissions can be calculated with Equation 4.3.

$$E_{trans} = E_{total} - 12 * 60 * 60 * P \tag{4.3}$$

Using the fact that a single transmission consumes about $0.04968\ J$ [37], the number of transmissions that can be done is calculated using Equation 4.4. This number is returned by the function.

$$N_{trans} = \frac{E_{trans}}{0.04968} \tag{4.4}$$

In Figure 4.2, the number of transmissions that can be done with the available energy level at a certain measured voltage is plotted. At a measured voltage of around $1.15\ V$ the system has just enough energy to sustain the nodes energy consumption without transmitting.

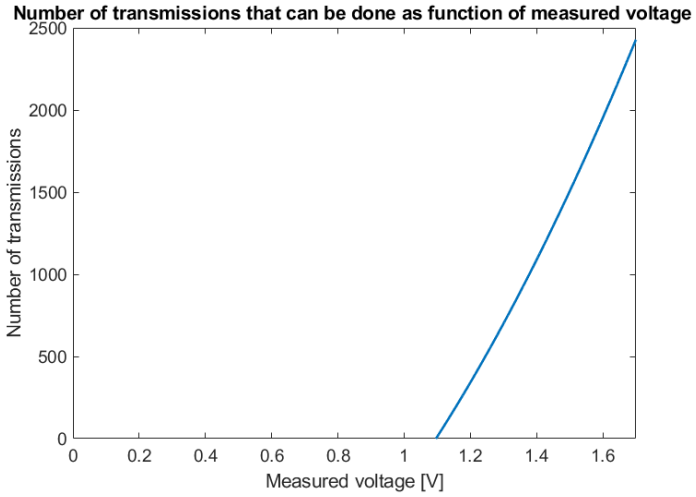


*Figure 4.2: The number of transmissions that can be cone with the available energy corresponding to the measured voltage.*

### 4.1.3 Normal state

In the normal state, the system has to actively measure to check for fruit frost. The measurement frequency is adaptive, and transmissions with temperature measurements can be done either every 5, 10, 15, 20 or 30 minutes. The frequency is limited to a maximum of 1 measurement per 5 minutes. This value is chosen as the maximum temperature slope during spring is $-8.3\ °C$ per hour as is found in Appendix B. A change $-1\ °C$ can thus happen in 7 minutes. It is therefore decided to measure at once per 5 minutes to be able to detect such changes. The minimum frequency of once per 30 minutes is set in order to comply with both Requirement 20 and Requirement 23. The exact measurement frequency is determined with the current energy level and temperature.

Firstly, upon entering the normal operation state, the system transmits its current energy level calculated with the `checkEnergy` function and its temperature data, which it has acquired using the `tempMeasurement` and `transmit` functions. Following this, the system calculates the time of the next transmission. A flowchart for the whole normal state function can be found in Figure H.2.

#### Temperature measurements

The temperature measurements are acquired from the sensors via an I2C interface with the function `tempMeasurement`. This function puts the sensor in single-shot by changing the configuration register of the sensor. After the sensor is done converting, it puts the measured value in the tvalue-register, from which it can be read from the MCU. The interpretation of the bits is done as prescribed in the datasheet of the sensor [18]. This process is repeated sequentially for the 5 sensors. The data from the sensors is put in an array for transmission.

#### Transmission

When a transmission needs to be done in normal mode, the `transmit` function is called. The energy-level along with the temperature are first converted into hexadecimal. Subsequently, the two hexadecimal values are put together in a single string following the format described in the report of the wireless communication part [9]. Then the string containing the data is sent along with a transmission command to the RN2483 communication module over the UART interface. Every transmission, there is a probability that the node receives a downlink message that synchronises the system, adjusts the time-slot and critical temperature if necessary, and indicates whether off-season mode should be activated. If no down-link message is received within 20 transmissions, it can be assumed that the system has failed and that it should rejoin the network [9]. Here the same `request` function is called as in the join procedure which returns a new time slot and the current system time.

### Calculating the next transmission time

After the measurement and transmission of the temperature data has been completed, the time of the next transmission is determined. The system first calls the `checkEnergy` function and determines the number of transmissions the system can still do within the next 12 hours with the current energy level. Based on the number of transmissions possible with the current energy, the maximum measurement frequency that can be sustained for 12 hours is calculated. The results achieved with this process are illustrated in Figure 4.3.

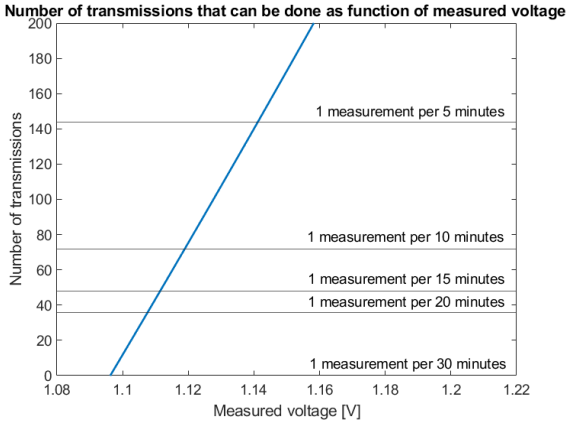


*Figure 4.3: The number of measurements that can be done as function of measured voltage.*

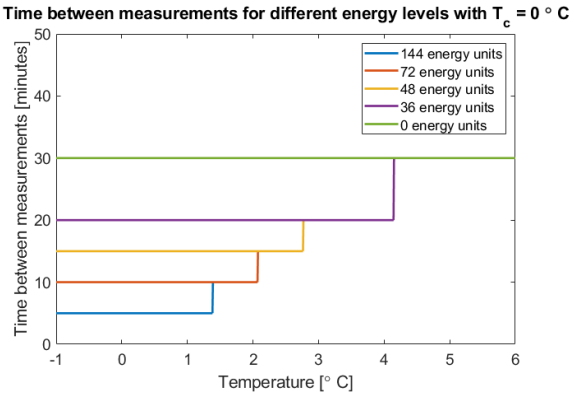


*Figure 4.4: The time between measurements as function of temperature for different energy levels*

After that, the temperature is measured using the `tempMeasurement` function, which returns the temperatures from the 5 sensors. Subsequently, `closestPeriod` finds the minimum measurement frequency needed to detect a temperature under the critical temperature. From the resulting two measurement frequencies, which are based on the current temperature and the available energy, the minimum frequency is chosen as the operating frequency. This is done so that the system can always sustain itself for 12 hours in the chosen operating mode, and the measurement frequency which is closest to the frequency needed to detect fruit frost is chosen. In Figure 4.4 the time between measurements is plotted as a function of temperature for different energy levels with a critical temperature of $T_c = 0\ °C$. Here 1 energy unit is defined as the amount of energy needed for 1 transmission. As can be seen, more measurement frequencies are available if enough energy is present in the system. Also, when the temperature is close to the critical temperature, the measurement frequency increases.

For the design, it is desirable to have a constant flow of temperature measurements at the base station, even at low measurement frequencies. Therefore, the system should ideally spread out the transmissions of different nodes over time.

Every hour is split into 12 time-segments of 5 minutes. During each time-segment, only a certain set of nodes can transmit data. This ensures that there are transmissions during all time-segments. For each of the measurement frequencies there is a different number of sets ($N_{set,total} = 12/f_{meas}$) where $f_{meas}$ is the number of measurements per hour. To determine which nodes belong to a certain set, the nodes are assigned a group number from 0 to 11 upon fabrication. The set-number of a node ($n_{set}$) can be found using the total amount of sets $N_{set,total}$ and the group number $n_{group}$ with Equation 4.5. The set number ranges from 0 to $(N_{set,total} - 1)$.

$$n_{set} = n_{group} \pmod{N_{set,total}} \tag{4.5}$$

A quantitative representation of the set division of the different groups at the various measurement frequencies is given in Table F.1.

To determine which set can transmit in what time-segment Equation 4.6 can be used. Here $ts_{set}$ is the set containing all time segment numbers of a particular set.

$$ts_{set} = \{N_{set,total} * k + n_{set} \mid ts_{set} <= 11 \mid k = \{0, 1, 2, ...\}\} \tag{4.6}$$

To further illustrate this concept, two examples will be given. First a case with $f_{meas} = 2$ measurements per hour, followed by a case with $f_{meas} = 4$ measurements per hour.

**2 measurements per hour** For this example, the procedure is followed for a node with group number 0. At the specified measurement frequency, there are 6 sets, each containing nodes from 2 different groups. Here group 0 belongs to set 0. The time segments that belong to this set are found to be time segment 0 and 6. This means that the node can transmit during its time-slot in these time segments. These segments correspond to the minute 0 to 5 and minute 30 to 35 respectively. An illustration of this concept is shown in Figure 4.5. Here the time segments corresponding to set 0 are coloured in blue. On the right, the nodes that can transmit at this time in at this measurement frequency are coloured.

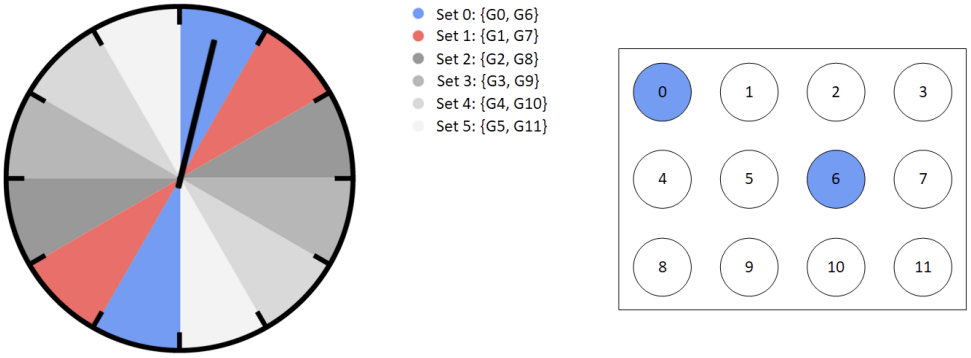


*Figure 4.5: For a measurement frequency of twice per hour, groups 0 and 6 can transmit in time segment 0 and 6.*

In this same example, a node with group number 7 ends up in set 1 (coloured in red). Given that it also measures twice per hour, it can transmit between minute 5 and 10 as well as between minute 35 and 40. This is illustrated in 4.6.

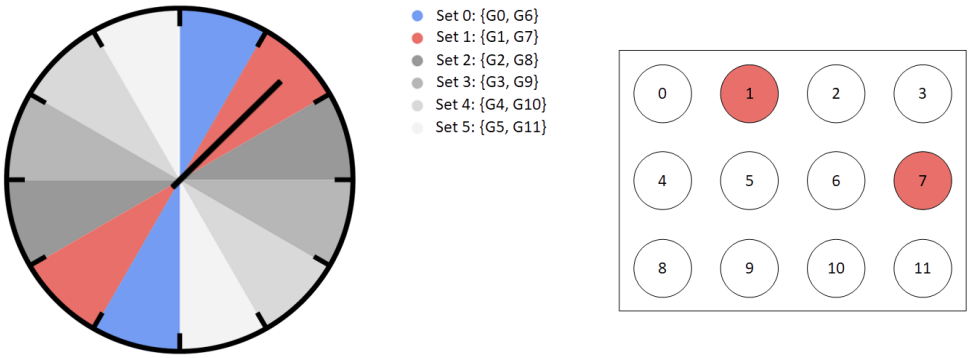


*Figure 4.6: For a measurement frequency of twice per hour, groups 1 and 7 can transmit in time segment 1 and 7.*

**4 measurements per hour** At a measurement frequency of 4 measurements per hour, there are only 3 sets containing 4 groups each. A node with group number 0 belongs to set 0. The node can thus transmit during time-segment 0, 3, 6 and 9, as can be seen in Figure 4.7.

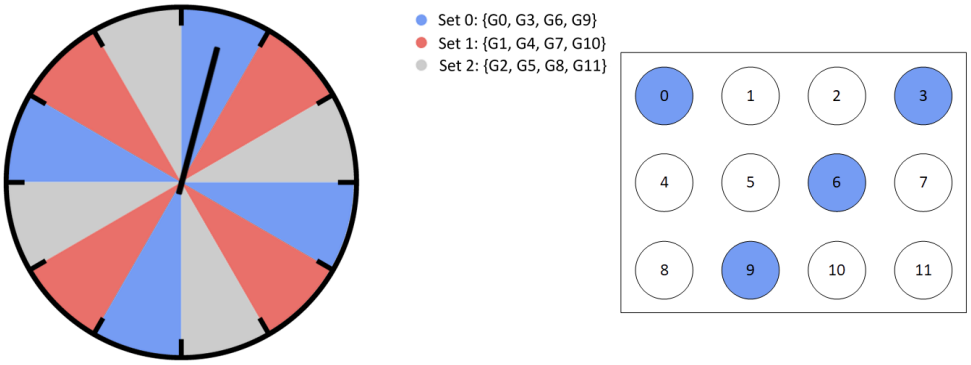


*Figure 4.7: For a measurement frequency of 4 times per hour, groups 0, 3, 6 and 9 can transmit during time segment 0, 3, 6, and 9.*

In the same example, a node with group number 7 ends up in set 1. It can transfer during time-segment 1, 4, 7 and 10, as can be seen in Figure 4.8.

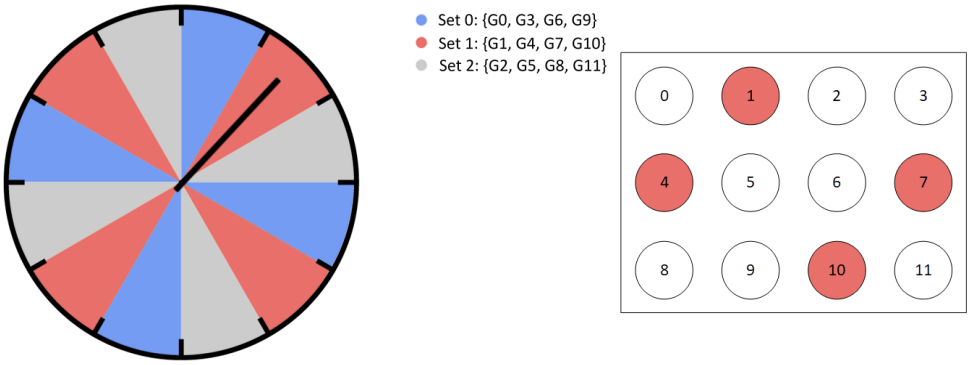


*Figure 4.8: For a measurement frequency of 4 times per hour, groups 1, 4, 7 and 10 can transmit during time segment 1, 4, 7 and 10.*

After the available time segments have been calculated, the node determines what time segment is closest to the one it is currently in. This is done by the function `nextTimeSeg`, which is tested in Appendix G.1. Then, a wake-up alarm is set for that time and the system enters its low power mode. When it wakes up from this mode, it returns to the power saving mode where the cycle is repeated again.

The advantages of the group and set system are not limited to temporal spreading. It has the added benefit that it also allows the measurements of one time-segment to be spread over the whole field as opposed to just a single area. The nodes can be placed in the field such that nodes with different group numbers are placed together as is shown in Figure 4.9. This maximises the probability that at least 1 measurement is received from each of the areas indicated by the boxes. The `closestPeriod` function is tested in Appendix G.2.

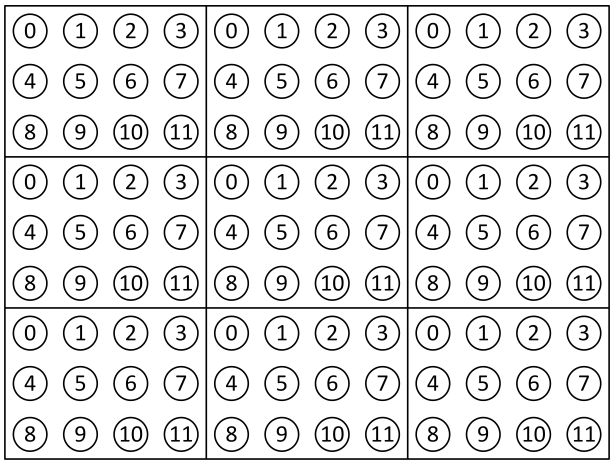


*Figure 4.9: An example of how the group numbers can be distributed over a field of about 1 hectare.*

### 4.1.4 Off-season state

The off-season state is an operation mode that specialises in maximising the uptime of the system when the farmer is not particularly interested in frequent temperature measurements. The farmer can indicate via the base station when this mode should be active. This mode is necessitated since, during winter, early spring and late autumn little solar energy can be harvested. To avoid complete outages that would require the nodes to have to reconnect to the network, energy usage should be decreased. Since no fruit frost occurs during these periods, because the trees do not have fragile blossoms or buds, there is no need for frequent temperature updates at the base station. Instead, multiple temperature measurements can be collected and transmitted together at the same time. There is also no need for dynamic temperature measurements, so the system can lower its measurement frequency down to a fixed 1 measurement per hour.

The wireless communication module supports payload sizes that can be large enough to transmit data of up to 4 different measurements at once. By sending multiple measurements over the same transmission, less energy is lost due to the overhead of the protocol. In the program, measurements made at 4 different times are stored in an array. When the array is full after 4 hours, the system transmits the data to the base station. This is done in a similar way as during normal operation. However, the system rejoins after 5 transmissions without confirmation [9]. A flowchart of the off-season state can be found in Figure H.3.

### 4.1.5 Watchdog timer

The watchdog timer (WDT) is used to detect and resolve malfunctions due to software failures. A reset sequence is triggered on the MCU if no reset signal is provided within a certain time window. This window is programmed to have the maximum value of 32.8 seconds to minimise the number of times the system has to wake up from sleep. At the beginning and end of every function call, the watchdog is reset. During the low power mode, the system wakes up periodically every 30 seconds to reset the timer.

# Chapter 5

# Power breakdown

To get an estimate on how much power the measurement and control subsystem uses, first the power usage of the different components is looked at separately, after which a worst-case power usage of the whole measurement and control system is calculated. In this chapter, all calculations are based on the most power-consuming mode of the system, which is when it measures once every 5 minutes. Furthermore, all components are supplied with a voltage of $V_{dd} = 1.8V$.

The following factors contributing to power are considered:

**Temperature sensor**

- Standby power: the power used by the temperature sensor when it is in standby mode.
- Active power: the power used by the temperature sensor when it is actively measuring the temperature and sending data over I²C.

**Microcontroller**

- Stop mode power: power usage when the MCU is in stop mode with all peripherals turned off.
- Active power usage: power used when the MCU is in LP run mode, where the CPU is active.
- Peripheral power usage: power usage by the different peripherals of the MCU, such as the I²C- and UART interface and the WDT.

**I²C**

- Dynamic power usage: Power used by the transmission of logic zeros on the I²C data-bus and clock line.

**UART**

- Dynamic power usage: power consumed by the transitioning between a logic high and low on the data lines.

First, the power usage of the temperature sensor is discussed, followed by the power usage of MCU and communication protocols. To conclude an overview of the found results is presented in Table 5.3 and Table 5.4.

## 5.1 Temperature sensor power usage

The average power usage of the temperature sensor can be calculated by multiplying the average running current and the operating voltage, which results in an average power usage of $0.2\ \mu W$. In Table 5.3 this power usage is multiplied by 5 to take all 5 sensors into account. The peak instantaneous current of the sensor is not listed in the datasheet. The value found for the peak instantaneous current in Table 5.3 is deduced from the datasheet values by using Equation 5.1. By subtracting the standby power from the average power given in the datasheet for 4 conversions per second, the additional energy consumed by 4 measurements can be found. Dividing by 4 then gives the additional energy consumed by a single conversion. This energy can then be divided by the conversion time to find an approximation for the power usage during one conversion. This is added on top of the standby power usage to find the peak instantaneous power. A visual representation of how this calculation works can be found in Figure 5.1. Here the areas coloured in blue represent the additional energy consumed by the sensors due to the conversions, which is equal in both graphs. Here the assumption is made that the power consumption is constant during

conversion, and the sensor can immediately switch between active and standby mode. This assumption is based on a figure found in the datasheet of the temperature sensor where the power usage over time is depicted in a similar manner [18, Figure 16].

$$P_{active} = E_{conv}/t_{conv} + P_{standby} = \frac{P_{avg@4Hz} - P_{standby}}{4} * \frac{1}{t_{conv}} + P_{standby} \tag{5.1}$$

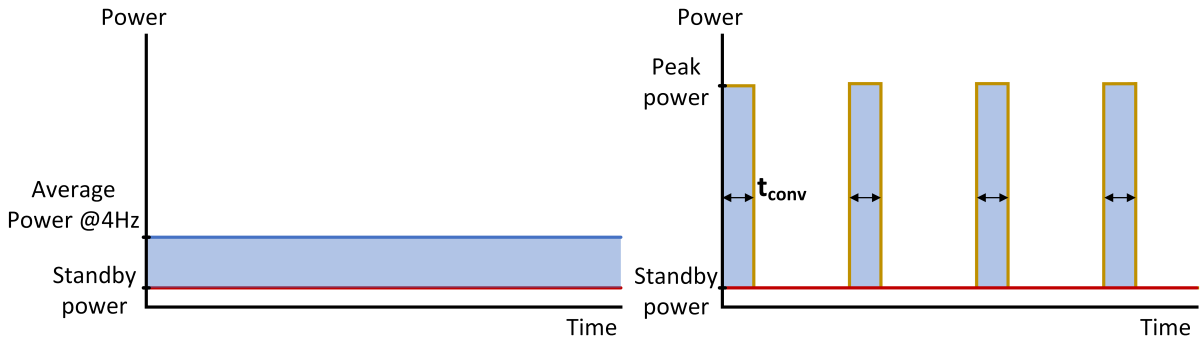


*Figure 5.1: Figure illustrating the way the peak power usage of the AS621x temperature sensor is found using the average current given in the datasheet for 4 measurements per second. (Not on scale)*

## 5.2 Microcontroller power usage

The microcontroller spends most of its time in a low power mode. For the chosen MCU this is the stop mode, as this is the lowest power mode which keeps the RTC active while maintaining the program state. This is important since the software is developed such that the program should continue where it left off when it entered its low power mode. The MCU periodically exits this state to perform a measurement and a transmission in its low-power-run-mode (LPrun). This mode is chosen as it is the mode with the lowest power usage while being able to operate the I²C and UART interface and perform calculations. Without any peripherals on, and the MCU core running at $f_{core} = 32\ kHz$, the MCU uses $24\ \mu A$ of current in low-power-run-mode while executing code from flash memory and keeping the RTC on, which results in a running power of $43.2\ \mu W$ in the low-power-run-mode. This, however, does not include the peripheral power used by the MCU.

To obtain the total active power, the MCU peripheral power should be added to the running power. The MCU peripheral power consists of the power used by the I²C and UART interfaces, GPIO pins, ADC, RTC and watchdog timer. The power contributions in the LPrun mode are divided into frequency-dependent power contributions and static power contributions. An overview of the frequency-dependent power usage of the peripherals is provided in Table 5.1 and the static power contributions of the peripherals are listed in Table 5.2.

*Table 5.1: Overview of frequency-dependent MCU power usage in LPrun mode*

| Peripheral | Current per MHz [µA/MHz] | Current at 32KHz [µA] | Power usage at 1.8 V at 32KHz [µW] |
|---|---|---|---|
| **UART** | 7.5 | 0.2 | 0.4 |
| **I²C** | 6.5 | 0.2 | 0.4 |
| **GPIO (x2)** | 7 | 0.2 | 0.4 |
| **ADC** | 9.0 | 0.3 | 0.5 |
| **Total** | 30.0 | 1.0 | 1.7 |

*Table 5.2: Overview of static MCU power usage in LPrun mode* [19]

| Peripheral | Static power consumption [µW] |
|---|---|
| **MCU (LPrun mode)** | 43.2 |
| **RTC** | 0.85 |
| **WDT** | 0.45 |
| **Total** | 44.5 |

Adding the frequency-dependent and static MCU peripheral power to the LPrun mode power, the total power used by the MCU in LPrun mode equals $P_{lpr} = 46.2\ \mu W$.

In stop mode, the MCU consumes a current of $1.2\ \mu A$, meaning that it uses $2.2\ \mu W$ of power. This already includes the power used by the RTC. During this mode, all peripherals but the watchdog timer and RTC are switched off. Adding the peripheral power used by the WDT gives a total power usage of $P_{stop} = 2.65\ \mu W$ in stop mode.

The average power cannot accurately be calculated since the active time of the MCU is unknown. However, an assumption can be made on the contribution of the active power usage to the average power usage. The ratio $R_{lp,stop}$ between the LPrun and stop mode power usage is calculated in Equation 5.2. From this, it follows that the LPrun mode consumes 17.43 times more power than the stop mode.

$$r_{lpr,stop} = \frac{P_{lpr}}{P_{stop}} = \frac{46.2}{2.65} = 17.43 \tag{5.2}$$

To even reach 10% of the stop mode power usage, the subsystem would have to be active for $(5{*}60/17.43){*}0.1 = 1.72$ seconds in which $1.72 * 32768 = 56361$ clock cycles would have to elapse for every measurement. This not expected for the application at hand. Based on this, the assumption is made that the running mode power usage is negligible compared to the stop mode power usage. This could be verified by power measuring equipment. For this reason, the average power usage of the MCU is estimated to be roughly equal to the stop-mode current.

## 5.3 Serial communication dynamic power usage

The dynamic power usage of the serial communication protocols can be calculated based on the amount of data that is transferred. Here the power usage of the protocols is discussed for the use case in this design.

### 5.3.1 I²C

From the datasheet of the temperature sensor, it can be found that in single-shot mode a transmission requires 87 bits of information to be sent with I²C. For the calculation of the power usage, the lowest pull-up resistor value is chosen, as this results in the most stable I²C output and the worst-case power usage. The lowest resistor value can be found using Equation 3.3, which gives a resistor value of $R_{min} = 467\ \Omega$. The peak instantaneous power can then be found with Equation 3.5. This results in a peak power of $6.9\,mW$ per line. This power usage is sustained during the time that a 0 is transmitted over the I²C lines. In the situation where the half of the transmitted bits are 0's, the total time spent by the data line transmitting 0's can be calculated with Equation 5.3. Here $t_0$ stands for the time spent transmitting 0's and $n_0$ for the number of 0's present in the transmitted message.

$$t_0 = \frac{n_0}{f} = \frac{0.5 * 87}{32000} = 1.36ms \tag{5.3}$$

The energy consumed during this time can be calculated using Equation 3.6. The power consumed in the clock-line can be calculated in the same way, again with the clock line transmitting 0's half of the time. Averaging the energy consumption of both lines over 5 minutes results in an average power of $63\ nW$.

[19] The contribution of the ADC to the power usage in LPrun mode is not considered since the ADC only needs sample 1 voltage per measurement, and the ADC is only on during that time. The typical conversion time for the ADC is $0.56\ \mu s$ and the running current of the ADC is $1450\ \mu A$. So each measurement, the ADC only uses $0.82\ nJ$. This means that if the node measures every 5 minutes, the average power contribution of the ADC will be $0.82\ nJ/(5 * 60\ s) = 2.71\ pW$, which is negligible compared to the total average power that is found. However, it is considered in the peak power consumption, which can be found in Table 5.4.

Multiplying this by five to account for the five sensors yields a total average dynamic power usage of $315\ \mu W$ for the I²C communication.

### 5.3.2 UART

To transmit a message to the wireless communication module and to receive a reply, a total of 44 bytes of data is transmitted using UART. Via Equation 3.2 the dynamic energy required ($E_{dyn,UART}$) for the transmission of 44 bytes can be calculated. The calculated energy can be averaged out over 5 minutes to find the average dynamic power usage, as is demonstrated in Equation 5.4. This results in an average dynamic power of $P_{avg,UART} = 65\ pW$.

$$P_{avg,UART} = N_{sensors} * \frac{E_{dyn,UART}}{t} = 5 * \frac{1.95 * 10^{-8}}{5 * 60} \tag{5.4}$$

Here $N_{sensors} = 5$ is the number of temperature sensors per pole. For the calculation of the peak instantaneous power usage of the UART communication, the typical rise time of the voltage along a line is needed. As no clear values of this are specified in the datasheet of the chosen MCU and temperature sensor, the maximum I/O pin current $I_{IO} = 4\ mA$ and maximum voltage $V_{OH} = V_{dd} = 1.8\ V$ are used to make a worst-case estimation of the peak power. This results in a peak instantaneous power of $7.2\ mW$.

## 5.4 Power consumption of the measurement and control system

The average power consumed by the various part of the measurement and control system is presented in Table 5.3. In this table, the average power of the 5 sensors combined is used. The average power usage of all components adds up to a total of $4\ \mu W$.

*Table 5.3: Overview total average power usage of the measurement and control system*

| Component | Average current [µA] | Average power [µW] |
|---|---|---|
| **Microcontroller** | 1.5 | 2.65 |
| **Temperature sensor (x5)** | 0.5 | 0.9 |
| **I²C (dynamic)** | 1.7e-1 | 0.3 |
| **UART (dynamic)** | 1.8e-4 | 3.3e-4 |
| **External crystal** | 5.6e-2 | 0.1 |
| **Total** | 2.2 | 4.0 |

The peak instantaneous power is presented in Table 5.4. Here the instantaneous power of a single temperature sensor is used as the sensors are activated sequentially, and there is no situation where more than 1 sensor is active at the same time.

*Table 5.4: Instantaneous power for different components.*

| Component | Peak instantaneous power [µW] |
|---|---|
| **Microcontroller**[22] | 46.2 |
| **ADC on Microcontroller** | 2610 |
| **Temperature sensor** | 41.15 |
| **I²C (dynamic)** | 13886.8 |
| **UART (dynamic)**[23] | 7200 |
| **External crystal** | 0.5 |

[20] Values specified for the microcontroller include peripheral power usage.

[21] Based on the typical power usage specified in [32].

[22] Values specified for the microcontroller include all power usage of all peripherals except for the ADC, which is considered as a separate component. This is done because in the case the system consumes the most power, the ADC is not active and thus cannot be added to the microcontroller peak power.

[23] worst-case instantaneous power estimation for UART based on the maximum output current $I_{IO} = 4\ mA$ and output voltage $V_{OH} = V_{dd} = 1.8\ V$ of an I/O pin of the STM32L151CBT6TR [31, Table 43].

To estimate the peak power used by the whole system, powers cannot simply be added, as not all systems are active at the same time. The worst-case instantaneous power usage occurs when both the microcontroller and temperature sensor are active and are transferring data over I²C. In this case, the UART interface is not active, and also the ADC is not active on the MCU. This results in a peak instantaneous power of the measurement and control system of about $14\ mW$.

# Chapter 6

# Discussion and Conclusion

The goal of the project was to design a wireless self-sustaining sensor system that smartly gathers temperature data to create a 3D temperature map of an orchard. The focus of the smart measurement and control subsystem was on designing a sensor system and a control system capable of measuring the temperature at different heights and communicating with the wireless communication module. Additionally, a system is created that allows for dynamic temperature measurements based on the available energy level and the ambient temperature.

To this end, first different options for serial communication protocols are considered for the data transfer between the sensor and control unit, based on energy usage and topology. After this, a selection of temperature sensors is made, which is primarily based on power usage, measurement accuracy, interfaces, error-checking capabilities and price. Following this, a selection of six suitable temperature sensors is found, of which the I²C-based AS6212 came out as the best option. A compatible control unit is chosen based on energy usage and price. A selection of six microcontrollers that fit the requirements of the design was put together, from which the STM32L151CBT6TR was chosen as the best option.

After component selection, the software is developed to control the various components of the subsystem, using the Arduino framework. An algorithm is developed to optimise the power usage of the system by dynamically adapting the measurement frequency. This is implemented in such a way that a sufficiently high measurement frequency is maintained to detect fruit frost. This is achieved by making use of a temporal and spatial spread of the measurements in the orchard.

An estimate of the energy consumption, as well as the peak power requirements, is made based on information in the datasheets of the chosen components and technologies. It is found that the subsystem requires an average power of $4\ \mu W$, and a peak power of $14\ mW$.

**Recommendations for future work**

Viewing the energy consumption of all subsystems in the node, it turns out that the smart sensing and control subsystem has a relatively small contribution to the energy usage of the complete system. The energy expended during 4 hours by the subsystem roughly equates to the energy expended during a single transmission. Also, the quiescent energy consumption of the other subsystems is significantly higher than the measurement and control system. Therefore, more emphasis could be put on factors other than low energy consumption such as cost or other features. For example, more advanced microcontrollers that support higher tolerance oscillator-crystals and RTC calibration features could be chosen, that were previously rejected due to their higher energy usage.

To verify the design, a prototype should be built. This could confirm or invalidate the theory-based estimations, calculations and assumptions made in this design. The energy consumption of the various components could be measured, which would lead to a more accurate power breakdown. Implementing the software on the prototype would give the possibility to find the execution time, which verifies whether the active energy consumption is rightfully deemed negligible. Furthermore, a prototype could potentially bring to light unforeseen issues in the design. For example, noise is not taken into account when considering the serial communication between temperature sensor and microcontroller. This could potentially result in erroneous temperature data being sent to the base station. This would warrant a reconsideration of the design choices such as a different serial communication protocol or additional error-checking.

# References


[1] H. v. d. Meulen, "Fruitareaal en aantal bedrijven," *Fruitareaal*, 2020. [Online]. Available: https://www.agrimatie.nl/SectorResultaat.aspx?subpubID=2232&sectorID=2237

[2] J. Rodrigo, "Spring frosts in deciduous fruit trees. Morphological damage and flower hardiness," *Scientia Horticulturae*, vol. 85, no. 3, pp. 155–173, 2000.

[3] A. A. Ghaemi, M. R. Rafiee, and A. R. Sepaskhah, "Tree-Temperature Monitoring for Frost Protection of Orchards in Semi-Arid Regions Using Sprinkler Irrigation," *Agricultural Sciences in China*, vol. 8, no. 1, pp. 98–107, 2009. [Online]. Available: http://dx.doi.org/10.1016/S1671-2927(09)60014-6

[4] A. C. Ribeiro, J. P. De Melo-Abreu, and R. L. Snyder, "Apple orchard frost protection with wind machine operation," *Agricultural and Forest Meteorology*, vol. 141, no. 2-4, pp. 71–81, 2006.

[5] S. R. Prathibha, A. Hongal, and M. P. Jyothi, "IOT Based Monitoring System in Smart Agriculture," *Proceedings - 2017 International Conference on Recent Advances in Electronics and Communication Technology, ICRAECT 2017*, pp. 81–84, 2017.

[6] M. S. Mahmoud and A. A. H. Mohamad, "A Study of Efficient Power Consumption Wireless Communication Techniques/ Modules for Internet of Things (IoT) Applications," *Advances in Internet of Things*, vol. 06, no. 02, pp. 19–29, 2016.

[7] F. J. Pierce and T. V. Elliott, "Regional and on-farm wireless sensor networks for agricultural systems in Eastern Washington," *Computers and Electronics in Agriculture*, vol. 61, no. 1, pp. 32–43, 2008.

[8] G. Sushanth and S. Sujatha, "IOT Based Smart Agriculture System," *2018 International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2018*, pp. 1–4, 2018.

[9] A. Bleeker and R. Wijnands, "Autonomous Temperature Sensor for Smart Agriculture - Wireless Communication," 2020.

[10] B. Pinty, F. Roveda, M. M. Verstraete, N. Gobron, Y. Govaerts, J. V. Martonchik, D. J. Diner, and R. A. Kahn, "I2C-Cabling," Analog Devices, Tech. Rep., 2013.

[11] T. Brand, "Isolated SPI Communication Made Easy," *Analog Devices*, 2019. [Online]. Available: https://www.analog.com/en/technical-articles/isolated-spi-communication-made-easy.html

[12] L. Asena, S. G. Güngör, and A. Akman, "Comparison of keratometric measurements obtained by the Verion Image Guided System with optical biometry and auto-keratorefractometer," *International Ophthalmology*, vol. 37, no. 2, pp. 391–399, 2017.

[13] A. Gloria, F. Cercas, and N. Souto, "Comparison of communication protocols for low cost Internet of Things devices," *South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference, SEEDA-CECNSM 2017*, 2017.

[14] K. Mikhaylov and J. Tervonen, "Evaluation of power efficiency for digital serial interfaces of microcontrollers," *2012 5th International Conference on New Technologies, Mobility and Security - Proceedings of NTMS 2012 Conference and Workshops*, 2012.

[15] *CAT5e Horizontal, 4pr, UTP, PVC Jkt, CMR*, Belden, 2020, rev. 0.405. [Online]. Available: https://catalog.belden.com/techdata/EN/1583A_techdata.pdf

[16] *I2C-bus specification and user manual*, NXP, 2014, rev. 6. [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[17] *Low-voltage, ultra-low-power, 0.5 °C accuracy I²C/SMBus 3.0 temperature sensor*, ST-Microelectronics, 2019, rev. 4. [Online]. Available: https://www.st.com/resource/en/datasheet/stts22h.pdf

[18] *AS621x Digital Temperature Sensor*, AMS, 2020, rev. 2. [Online]. Available: https://ams.com/documents/20143/36005/AS621x_DS000677_2-00.pdf/a90b32a5-d1f8-e6df-5327-bec856e093dd

[19] *TMP102 Low-Power Digital Temperature Sensor With SMBus and Two-Wire Serial Interface in SOT563*, Texas Instruments, 2018. [Online]. Available: http://www.ti.com/lit/ds/symlink/tmp102.pdf?&ts=1589811705241

[20] *TMP107 Digital Temperature Sensor With Bidirectional UART One-Wire Interface and EEPROM*, Texas Instruments, 2020. [Online]. Available: https://www.ti.com/lit/ds/symlink/tmp107.pdf?ts=1591359280104

[21] *Low-voltage, ultra-low-power, 0.5 °C accuracy I²C/SMBus 3.0 temperature sensor*, Tyco Electronics, 2015. [Online]. Available: https://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=TSYS02S&DocType=DS&DocLang=English

[22] *Digital Thermometers and Thermostats with SPI/3-Wire Interface*, Maxim Integrated, 2015, rev. 2. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX31722-MAX31723.pdf

[23] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*, 3rd ed. USA: Prentice Hall Press, 2008.

[24] J. Borgeson, S. Schauer, and H. Diewald, "Benchmarking MCU power consumption for ultra-low-power applications," Texas Instruments, Dallas, Tech. Rep., 2012.

[25] M. Hayashikoshi, H. Kawai, H. Ueki, and T. Shimizu, "Normally-off MCU architecture and power management method for low-power sensor network," *ISOCC 2015 - International SoC Design Conference: SoC for Internet of Everything (IoE)*, pp. 151–152, 2016.

[26] *MCU with Integrated 240–960 MHz EZRadioPRO® Transceiver*, Silicon Laboratories, 2013, rev. 2. [Online]. Available: https://www.silabs.com/documents/public/data-sheets/Si106x-8x.pdf

[27] *EFM32 Zero Gecko Family*, Silicon Laboratories, 2013, rev. 2. [Online]. Available: https://www.silabs.com/documents/public/data-sheets/efm32zg-datasheet.pdf

[28] *AVR® Microcontroller with Core Independent Peripherals and PicoPower® Technology*, Microchip Inc., 2018. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/40001906C.pdf

[29] *8-bit ultra-low-power MCU, up to 64-KB Flash, 2-KB data EEPROM, RTC, LCD, timers, USARTs, I2C, SPIs, ADC, DAC, comparators*, ST-Microelectronics, 2018, rev. 11. [Online]. Available: https://www.st.com/resource/en/datasheet/stm8l152k8.pdf

[30] *Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 256KB Flash, 64KB SRAM, analog, audio*, ST-Microelectronics, 2018, rev. 3. [Online]. Available: https://nl.mouser.com/datasheet/2/389/dm00257211-1798949.pdf

[31] *Ultra-low-power 32-bit MCU ARM®-based Cortex®-M3, 128KB Flash, 16KB SRAM, 4KB EEPROM, LCD, USB, ADC, DAC*, ST-Microelectronics, 2016, rev. 12. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32l151cb.pdf

[32] *ABS07-166-32.768KHZ-T; TUNING FORK CRYSTAL*, Abracon, 2008, revised 19 March 2018. [Online]. Available: https://nl.mouser.com/datasheet/2/3/ABS07-166-32.768kHz-T-1359238.pdf

[33] *Quartz Crystal Ageing*, IQD. [Online]. Available: https://www.iqdfrequencyproducts.com/media/pg/1589/1459502414/quartz-crystal-ageing.pdf

[34] *RN2483 LoRa Technology Module Command Reference User's Guide*, Microchip, 2015, rev. B. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/40001784B.pdf

[35] M. Huiskes and M. Miao, "Measurement and control software," https://github.com/MichaelMiao99/BAP_measurement_control, 2020.

[36] M. v. d. Linden, "De dagen zijn weer langer dan de nachten," 2020. [Online]. Available: https://www.weeronline.nl/nieuws/de-dagen-zijn-weer-langer-dan-de-nachten

[37] M. Hubers and R. van der Sande, "Autonomous Temperature Sensor for Smart Agriculture - Energy Harvesting and Control," p. 70, 2020.

[38] J. Bloksma, "Basiskennis Hoogstamfruit - Deel 1," Tilburg, 2020.

# Appendix A

# The top-level system

## A.1 Top-level system implementation

Before the three subsystems are designed the implementation of the entire system is considered. The goal of the implementation is such that it fits the requirements of the entire system and that each subsystem can comply with their own requirements. When all these requirements are met, aspects that have to be reduced are manufacturing cost, deployment cost and maintenance cost. Furthermore, durability has to be taken into consideration. To prevent further confusion, a single-node is regarded as a subsystem with one wireless communication module, one microcontroller and one energy harvesting module. The possible implementations of the system that were considered that fit into the requirements of the system are the following:

- **Tree multi-node:** An implementation where each single-node has one temperature sensor. As a result, a tree contains 5 separate nodes all on one tree. These multiple nodes are placed at different heights in the tree. This is a multi-node implementation in a tree.
- **Tree single-node:** A system where every node consists of a single MCU, wireless communication module and energy harvester, while multiple temperature sensors are connected to this single MCU. All sensors are placed in a tree at different heights.
- **Pole multi-node:** A system where every node contains a single temperature sensor. Multiple nodes are placed on different heights on a pole.
- **Pole single-node:** A system where every node has multiple connected temperature sensors and every node is integrated on a pole with the sensors at different heights.

To come to a design decision each option is weighted in the sections below. Based on this weight and the importance of each aspect an end decision is made which implementation is ultimately chosen.

#### Power consumption

The main power consumer is the wireless communication module. In a single-node implementation, four wireless communication modules and MCUs are spared and thus this implementation is more preferable over a multi-node implementation. The power consumption does not differ between a pole or tree implementation.

#### Cost

By implementing a single-node structure, four wireless communication modules, four microcontrollers and four energy harvesting modules are spared compared to a multi-node structure. This gives a single-node structure more advantages. The purchase of a pole gives a tree implementation preference compared to the pole implementation, although this can be offset by the lower installation cost.

#### Durability

Durability assesses the lifetime of the system. A pole implementation is considered more durable than a tree implementation since it has a more robust structure than the more fragile branches of a tree. For the pole implementation, it is difficult to assess whether a multi-node or a single-node implementation is better. However, in a single-node implementation, fewer components are used and thus the chance of component failure is lower. On the other hand, in a single-node implementation, temperature sensors are connected by wire and it depends on the type of integration whether this can have a signification negative impact on the device its lifetime.

**Deployment**

The deployment of the system assesses how easily the system can be installed. A significant advantage of a pole implementation is that it can be prefabricated and quickly installed by only planting the pole. For the tree, the on-sight installation has to be performed which can be quite work-intensive, since sensors have to be installed on a certain height. Regarding a multi-node or single-node solution, the only main difference is the wiring which comes with a single-node system. This is concerned to be more dramatic in a tree than on a pole since wiring of prefabricated poles can be implemented more easily.

**Placement consistency**

The consistency of the system assesses how uniform the heights of the sensors are for each of the trees/poles. For an implementation with a pole the distance between sensors can easily be made very consistent, whereas, for the implementation in a tree, heights can be slightly different due to the fact that trees have different shapes. For this factor, it does not matter whether a single-node or multi-node implementation is chosen.

**Efficiency**

The efficiency of the system describes how efficiently the system can harvest energy and transmit signals. For a pole implementation with the antenna on top, the transmitted signal likely comes across fewer obstacles. Also with the solar panel on top of a pole instead of on a tree, the panel will receive a lot more solar energy due to the fact that there is less shadow from the three itself. The difference between a multi-node implementation and single-node implementation also give rise to different signal transmission- and energy harvesting efficiency. In the multi-node case, the solar panels are positioned on different heights. The nodes that are positioned higher will likely receive more solar radiation than lower positioned nodes. Furthermore transmitted signal from the nodes that are positioned lower will likely come across more obstacles.

**Flexibility**

The flexibility of the system describes the ease of changing the number of sensors at a location. For a multi-node pole implementation, the number of sensors is changed with more ease than for a single-node implementation. This is because, for a single-node implementation, a sensor should be added or removed from the MCU, while for a multi-node implementation only a complete module should be placed or removed without the need of connecting or disconnecting wires. The same holds for a tree implementation.

**Maintainability**

An important measure for designing a system is minimising the cost of maintenance after the system has been designed to reduce further costs and time. To maintain the system where each node has a single sensor greatly increases the amount of hardware that can fail and needs to be maintained. The time it takes to replace a node can be neglected in the comparison between multi and single-node as it will have much less influence on the total maintenance cost than the total amount of hardware failures. To compare the tree and pole solution it comes to the same comparison as with deployment namely disassembling the node and installing it again.

**Expandability**

By implementing a single-node structure, more nodes can be added before the communication system gets saturated. This because of the limitation in the number of wireless communication nodes that can be used for a certain gateway. By implementing a multi-node structure, the maximum number of nodes gets reached more easily, and so the size of the sensor network becomes more limited.

*Table A.1: Decision matrix for a tree or pole and multi- or single-node implementation*

| | | Tree multi-node | | Tree single-node | | Pole multi-node | | Pole single-node | |
|---|---|---|---|---|---|---|---|---|---|
| **Criterion** | **Weight** | **Score** | **Weighted** | **Score** | **Weighted** | **Score** | **Weighted** | **Score** | **Weighted** |
| **Power consumption** | 5 | 3 | 15 | 10 | 50 | 3 | 15 | 10 | 50 |
| **Cost** | 7 | 6 | 42 | 10 | 70 | 5 | 35 | 9 | 63 |
| **Durability** | 9 | 4 | 36 | 3 | 27 | 9 | 81 | 8 | 72 |
| **Deployment** | 2 | 4 | 8 | 3 | 6 | 10 | 20 | 10 | 20 |
| **Placement consistency** | 2 | 6 | 12 | 6 | 12 | 10 | 20 | 10 | 20 |
| **Efficiency** | 5 | 1 | 5 | 5 | 25 | 3 | 15 | 10 | 50 |
| **Flexibility** | 3 | 8 | 24 | 2 | 6 | 8 | 24 | 2 | 6 |
| **Maintainablity** | 7 | 1 | 7 | 8 | 56 | 2 | 14 | 9 | 63 |
| **Expandability** | 2 | 2 | 4 | 10 | 20 | 2 | 4 | 10 | 20 |
| | | | 153 | | 272 | | 228 | | 364 |

### Final decision

From Table A.1, it clear that, with a score of 364, a single-node pole implementation is the best option for this application. The sub-modules in this project are designed according to this specific implementation. For the remainder of the thesis, a pole-length of 3 meters is considered as this is the typical maximum length of a dwarf or semi-dwarf apple tree [38, Page 11], which are common in the commercial fruit-orchards.

# Appendix B

# Maximum temperature slope

In order to determine the frequency with which the system should measure so that the system will give an alert on-time, the maximum change in temperature over time is a necessary parameter. For estimating this temperature change, data from the KNMI (Royal Dutch Meteorological Institute) is used. The dataset used contains the temperature measured in an hourly interval for 50 weather stations spread over the Netherlands, for a period of 20 years.

The change in temperature varies at different times of the year and at different temperature ranges. For the purposes of this project, only temperatures around the freezing point that were measured during spring are considered.

In Figure B.1, the maximum drop in temperature between when the temperature first drops below 0 degrees and an hour before is plotted. These are the maxima of values collected in between 1 February and 1 June in the 20 years between 2000 and 2020. From this figure, it can be seen that the maximum temperature slope found is around $8.3\ °C/hour$. This measurement is an extreme outlier, but the designed system should be able to detect such changes, as that is what farmers are most interested in.

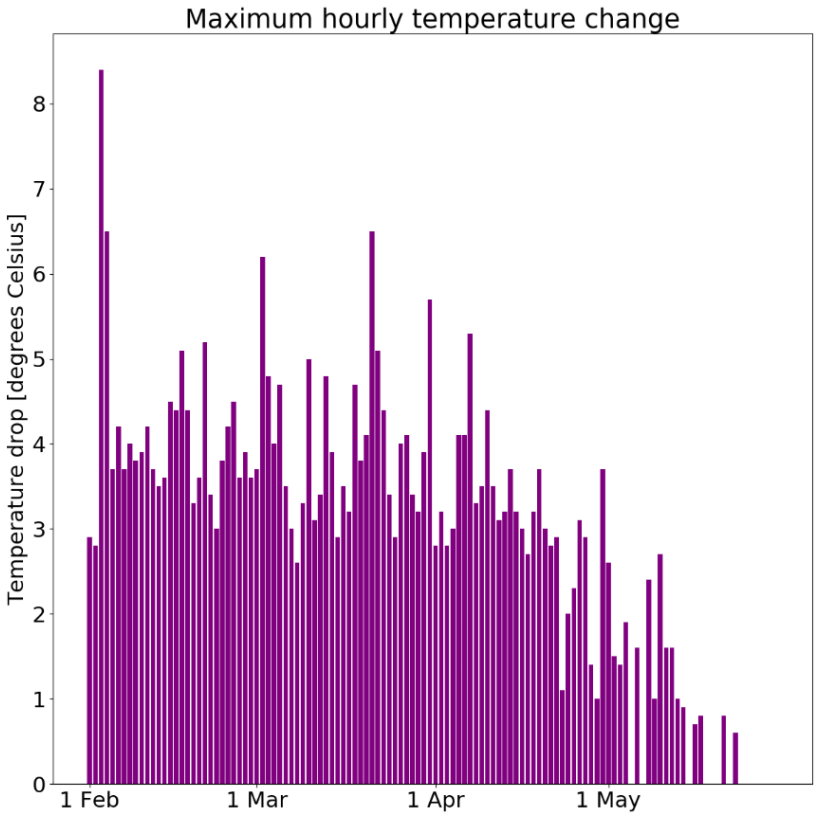


*Figure B.1: The maximum hourly temperature change that results in a sub-0 temperature.*

# Appendix C

# System clock overview

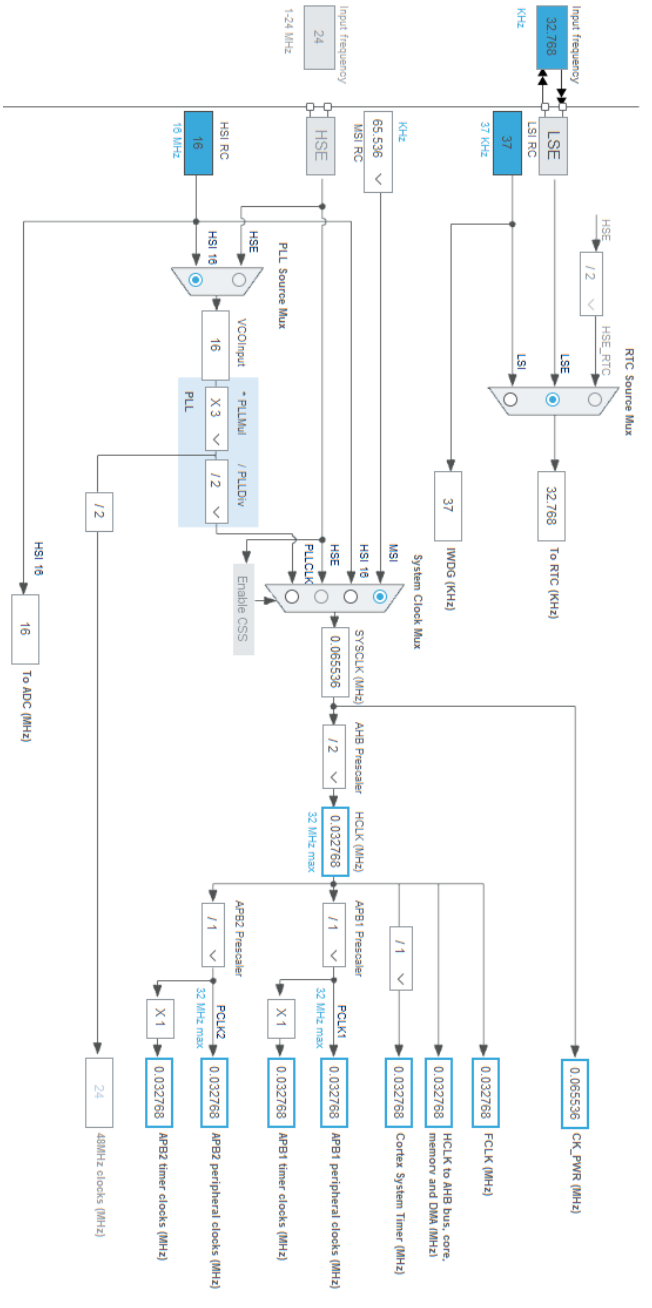


*Figure C.1: An overview of the clock system in the MCU.*

# Appendix D

# System overview

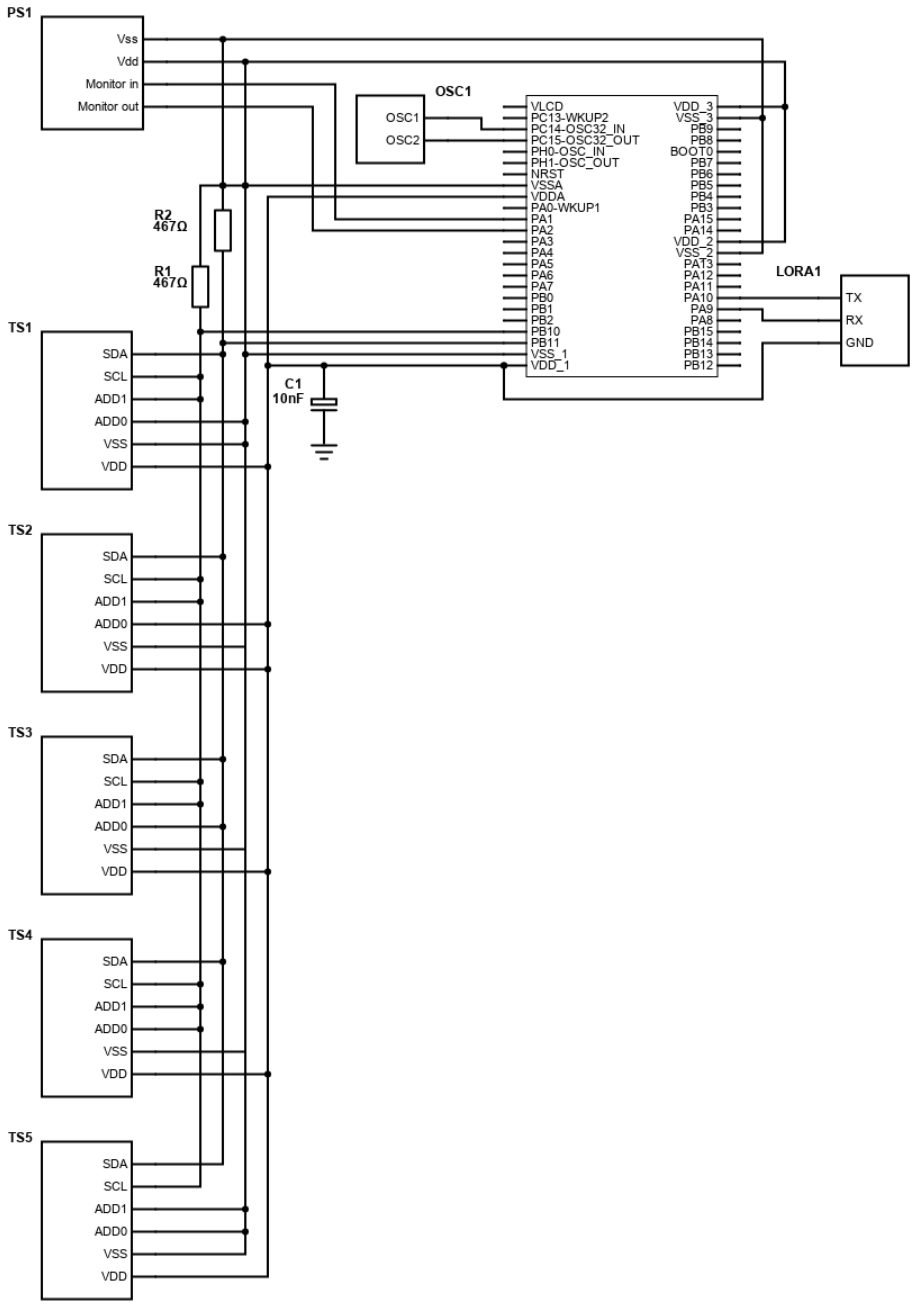


*Figure D.1: An overview of the measurement and control subsystem*

# Appendix E

# Software parameters

In Table E.1 below, the configurable parameters of the software can be found, including their meaning and default values.

*Table E.1: Configurable parameters in the software*

| Parameter | Function | Default value |
|---|---|---|
| **threshold_PSM** | Indicates the minimum energy value needed in order for the device to come out of power saving mode (PSM) | 2 |
| **threshold_Join** | Indicates the minimum energy needed to be able join the LoRa network | 2 |
| **sensorCount** | Indicates the amount of temperature sensors connected to the MCU | 5 |
| **maximumTemperatureSlope** | The maximum hourly temperature change that can occur in the area the device is used | 8.3 $°C/hour$ |
| **offSeasonIterations** | Amount of measurement times before transmission in off-season mode | 4 |
| **OffSeasonSleepDuration** | The amount of time between measurements in off-season mode | 60 minutes |
| **groupNumber** | The group number assigned to the node | - |
| **deveui** | LoRa device ID | - |
| **sf** | Spreading factor | - |
| **nwkskey** | LoRa network session key | - |
| **appskey** | LoRa application session key | - |

Threshold_PSM has been given a default value of 2. This is equivalent to the energy needed for 2 measurements and transmissions (see Equation 4.4). The limited resolution of the ADC gives an inaccuracy of $0.032\ J$ in the energy level measurement.
This is 64% of the energy consumed by 1 measurement. The threshold is given a value of 2 as it provides a small overhead on the amount of energy needed for a single transmission and to compensate for the inaccuracy of the energy level measurement. The join threshold threshold_Join is given the same value for the same reason.

# Appendix F

# Set division

*Table F.1: Set number for different group numbers and number of measurements per hour*

| Group number / Frequency | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **2** | 0 | 1 | 2 | 3 | 4 | 5 | 0 | 1 | 2 | 3 | 4 | 5 |
| **3** | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| **4** | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
| **6** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| **12** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Appendix G

# Software function tests

In this chapter the behaviour is tested of the `nextTimeSeg` and the `closestFrequency` functions. The functions are run with different input combinations and the output of the functions is checked to verify whether the functions are working as intended.

## G.1 nextTimeSeg

The code seen in Listing 1 is used to test the `nextTimeSeg` function. The group number, time segment the device is currently in and $f_{meas}$ can be set. The test code will then keep calculating the next time segment based on the last time segment. Note that the `nextTimeSeg` function is not included in Listing 1, but can be found the in GitHub repository [35].

*Listing 1: Test code for the next time segment function.*

```
1  int groupNumber = 0; // Set ground number
2  int currentTimeSegment = 2; // Set the time-segment the device starts in
3  int f_meas = 4;             // Set the measurement frequency per hour
4
5  int measurementPeriod = 60/f_meas; // Calculate the period between measurements
6
7  void setup() {
8    // put your setup code here, to run once:
9    Serial.begin(9600);
10 }
11
12 void loop() {
13   // put your main code here, to run repeatedly:
14   while(1){
15   currentTimeSegment = nextTimeSeg(currentTimeSegment,measurementPeriod);
16   Serial.println(currentTimeSegment); //print output
17   delay(500);
18   }
19
20 }
```

The output for two different cases are shown in Figure G.1 and G.2, which behave as expected following the logic explained in Chapter 4.1.3.

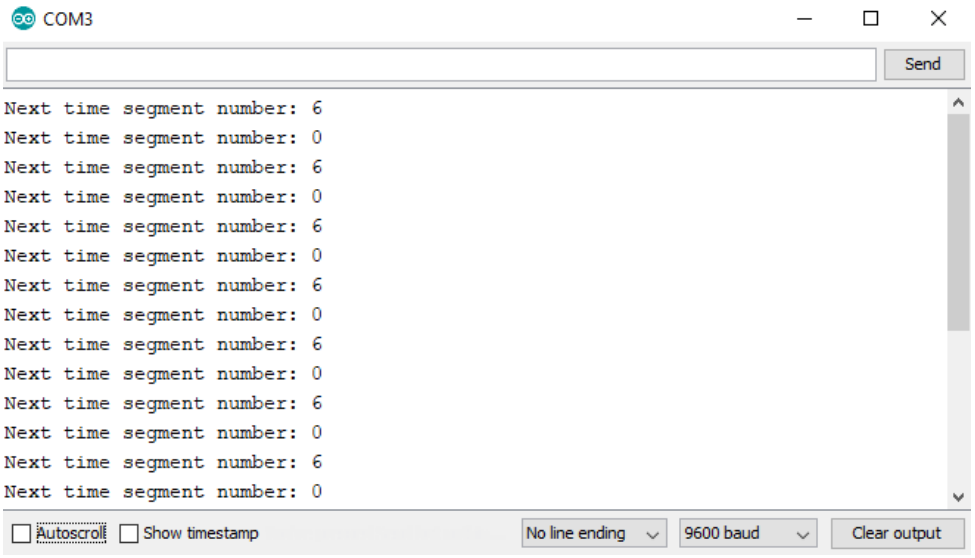


*Figure G.1: Result of the nextTimeSeg function for a device with group number 0, $f_{meas} = 2$ measurements per hour and which starts in time-segment 2*

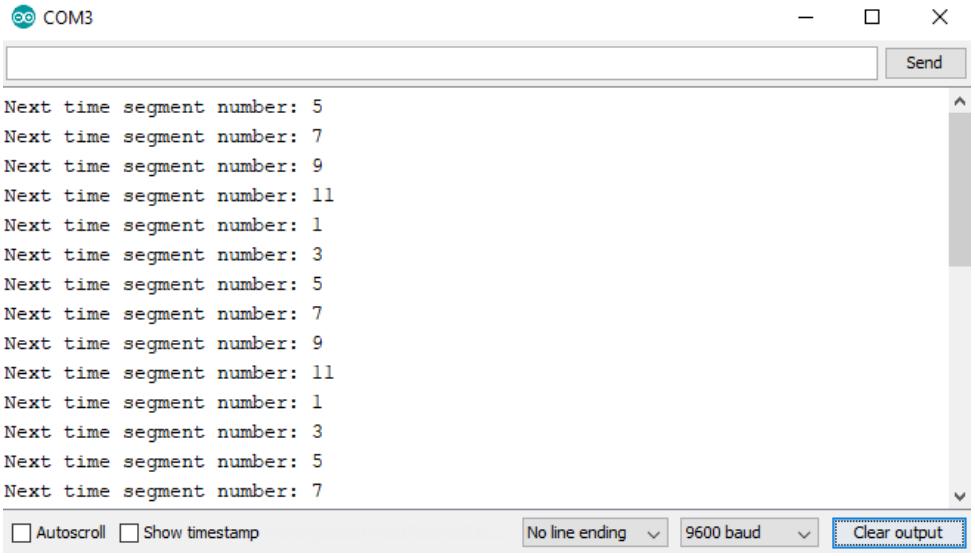


*Figure G.2: Result of the nextTimeSeg function for a device with group number 3, $f_{meas} = 6$ measurements per hour and which starts in time-segment 4*

## G.2 closestPeriod

The code seen in Listing 2 is used to test the `closestPeriod` function. Here the temps-variable represents the minimum temperature measured by the sensors from the last measurement. The values in the temps-array are the test values. The criticalTemp represents the set critical temperature, which for this example is set to $1\ °C$ and the maxTempSlope variable is the maximum temperature slope per minute as discussed in B.

*Listing 2: Test code for the closestPeriod function.*

```
1 int temps[8] = {-60,10,20,30,40,50,60,180}; //current temperature value x10, so
  ↪  -6,1,2,3,4,5,6 and 18 degrees
2 int criticalTemp = 10; // critical temperature x10 -> critical temp is 1 degree celcius
3 float maxTempSlope = 83.0/60; //temperature slope per minute
```

```
int temperature;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(2000);
  for (size_t i = 0; i<8; i++)
  {
    temperature = temps[i];
    int closestPer = closestPeriod();
    Serial.print("Time between measurements when the current temperature is ");
    Serial.print(temps[i]/10);
    Serial.print(": ");
    Serial.println(closestPer);
    Serial.print(" minutes");
    Serial.println();
  }
}

int closestPeriod()
{
  int minTimeToNext = (temperature - criticalTemp) / maxTempSlope;
  int availablePeriods[5] = {30, 20, 15, 10, 5};
  int minPeriod = 5;
  for (size_t i = 0; i < 5; i++)
  {
    if (availablePeriods[i] < minTimeToNext)
    {
      minPeriod = availablePeriods[i];
      break;
    }
  }
  return minPeriod;
}
```

The results of the code presented in Listing 2, which calculates the needed measurement interval for the different temperatures in the temps array, is shown in Figure G.3. The results are as expected, as the resulting measurement intervals are such that a temperature under or equal to the critical temperature is detected with a minimum time period of 5 minutes.

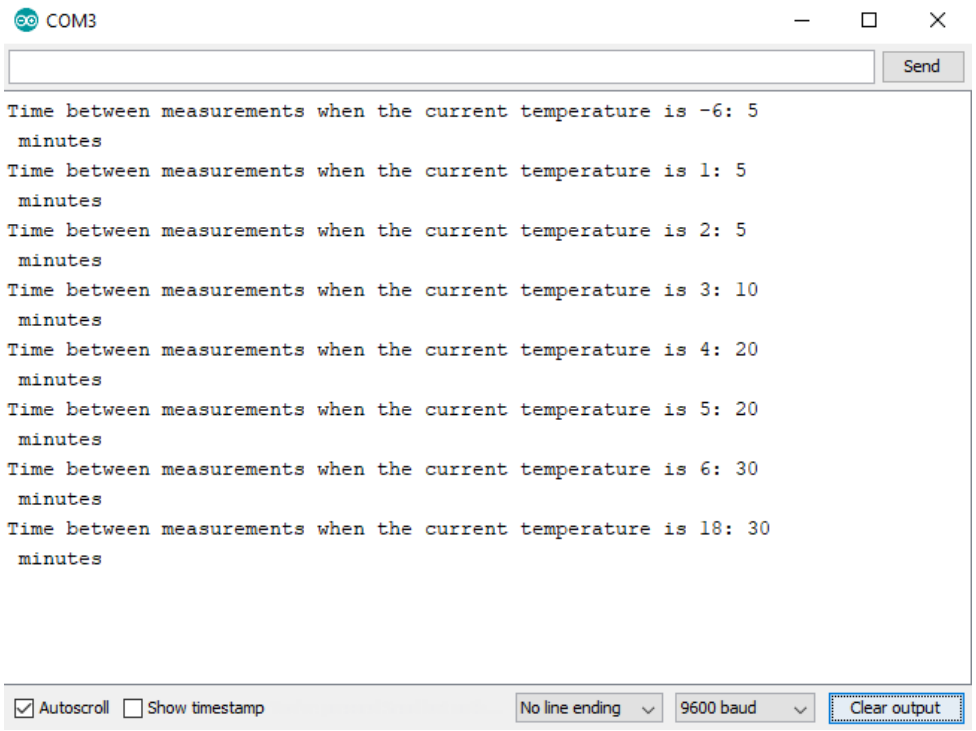


*Figure G.3: Results for the test function (Listing 2) written for the closestPeriod function.*

# Appendix H

# Software flowcharts

## H.1 Flowchart for power saving mode

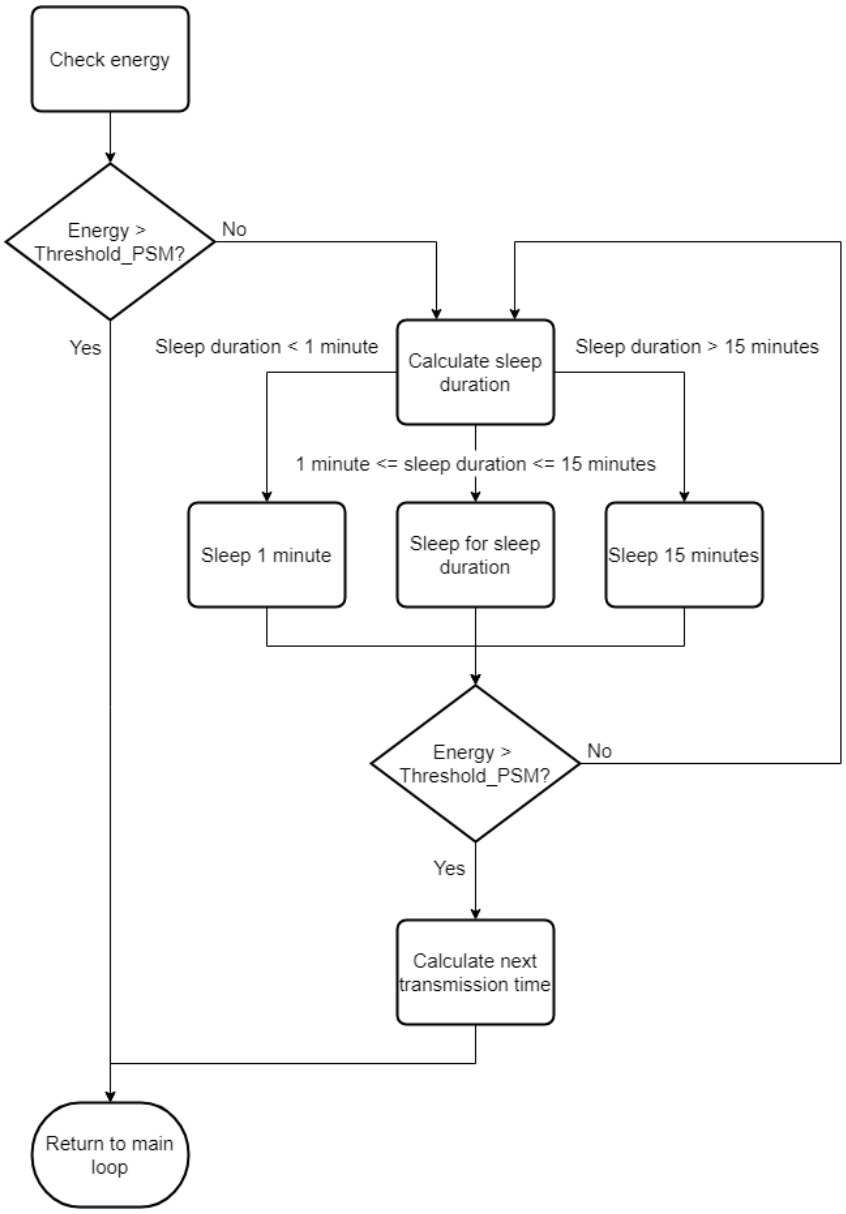


*Figure H.1: Flowchart of the software behaviour of the power saving mode*

## H.2 Flowchart for normal mode

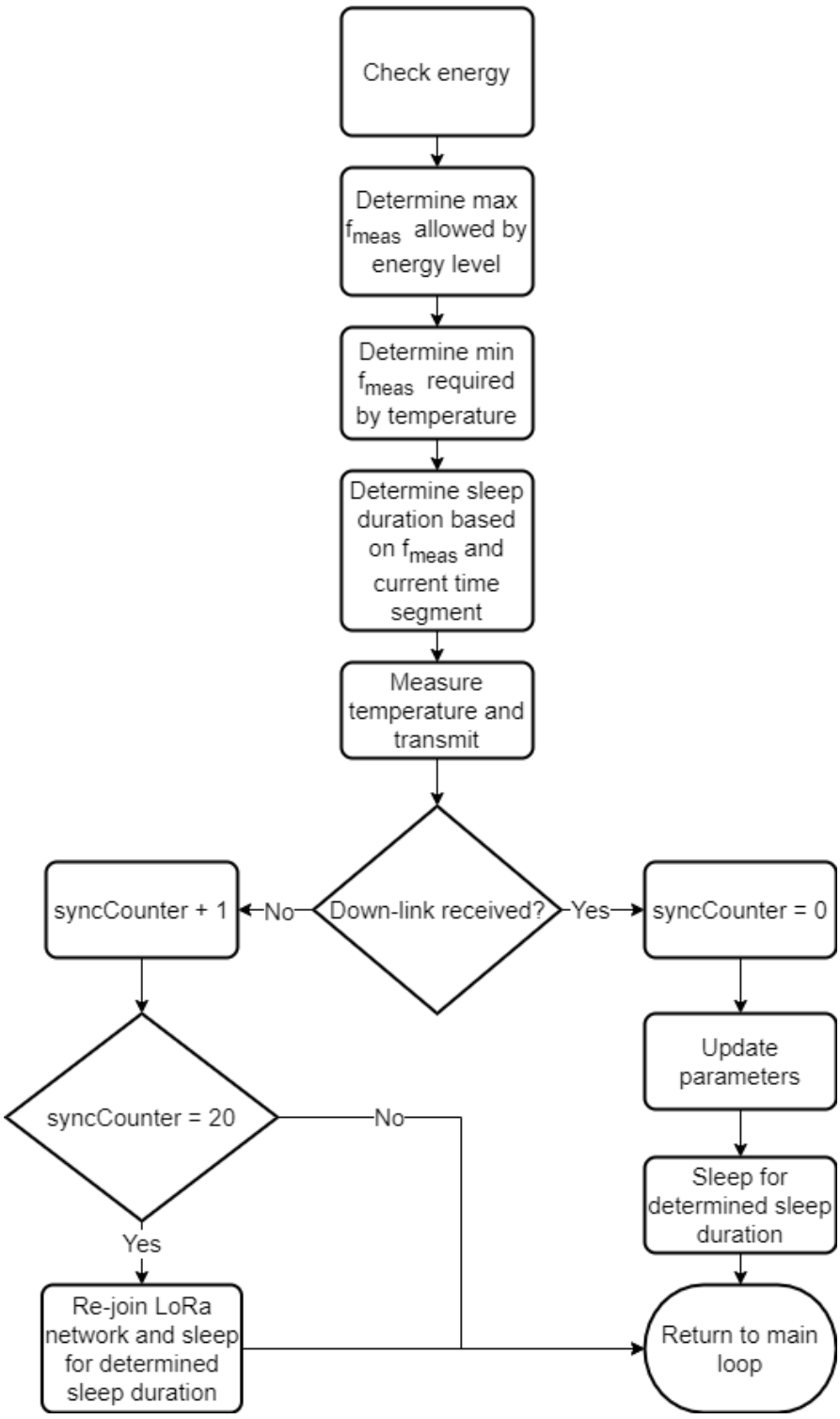


*Figure H.2: Flowchart of the software behaviour of the normal mode.*

## H.3 Flowchart for off-season mode

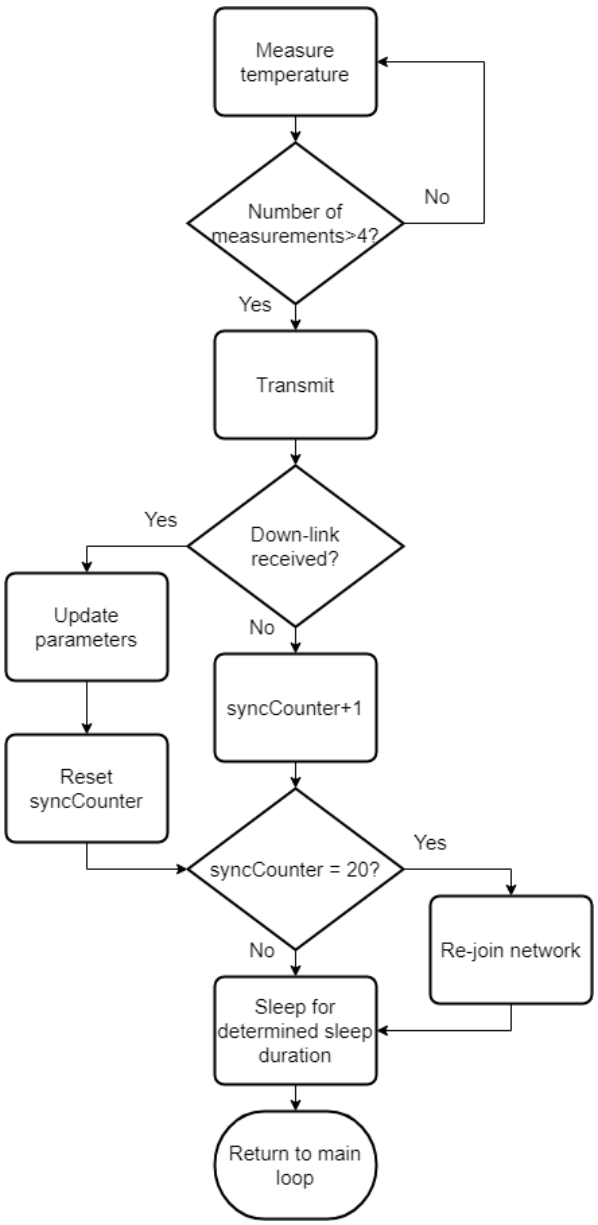


*Figure H.3: Flowchart of the software behaviour of the off-season mode.*