NTNU

Norwegian University of
Science and Technology

Wireless Surface Interface for Subsea Instrumentation

Ove Nesse

Master of Science in Cybernetics and Robotics
Submission date: June 2016
Supervisor: Jo Arve Alfredsen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

This report is the result of a master's thesis that was carried out during the spring semester of 2016. The thesis is a part of the 2-years master's degree programme in Cybernetics and Robotics. Apart from evaluation, this report is meant for those who will be applying the buoy controller platform, by providing an understanding of the design to allow further development of the design. It will for those provide a valuable insight.

It is expected that the reader has basic knowledge about electronics and serial communication interfaces, as well as knowledge about software design and the C programming language. Most of the information needed for the thesis have been found in the data sheets, on the Silicon Labs website, or provided from the supervisors.

Trondheim, June 21, 2016

_____

Ove Nesse

Acknowledgment

I would like to thank the following persons for their help and contributions during the work with this master's thesis:

My supervisor Jo Arve Alfredsen, who has helped guide me through this master's thesis, and provided me with tips and information.

My co-supervisor Artur Piotr Zolich, who provided valuable software in the beginning of the semester, acting as a basis for the further software development.

Jørn Iversen Efteland, who gave me software examples during the last days of the semester, to avoid extra time spent on unnecessary debugging.

O.N.

Sammendrag

Denne masteroppgaven ble gjennomført for å kunne tilby et forslag til design av en lavenergi bøyeplattform, for å lette på noen av utfordringene som oppstår i sammenheng med sporing av fisk, dens oppførsel, og dens migrasjonsmønster. Akustisk telemetri er en metodologi som gjerne brukes til å oppnå dette, som består av akustiske mottakere som blir satt ut under sjøoverflaten, sammen med små akustiske sendere som blir festet på fisken. Siden de akustiske mottakerne ikke befinner seg over sjøoverflaten, betyr dette at de heller ikke kan nås vha. trådløs kommunikasjon. Dette fører til at brukerne ikke har mulighet til å innhente data i sanntid fra de akustiske mottakerne, som igjen fører til at all datanedlasting fra mottakerne må gjøres manuelt. De akustiske mottakerne må da oppsøkes der de er utplassert, gjerne i en fjord eller elv, og data må lastes ned for én etter én mottaker. Dette fører til at datainnsamling gjerne er en tungvint prosess.

Bøyekontrolleren muliggjør datainnsamling fra den akustiske mottakeren, i tillegg til de andre implementerte enhetene og instrumentene som kan tilkobles plattformen. Dette bidrar til at bøyekontrolleren kan tilby funksjonalitet som ikke bare er tilknyttet en spesifikk anvendelse, som potensielt gir den et mye større bruksområde. Normal drift av bøyekontrolleren innebærer datainnsamling fra tilkoblede enheter og instrumenter, som deretter skrives til loggfiler på SD kortet. I tilfeller der flere bøyekontrollere anvendes i forbindelse med hverandre, vil nettverkskommunikasjonen som er implementert i bøyekontrolleren bli anvendt for å opprette et kooperativt nettverk. Vha. nettverket blir loggdata fra hver bøyekontroller gjort tilgjengelig, som da blir samlet inn av én bøyekontroller. Når denne kontrolleren blir utrustet med et GSM/GPRS modem, muliggjør den fjerntilgang til hele systemet, og fungerer derfor som en gateway for brukere som kobler seg til. Dette betyr at all loggdata fra hele systemet blir gjort tilgjengelig fra den ene bøyekontrolleren.
Det utviklede designet tilbyr i tillegg flere lavenergiløsninger, som for eksempel muliggjør enkel strømstyring for alle instrumentene og enhetene som kan tilkobles bøyekontrolleren, i tillegg til enhetene som finnes på kontrolleren. Intervaller kan defineres for hver enhet eller instrument, hvilket betyr at de kan slås av og på i henhold til de definerte intervallene. Dette fører til at de

instrumentene eller enhetene som ikke er i bruk, blir slått av eller satt i en lavenergimodus. I tillegg er mange av lavenergifunksjonene som finnes i mikrokontrolleren implementert i programvaredesignet.

Mange funksjoner har blitt inkludert i designet for bøyekontrolleren. De mest nevneverdige er kanskje tidssynkronisering og posisjonering som kan utføres vha. GPS modulen, nettverkskommunikasjon muliggjort av radiomodulen, datalogging på det flyttbare SD kortet, og fjerntilgang via kommunikasjonsgrensesnittet som er etablert på bøyekontrolleren. I tillegg er en separat solcellemodul konstruert, som tilbyr måling av strømforbruk og batterispenning, og lading av batteriet vha. et tilkoblet solcellepanel. Dette kan bidra til å øke brukstiden til bøyekontrolleren.

En felttest ble utført for å kunne verifisere noen av kravene som ble satt for bøyekontrolleren. Testen ble gjennomført ved bruk av to bøyekontrollere montert på hver sin bøye. Testen ble gjennomført med gode resultater, der funksjonaliteten som ble testet fungerte som ventet. Flesteparten av designkravene kan også regnes som oppfylte.
Bøyekontrolleren kan imidlertid ikke regnes som ferdig utviklet, da flere designforbedringer er anbefalt gjennomført. Dette kan for øvrig forventes på dette stadiet av en designprosess. Driften av den akustiske mottakeren bør også forbedres, noe som kan føre til større strøminnsparinger for bøyekontrolleren. Videre programvareutvikling anbefales også. Brukervennligheten og driften av fjerntilkoblingen til bøyeplattformen foreslås forbedret, da den implementerte løsningen gjerne kan tolkes som midlertidig og lite brukervennlig. Uansett vil kretskortene og programvaren som er designet fremstå som en solid plattform for videre utvikling av bøyekontrolleren.

Summary

This master's thesis was carried out to provide a proposal for a low power buoy controller platform, which could help alleviate some of the challenges that arise in relation to fish behaviour -and migration pattern tracking. Acoustic fish telemetry represents a methodology used to achieve this, which comprise of subsurface acoustic receivers, and small acoustic transmitters fitted on the fish. However, since the acoustic receivers are subsurface instruments, they cannot be accessed through radio communication. This prohibits the users to acquire fish telemetry data in real time, and the need of manual operation and data downloading makes the data much less accessible. When the acoustic receivers are located far out in a river or fjord, it means that these operations happen seldom.

The buoy controller platform enables data acquisition from the connectable acoustic receiver, in addition to the other implemented log devices and connectable instruments. This allows the buoy platform to provide functionality beyond a specific application, thus enabling a wider range of use. The basic operation of the buoy controller platform consists of collecting data from its environment and instruments, and utilizing the data logging functionality for storing the data. In cases where several buoy controllers are deployed, they make use of the network capabilities of the radio module to enable a cooperative network. Through the network, each buoy controller will provide its data logs, which is collected by the buoy controller gateway. When equipping the gateway buoy controller with a separate communication module, remote access to the system will be enabled, thus allowing data log download from the entirety of buoy controllers.

The design provides power-friendly solutions, such as the possibility to control the power to all connected devices and instruments, and to define intervals for when to power them on, or off. The design also makes use of many of the integrated power saving features of the microcontroller.

A great deal of functionality have been included in the design of the buoy controller platform. Most mentionable may be the time synchronization and positioning abilities using the on-board GPS module, network communication provided by the radio module, data logging capabilities

by use of a SD-card, and remote connectivity through the implemented communication interface. In addition, a separate harvester module provides power consumption measurements and the ability to connect a solar panel for operation over longer periods of time.

A field test was carried out to verify some of the design requirements for the platform, which consisted of using two buoy controllers mounted on separate buoy platform structures. The test provided positive results, as all tested functionality worked as intended. Most of the design requirements for the platform has also been considered fulfilled.
However, the buoy controller platform is still subject for design improvements, which may be expected in this stage of development. The operation of the acoustic receiver should for example be improved to allow a noticeable reduction in the total power consumption. Further development should also be considered, such as improving the user-friendliness and operation of the remote connection to the buoy platform. Regardless, the designed hardware -and software solution provide a solid basis for further development of the buoy controller platform.

Acronyms

PCB Printed Circuit Board

PV Photo Voltaic cell

MCU Micro Controller Unit

IMU Inertia Measuring Unit

BOM Bill of Materials

GPS Global Positioning System

UART Universal Asynchronous Receiver Transmitter

USART Universal Synchronous Asynchronous Receiver Transmitter

I2C Inter-Integrated Circuit

SPI Serial Peripheral Interface

GPIO General Purpose Input Output

LDO Low Dropout Regulator

MPPT Maximum Power Point Tracking

SMD Surface Mounted Device

IDE Integrated Development Environment

LFXO Low Frequency Crystal Oscillator

HFRCO High Frequency RC Oscillator

HFPERCLK High Frequency Peripheral Clock

RTC Real Time Counter

UTC Coordinated Universal Time

RTS Request To Send

CTS Clear To Send

BMP Buoy Message Protocol (Proprietary)

Contents

Chapter 1

Introduction

This master's thesis has been carried out to provide a proposal for a hardware -and software design for the buoy controller platform. The main objective has been to design a complete software solution based on the already designed hardware during the preliminary project. It should implement solutions for local data acquisition from sub-sea instruments, network capabilities to enable communication between multiple buoy controllers, and gateway functionality to enable remote access to the system and the available data logs. The design should also feature low power solutions, including energy harvesting, to accommodate operation over longer periods of time.

1.1 Background

To get a better understanding of fish behaviour and migration patterns, acoustic fish telemetry represents a methodology to achieve this, by equipping the fish with small acoustic transmitters. However, due to the acoustic receivers being subsurface instruments, they cannot be accessed through radio communication. At the present, no platform exist to utilize the acoustic receiver to provide remote access to the fish telemetry data. There exist numerous areas of application for such a platform, some of which will be elaborated here.

Sustainable management of fishery resources requires understanding of fish behaviour and how the fish interact with the environment they inhabit. By observing the fish populations in rela-

tion to their habitat, a better understanding of their migrations and distribution through space and time can be acquired. Acoustic fish telemetry provides the means to achieve this, though limitations prohibits the possibility to acquire data in real time. Manual download of data from the receivers are normal practice for receivers placed out in rivers and fjords, which contributes to time consuming data collection. As a result, the receivers may stay in the water for longer periods of time, thus wasting the benefit real-time updates would provide.
This would provide an obvious application for a surface buoy platform. By incorporating the acoustic receiver in the platform, fish telemetry data could be accessed. The platform could provide a communication interface, thus enabling remote users to access the fish telemetry data remotely through the communication link.

The fish farming industry faces several challenges related to salmon farming. To better understand the fish behaviour and their distribution in a fish farm net cage, acoustic fish telemetry can be applied. The acoustic receivers provide accurate time synchronization features, which enables the receiver to be used in positioning applications. By forming a grid of receivers in the net cage, the position of salmon fitted with acoustic transmitters can be determined.
However, shall the acoustic receivers provide accurate results, they must all be synchronized to a common clock. The most prominent mean to provide a common clock source, is by utilizing a GPS module.
Other positioning applications may as well be relevant, such as positioning of a underwater vehicle. The same principles will however apply in every application. To provide this functionality through the buoy surface platform will therefore enable a wider range of use, thus bringing it more value as a generic platform for various applications.

1.2 The Preliminary Project

The preliminary project was carried out during the fall semester of 2015. It comprised the hardware design of the buoy controller platform, which included the design of two PCBs.
The first objective of the preliminary project was to select appropriate components for the design; everything from small passive components, to the main components such as the MCU or

the power supply ICs. Energy efficiency was carefully considered when selecting the components, as low power solutions are essential when designing a system for battery operation. This resulted in power saving design elements such as MCU controlled power switches for some of the peripherals.
The second objective required the development of the buoy controller design, including schematics and PCB layout.
To accommodate a wide range of use for the buoy controller platform, a great deal of interfaces and peripheral devices was included in the design. A energy harvester solution was designed as a separate module, which lets the module be upgradable or removable if the need should arise.

Three copies of each PCB was ordered, to act as a test -and development platform for later software development. However, no testing of the hardware was carried out, and had to be postponed. In chapter 3, the hardware design from the preliminary project is described, to provide a better understanding of the implemented components and the hardware design.

1.3 Design Tools

During the preliminary project, the hardware was designed by using the Altium Designer software. It was used for both the schematics and the PCB design, which provided professional tools for creating good solutions for the design.
Since a Silicon Labs microcontroller is implemented, it made sense to make use of the development tools available from Silicon Labs. Simplicity Studio provides a development environment with great debugging tools, which are valuable during the design process.

1.4 Objectives

The main objectives of this thesis is to:

1. Propose a hardware design for a generic buoy platform implementing interfaces for underwater instrumentation, wireless communication, data storage, positioning, energy harvesting, and power management.

2. Design a dedicated PCB that will serve as the main control unit for the platform.
3. Propose a software design which implements all of the features needed to fulfill the design requirements specified in section 1.5.
4. Carry out a test program to verify the design.

This master thesis report includes both the hardware -and software design of the buoy controller platform, and consequently does the objectives involve both the preliminary project and the master thesis.

1.5 Design Requirements

The detailed requirement specifications are as follows:

1. The buoy controller should be regarded as an autonomous unit, meaning no remote interaction should be needed for normal operation.
2. The system should be easily scalable, meaning that small, if any adjustments are needed to add or remove nodes to / from the system, up to a certain limit.
3. Both the hardware and the software should be designed on the premise of low power consumption, to enable operation over a minimum of six months while powered by battery and solar energy.
4. All on-board -and connectable devices needed for basic operation should be implemented in the software design.
5. All of the nodes in the system should incorporate features to enable communication through neighboring nodes to a "master node".
6. A communication interface should enable a separate communication module (GSM/GPRS, Iridium) to be connected, to let the master node act as a gateway for remote users.
7. The software should be based on a fail-safe design, meaning any errors that may occur during operation, will be handled by the buoy controller itself, if possible. Any major errors must be reported to enable the users to handle them accordingly.

8. Each node should be able to acquire data from connected instruments and sensors that produces data, and store them locally. This data should be accessible to the master node, which again should be made accessible for remote users of the system through the gateway.

1.6 Structure of the Report

The rest of the report is organized as follows. Chapter 2 provides an overview of the buoy controller design. Chapter 3 describes the hardware design of the preliminary project, to give an overview of the implemented components and solutions. Chapter 4 describes the software design in all aspects; from the high level design, to the detailed descriptions of each software module. Chapter 5 provides the power consumption estimations for the system, as well as solar radiation -and solar panel calculations. In chapter 6, the field test, requirement evaluations, and assessment and summary are presented. Chapter 7 suggests design improvements and recommendations for further work. In the appendices, some software modules and calculations are described, in addition pin maps, connector pin configurations, and modifications and adjustments. In the enclosed electronic appendix, the listed files can be found:

- Data sheets for the components used in the hardware design.
- Updated Altium project files for the hardware design.
- Updated Gerber fabrication files for the two dedicated PCBs.
- BOM (bill of materials).
- Updated schematics and assembly outputs of the hardware design.
- Software project files (Simplicity Studio) and C files.
- A power consumption estimation excel worksheet.
- An example of a user configuration file.
- A TCP client/server application used for testing.

Part I

Design

Chapter 2

Design Overview

The resulting design of this master's thesis will be applied as a controller unit on a floating buoy platform. The buoy controller may have various applications, where it can be applied as a stand-alone node, or part of a larger buoy controller network. Independent of application, the basic operation of the buoy controller will consist of collecting data from connected instruments and devices. The data will be locally stored on a SD-card, where a data logging system manages log files for each connected instrument and device that produces data.
Embedded in the buoy controller, a radio module provides radio connectivity by implementing mesh network capabilities by use of the built-in TinyMesh protocol. This enables several buoy controller nodes to be connected together in a mesh network, thus allowing a single buoy controller to act as a gateway controller for the other buoy controller nodes. The gateway node implement functionality that allows it to acquire log data from the other buoy controller nodes in the network, to act as a centralized access point that can provide log data and status information from all buoy controller nodes connected to the same network.
To enable remote access to the gateway controller, a GSM/GPRS modem has been implemented through a communication interface on the buoy controller, which lets the buoy controller connect to a remote host, thus enabling easy access to the entire system through a single connection to the gateway controller.

The design presented in this part encompass both hardware and software. The hardware design was developed in the preliminary project and covers the chosen components, the schematics

and the PCB design. The developed software utilizes the hardware and together presents a complete solution for a low power generic platform for data acquisition. Together with the communication capabilities, the numerous interfaces, and the integrated peripheral devices, a highly configurable platform is provided.
The software design implements most of the devices integrated in the developed hardware, as well as the acoustic receiver TBR700[1] and the GSM/GPRS modem.

The peripheral devices integrated on the main PCB, such as the radio -and the GPS module, are considered to be applied in most applications of the platform, whether it is applied in a system for data acquisition, or some sort of positioning system. The GPS module provides a highly accurate clock, which is essential in positioning systems. A time reference for the system clock also proves valuable where precise timing of events is needed. The radio module provides a master-slave mesh network allowing a master node to communicate and collect data from slave nodes. The acoustic receiver TBR700 is considered a main component on the platform, even though it is not a part of the hardware design. Communication is enabled by a RS485 transceiver that lets data be sent in real time to the MCU. Together, these components forms the foundation of the platform, including the flash based SD-card storage.

Both the hardware and software have been designed to best satisfy the low power demands for the system. A battery powered system means limited power resources, and while quite powerful batteries can be acquired to extend the operational time, a power efficient design assures long operation of the system. A separate energy harvester module provides a solar panel input making it possible to extend battery life, or even make the system self-sustainable on power.

Most of the software components are designed to be generic and reusable for other applications, and ensures that mostly higher level application software needs to be developed / extended to adapt the platform to another application. The design is also highly configurable, allowing functionality to easily be adjusted, removed, or just switched off.

[1] www.thelmabiotel.com

Chapter 3

Hardware Design

This chapter will describe the hardware design of the preliminary project. All of the implemented peripherals and power solutions will be elaborated to provide a better understanding of why the components were included, which features they present, and what they provide to the buoy platform.

In the block diagram in figure 3.1, an overview of the hardware design is illustrated. All of the integrated main components are illustrated, as well as the harvester module (harvester module below the "harvester module interface"). The sketch does not include all of the connections that are present in the design, rather it shows the main communication interfaces and main components that was integrated. In the enclosed files, the detailed schematics of the hardware design can be found.

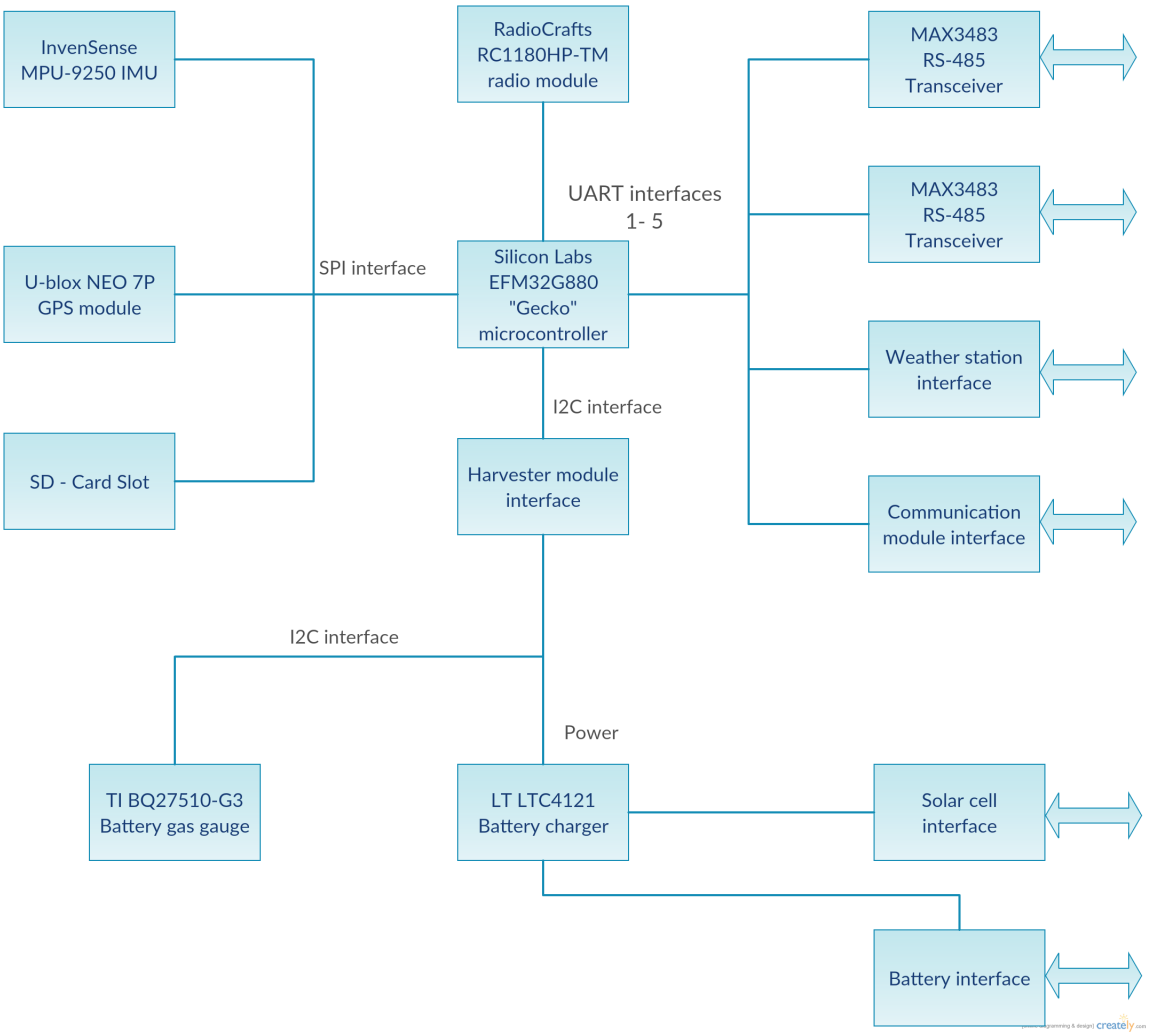

Figure 3.1: High level overview of the hardware design.

3.1 Component Description

3.1.1 Microcontroller

The MCU was chosen due to its beneficial power saving features, and the available number of interfaces needed for the peripherals and interfaces on the controller. Figure 3.1 shows the need for five UART interfaces, one SPI interface and one I2C interface.

The *Silicon Labs EFM32G880* featured the required functionality of the buoy controller, and was therefore chosen for the design. It also include valuable functionality such as low energy modes,

a real time counter, Low energy UART, direct memory access etc. The MCU is based on the ARM Cortex M3 core with maximum frequency of 32 MHz. Regardless, only a LFXO of 32.768 kHz was implemented in the design to supply a stable clock for the RTC. The *HFCLK* is supplied by the built-in 28 MHz *RCO*.
The available flash program memory is 128 kB, and the available RAM is 16 kB. This proved to be more than sufficient for the implemented design, leaving the larger part of the memory free for feature expansions. In appendix B.1 all of the connections to the MCU are listed.

3.1.2 Peripherals

The hardware design encompasses several different peripherals, and several interfaces to other modules and devices as well. As part of the hardware design process, these devices had to be selected to best fit future applications. A part of this process was also to select the passive components needed for operation. More information beyond what is provided here, can be found in the enclosed data sheets.

GPS

A requirement for the design was to include a GPS module. To have an on-board GPS module turns out to be an excellent decision due to its many features, and the possibilities they enable. In many applications, accurate positioning of the floating buoy proves to be very beneficial, especially if the buoy is floating freely. An even more favourable feature is however the ability to use the GPS as a provider of a very accurate clock.

The *U-blox NEO-7P* module was chosen for this purpose. It features several low power modes, an accurate time pulse output, and a high precision positioning feature which enables an accuracy of 1m or less.

The GPS module has also been coupled with a MOSFET power switch, as described in section 3.1.4. This means that the power to the GPS module can be completely shut off through the software by use of a GPIO pin on the MCU. This proves to be a good solution, as the GPS module does not have a low power shut down mode. The design also requires an active GPS antenna

to be used, which increases the power consumption of the GPS. The antenna power output is directly connected to the GPS module power pin, and may not be controlled internally by the GPS module.

The GPS module has got three major operational modes: continuous, cyclic, and on/off. Each mode has different ranges of use, and with different power consumption. These modes may however not need further consideration if the power switch is used. As can be seen in the enclosed schematics, the GPS module has got a backup power supply pin, which is always connected. This enables the GPS module to let the internal *LFXCO* be running when the GPS is powered off, thus keeping the GPS module's clock and position reference active, to allow a "hot start" when powered on.

Wireless Network Communication

A most prominent feature of the buoy controller is the ability to allow communication between neighbouring buoy controller nodes. The idea of the buoy controller platform is to enable this communication using a integrated radio device. The integrated *Radiocrafts TinyMesh RC1180HP* module provides a fully implemented mesh network topology for wireless communication between the buoy controllers.
The mesh topology provides a way of communication different from the usual star topology, or node-to-node topology. The TinyMesh protocol extends the mesh topology and provides several features implemented in the radio modules. The modules are for example configurable to act as three different types of devices: gateway, router and end. One TinyMesh network will in normal operation only allow one module to be configured as a gateway for each network. The gateway is special in the way that all messages sent from it is received by all of the other devices in the network. The router devices have the ability to forward messages to nodes that do not have direct contact with the gateway device. All messages sent from either the "router", or "end" devices in the network will on the other hand only be received by the gateway device.
The radio module is also a high power version, allowing extended range over the non-HP modules. When combining the use of HP modules with the mesh network topology, the range of the network can stretch over long distances, only limited by the distance between each module.

IMU

Since the buoy controller is a floating unit, some applications may involve measuring the movement of the buoy in the sea. The *InvenSense MPU-9250* IMU was therefore integrated in the design. It is a simple energy efficient chip providing 9 degrees of motion. The IMU has however not been implemented in the software design of this thesis and will therefore not be elaborated any further.

RS485 Transceivers

The integrated *MAXIM MAX3483* RS485 transceivers provides two half duplex RS485 network interfaces to the buoy controller. Two transceivers was integrated to enable communication with two different devices, where one of them may implement the Modbus protocol(specifically the AquaTroll multiparameter instrument). They also implement driver/receiver enable to control the data direction.

3.1.3 Interfaces

The buoy controller is designed to interface with several different modules and devices. A communication interface is implemented to enable the communication capabilities of a separate module, such as a GSM/GPRS modem or a satellite modem. This fulfills part of the design requirement, which states that the buoy controllers must be able to not only provide local radio communication, but also allow remote connections from users on-shore. This enables control and data acquisition for the user, and thereby widens the usability of the system greatly. Since the modem is not integrated in the design, the use of a modem is optional and can be implemented if the application requires it.

The harvester module interface enables features which may not be paramount for every application, and may therefore be omitted. The harvester module designed in the preliminary project adds the possibility to charge the battery using a solar panel, and to measure the voltage and current flow from the battery.

A weather station / UART interface was also added. This interface has not been utilized in the software design, nor is it investigated if any weather station would fit the system. The interface simply exposes a UART interface and a selectable voltage of either 3.3V, 5V or 12V using a jumper selector on the main PCB.

The SD card slot lets the MCU connect to a SD card for storing measurements and other data. The SD card storage provides a good solution since the card is swappable, which lets the user connect the card to a computer and read or write data if needed. The size of the storage can also easily be changed by installing a larger SD card.

Two RS485 interfaces was implemented to provide connections to the RS485 transceivers. In addition to exposing the RS485 interface, a integrated power supply can provide 12V on the interface, being able to provide enough power for the connected device. A MOSFET power switch was added for each interface to provide simple on/off control, bringing power saving features if needed. In Appendix B.2, pin maps of all of the connectors are listed.

3.1.4 Power Supply

The power supply was designed to handle the peak current consumption of the system. The power supply is situated on the back side of the main PCB, where three DC-DC converters supply the system with 3.3V, 5V and 12V. Most of the components and peripherals on the buoy controller require the 3.3V supply, while the 5V is accessible only through the communication interface and the weather station interface. The 12V supplies the two RS485 interfaces if needed.

In table 3.1, all of the main components are listed, showing their peak current consumption at the defined voltage. The RS485 interfaces were introduced in the design to accommodate the subsurface instruments *TBR700* and *Aqua Troll*, and they are therefore listed for power supply dimensioning purposes. The 5V power supply was dimensioned to support the *Iridium 9602* satellite communication unit, or a GSM/GPRS modem. The chosen DC-DC converters was oversized to ensure that they would handle the current consumption of the system. All of the values in table 3.1 have been extracted from the data sheets for the specific components.

Table 3.1: Max current consumption and required voltages.

Component	Peak (max) current	Voltage
Silicon Labs EFM32G880	20mA(estimated)	3.3V
Radiocrafts TinyMesh RC1180HP	560mA	3.3V
U-blox NEO-7P	60mA	3.3V
SD Card(Transcend 2GB)	50mA (estimated)	3.3V
MAXIM MAX3483	2∗2mA	3.3V
Invensense MPU-9250	4mA	3.3V
Aqua Troll Multiparameter instrument	32mA	12V
TBR700 Acoustic receiver	Unknown	12V
Optional communication module	1.5A	5V

5V supply

The *Iridium 9602* and a GSM/GPRS modem were reference components for the design of the 5V power supply, and their peak current and maximum allowable voltage drops have therefore been investigated. Common for this type of communication is that it requires high peak currents during small time slots when transmitting. The power supply therefore had to be dimensioned to handle these peak currents, while maintaining the voltage within a specific range. The *MAXIM 1708* was chosen, which is a powerful 5V Boost DC-DC converter able to handle continuous currents of 2A at 5V, and large current transients without large voltage ripples. Several different options were evaluated, such as using a smaller Boost DC-DC converter with large output capacitors to provide current for the fast transmit transients. However, due to large aluminium electrolytic capacitors, this solution would occupy larger parts of the PCB, in addition to increasing the cost, and was therefore discarded.

3.3V supply

The system is designed to be powered by a single cell Li-Ion/LiPoly battery, which has a nominal voltage around 3.7V. However, it may vary between 2.7V-4,2V, depending on battery charge. The power supply for the 3.3V therefore had to be able to handle voltages above- and below the output voltage. This meant that a Buck/Boost DC-DC converter had to be acquired for the purpose. The *Texas Instruments TPS63000* is a powerful DC-DC converter that can deliver up to 1.2A of continuous current in step down mode (buck), and up to 0.8A in step up mode (Boost). This means that it will be sufficient to power all of the 3.3V components whether in step down- or

step up mode. It also provides a built-in power save mode for times when the power consumption is low.

12V supply

The chosen 12V power supply was not designed, but instead a complete voltage regulator was chosen. This was mostly due to time limitations during the preliminary project. The regulator will however serve the purpose, providing at least 200 mA at the output voltage of 12 V, and input voltage of 2.7V. It may however be considered to be replaced with a more integrated solution in later revisions of the buoy controller.

Power switches

Several of the main components included in the design of this master's thesis does not have an available low power sleep mode. To satisfy the low power design requirements, power switches had to be integrated. Simple P-Channel MOSFET power switches was therefore integrated in the design, allowing complete shut down of the connected components. This feature was integrated for the components (and connected instruments) that did not offer a low power shut down mode:

- GPS module
- SD card
- 12V regulator
- Weather station interface
- Both RS-485 interfaces (underwater instruments)

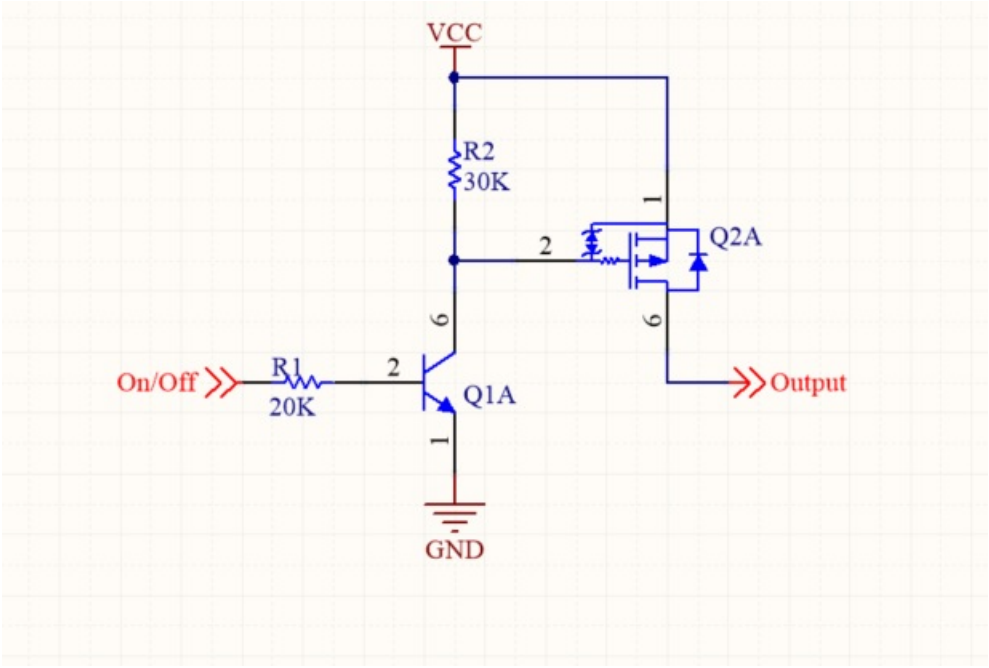

Figure 3.2: Illustration of the power switch connection.

The figure above shows how the power switches are connected. They are also found in the enclosed schematics. NPN transistors were included as pre-switches to ensure that the voltage at the Gate pin of the P-channel MOSFETs are pulled to the Source pin voltage when in off state. This ensures that the MOSFET is totally shut off (blocking). The resistor values was chosen to make sure that the transistors and MOSFETs are saturated.

3.1.5 Energy Harvesting

The buoy controller platform is going to be battery powered, and to prolong battery life, a solution for harvesting of solar energy was developed in the project. There are a few energy harvesting solutions available on the market, where an ultra low power unit incorporates several functions on one chip. These functions are normally battery charging, Buck- and/or Buck/Boost DC-DC converters, and an input prioritizer to select the input to use depending on how much energy is available (solar- or battery input).

Solar cells also have a maximum power point, which means that maximum efficiency is reached at a certain voltage. The energy harvester chips therefore have a built-in MPPT (Maximum

Power Point Tracking) algorithm, which measures the solar cell voltage and regulates the throughput to keep the voltage at the maximum power point. Other components that incorporates this feature also exists, such as MPPT battery chargers, and MPPT DC-DC converters.
It was however decided that none of the all-in-one chip solution would suffice. This was due to the maximum currents they could provide, which was not near as much as the peak currents of the system. A simple alternative was therefore established, which exists of a battery charger able to harvest energy from solar cells.
The *Linear Technology LTC4121-4.2* was selected for the purpose, which is a buck charger that handles a wide input range from 4.4-40 V. It provides a constant charge voltage of 4.2V, thus only allowing one-cell Li-Ion or Li-Po batteries to be connected. It can deliver up to 400mA of charge current, which is a considerably higher charge current than for the low-power harvesting chips mentioned above. The charger starts and stops charging depending on the battery voltage. A possibility to shut down the charger has also been added via the battery gas gauge. This option lets the gas gauge shut down the charger if for instance the battery temperature exceeds a set limit (if a battery with a thermistor is used).

3.1.6 Power Consumption Measurement

An embedded system with power efficiency requirements will benefit from having the possibility to measure the power consumption of the entire system. A battery gas(fuel) gauge IC is a unit that is specifically designed for low power applications where a battery is used as the primary power source. It measures the net current flow to the battery, which means that it measures both the current flowing in to the battery (charging), and out (consumption). Its registers are accessible through I2C, where data about current consumption, charging, battery charge etc. is stored.
The battery gas gauge *Texas Instruments BQ27510-G3* was therefore integrated in the design, more specifically on the harvester module where the battery is connected. An I2C interface is connected to the harvester module interface so that the MCU can communicate with it. The gas gauge also provides battery status monitoring and battery temperature monitoring (if implemented), amongst other things.

Chapter 4

Software Design

The software design employs the hardware developed in the preliminary project, which is described in chapter 3. As mentioned in the Design Overview, the software continues the low power generic design of the hardware solution. Several of the implemented software modules are highly generic and may just as well be applied in other applications without the need for redesign. Should software changes be needed, the design has been focused on making it easy to extend modules, or add new features that may be needed. Several features, such as an accurate system clock, time synchronization, an event scheduler, to name a few, are widely applicable.

In figure 4.1, a UML diagram of the design is presented. It provides an overview of all of the software modules in the design. However, not all of the dependencies between the modules are illustrated, which mostly concerns the *stream*, *fifo*, *system*, *err* and *shared* modules. Many of the other modules implement these modules, such as many of the drivers, the *com* modules, etc. The UML diagram has also been simplified, making it clearer and more understandable, only revealing the most important functions. The function arguments and return types have been omitted as well.
The rest of this section will be split into several subsections where each software module is described in detail. More detailed UML diagrams are also presented in each subsection.

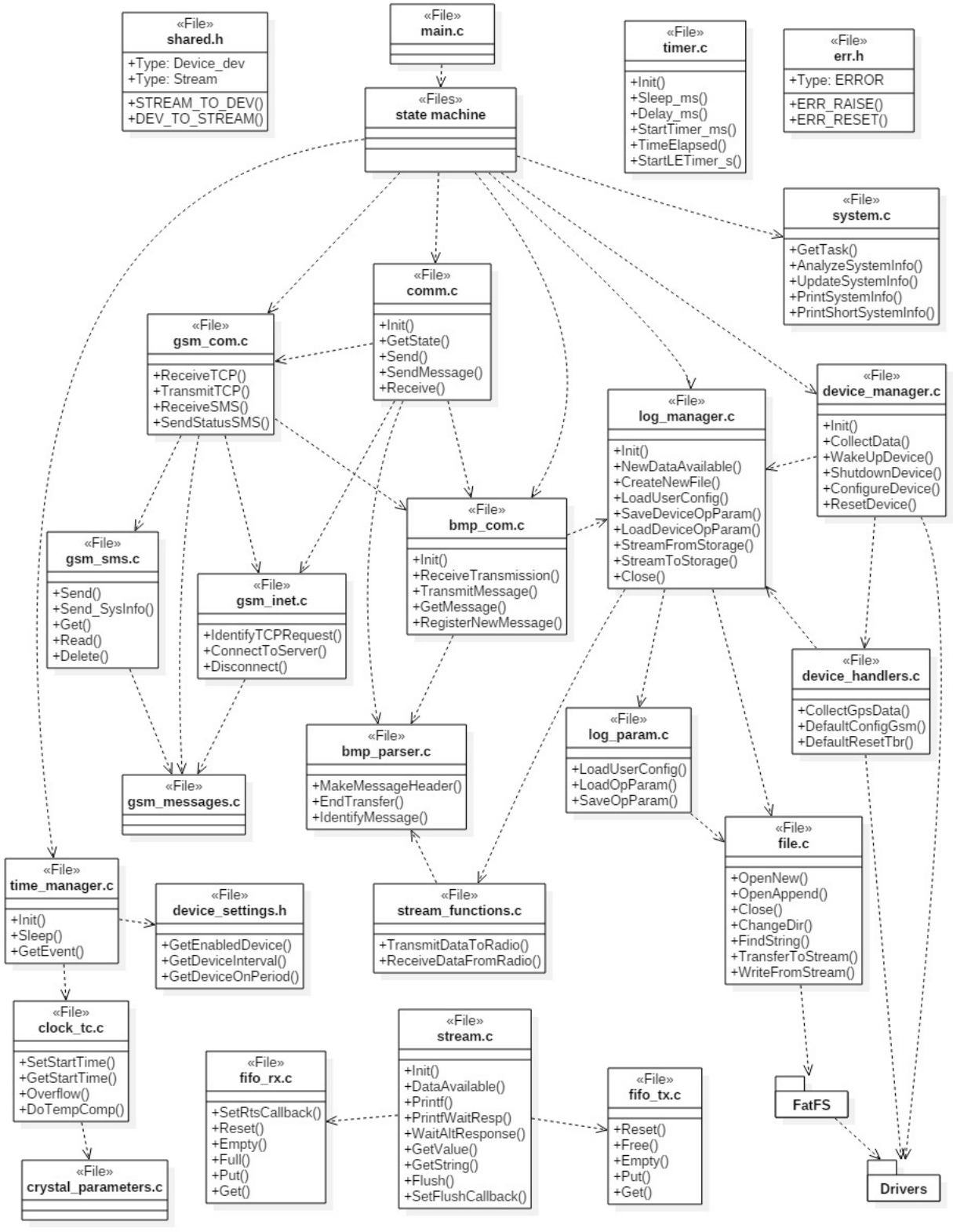

Figure 4.1: UML overview of the software design.

4.1 Application Flow

The software is built upon a interrupt -and task driven design, where the existing interrupt functionality of the MCU is utilized to enable the required operation of the system. By using the available energy management library, the MCU is operated as energy friendly as possible, being led into a low power energy mode when there are no pending tasks. A state machine is implemented to move the system between different states of operation as to which tasks are pending. In section 4.2, the main state machine is further described. In figure 4.2, a flow chart illustrates the state machines operation in a simplified form.

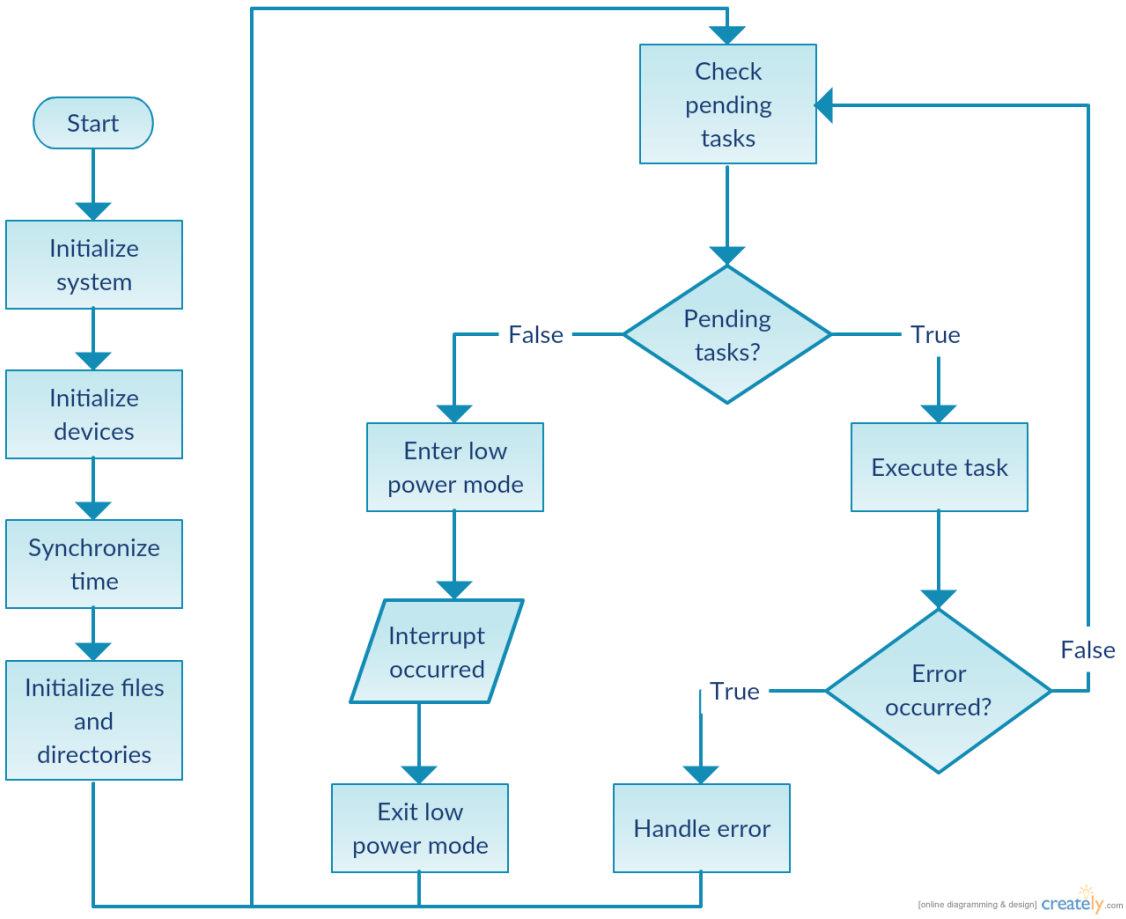

Figure 4.2: Simplified flow chart of the state machine.

4.1.1 Application Flow Examples

In figure 4.3, an example of a typical event triggered action is illustrated. While the illustration has been simplified a bit, it still shows the flow of the system when incoming data from the radio module causes an interrupt to trigger the MCU out of one of the lower energy modes (typically EM1). The MCU will thereafter continue execution of code from where it left off before it went to "sleep", and the state machine will transition to the "check states", where all of the possible interrupt sources will be checked.

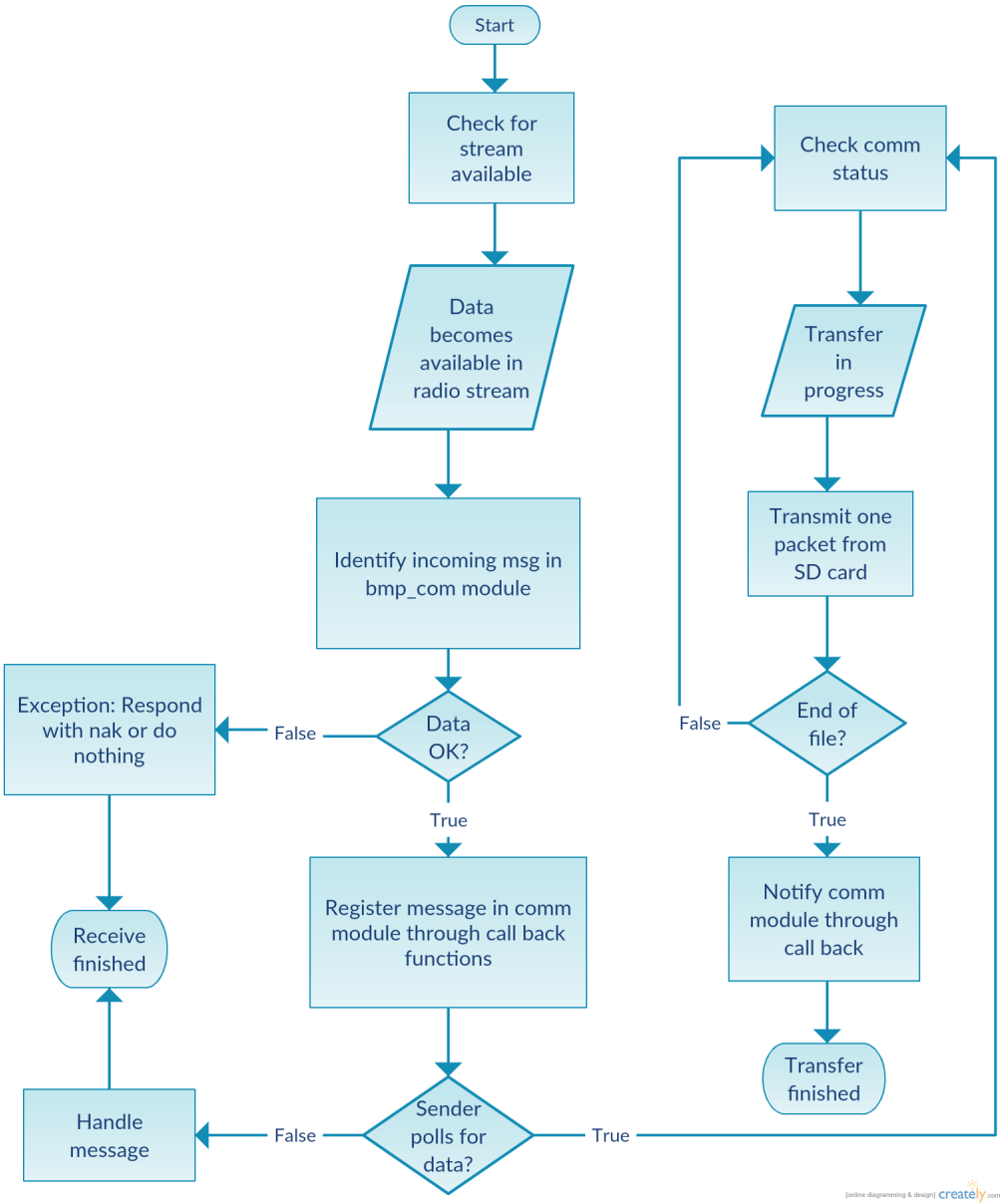

Figure 4.3: Flow chart for receiving a message.

The state machine will next transition to one of the "com" states, where the appropriate module will be invoked to identify the incoming data. In this case, the *bmp_parser* and the *bmp_com* will be required to identify the message. If the received message is a "poll" message, it means that the sender asks the receiver for data. The execution therefore ends and returns to the "check states" where the return value from the *comm* module's *COM_GetState()* has been changed to "transfer in progress". This leads to the transition to the "transmit data" state, and the *bmp_com* module is again accessed, but this time to transfer the data specified in the received message.

Since the TinyMesh protocol allows for a max packet length of 120 bytes, the transmission will stop after transmitting maximum 120 bytes. It may also stop earlier, due to the fact that the outgoing stream function for the radio module is designed to only transfer complete lines of data. The reason for this is that the gateway node will have to handle incoming data from several buoy controller nodes at the same time. The data logging design also defines that all incoming data for the same log device will be written to the same log file. To avoid incomplete, and thereby unusable data, each packet will therefore only contain complete data.

4.2 Main State Machine

The main state machine is designed in accordance to the STATE pattern, which implements the state machine by use of functions for each state and state transition, rather than the use of a switch case pattern. This lets the state machine be scalable, since it doesn't grow out of proportions as it can be distributed over several files. The design also encapsulates each state, and lets the behaviour of the state machine be controlled by the state transition functions. New states and transitions can easily be added to extend or change the functionality of the system.
The state transitions is configured to always lead the system into the "check states", where several modules are polled for pending tasks, incoming data, or any other conditions, before going into a low energy mode. In figure 4.4 a more detailed illustration of the state machine is shown. In the implemented design the first state is the "check stream" state, which is seen as the top state in the figure.

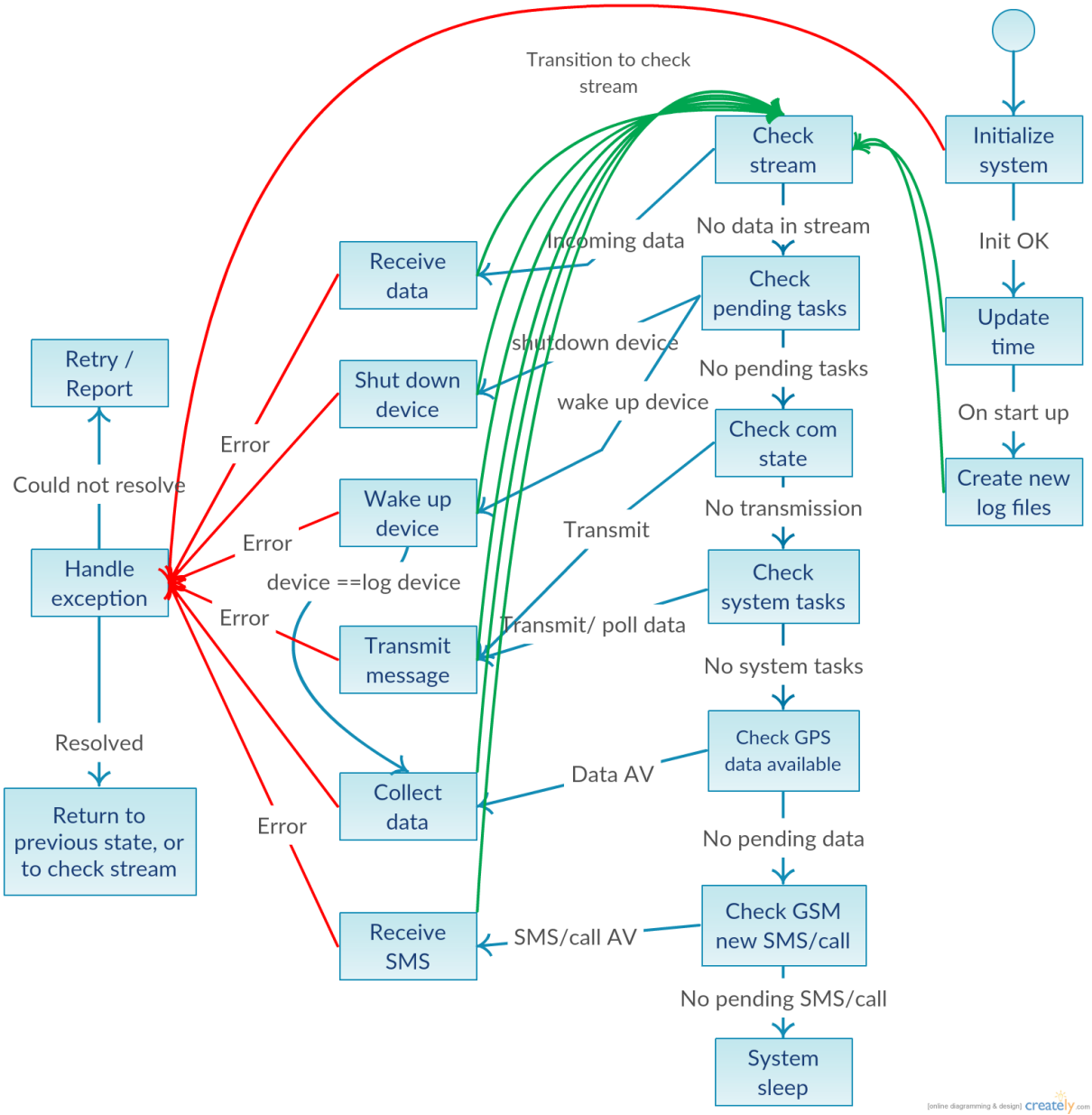

Figure 4.4: State chart of the main state machine.

4.2.1 States and Transitions

The state machine is "partitioned" into several source files, which is one of the advantages with the STATE pattern. The "check states" includes all of the states and transitions relevant to checking for pending tasks, polling for available data in stream, available data in device TX buffers, and so on. The "update states" includes all of the states and transitions relevant to making up-

dates. This can for example be to synchronize the system clock with the UTC time provided by the GPS module. The "device states" comprise all of the device related states, which for example can be "wake up" of device, or collecting data.
The "com states" encompass all communication-related states and transitions, which can be communication to/from any of the "stream" devices, which are the devices that are connected to one of the UART interfaces. This can be the radio, the GSM/GPRS modem, the TBR700, or any of the other stream devices, if implemented. The "init state" and the "exception state" are self-explanatory.

4.3 Time Manager

The time manager module has got two major functions in this design. As the name implies, one of the them is to provide an accurate system clock by utilizing the real time counter (RTC). The second function of the time manager is to provide a event scheduler system, which produces interrupts to wake the MCU from sleep when a specific event is scheduled to be executed, also driven by the RTC. The event scheduler can therefore be considered to be the major driving force behind the interrupt -and task driven design. Without it, the system would have needed to be operated in a totally different fashion, and in a much less power efficient one for sure. The event scheduler provides one of the most power saving features in the design, letting devices be powered on - and off at separate configurable intervals.

To drive these functions, the RTC provides a 32.768kHz counter by utilizing an external crystal oscillator (LFXO). This oscillator is available in three of the MCUs available energy modes: EM0, EM1 and EM2. This means that the RTC will run and provide a fairly stable counter when the MCU is operation in a low power mode, allowing it to be in the most power efficient mode possible at all times.

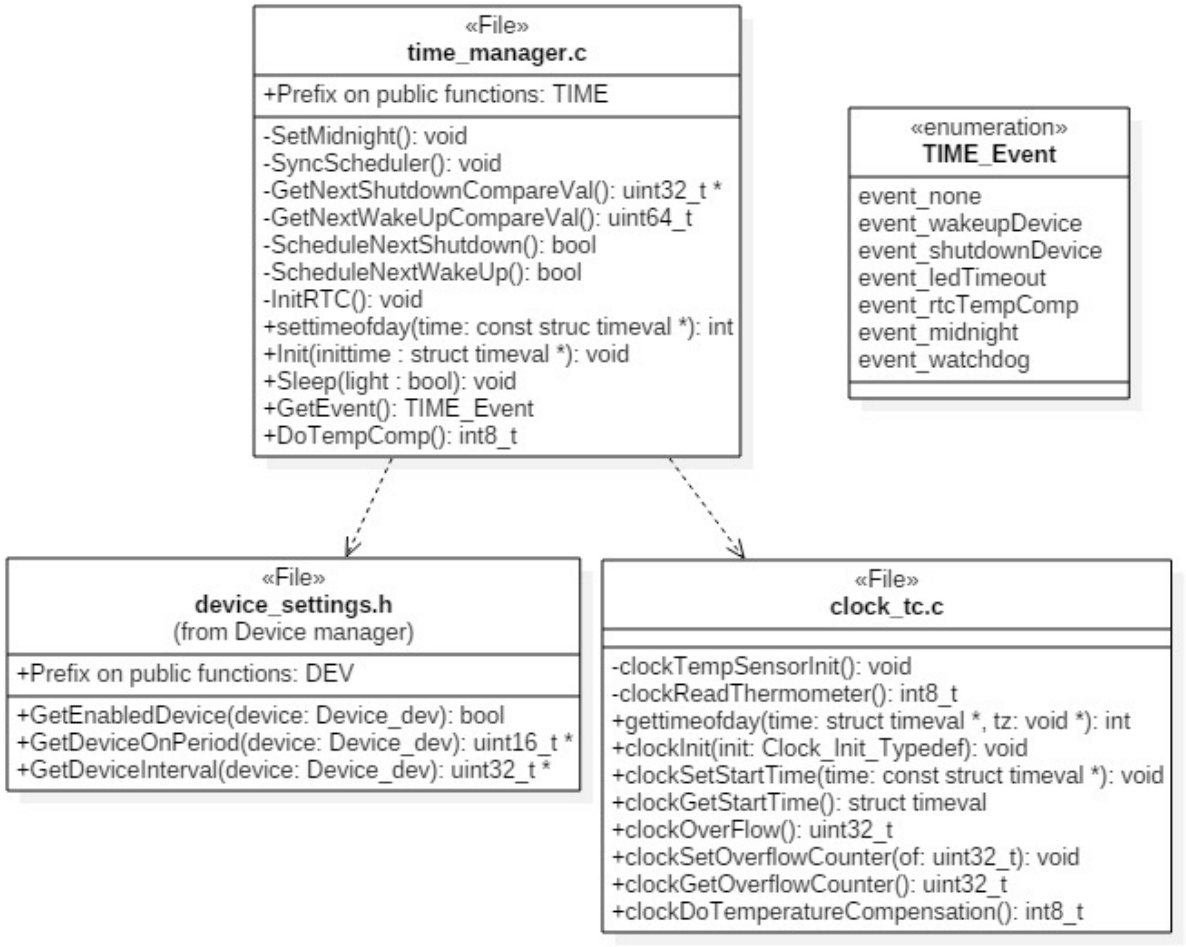

Figure 4.5: UML diagram of the time manager modules.

4.3.1 System Clock

A crucial design requirement for the buoy controller is to be able to keep track of time. This feature is utilized by several other modules in the design, and may be required by other applications of the platform as well. As mentioned in section 4.3, the hardware design includes a separate LFXO to provide better accuracy for the RTC. To be able to provide a valid time reference, the clock must however be coupled with the on-board GPS module. As mentioned in section 3.1.2, the most vital reason to include the GPS module in the buoy controller was the possibilities it gave by providing an accurate UTC time. This is highly required in application where timing is crucial, such as in applications where the acoustic receiver TBR700 is used for

positioning purposes (more on this in section 4.3.2). In such applications, the system clock may not suffice, and a highly accurate clock source can therefore be provided by the GPS module through the time pulse output. This feature is described in section 3.1.2, as well as in the module's data sheet. In this application, the time pulse is not utilized, but can easily be enabled if needed.

As seen in figure 4.5, the time manager includes the *clock_tc* module. This is a partially modified module provided from Silicon Labs. The module enables the needed clock functionality, and extends the GNU C libraries "time" and "sys/time" to provide the functions *gettimeofday* and *settimeofday*. These functions lets the system time be updated or retrieved in the form of a UNIX time stamp. The libraries also provides calendar functionality, which can be valuable for time stamping of data logs. The *clock_tc* module also provides a temperature compensation feature which uses the internal temperature sensor of the MCU to add a bias to the time output.
The resolution of the clock is adjustable by changing the prescaler division of the RTC. This can be adjusted from one tick per second, to the highest resolution of one tick per 30.5 us. All of the possible resolutions is however allowed in the application, which makes no reason to not use the highest resolution. More on this is found in the MCUs reference manual.

4.3.2 Time Synchronization

As mentioned in 4.3.1, most applications require some sort of clock reference to be able to allow time stamping, task synchronization, accurate timing etc. This system is therefore designed to provide an accurate UNIX time reference by utilizing the real time counter and the external LFXO, as mentioned in section 3.1.1. For applications where high accuracy is required, such as in positioning applications, the configurable time pulse output from the GPS module can easily be used to provide an more accurate counter. In the default configuration, the time pulse is configured to provide one pulse every second. This will however require that the GPS module is powered on and configured in one of the continuous modes (see the enclosed u-blox receiver description for more info), and may not always be a wanted solution due to system power limitations. However, the design provides the possibility to choose the solution that best suits the needs for the application.

4.3.3 Event Scheduler

The event scheduler was created to enable a low power operation of the system. To make the power demand of the system as low as possible, it is crucial that devices that do not need to be powered on, be shut off or put in a very low power sleep mode. Therefore, an event scheduler has been developed to enable this functionality, which is a major component in the software design. The scheduler employs the compare registers of the RTC to schedule the occurrence of the next events. Compare register 0 is used to schedule "wake up" of devices, while compare register 1 is on the other hand used to only schedule shut down of devices.
The schedulers basic operation is illustrated in the flow chart in figure 4.6. In addition to scheduling wake up and shutdown events for the devices, the scheduler can also schedule other events. This can enable different actions at specific intervals, or at specific times if configured so. By using the function *TIME_GetEvent*, the occurring task or event can be read.

To schedule the times at which the events shall occur, the scheduler accesses interval variables that contains the time in seconds between each occurrence of the event. In addition, for the devices, it also accesses variables containing the time in seconds they shall be active, before being put in sleep mode or shut down. The variables for the devices are accessed through the device manager, which holds the configuration for each device in the *DEV_Settings* structure. The miscellaneous intervals can be configured within the time manager, as well as new events easily can be added.

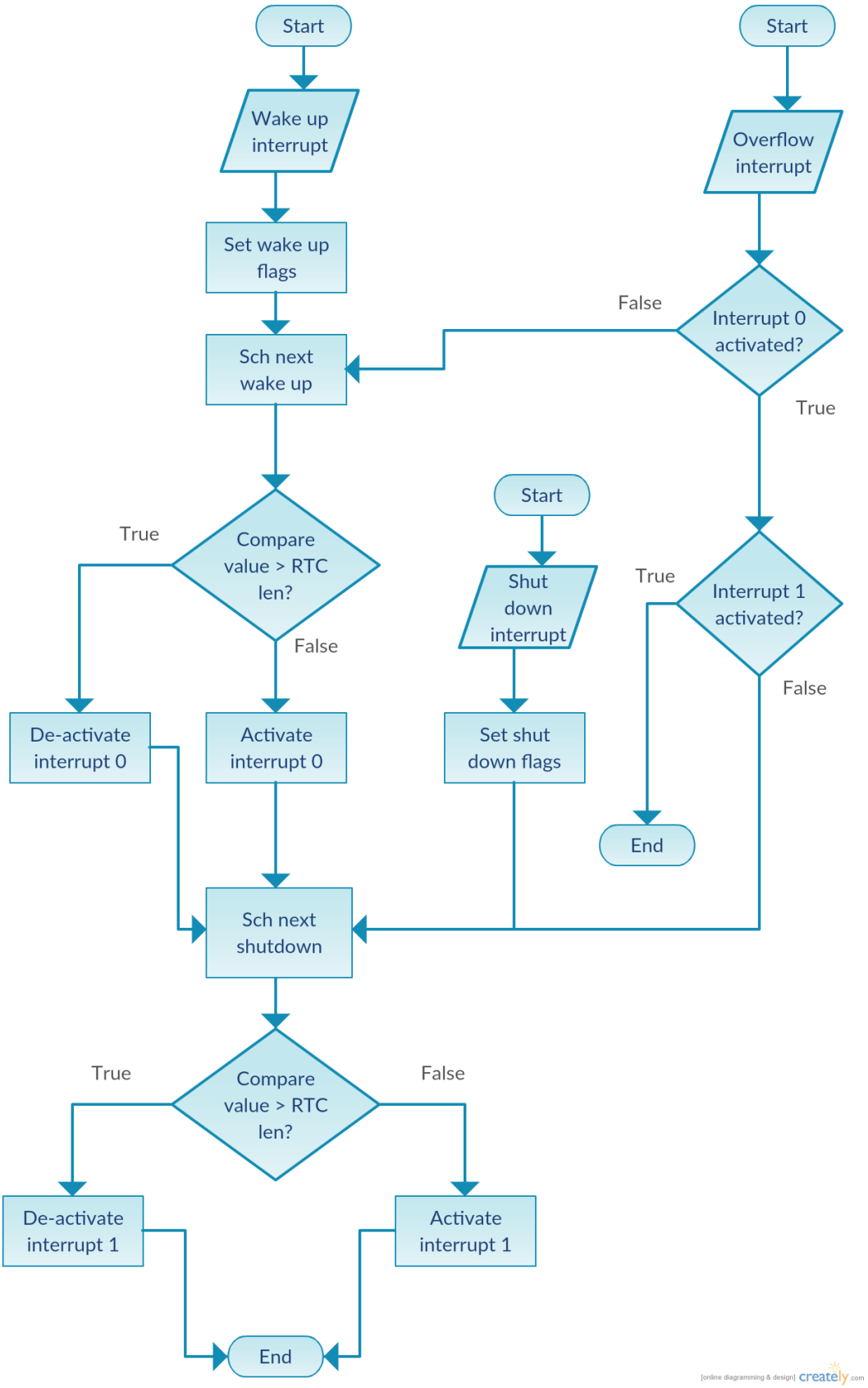

Figure 4.6: Flow diagram of the event scheduler.

The event scheduler is made up by several algorithms that calculates the next compare values to be set in the RTC compare registers. The RTC has got two compare registers, where one is used as a "wake up" compare register, and the other as a "shut down" compare register. The RTC value is constantly compared to these registers, and when one of them equals, an interrupt is generated.

$$W = \min \begin{cases} \min_{i=[0,n_{devices}\rangle} \left\{E(i) * I_{dev}(i) * C(i) - S_{comp}\right\} * RTC_{cps} - (OFC_{current} * RTC_{len}) \\ \min_{j=[n_{devices},n_{misc}\rangle} \left\{I_{misc}(j) * C(j) - S_{comp}\right\} * RTC_{cps} - (OFC_{current} * RTC_{len}) \end{cases} \tag{4.1}$$

$$S = \min_{n \forall A_{devices}} \left\{RTC_{start}(n) - O(n) * RTC_{cps} - \left(OFC_{current} - OFC_{start}(n) * RTC_{len}\right)\right\} \tag{4.2}$$

Where:

- W = The compare value to be loaded in the RTC compare register 0.
- S = The compare value for shutdown of device to be loaded into the RTC compare register 1.
- E = A Boolean variable describing whether the device is enabled or not.
- I = The interval for either a device or misc. event.
- C = The interval count for each device or misc. event. Keeps count of how many times the event has occurred.
- O = The time in seconds the device is supposed to be active.
- S_{comp} = The scheduler compensation for synchronization with the UTC time.
- RTC_{cps} = The defined value for how many counts the RTC make each second. Can be changed with a prescaler.

- $OFC_{current}$ = The current overflow counter value.
- RTC_{len} = The length of the RTC counter. In this case, the length is 24bit.
- RTC_{start} = The RTC counter value when the device was scheduled to "wake up".
- OFC_{start} = The RTC overflow counter value when the device was scheduled to "wake up".

With boundary conditions:

1. $I(n) > 0$
2. $RTC_{current} < W < RTC_{len}$

While W must be smaller than RTC_{len}, W cannot be limited to RTC_{len}. This would mean that the interval I had to be limited to a very short amount of time, depending on the resolution of the RTC, the RTC_{cps}. To overcome this, the overflow interrupt is utilized. This is illustrated in figure 4.6.
Equation 4.1 is implemented as an algorithm, where the interval count I_{dev} is incremented each time a device has a wake up event. That way, the total amount of time a device has been active is kept track of. When calculating the next compare value, the device that is closest to its next "wake up" interval is selected. If however a misc. interval is considered to be closer, it will be chosen instead. If two intervals initiate at the same exact time, both of them will be scheduled. In addition, the number of RTC counter overflows must be subtracted, since the interval count is never reset.

Equation 4.2 provides the algorithm for scheduling the shutdown compare values. Here, the RTC counter value at "wake up" of the current device is temporary stored. The next shutdown compare values is thereafter found by comparing all of the currently active devices. The one that is closest to its shut down value will be scheduled next.

The scheduler also includes a synchronization feature to enable synchronization to the system clock. By implementing this feature, all of the events will in practice happen at specific times

during the day, which is a major advantage when operating the radio module during short periods of time. This way, all of the nodes in the network will be scheduled to enable radio communication at the same time, thus enabling complete data acquisition between the network nodes during short time slots.

The synchronization feature is activated each time the system clock is synchronized with the GPS UTC time. What happens is that the number of seconds to midnight is calculated. A scheduler compensation value is thereafter calculated so that a radio "wake up" event is scheduled exactly on midnight. This value is added to the algorithms calculating the next "wake up" compare register values, which can be seen in equation 4.1 as S_{comp}.

Algorithm 1 shows the pseudo code for the synchronization feature. I is the configured interval for the radio module. $s_{midnight}$ represents the number of seconds to midnight.

Algorithm 1 Scheduler sync

```
1: procedure
2:     if "wake up" for radio then
3:         n ← 0
4:         while I * n < s_midnight do
5:             n ← n + 1.
6:         if I * (n − 1) − s_midnight >= I/2 then
7:             S_comp ← S_comp + s_midnight − I * (n − 1)
8:         else
9:             S_comp ← S_comp − I * n − s_midnight
```

Algorithm 2 shows the pseudo code for the adjustment algorithm, where the interval count C gets adjusted so that any changes in the scheduler compensation S_{comp} or the interval I gets compensated for. This keeps boundary condition 2 maintained.

Algorithm 2 Interval Count adj

```
1: procedure
2:     n ← 0
3:     for All enabled devices + number of misc. intervals; n ← n + 1 do
4:         while I(n) * C(n) * RTC_cps > RTC_current + OFC_current * RTC_len + S_comp * RTC_cps do
5:             C(n) ← I_count(n) − 1.
6:         while I(n) * C(n) * RTC_cps <= RTC_current + OFC_current * RTC_len + S_comp * RTC_cps do
7:             C(n) ← I_count(n) + 1.
```

4.4 Device Manager

The device manager module handles all of the contact with the device drivers. It acts like a middle manager module between the main state machine and the drivers, in that way encapsulating the communication with the drivers. This can be seen in the UML diagram in figure 4.1. The device manager provides easily understandable functions, such as *DEV_WakeUpDevice()*, or *DEV_ConfigureDevice()*. In addition to controlling the devices, the device manager also implements the log manager to enable access to the log files. This allows the *DEV_CollectData()* function to poll the devices for data, and to store it in the appropriate log files.

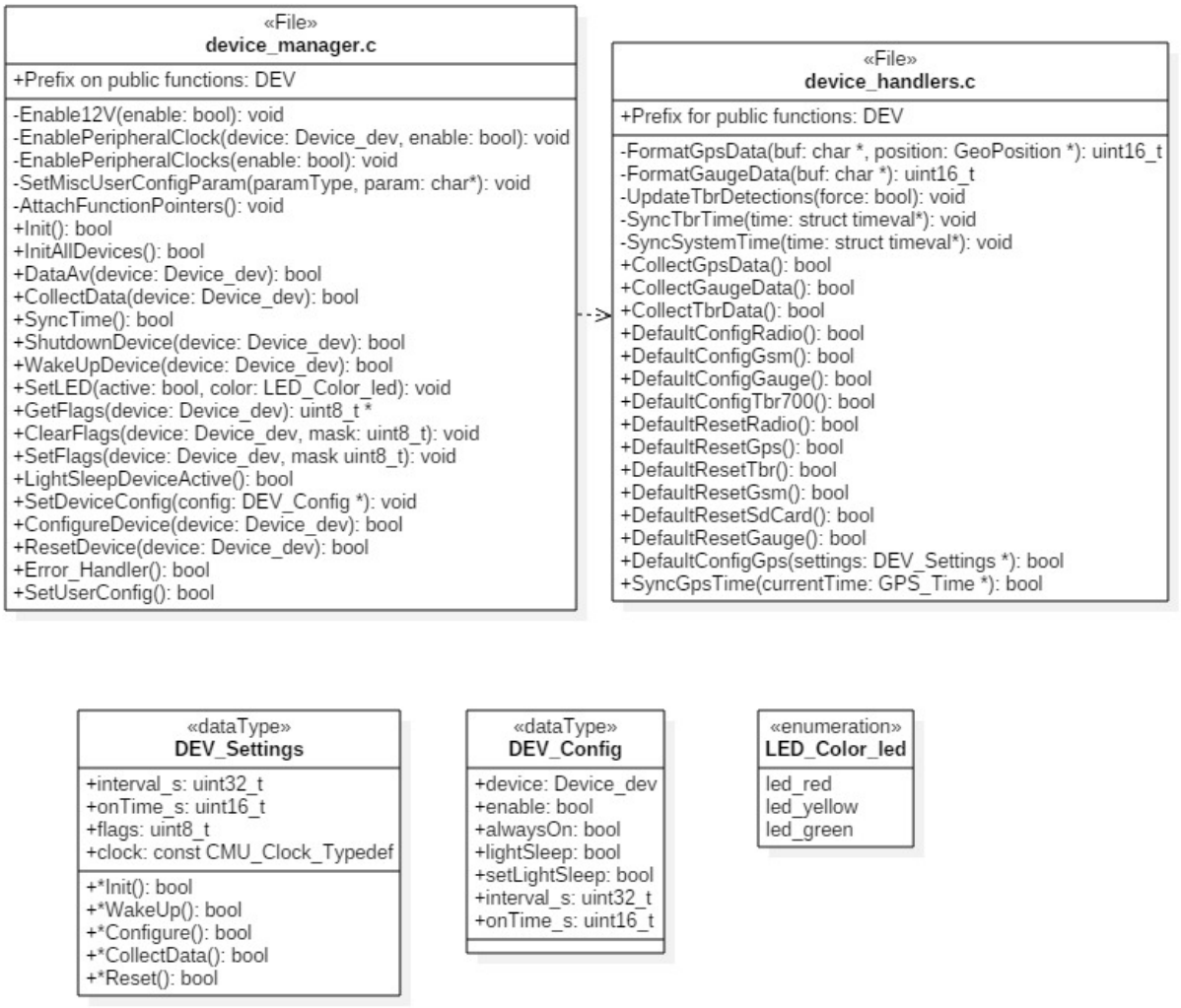

Figure 4.7: UML diagram of the device manager.

The device manager also handles all of the GPIO interrupts not related to any specific peripheral interface. The MCU incorporates two interrupt handlers for GPIO interrupts: one for odd numbered pins, and on for even numbered pins. These handlers has therefore been placed in the device manager, where appropriate functions can be called, or variables be set, when an interrupt is triggered. The device manager is also responsible for accessing the user configuration file on start up. This file is described in Appendix A.3.

The *DEV_Settings* structures also resides within the device manager, where the settings for each device is stored, such as the device flags, interval times and "on" periods. These are utilized by the time manager to calculate the next wake up -and shutdown events.
The *device_ handlers* module incorporates many of the device related functions, such as configuration functions, reset functions, etc. Since the functionality is split into functions for each device, it is easy to extend the module, or replace the functions with new ones if needed.

4.4.1 Device Operation Configuration

Through the *device_config.h* file, initial values for the *DEV_Settings* structures can be set. Specifically, the wake-up intervals, the active times, and the flags, which are used to define the operational modes for each device. Flag masks have been defined to make the configuration easy. For example, a specific device can easily be disabled if it will not be needed in the application. This will result in that the device will either be powered off or configured to a low power mode, depending on implementation, thus ensuring the most low power operation possible.

4.5 Log Manager

While the device manager controls the connected peripherals, the log manager is the module which enables easy access to the log files. It encapsulates the lower level *file* module, and provides functionality to support configuration files, and easy access to the log files.
The log manager keeps track of all the current log files that are used for data logging. The structure *LOG_Properties* holds information about current log files, last read log files, and log directories.

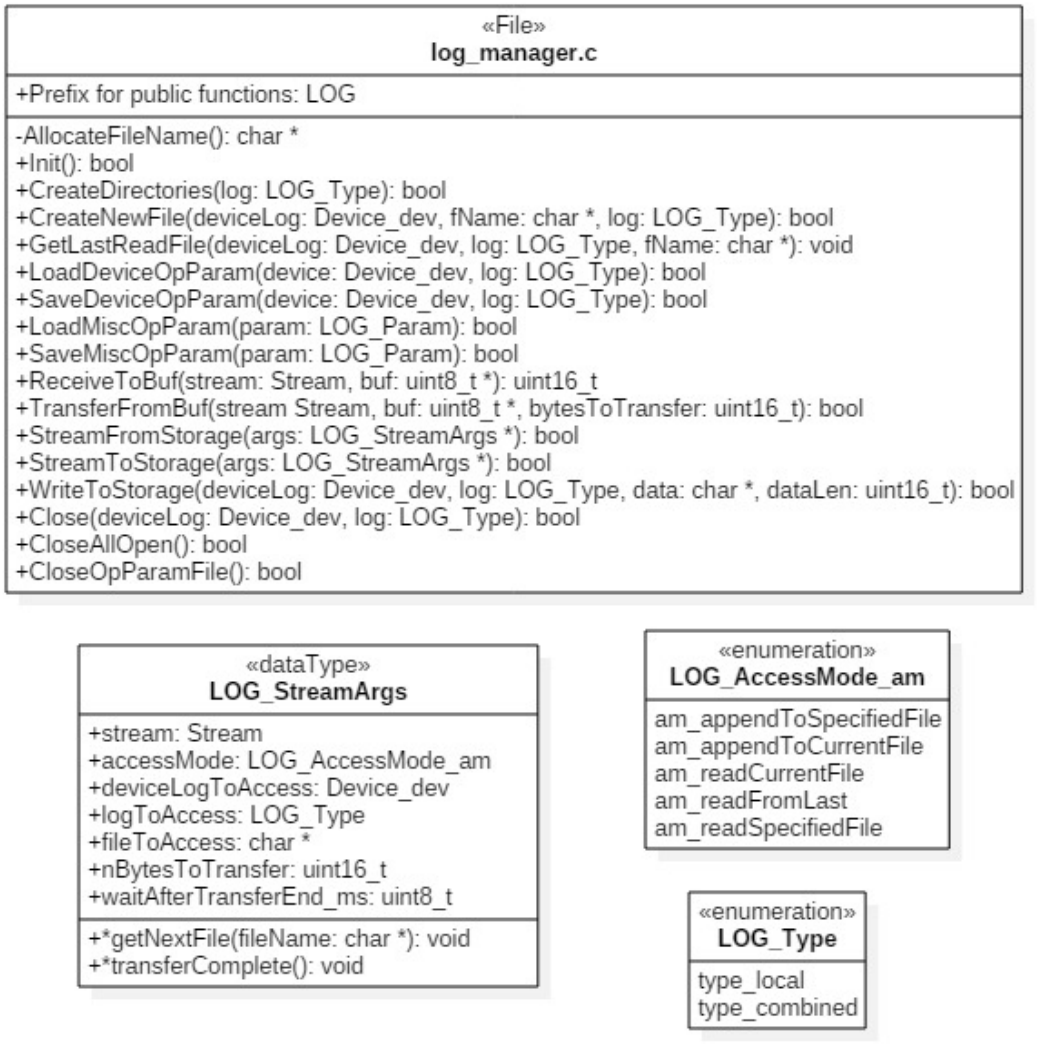

Figure 4.8: UML diagram of the log manager.

The data logging system is created to handle several log files for each log device. When a new log file is created, it will be assigned as the current log file. All write actions will thereafter happen to that file. On the other hand, when for example the network gateway node is polling for data, it may want to only access new data.

The log manager therefore keeps track of the file names of the last accessed files, and file positions at where it has currently read from. By calling on the *LOG_StreamFromStorage()* function with the specific access mode *am_readFromLast*, the log manager initiates data transfer from the particular file position, and reads to the end of file. For each data packet, the current file and file position is stored. When the end of file is reached, the log manager notifies the higher

layer module via the call back function pointer *getNextFile()* provided in the *LOG_StreamArgs* structure. This way, the actual implementation can be handled in a higher layer module.

4.5.1 Log Structure

The log files for each device log are stored in separate directories corresponding to the device name. This way, manual access to the log files is made more intuitive and understandable. At the parent level, the device log directories are put in a directory indicating that they are log files of the specific buoy controller, namely "LOC_LOG" (local log). The reason for this is that for the buoy controller that act as a gateway node, a new log directory will be created, named "COMB_LOG (combined log). The combined log will be structured in the same way as the local, only difference being that the combined log will contain the log data collected from the other buoy controller nodes in the network.

As mentioned in the start of section 4.5, a particular log file system has been created. The implementation of this system is fairly simple, and exists in the "update" state *CreateLogFiles*, and the *comm* module's implementation of the callback function *getNextFile()* from the *LOG_StreamArgs*, respectively. Another log file system can therefore easily be implemented by extending these modules.
The implementation depends on the *time_manager* module to provide a time stamp of the current date. The date is thereafter used directly to name the log file, along with two identifies: one for the device log, and one for the log type (local or combined). The *time_manager* is configured to issue an event at midnight, which leads to the *CreateLogFiles* state, and the creation of new log files for the coming day.

4.5.2 Configuration Files

To enable easy configuration of the system without having to make changes in the source code, the use of configuration files have been implemented in the log manager. This comprise of the public functions which includes *OpParam* in the name, and the *LoadUserConfig()* function.
To sorts of configuration files are implemented: One "user config" file, to enable some configu-

ration of the system, and a "operational parameters" file, which is used to store *LOG_Properties* parameters and other parameters that are considered important to store in a non-volatile memory. The latter file is created and managed by the *log_manager* itself, while the former must be created and edited by the user. In Appendix A.3 the configuration files are described in more detail, including the implemented configuration parameters.

4.5.3 File Access

The log files are accessed through the *file* module, which encapsulates the *FatFS* module. The *FatFS* is a generic FAT file system module freely available. The module has been implemented in the design by use of Silicon Labs own implementation, which includes a lower layer disk access module, and a SD card driver. The SD card driver has also been given some minor changes to accommodate the design of the lower layer SPI driver.
The *file* module provides standard functionality such as opening files, close files, change directory, and write and read to/from file. It also provides the stream functions which are accessed when file data is transferred over the mesh network, or via the communication module.

The module also implement functionality that enables several files to be open at the same time. The private functions *GetNewFileObj()*, *ReleaseFileObj()* and *IdentifyFileObj()* implements this features. The implementation allocates a certain number of *FIL* structures for the file system, and char arrays for the file names. When a calling function wants to open a file, one of these structures and char arrays are reserved for the file. The file name is stored in the char array, thus being kept track of. The next time a functions wants to open the file and it has not been closed, it will be identified as open, and the corresponding *FIL* structure is returned for file access. This solution does however require that every file name is unique, independent of what directory it is stored in, and must be taken note of if the file naming convention is changed.

The *file* module also incorporate an error handler, which is more like an interpreter for the *FRESULT* return values of the *FatFS* module functions. If an error occurs in the lower layer *FatFS* module, a corresponding error is raised via the *ERR_RAISE* macro described in section 4.8.

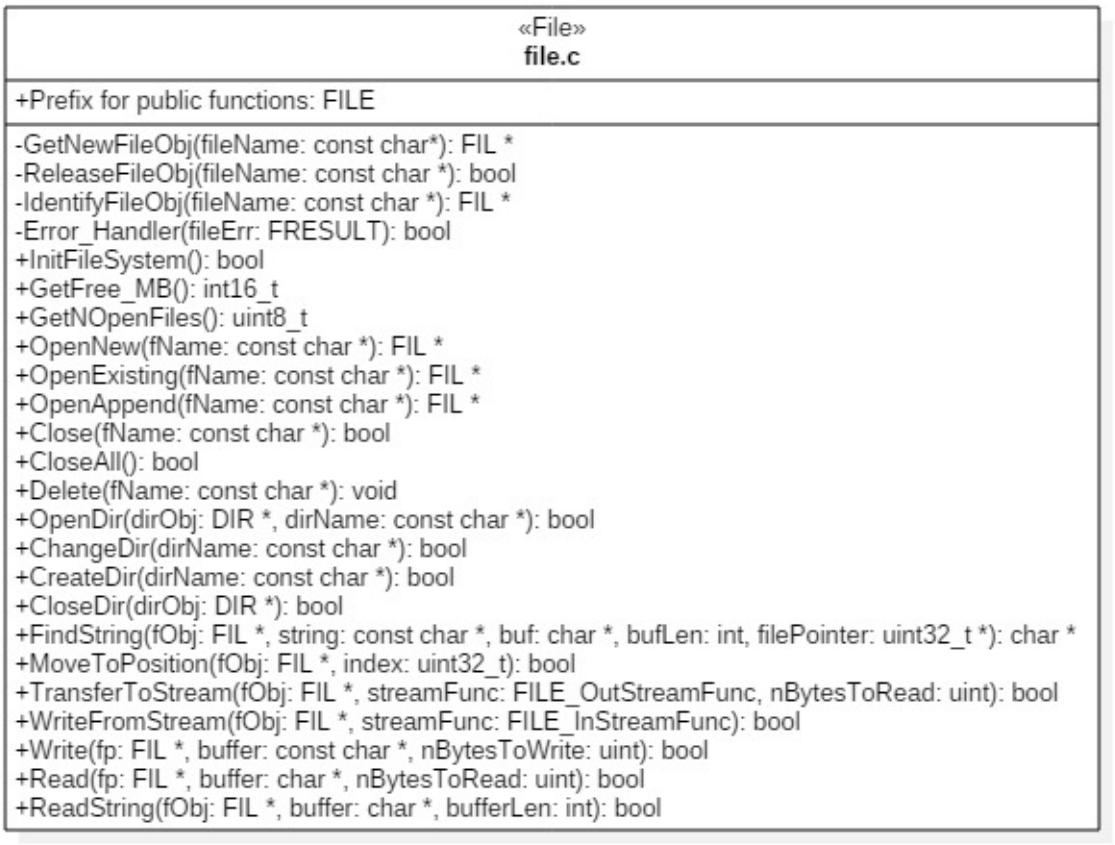

Figure 4.9: UML diagram of the file module.

4.6 Communication

Whilst there are a lot of different communication interfaces available on the controller, the communication modules encompass the communication via the radio module and the communication interface. All incoming -and outgoing data from/to these devices are buffered in two allocated FIFO queues, just as with the other devices that are, or will be connected to on the MCUs UART interfaces. In figure 4.10, 4.12 and 4.13 the underlying modules are illustrated. The higher level *comm* module encapsulates these modules and presents a single communication interface to be accessed. All message handling is based on the buoy message protocol, which means that if any other protocol will be used, such as may be the case with the GSM/GPRS module, another module will need to implement a conversion to the BMP. In the integrated solution

for the GSM/GPRS modem, the *gsm_com* module has been designed to handles this.

4.6.1 Buoy Message Protocol

A simple message protocol was included in the design to allow identification of each data packet sent to or from the buoy controllers. The protocol is described in Appendix A.1, where the structure of the packet header, including the available commands are illustrated. The *bmp_parser* module was designed to bring the BMP into the design. Its purpose is in effect to parse incoming data from a selected stream, and compare it up against allowed values. It also assembles the header of an outgoing packet in the correct format.
While the buoy message protocol was created for message identification and data transfer over the local mesh network, the protocol was easily identified to be applicable on any interface. The protocol therefore act as an application layer protocol, and is in the design applicable on any interface, which means that it can be used for communication via the integrated radio module, the implemented GSM/GPRS modem, or any other connectable communication module.

The radio packet size of maximum 120 bytes was taken into consideration when creating the protocol. This means that the header is kept fairly short with a length of 28 bytes when transferring file data (includes the file name), and 14 bytes when sending simple commands and updates. The header does not include a packet length field, instead a single "end of transmission" byte is added as the last byte, to signal end of packet. This allows streaming of data directly from the log files without knowing the total packet size before transmission starts. A length field would neither have any practical purpose, since the data is written directly to flash storage on the receiver side.
For most messages, no payload is added. This mostly applies to the *poll*, *update* and *response* access mode types. All *write* messages are however expected to have payload, but the total message size is ensured to never exceed the max packet size of the underlying transfer protocol.

The design of the *bmp_com* module ensures that the header and EOT char is added to every packet. While it may lower the throughput of the transmission line, it ensures that all of the transmitted data is identified and handled correctly as defined by the protocol. This can prove

to be crucial when the gateway controller is simultaneously receiving data from all of the connected buoy controller nodes.

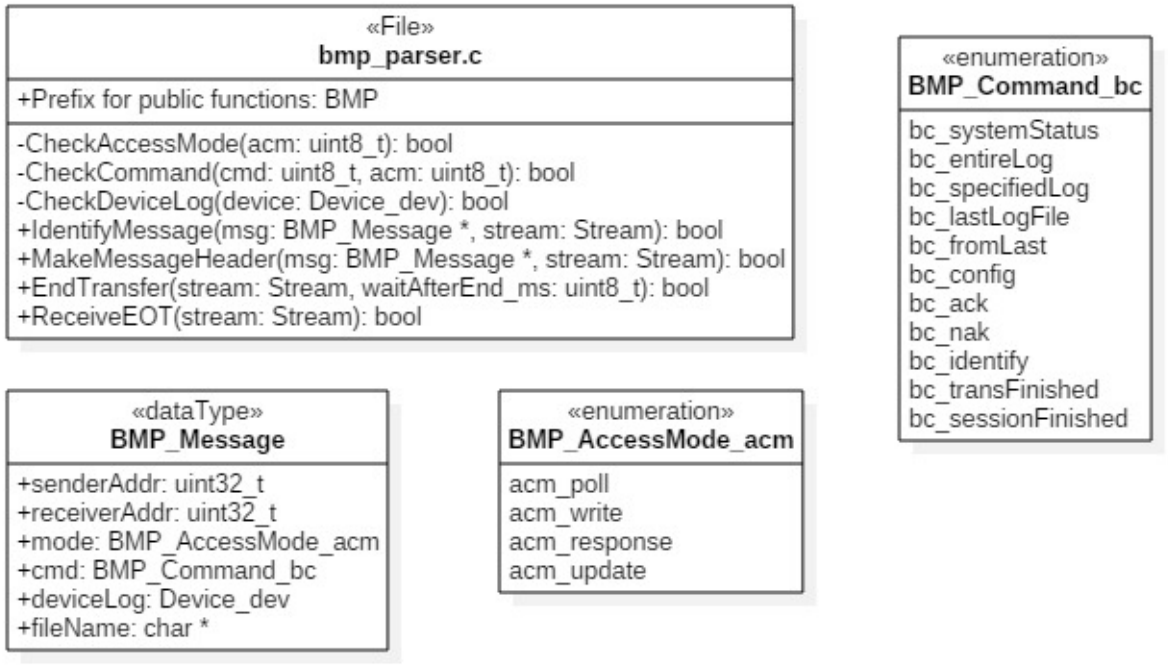

Figure 4.10: UML diagram of the BMP parser.

In figure 4.10 the *bmp_parser* module is illustrated in form of a UML diagram. The *BMP_Message* structure defines the content of the BMP header, and the two enumerations defines the type of commands and requests that can be issued. The *bmp_parser* can easily be expanded if the need for other commands emerge. In addition, the *bmp_com* module, which handles the BMP messages, must off course be extended as well to accommodate the new commands.

4.6.2 Data Transfer

How data is transferred from the buoy controller is heavily influenced by how the application flow of the system works, and the way the main state machine operates. An example of this is illustrated in the flow chart in figure 4.3. As described in section 4.2, the state machine implement a sort of task prioritizer based on the sequence the different "check" states. To ensure that the MCU is not "blocking" in the transfer state for a longer period of time, in case of larger

data transfers, the data transfer is stopped before the max packet size of the underlying protocol is reached. The state machine ensures that it returns to a "check" state to receive the next task. This way, other tasks with higher priority are ensured to be executed before the transfer is resumed, reducing any task execution delay to a minimum.

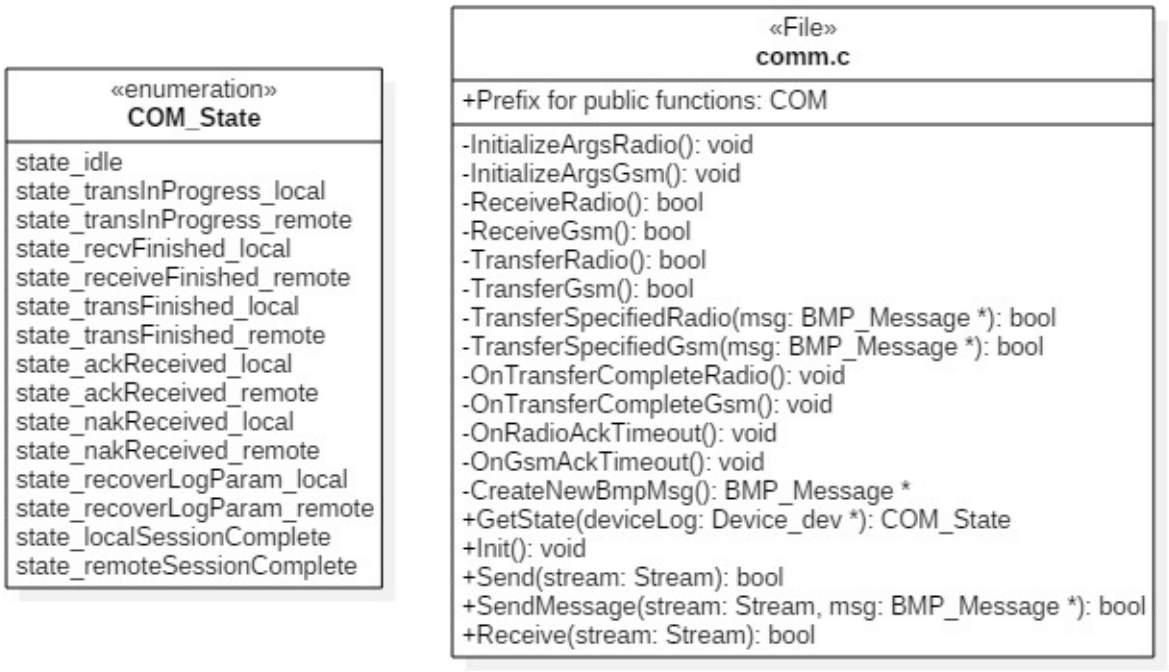

Figure 4.11: UML diagram of comm module.

The *COM_GetState()* function of the *comm* module provides the current communication status of the system. It returns an enumerator of the *COM_State* enumeration, which is illustrated in figure 4.11. The returned value represents the commands of any incoming -or outgoing BMP message. For example: the enumerator *state_transInProgress* informs that a transfer from this buoy controller has started, and to continue the transfer, the *COM_Send()* function must be called. When the data transfer is finished, the lower level log manager will inform the *comm* module, which result in a change of state to *state_transFinished*. Most of the communication state changes are implemented by the buoy message protocol in the *bmp_com* module, by analyzing the incoming BMP messages. In the *gsm_com* module, incoming TCP requests are analyzed in respect to the *GSM_TCPCmd_tcmd* enumeration from the *gsm_inet* module illustrated in figure 4.13.

4.6.3 Message Handling

The *bmp_com* module has been designed to handle the outgoing and incoming BMP messages. This module depends on the *bmp_parser* -and the *log_manager* modules, which allows it to read -and write BMP messages, and to access the log files specified by the outgoing, -or incoming BMP message.

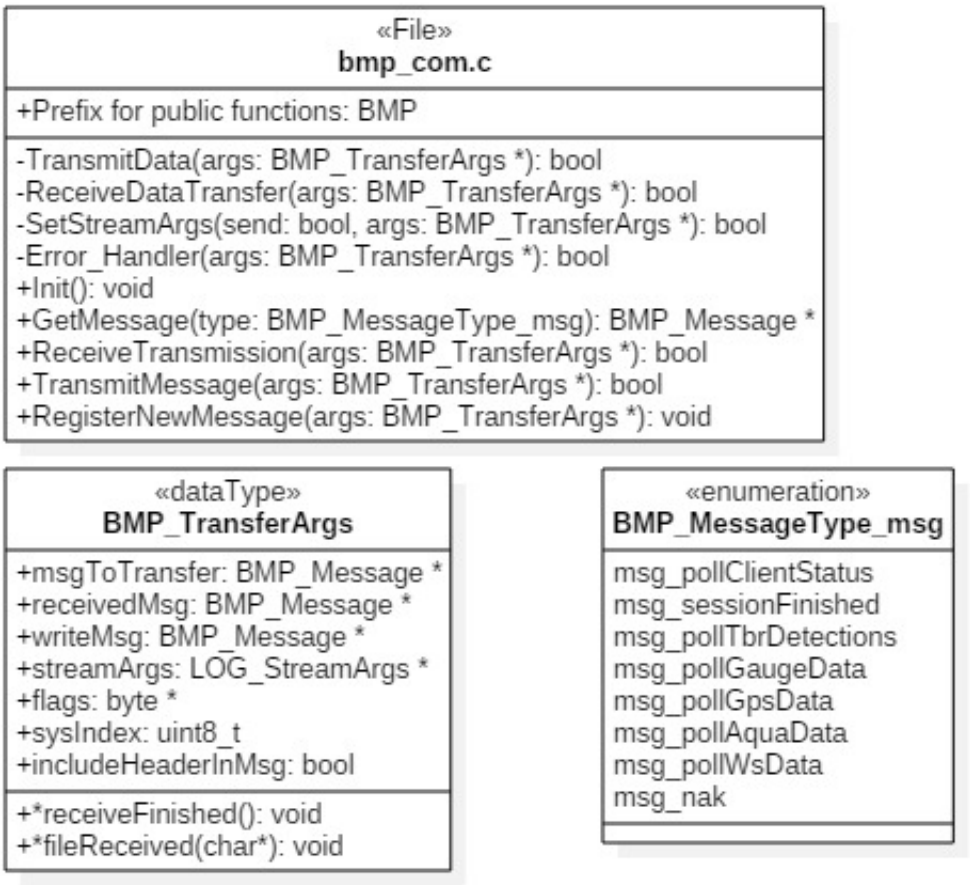

Figure 4.12: UML diagram of the bmp com module.

The public *RegisterNewMessage()* function analyzes the incoming BMP message, and registers it in the provided *BMP_TransferArgs* structure illustrated in figure 4.12. The structure defines several pointers to which data fields and structures must be implemented in a higher level module. This way, the *bmp_com* module can make changes to these fields to implement the functionality of the protocol.

The private *SetStreamArgs()* function defines what the specified BMP access mode -and command should convert to in terms of the *LOG_AccessMode_am* enumeration.
By implementing this "open-closed" design pattern, the higher level *comm* module is allowed to add, or to change some of the *bmp_com* module's functionality, without the need of making changes to the module itself. All function pointers and settings are defined in the *BMP_TransferArgs* structure.

4.6.4 Radio Communication

The radio communication is enabled using the integrated Radiocrafts TinyMesh module described in section 3.1.2. The module handles all of the network functionality, such as setting up the network connections, sending out beacons, and handling the traffic. The network topology lets one of the network nodes act as a *gateway*, and the others as *router* or *end* devices. The gateway device will receive any data sent from the other network nodes.
The software implementation therefore consists of utilizing the buoy message protocol to allow message exchanges between the gateway node and all of the other nodes in the network. The module is configured in "transparent" mode, which means that all data transferred to the radio module from the MCU will be considered data to be assembled into a TinyMesh packet. All data received by the radio module is transferred to the MCU immediately, unless the RTS signal is configured and active.

4.6.5 GSM/GPRS Communication

As a part of the preliminary project, a communication interface was integrated in the hardware design of the buoy controller to accommodate an external communication module. In this thesis, a GSM/GPRS module was bought off the shelf to avoid a new hardware design process. The module enables SMS and GPRS communication between the buoy controller and a mobile device or a computer/server. Several "GSM" commands have been implemented in the design to enable easy access and control of the buoy controller, all of which are listed in Appendix A.2.
The module is controlled by standard AT commands, which are normally used to control most GSM/GPRS modems. The software modules that communicates with the modem, namely the

gsm_sms, *gsm_inet* and the *gsm_driver*, can all be reused if any other GSM/GPRS modem will be applied in the future.

The *gsm_com* module in figure 4.13 was integrated in the design to act as a interface between the higher level *comm* module and the lower level *gsm_inet* module. The GSM commands received through the *gsm_inet* -and the *gsm_sms* modules are handled and analyzed, and converted to corresponding BMP messages. By doing this, the messages can be registered in the *bmp_com* module by use of the *BMP_RegisterNewMessage()* function shown in figure 4.12. This lets the messages be handled in the same way as a regular BMP message received via the radio module. The advantage is that the already existing *comm* and *bmp_com* modules are utilized, by only providing the *gsm_com* module as a interface.

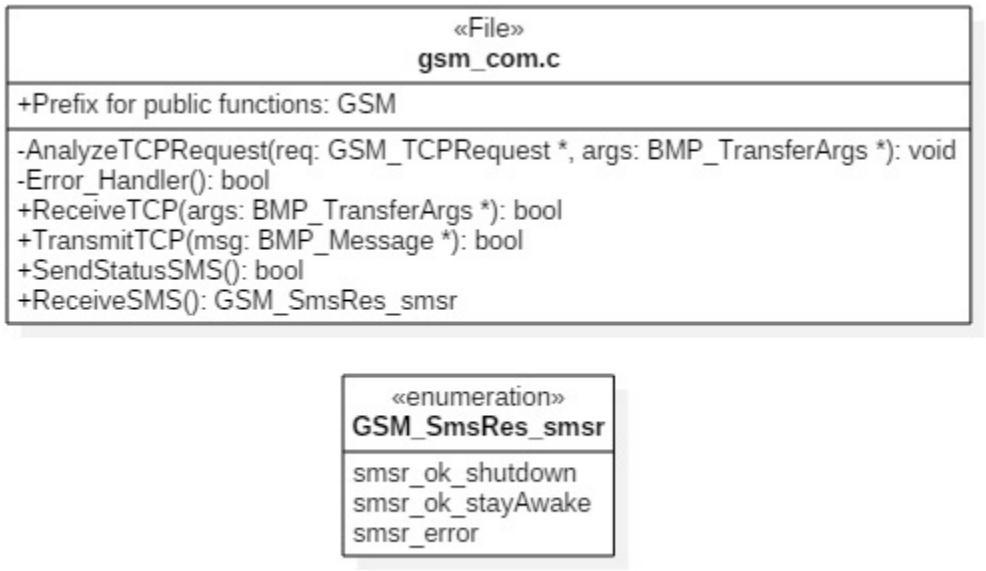

Figure 4.13: UML diagram of the *gsm_com* module.

The *gsm_com* module also handles all of the SMS communication by encapsulating the *gsm_sms* module. The mentioned GSM commands are integrated in both the *gsm_sms* -and the *gsm_inet* module, allowing some common commands on both interfaces. In addition, the buoy message protocol can easily be applied on the GPRS interface trough a setting in the *shared.h* file.

The GSM/GPRS module incorporates an embedded TCP/IP stack, and the *gsm_inet* module

utilize this functionality by providing basic TCP connectivity through either a TCP server or a TCP client. To establish the TCP server, a user must request it by sending an SMS to the buoy controller. When the server is running, the buoy controller returns the IP address and port for connection. The buoy controller can also act as a TCP client, being able to connect to an already established TCP server. When a connection is established, all of the communication will be handled through the *comm* module, in the same way as for the local radio communication. The GSM commands and the BMP are described in further detail in Appendix A.1 and A.2.

4.7 System Info

The *system* module act as a storage "class", where the *SystemInfo* structure is managed. The *SystemInfo* structure was designed to provide easy access to the most important information about the system. For the buoy controller gateway node, the *system* module manages an array of these structures, one for each node in the network. This way, the system info for all nodes can be accessed from the gateway node.

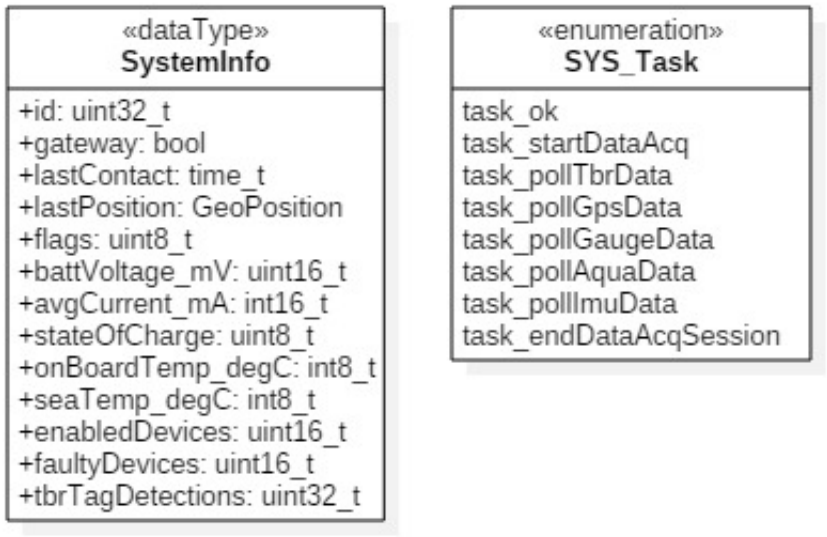

Figure 4.14: UML diagram of the *SystemInfo* structure.

The *system* module also provides the "system tasks" through the function *SYS_GetTask()*. The return value *SYS_Task* enumerator illustrated in figure 4.14 represents the next action for the

system to take, in terms of acting on received data, or polling data from the buoy controller nodes. New system tasks can easily be added if needed.
The *system* module also includes functionality for serialization and de-serialization of the *SystemInfo* structure, in addition to other system-related functions.

4.8 Error handling

The C language does not implement a specific error handling mechanism like for example "try catch" in C++. Error handling functionality was therefore implemented through the *err* header file, where the *ERR_RAISE* macro is used to register any unexpected event or result in the "extern" ERROR structure.
The *err* file is included by most modules in the design, where errors can be raised if something goes wrong. All functions that raise an error must therefore also be able to signal that an error occurred by use of a return value. If an error occurs, the calling functions will be signalled until an error handler is reached.

When errors originate in the device drivers, the "origin" field of the ERROR structure is set to the device number corresponding to the *Device_dev* enumerator, so that the originating device is registered. The errors are handled at different levels in the design, depending on their type and origin. The device manager module implements the *DEV_Error_Handler*, which handles most errors related to the devices. However, if the handler cannot handle or resolve the specific error, it is expected that a higher level error handler will. The state machine therefore implements the "exception state", which the system will end up in if an error occurs.
The *ERR_ReturnValue_ret* field of the *ERROR* structure informs the higher level error handler if the error has been handled at a lower level. Three types of return values exist: *resolved*, *notResolved* and *resolvedRetry*. This way, the error handler in the "exception state" is informed on what actions has been made to resolve the error. An error counter value is also added, which is incremented if the error occurs over again. If another error occurs in between, the counter is however reset.

4.9 Shared Files

A separate *shared* header file implements the enumerations, macros and settings that are needed by several of the modules in the design. It therefore serves as a configuration file for the system. It defines both the *Device_dev* -and *Stream* enumerations, which are used throughout the design. By making these shared, indexing arrays and accessing device specific functions are simplified to the extent that little type conversion is needed. Two macros are defined, which converts a *Stream* enumerator to a *Device_dev* enumerator, or the other way around.

The timer module provide five separate timers by utilizing the three available 16 bit timers, the SysTick timer, and the Low Energy timer. The *TIMER0*, *TIMER1* and *TIMER2* are 16 bit counters that are clocked by the peripheral clock *HFPERCLK*. They can however only be divided by maximum prescaler of 1024, giving them a quite short range of about 2.2s before overflow, depending on the peripheral clock frequency. To cope with this, the overflow interrupts have been utilized in much the same way as with the event scheduler in the time manager module (see figure 4.6). This way, the timers are only limited by the argument length (2^{16} ms), making them much more usable. These timers are applied in the *StartTimer_ms*, *StartTimer_s* and *Delay_ms* functions. The timers are employed throughout the design, but must be used carefully, as an implementation at a lower level may interrupt with the higher level implementation.
The *Sleep_ms* function does on the other hand employ the *SysTick* timer, which is a lower level counter implementation provided through the Cortex CMSIS library. The counter is 24 bit in length and configured to be clocked by the RTC counter. The Low Energy timer is employed as a low resolution timer allowing a callback function to be invoked when the time has elapsed.

4.10 Drivers

The device drivers are software modules that act as the lower software level, creating an interface to all of the peripheral devices integrated in the hardware design, as well as the connectable devices and instruments through the hardware interfaces. The drivers have been developed with a basis in the devices' data sheets, and most of the device functionality have been implemented in the software.

The drivers are created in a similarly fashion, where their main responsibility is to initiate the MCUs communication interface and to enable the appropriate GPIO pins, including controlling the power to the modules, configuration, and for some drivers, data transfer.
Since not all of the devices implement their own low power sleep mode, MOSFET power switches was implemented in the hardware design to enable total shut down of these devices (see section 3.1.4). These switches are connected to the GPIO pins of the MCU, making it easy to control them from the driver software.

4.10.1 UART Devices

The MCU can interface with five UART devices, most of which are connected to the different interface connectors integrated on the buoy controller PCB. In the software design, drivers have been developed for the integrated radio module, the TBR700 via the RS485 interface, the GSM/GPRS module, and partly the AquaTroll multimeter instrument for connection via the second RS485 interface. A last UART interface is connected to the "Weather Station / UART interface" of the buy controller PCB, which exposes the UART interface for future implementation.
Common for all of the UART devices is that they are all assigned two corresponding FIFO queues: one for TX, and one for RX. The size of the buffers can be set independent of the others in the *fifo_config* file. This provides the possibility to minimize the memory usage for the FIFO queues by choosing the most appropriate size for each queue. The UART TX -and RX interrupt handlers have been designed to put all incoming data from the UART RX registers into the RX FIFO buffer, and to load all outgoing data from the TX FIFO buffer into the UART TX registers. The drivers do not handle data transfer beyond this, instead it is left to be processed by higher level software modules.

4.10.2 SPI Devices

Three on-board modules have been integrated with the SPI through one of the available USART ports on the MCU, namely the GPS module, the IMU, and the SD card slot.
A simple driver for the IMU has been developed, however only for testing purposes, and to configure it to a low power mode. However, most of the framework exists, and a future implemen-

tation of the IMU can therefore easily be added. The SD card driver was adopted from Silicon Labs' implementation of the *FatFS* module described in section 4.5.3.

Part II

Discussion And Evaluation

Chapter 5

System Power Consumption

One major premise for the design of the buoy controller was to ensure that low power operation of the system would be possible, as stated in section 1.4 and 1.5, to enable continuous operation over a period of at least six months. In the hardware design process, the focus was to select components and solutions that could be utilized in the software design to enable the most energy efficient operation possible. Through the software design process the same focus has been continued, and software functionality has been implemented to enable efficient operation.

For example: the radio is a high power device being able to pull as much as 560mA at 3V at the highest gain setting when transmitting. However, this state will not be kept for a very long time. The design lets the gateway controller collect the available data, and when finished, it will issue an "session complete" command (see Appendix A.1), which tells the other nodes to turn off their radios. The event scheduler will also issue a shutdown of the radio after a set time to be certain it will be shut down. This way, the radio will most of the time stay in a low power mode, only consuming current in the uA range.

Each buoy controller node must provide its own power supply and energy harvester. The main power supply will be a battery, and a solar battery charger integrated on the harvester module allows the battery to be re-charged using a connected solar panel. Battery life longer than six months may be desirable, but there may also be a trade-off between battery and solar panel cost and longevity. The current charger chip does also have a limitation of max. 400 mA of charge

current, which may limit the charging capacity. In this chapter, the power consumption for a specific case will be estimated, and will together with solar radiation estimates make a basis for solart panel -and battery dimensioning.

5.1 Power Consumption

To be able to estimate the total power consumption, the current consumption for each of the connected instruments and on-board peripherals must be recognized. This estimation does not include any actual measurement of the current consumption for each device, but will instead be based on the current consumption data acquired from the data sheets.
Most of the devices have several states they operate in, such as "idle", "low power mode", "transmit/receive", "shutdown" etc. In each of these modes, the devices consume different amounts of power. One of the challenges is therefore to estimate how much time will be spent in each state, to be able to correctly estimate the total current consumption. The software also enables user configuration of how "active" a device / peripheral should be, which will affect the overall power consumption (see section 4.4).

The following presumptions about data acquisition have been taken, to give a basis for calculating the overall power consumption:

- A total of 11 buoy controllers will operate in a single network (one gateway node and 10 router nodes).
- A single buoy controller node will receive 1000 tag detections via the acoustic receiver every day.
- The gateway buoy controller will be accessed every hour during the day, by a remote user via the GSM modem's TCP server.
- The gateway buoy controller will poll the buoy controller nodes for data once every hour.
- The acoustic receiver TBR700 will have a continuous connection to the buoy controller, meaning that the RS485 connection must be active at all times.

- GPS position will be logged once every 30 minutes, and the data will be collected by the gateway. For clock synchronization, the GPS module is however expected to be active every 10 minutes. In between, the module is expected to be shut off.

- Battery voltage and current consumption will be logged every 30 minutes and will be collected by the gateway as well.

As mentioned earlier, the software integrate functionality to ensure that the devices will be shut off, or set in low power modes when their full operation is not needed. The amount of data logged by each buoy controller node will therefore directly affect the power consumption of the system, as the most power demanding devices, such as the radio module, the GPRS modem and the GPS module are used either to generate data, or to transfer it. Moreover, the gateway node will need to handle data amounts corresponding to the number of buoy controller nodes that operate within the same network. Two calculations have been carried out, one for a single router node, and one for a gateway node in a network of totally 11 nodes.
The generated data is stored on the SD-card with standard ASCII coding, were one line in the log file represent one measurement. For a GPS position fix, about 44 bytes of data is generated. For a Gas gauge or a TBR700 tag detection, about 40 bytes of data is generated. When transferring data over the TinyMesh network via the radio module, the BMP header of about 28 bytes is also added for each data packet. By using these known values together with the presumed data acquisition, -and generation during a day, an estimation for the current consumption for each device has been calculated.

Table 5.1: System power consumption estimate pr. day

Device	Power consumption gateway node	Power consumption router node
Radio module	62.2 mWh	20.2 mWh
IMU	0.06 mWh	0.06 mWh
SD-card	2.1 mWh	0.22 mWh
TBR700	0(190) mWh	0(190) mWh
MCU	134.6 mWh	122.5 mWh
GPS	36.4 mWh	36.4 mWh
GPS antenna	34.8 mWh	34.8 mWh
GSM/GPRS modem	204 mWh	-
Gas gauge	6 mWh	6 mWh
RS485 receiver/bias	301 mWh	301 mWh
Total Consumption	0.77(0.96) Wh	0.52(0.71) Wh

Table 5.1 presents estimated total power consumption per day for a gateway -and a router node. Timings and current consumption have been found in the data sheets. The power consumption of the TBR700, RS485 receiver and the bias resistor network is identified to be the largest contribution to the total consumption. The current implemented operation of the TBR700 receiver keeps a continuous connection from the MCU via the RS485 transceiver to the TBR700. This means that all real time tag detections are consequently transferred to the MCU and interpreted instantaneously. Since the connection to the TBR700 is active all the time, it means that the RS485 transceiver and the RS485 bias network must be active. The 12V regulator can also supply the TBR700 with power, which leads to the consumption seen in parenthesis.
To further reduce the power consumption of the system, the operation of the TBR700 should therefore be evaluated. It also keeps the MCU from entering low power sleep (EM2 mode), which can reduce the power consumption of the MCU to almost nothing (about 1mWh pr day). This will also be discussed in the recommendations in section 7.4.
A battery gas gauge was also implemented in the design during the preliminary project, to make battery voltage and current flow measurements possible. This component is also integrated in the software design, and enables observation and logging of battery voltage and current consumption, and can therefore provide more accurate measurements of the total power consumption.
For more information and data about the power consumption, both the data sheets for all of the devices, and the excel sheet used for the calculations (*Power consumption estimation tool*) can be found in the enclosed files.

5.2 Solar Radiation

The power consumption of the buoy controller will vary according to the workload it is given, and consequently so will the requirements for the solar charging solution. However, the provided solar radiation data will be applicable no matter what the power consumption may be. Solar radiation data have been acquired through the European Joint Research Centre, by using their available Photovoltaic Geographical Information System (PVGIS)[1]. This tool can be used to

[1] http://re.jrc.ec.europa.eu/pvgis/apps4/pvest.php

calculate the expected yield from a PV cell (solar panel) at a specific geographic position, for an entire year. However, the tool presents several uncertainties, such as PV technology, efficiency, mounting angle, and the azimuth (the direction of the PV cell with respect to south).
Four calculations have therefore been done, one for each cardinal direction. A fixed system loss of 14 % was added, as the default value. This is meant to compensate for loss in the charger, in addition to weather inflicted conditions such as ice and snow that may cover the cell. Two different locations have been evaluated, one in Sognefjorden, and one in Trondheimsfjorden, which represents plausible locations for deployment of the buoy controller system.

Table 5.2: Average daily output of a 1 kW PV cell (values in kWh)

Month	Location 1: Sognefjord	Location 2: Trondheimsfjord
Jan	0.07	0.07
Feb	0.51	0.61
Mar	1.27	1.45
Apr	2.26	2.36
May	3.05	3.25
Jun	3.54	3.60
Jul	3.01	3.16
Aug	2.19	2.34
Sep	1.40	1.40
Oct	0.63	0.66
Nov	0.11	0.19
Des	0.03	0.01

The output values from the PVGIS tool are listed in table 5.2. The values have been averaged for the four cardinal directions. It is easy to identify the month were the output is at the lowest. The days in December are short this far north, and as a result the solar radiation is very low compared to June. As a result, the battery and the PV cell must be dimensioned to allow operation of the system in December, just as in June, if operation throughout the year is required.

$$Bat_{min} = \left(E_{sys} - (E_{month}/1kW) * P_{PVcell}\right) * n_{days}/V_{bat}$$

In the equation above, the smallest required battery capacity (Ah) can be calculated for a selectable month. The P_{PVcell} (rated PV cell power output) must be selected to an appropriate value. However, if December is to be operated, it seems that a PV cell of 20-30 W may be needed, when disregarding the battery. These numbers must however be considered to be guiding at

best, since small deviation cause large variations in the requirements. Looking past December, in location 1, January doubles the radiation compared to December, even more in location 2. When doing the calculation with the values for January, a PV cell of 30 W will already lead to generating more power than needed by the system, and to charging of the battery.
If looking at the seasons as a whole, the requirements for location 1 for the router node power consumption estimate (0.52 Wh pr day) looks like this:

- Nov - Feb (Winter) : 2.9 W PV cell in average
- Mar - May (Autumn) : 0.2 W PV cell in average
- Jun - Aug (Summer) : 0.2 W PV cell in average
- Sep - Nov (Fall) : 0.7 W PV cell in average

And for the same location, but for the gateway node power consumption estimate (0.77 Wh pr day):

- Nov - Feb (Winter) : 4.3 W PV cell in average
- Mar - May (Autumn) : 0.4 W PV cell in average
- Jun - Aug (Summer) : 0.3 W PV cell in average
- Sep - Nov (Fall) : 1.1 W PV cell in average

This shows that the power output requirement for the PV cell for most of the year is quite low. Operation between March and October requires a 0.2 W PV cell on average for the router node estimate, and 0.4 W for the gateway node estimate. The above number do however not take into consideration that a battery is used, which may reduce the PV cell requirements. A fairly small PV cell can therefore be chosen for the operation during most of the year, where a trade-off can be made between the battery -and PV cell size.

5.3 Uncertainties

As mentioned several times, the power consumption estimation relies on many uncertain values, as well as assumptions about the expected operation and time consumption for the dif-

ferent tasks of the buoy controller. While many of the values for current consumption may be accurate, the expected time the devices use in the different power modes are based on a specific data/measurement flow, and may not be accurate for all applications. When moving to the solar radiation estimation, the PVGIS tool has got several adjustable parameters, which provides uncertainty due to unknown factors around the installation and operation of the PV cell. The resolution of the values from the tool is also fairly limited, due to the fact that it is scaled for larger systems where PV cells in the kW range is used.

Regardless, the calculations seem to prove that the available power during most of the year makes the use of both low power PV cells, and reasonably small batteries possible. To enable operation throughout the winter will however require an increase in both battery -and PV cell size. The acquired data for the winter months are however so small, with a corresponding low resolution, that only small deviations cause large variations in required battery -and PV cell size, and must be considered highly uncertain.

Other solutions, such as decreasing the power consumption when battery is low, have not been implemented in the design, but can certainly be a possibility to cope with the low solar radiation during the winter. If operation is not critical during winter, a consideration can also be made to move the system into a low power state until the battery voltage increase, and thereby continue normal operation.

An integrated software solution can make sure that the buoy controller always acquires all of the registered tag detections from the TBR700, which is also discussed in the recommendations in section 7.4.2. Since the TBR700 is self-sustained by having a separate battery, it can in fact be fully operational even though the buoy controller is shut down / stays in a low power mode. When the buoy controller is re-activated, it can collect all of new data from the TBR700, and get up to date. This may be a reasonable solution in some cases, thus avoiding the higher cost of a larger battery -and PV cell that may be required for full operation during the winter.

Chapter 6

Evaluation and Results

This chapter will comprise an evaluation of the master's thesis, a description of the hardware prototypes and the test setup, and a discussion of the results.

6.1 Test Setup

A field test was included as one of the objectives for this master's thesis, to be able to verify the design requirements for the design. On of the most prominent features of the buoy platform enables wireless communication and data transmission between the buoy controller nodes, and consequently the field test had to include at least two buoy nodes to be able to test all of the functionality. In addition, the test included the acoustic receiver TBR700, a mobile device for SMS communication, and a computer for communication to the separate GSM/GPRS module. A simple TCP client application was installed on the computer to be able to connect and communicate with the TCP server established by the buoy controller. One of the buoy controller nodes acted as a slave node (router), and the other as a master node (gateway). The gateway node was equipped with the separate GSM/GPRS module connected to the communication interface on the main PCB, to enable SMS messaging and TCP communication.
The buoys were placed out on the sea with about 80m distance in between. Initially they were placed with a distance of 300m in between, but due to a wrong setting in the software, the TX power was too low for the radio modules to be able to communicate. An acoustic transmitter tag was situated in between the buoys, producing acoustic signals for the acoustic receivers.

Figure 6.1 shows the buoy setup. A slight difference from the actual buoys is that no solar panels were included in the testing; only small test batteries were used to power the system.

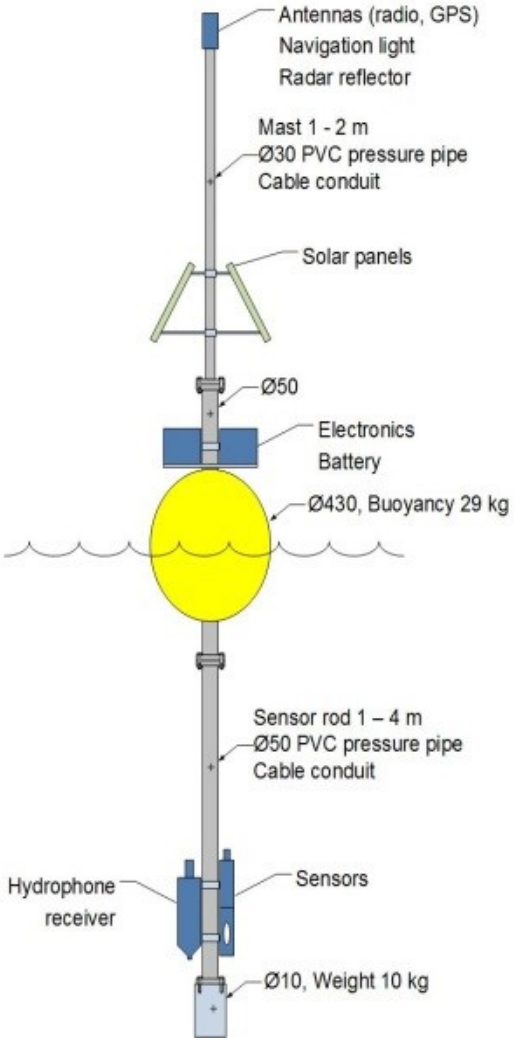

Figure 6.1: Buoy setup illustration.

6.1.1 Prototypes

The preliminary project resulted in the design and production of two separate PCBs, to act as preliminary proposals for the final buoy controller design. The designed prototypes integrate all of the requested features and peripherals devices, including interfaces to separate modules and instruments, battery, etc (see chapter 3 for more information about the hardware design). Some of the components had been tested and utilized in a project prior to the preliminary project, and they therefore made the basis for the buoy controller design.

The production and testing was however postponed and was included in this thesis.
The design of the preliminary project proved to work as intended. Some minor design flaws were discovered in the start phase, but were easily compensated for. However, corrections have been made in the revised hardware design, which is included in the enclosed files. All of the recommended hardware design revisions are described in chapter 7, along with other recommended improvements.

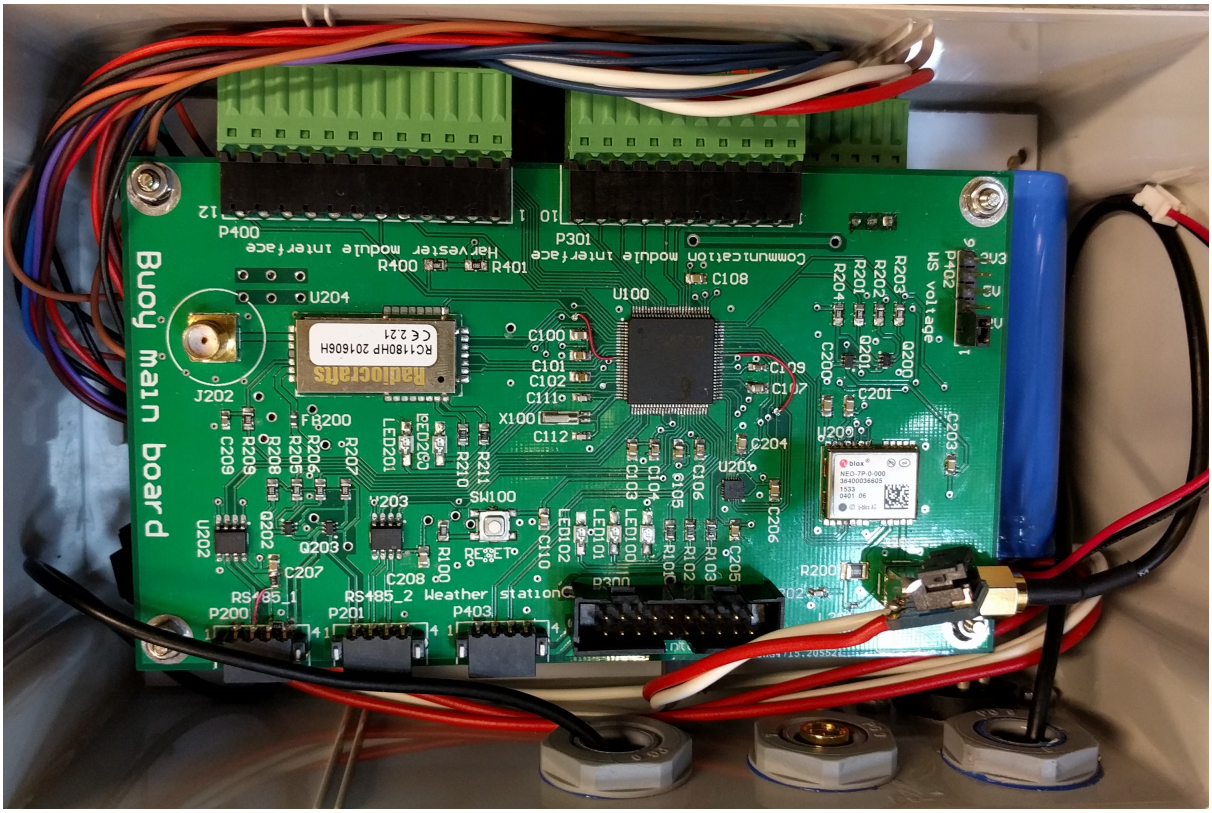

Figure 6.2: Buoy controller test setup.

6.2 Testing and Results

As defined in section 1.4, one of the objectives for this master's thesis was to perform a field test to verify the system design and the implemented features. The test should also reveal any discrepancies in the design that may not fulfill the requirement specification.

The main test objectives were as follows:

- Successfully complete the power up sequence of the buoy controllers, and acquiring a valid time stamp from the GPS module.
- Verify GSM commands and data transmission via a mobile phone and a Internet-connected computer.
- Validate synchronous operation of the nodes by acquiring data from the gateway controller, to see that it has successfully acquired data from the router node.
- Produce fish tag beacons by placing a stationary acoustic transmitter tag in near proximity to the buoy nodes, to act as a imaginary fish that the TBR700 acoustic receivers can detect.
- Connect to the gateway controller and poll for data from both buoy controllers.

The test objectives were derived from the requirement specification in section 1.5. Some of the requirements were however not included in the test objectives, mostly due to the fact that they would be hard to verify during the test.

Figure 6.3 is a photo from the test that was carried out. In the foreground one of the buoys has just been placed out. In the background the other buoy can be spotted, and approximately in the middle, a red float can be spotted, which carried the acoustic fish tag. The test was considered a success, as all of the main test objectives were fulfilled.

Figure 6.3: The two buoys anchored at sea.

6.2.1 Design Requirements Evaluation

In this subsection, each design requirement for the buoy controller design will be covered, to give a clear picture of the requirement, how it has been implemented and in some cases tested, and if it is considered fulfilled.

Autonomous Operation

"The buoy controller should be regarded as an autonomous unit, meaning no remote interaction should be needed for normal operation."

This requirement may be considered obvious and redundant, but nonetheless it defines how the buoy controllers are expected to operate. Several of the other requirements can even be considered to be a part of this requirement, as they expect autonomy as well. The interpretation of the requirement is that during the expected life time of the system, mostly being limited to the access of power, the system should be expected to operate as normal. If any problems or abnormalities should occur, the buoy controllers should handle them accordingly, and maintain operation to best efforts.
The requirement has been tested to some degree. The buoy controllers used in the test were placed out on the sea and left alone to collect data, and enabled communication as expected. This was however during a short time period of about two hours, and may not represent the much longer expected time of operation.

Scalability

"The system should be easily scalable, meaning that small, if any adjustments are needed to add or remove nodes to / from the system, up to a certain limit."
This requirement has been implemented on different levels in the design. There only exist one limitation in the software, which defines the maximum number of buoy controller nodes that can be added to the network. This limitation is only implemented to ensure that there is enough memory available for the "System info" structures described in section 4.7. Other than that, nodes can easily be added or removed from the network.

The functionality has been partly tested by adding one buoy controller node to the "existing" network created by the gateway controller. This does however not suffice to determine the full scalability of the system, being able to handle up to for example 50 buoy controller nodes. Timing tests was therefore carried out, to get a overview over the time it took for the buoy gateway controller to handle an incoming packet. The result is described in Appendix A.5, which demonstrates that the buoy controller should have no problem to handle incoming data at the maximum theoretical transfer rate of the TinyMesh network, at 76.8kbit/s, or 9.6kB/s. Since the radio modules maintain the network, and makes sure no data packets crashes, the gateway controller only need to be able to handle maximum amount of data. The requirement is therefore

considered fulfilled.

Low Power Operation

"Both the hardware and the software should be designed on the premise of low power consumption, to enable operation over a minimum of six months while powered by battery and solar energy."

The energy harvester module designed in the preliminary project integrates both a battery, -and solar panel interface. As described in section 3.1.5, a battery charger has been implemented on the harvester module to enable charging of the battery. However, this feature has not been tested. A solar panel must be obtained and connected to the solar interface, where the expected power output of the solar panel can be determined on the basis of the power consumption estimations of the system described in chapter 5. In Appendix C.3 adjustments needed for solar panel operation are described.

In addition, the hardware described in chapter 3 has been designed to best fulfill the requirement, by implementing low power components in the design, as well as other power saving features such as power switches (covered in section 3.1.4).

During the entire software design process, several design choices was made to accommodate the requirement for a low power design. The event scheduler is a good example of a software module that contributes greatly to reducing the power consumption of the system. As its most important tasks, it schedules the wake-up and shutdown of all of the connected devices. This way, any device that is not required to be powered on, will powered off.

When a device is to be shut down, the *device manager* module is called. It makes sure that the specified device's driver is called, which either puts the device in a low power sleep mode, or shuts it completely off by use of one of the integrated MOSFET switches, depending on the implemented hardware solution. After a successful shut down of the device, the *device manager* also turns off the peripheral clock for the peripheral interface that the device is connected to, which contributes to lowering the power consumption of the MCU.

The device manager also implement a function (see figure 4.7) that is specifically made to provide information about any currently active(powered on) device that do not allow deep sleep

(EM2 operation). This function is called before the MCU changes energy mode (see figure 4.4), to be able to determine if it should enter EM1 operation, or the much more power efficient EM2 mode.

The *comm* module also integrate a power saving feature, which exists of the *bc_sessionFinished* command implemented as a part of the BMP described in Appendix A.1.1. The *system* module is used to initiate data acquisition from the router nodes, and after all of the data has been acquired, it initiates a "sessionFinished" message to be sent. The message signals that no more data is requested, resulting in a shut down radio module of the radio module. This way, the radio module of each buoy controller will be active no longer than needed, thus making the operation more power efficient. A shut down of the radio module will also be issued after a defined time, so that it will never stay powered on any longer than the "active" time defined in the *device manager* module in section 4.4.
The GSM/GPRS module has also been integrated by utilizing its available low power operational mode, which enables it to be connected to the mobile network and receive incoming SMS, while consuming only a fraction of its normal "idle" current.

This design requirement is considered to be fulfilled. The power consumption estimation in chapter 5 also shows a modest consumption. A more detailed excel sheet can also be found in the enclosed files, which shows the calculations of the consumption. However, there is always room for improvements, some of which are described in the recommendations in section 7.3.

Full Device Support

"All on-board -and connectable devices needed for basic operation should be implemented in the software design."
This requirement can be considered fulfilled. There are only one on-board peripheral device that has not been fully implemented in the design, which is the IMU unit. This is a component that is not required for normal operation, but which can easily be implemented later. The framework for operation of the device is however implemented, which exists of the data logging functionality, the power control through the *device manager* module, and the driver module,

which was created to test the connection with the device, as well as to enable the low power sleep functionality.
In addition, the buoy controller provides two available communication interfaces, one which exposes a RS485 interface, and the other which exposes a standard UART interface. The RS485 interface was implemented to accommodate the AquaTroll multimeter instrument mentioned in section 3.1.3, and the UART interface to accommodate a potential Weather station. As with the IMU, frameworks for both the Aquatroll and the potential Weather station are included in the design to enable easy implementation at a later point in time.

Local Network

"All of the nodes in the system should incorporate features to enable communication through neighboring nodes to a master node."
This basis for this functionality was enabled through the integration of the *Radiocrafts TinyMesh* radio module described in section 3.1.2, during the preliminary project. The radio module has during this master's thesis been further integrated in the software design to enable communication via the mesh network that the radio module provides. The design allows the gateway controller be in control of data acquisition and communication, while the buoy controller nodes responds with updates and data. Functionality has also been implemented, which enables synchronization between the buoy controller nodes by use of the time stamp provided by the GPS, to let radio communication be enabled at the same time for each buoy controller. This ensures that all buoy controller nodes will be able to communicate at the same time.
This requirement is considered to be fulfilled.

Communication Link

"A communication interface should enable a separate communication module (GSM/GPRS, Iridium) to be connected, to let the master node act as a gateway for remote users."
In the preliminary project, a communication interface was added to enable GSM/GPRS, or Iridium communication. A GSM/GPRS modem was bought off-the-shelf to enable remote access to the system, which has been incorporated in the design. As described in section 4.6.5, the implementation enables both the use of a TCP server, and a TCP client. This widens the range of use

for the modem, and enables it to either connect to an already existing TCP server, or to establish one by itself. The SMS functionality also enables remote start-up of a TCP connection.
In addition, several "GSM" commands have been made available, which lets a user request data and status updates. The buoy message protocol can also easily be enabled on this interface as well, to be used instead of the mentioned "GSM" commands.
This requirement is considered to be fulfilled, but further improvements may be considered to make it more user-friendly.

Fail-Safe Design

" The software should be based on a fail-safe design, meaning any errors that may occur during operation, will be handled by the buoy controller itself, if possible. Any major errors must be reported to enable the users to handle them accordingly."
Error handling mechanisms have been implemented in the design, as described in section 4.8. This system is designed to handle most of the potential errors that may arise during operation, by handling errors within some of the different software modules, in addition to having a top layer error handler in the "exception" state illustrated in figure 4.4.
However, the implementation cannot be considered complete, as some mechanisms have proved not to function properly. Some errors may not be handled properly, as the implementation has not been fully tested. In an error should occur, no extra functionality has been implemented to report the error, such as for example an "error" command message via the BMP. However, if any connected (initially enabled) device is not recognized, either during start-up, or when trying to wake it up, it will be flagged "faulty" in the "System Info" structure as described in section 4.7, and in Appendix A.4.1. This way, the gateway controller, and thereby the remote user, can be given access to this information. In section 7.2, some of the recommended improvements are described.

Local and Central Data Acquisition

"Each node should be able to acquire data from connected instruments and sensors that produces data, and store them locally. This data should be accessible to the master node, which again should be made accessible for remote users of the system through the gateway."

This functionality is considered to be fully implemented. Every buoy controller node can collect data at selectable intervals, and store them in the local log files on the on-board SD-card storage. However, not all log devices are implemented in the design, as described under the "Full Device Support" requirement.
The local mesh network capabilities enabled by the radio module and the communication software modules, lets the gateway controller collect all of the local log data on each buoy controller node, and store it in its own "combined" log files. Together with its own local log files, the combined log data is made available to a remote host through the implemented GSM/GPRS solution.
This functionality was tested during the field test and confirmed working. A computer connected to the Internet was used on-shore to connect to the buoy gateway controller, where recent log data for both buoy controller was acquired.

6.3 Assessment and Summary

The design process of this thesis has accumulated in a complete software solution for the operation of the buoy controller. The preliminary project resulted in the production of two separate PCBs: the buoy controller main PCB, and the separate harvester module. The hardware from the project has been utilized for the software development and integration of the required functionality of the buoy controller platform. A final field test was carried out to verify the functionality of the system in a realistic environment for deployment. As described in the former section, most design requirement have been fulfilled. Identified bugs and recommended improvements are described in the next section.
The implemented software solutions have been developed, with best efforts, to accommodate further development of the system. This means that common software design patterns and design principles have been implemented where possible, to let the software be as understandable as possible, in addition to being easily expandable. This lets further software development be implemented as extended functionality, rather than modifications of already existing solutions, thus ensuring that less software bugs are introduced when adding functionality. Scalability can also be an issue when adding functionality to existing software. The main state machine has for

example been designed to avoid issues related to scaling, where all states and transitions exist in functions distributed in different modules/files corresponding to their functionality.
There may still exist software bugs in the design, which must be anticipated at the current stage of the design. Some design choices may also be considered less fortunate, or bad in some cases. Especially the communication solution including the *comm* module, the *bmp_com* module, and the *gsm_com* module have been subject for several re-design processes. Still, the chosen solution may appear complicated and entangled, as it is designed around the buoy message protocol and its commands. However, expansions and changes have been allowed with best effort. The *gsm_com* module does for example include a function for converting the implemented "GSM" commands to the BMP commands used in the *bmp_com* module, to allow the use of its functionality as described in section 4.6.5.

A power consumption estimation was carried out, as described in chapter 5. This can serve as a reference for applications of the system, where a particular case was used to calculate the consumption. A simple calculation tool has also been added in the enclosed files, where the power consumption can be calculated with other parameters than the ones used in the presented estimations.

Chapter 7

Recommendations

The test results in section 6.2 indicates that the designed system is functional and can in theory, in its current state be deployed without major software improvements. However, there are still functionality that must be considered for full operation of a larger buoy controller network, in addition to both general minor software improvements, and improvements concerning the fault-tolerance of the system.
If the hardware design is to be continued, some minor corrections and design changes should be evaluated.

7.1 Hardware Improvements

Since the preliminary project did not include any testing and completion of the produced hardware, no shortcomings of the design was identified during the project. However, after utilizing the hardware in the software design process, the hardware has been tested and used, and a clearer picture of its shortcomings has been acquired.
Only two PCB routing errors have been identified, which has been worked around by shorting pins on the MCU with small cables. For the next buoy controller revision, these errors have been fixed to avoid unnecessary work.
The PCB's were designed with quite large passive components, and with a lot of free space. It may be that a more compact hardware design will be beneficial for some applications, and several changes in the design can enable this. For example should smaller decoupling capacitors

be chosen to get them closer to the power pins of the components, while also freeing up a lot of space for routing, especially near the MCU.
The buoy controller main PCB was also designed with components on the back side. When soldering the boards, this proved to be a solution that makes the soldering more time consuming, since the components on the back must be hand soldered with a soldering iron and a heat gun. In a future solution it may be considered to remove the power components from the main PCB, and instead located them on a separate power module.
The RS485 interfaces normally require a bias resistor network. This was not implemented in the hardware design, and has instead been soldered on to the RS485 connectors. The resistors have also been connected to the 3.3V for low power consumption, but should be made able to disconnect from the power, to enable shut down of the bias network and thereby reducing power consumption.
There may also be other design flaws that has not yet been identified, but as of now, these mentioned items are the most noticeable. A revised hardware design has been included in the enclosed files, which removes the mentioned routing mistakes made in the preliminary project. The RS485 bias resistor networks were also missing, and is included in the revised design. The bias resistors is powered directly from the MCU, which is possible since each bias network consumes less than 2mA. By enabling power control of the bias network, it can be shut off when communication is disabled / not required.

7.2 Software Improvements

The resulting software design has been developed in accordance with the requirement specification specified in section 1.5. Some features and functionality have been developed that has not been specifically required, but that will be beneficial in the operation of the buoy controller. However, functionality will need to be added to improve the system, and to make the final approach to a system ready to deploy. Software bugs that will need to be straightened out still exists, which may be expected by a system in the current state of development.

7.2.1 Software Bugs

One of the more problematic software bugs that has been identified, is a bug that induces overwrites of existing log data. It is an issue that occurs in the gateway node when receiving log data from the buoy controller nodes and writing it to the combined log file. After about 500 bytes written to file, a variable length of data is overwritten by the next data input. The source of the bug has not been identified, nor in which buoy controller it originates, whether it is on the receiver side, or on the sender side. It does however seem most likely that the error occurs on the receiver side.

The FIFO queues have been cleared, and are not part of the problem. It may seem that the issue is a lower level problem, that may originate in the FatFs module, or in the FAT file system of the SD-card. However, this is considered an unlikely origin.

7.2.2 Time Consuming Operations

When the gateway controller receives data from the other buoy controller nodes, it is essential that it will not be interrupted for longer periods of time (in the ms range). A selectable sized FIFO queue is applied for incoming data, but it can only buffer so much data before overflow. This means that time demanding tasks that occurs while receiving data, may stop or corrupt the data transfer.

Certain time demanding tasks have been identified, specifically related to the GSM/GPRS modem. Experiences made during testing of the modem, shows that operations like receiving and sending SMS, and starting a TCP server/connecting to a TCP server, can take up to several seconds to complete. The buoy controller will during these operations wait for a response from the modem, and will therefore block any other operations that should have been executed. Any incoming SMS during data transfer has therefore been identified to be a potential error source. However, the radio module do implement a RTS functionality, which also has been implemented in the software design. It has not been tested thoroughly, but when the RTS pin is asserted by the MCU, the radio module will in effect disconnect the network, and thereby stop any ongoing data transfer. The connected buoy controllers will also cancel their ongoing transfer after a specified time, and resume to other tasks etc. This will result in the acquisition session being

cancelled, and no more data will be transferred until next session.
A more fail-safe solution to the problem could be to ignore any incoming SMS, or wait to establish a TCP connection while data transfer is active. This would effectively ensure that no activity related to the GSM/GPRS modem could interrupt any ongoing data transfer. It could however add a certain delay to any SMS response during data transfer, depending on the amount of data being transferred.

7.2.3 Fail-Safe Functionality

The implemented proprietary buoy message protocol implements an extended functionality when transferring "write" messages (described in Appendix A.1.1). The "ack" message is consequently sent after a "transmission finished" command is received, whether the data received before it was complete or not. This may be considered a "faulty" implementation, as the actual data that is being transferred is not really acknowledged, only the last "transmission finished" message. When transferring file data, this functionality is used to update the last read file -and file position of the accessed log. However, if a faulty data packet is received, the receiver will issue an "nak" message to the sender, and if log data is transmitted, the transmission is reset, and starts over again.

If a situation should occur, where several data packets would be lost due to unknown reasons, it may be that these packets are not registered by the receiver. If the receiver however receives the "transmission finished" message successfully, the transmission as a whole is considered finished and successful, and an "ack" response is issued.
A solution to making the data transfer more fail-safe, will either require an "ack" message is be issued on a packet-by-packet basis, or a solution where the number of packets to be sent is known on beforehand, so that the receiver can count the received packets and compare it to the expected number of packets. The first solution will require a significant amount of network capacity, as each data packet from each node in the network will have to be acknowledged. Since the TinyMesh network distributes the messages from the gateway node to all other nodes, each packet will have to be addressed specifically to each node. This will consequently add a lot of extra message handling for each node.

The second solution will avoid this problem, but may introduce another problem. When starting a data transfer from log file, the number of packets to send is not known. The number of unread bytes in the log file can however be retrieved, and thus calculating the number of packets needed to transfer all of the data. However, if a log write happens during a data transfer, it may be that the total number of packets to transfer will be higher than the reported number.

Before deploying the system in real applications, the fail-safe features of the system should be thoroughly investigated and tested, to ensure that the right actions are taken on any error. Especially dead locks have appeared during testing of the error handling mechanisms, and currently unknown ones may yet exist, which could be devastating in a deployed system.
The available watchdog timer, which is integrated in the MCU, has not been utilized in the design. It provides essential functionality to enable restart of the system should a unrecoverable error occur, and should be integrated in the final design before deployment.

7.3 Further Software Development

7.3.1 Buoy Gateway Access

A GSM/GPRS module (see Appendix C.1) was implemented in the design to enable access to the buoy controllers for remote users, as one of the design requirements stated. The implemented GSM/GPRS module has an embedded TCP/IP stack, which made a TCP/IP implementation easy. The communication module design therefore utilizes both the available TCP server, -and TCP client functionality, as described in section 4.6.5.

However, to improve the user-friendliness and range of use for the communication with the gateway controller, several extended solutions can be considered. One solution to be considered, is to keep the implementation mostly as it is. In addition, a full TCP server application is designed for user-hosting. The application can be considered to function as a gateway, while the buoy gateway controller act as a node, just as with the buoy controller nodes. This operation is

already implemented in the buoy controller design, where the BMP easily can be applied on the GSM/GPRS interface as well, providing extended functionality compared to the "GSM" commands. Most of the local buoy communication functionality can therefore be adopted, so that one TCP server can just as well function as a gateway for several buoy gateway controllers. This principle is illustrated in figure 7.1, just to give a clear understanding of the suggested solution. The solution can also take advantage of using a static IP address for the TCP server, so that the buoy gateway controller(s) can initiate connections to the server at fixed intervals, in the same way as with the radio communication at the local network level. This would remove the need for using SMS commands to activate a TCP connection, which simplifies the connection process a lot.

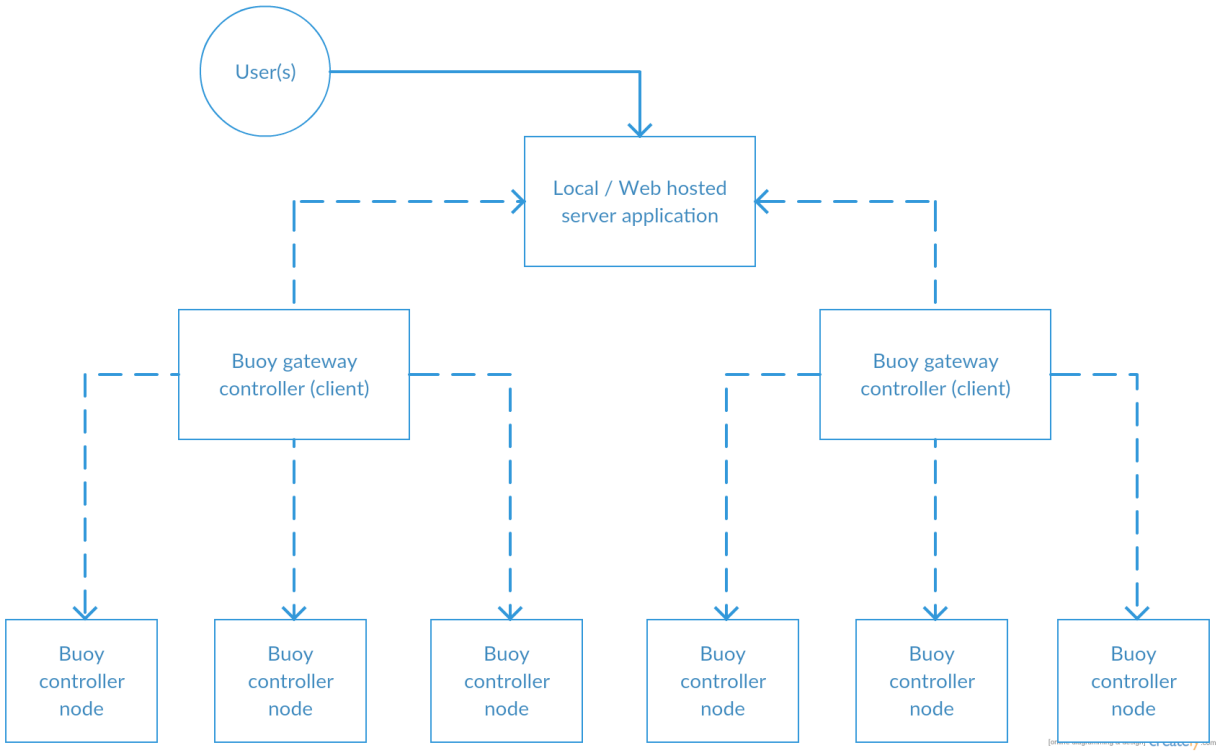

Figure 7.1: Buoy gateway access suggestion.

Another suggestion is to implement an application layer protocol on top of the existing TCP communication layer integrated in the design. This could enable web server functionality by either providing the web server on the gateway controller itself, or on another server location. This could provide a more simple and accessible solution than the current. If the server is to

be accessed at random points in time, the SMS start-up functionality will however have to be continued.
If hosted on a Internet-connected server, which both the users and the buoy controller could connect to, the solution could function in much the same way as the solution mentioned above. The only difference would be that, instead of the buoy gateway controller accessing a user-hosted TCP server, it could access a web-hosted server, which could also provide a web interface for users.

7.3.2 Watchdog Functionality

The time manager module implements a task called "watchdog". The interval can easily be adjusted in the header file, as described in section 4.3.3. The intention with this task, is that it for example can be used to check the connections to all of the devices. Depending on interval settings for each device, it can be that the time between communication with each device is quite long. Other tasks, such as checking the current remaining storage capacity, may be something worth implementing. Over long times of operation, the SD-card may end up being filled up, depending on the size, and a solution may therefore be to erase old log files after a certain threshold to make room for new log data.

7.3.3 Accurate Time Synchronization

The buoy controller design enables the use of the time pulse output of the GPS module. It can easily be enabled through the GPS driver, and can be integrated in the design to accommodate positioning applications where a highly accurate time reference is required. The TBR700 driver allows time updates and time synchronization of the receiver, making it easy to integrate the functionality in the existing design.

7.4 Power Consumption Reduction

Below, measures to further reduce the power consumption of the system are discussed.

7.4.1 Power Manager

For further development and extensions of the buoy controller, a power manager module may be a next step in the process. This module could serve as a separate manager module to control the power usage of the system. The input to the module could be the battery status and the current power consumption of the system. By using these parameters, it could make decisions on whether to execute requested power consuming tasks, or to lead the system into different power modes according to the currently estimated remaining battery capacity. This could for example keep the system running during the winter when the solar radiation is low, reducing the need of having a large solar panel or battery.

The radio module transmit power can also be adjusted by using the implemented radio driver. Since the radio module is a high power version, the difference in power consumption between the highest transmit power and the lowest is quite substantial. When the buoy controller nodes are placed in close proximity to each other, the highest transmit power may not be needed. A power manager could therefore also be used to adjust the transmit power by measuring the current RSSI (signal strength) of the network. Measuring the RSSI is simply done by requesting a measurement by the radio module, which is also integrated in the radio driver. Such a feature could therefore help to reduce the radio power consumption by dynamically adjust the RF output power of the radio.
However, the functionality will be limited to the fact that all radio modules that reside within the same network, must be configured to the same output power. Such a functionality should therefore be managed by the gateway node, and distributed by use of a BMP "config" message, which is mentioned in Appendix A.1.1. In addition, when configuring the radio module with a new output power setting, it is stored in a non-volatile memory, which only has a guaranteed number of write cycles of 1000. It may therefore be that the functionality is not applicable, and should instead be configured in software, or preferably in the user configuration file, before deployment.

7.4.2 TBR700 Operation

The TBR700 is connected to the buoy controller through one of the RS485 interfaces as described in section 3.1.3. In Appendix B.1, the RS485 transceiver connected to the RS485_1 interface, is connected to one of the Low Energy UART interfaces of the MCU. This interface can be clocked by the *LFXO* clock, which means that it can also function in energy mode 2 (EM2). This energy mode shuts down the *HF* clock, which means that only peripheral interfaces and timers clocked by the *LFXO* will function, in addition to RAM access through the direct memory access system (see EFM32 data sheet for more information).

The implemented solution do however not make use of the LEUART functionality, mainly because of the currently non-configurable baud rate setting of the TBR700, which is set to 115200 baud. For the LEUART interface to function in EM2, maximum baud rate is limited to 19200 baud. However, if a baud rate of 19200 baud is allowed, the LEUART features should be implemented. This will allow the MCU to enter EM2 operation when no other devices are active, thereby reducing energy consumption of the MCU to a minimum. In fact, most of the MCU power consumption seen in table 5.1 comes from the EM1 operation of the MCU.

When data is received from the TBR700, the MCU can be woken up, and data will be loaded into the FIFO buffer as usual, or the mentioned DMA (Direcet Memory Access) controller can be utilized. The DMA allows the data received through the LEUART interface to be loaded directly into RAM without waking the MCU. A defined number of bytes can be loaded into memory before the MCU is finally woken up, which then loads the buffered data from RAM, and writes it to the log file.

By enabling the LEUART features, the power consumption of the system can be reduced a lot, mostly due to EM2 operation being enabled.

If lowering the baud rate of the TBR700 is not possible, another power reducing measure consists of disabling the communication with the TBR700 most of the time, and wake-up the buoy controller at defined intervals to poll the TBR700 for new tag detections. This functionality can easily enabled, as most of the software solutions already exists through the *time manager* module and the *device manager* module. This would also enable the RS485 bias network to be shut off, as well as the RS485 transceiver, which would reduce the power consumption a great deal more.

These suggestions for further extension of the buoy platform may only be a few in many of possible future extensions. The implemented software design do however provide a solid basis for developing the platform further in the future, not only being limited to a specific application.

Part III

Appendix

Appendix A

Software Modules

A.1 Buoy Message Protocol

A.1.1 Description

The buoy message protocol was designed to accommodate the different messages and commands that may be sent between the buoy controllers in the mesh network enabled by the Radiocrafts TinyMesh radio modules. In addition, the protocol has been extended to be applied on any interface, such as a TCP connection implemented by the GSM/GPRS module. The protocol is designed to allow certain combinations of the *BMP_Command* and the *BMP_AccessMode* bytes. This is enabled through a system where the command byte also includes the allowable access mode it can be paired with. The *bmp_parser* performs an acceptance test on the incoming BMP message, where the allowable *BMP_AccessMode* is compared with the received access mode. The received command byte is compared with allowable *BMP_Commands*, in addition to being compared with the received access mode as well. This way, only valid combinations will be accepted. The different commands and access modes and their valid combinations are listed in table A.1, as well as in the *bmp_parser.h* file.

Table A.1: The implemented commands and access modes

Command	Access mode	Access mode
bc_systemStatus	*acm_poll*	*acm_update*
bc_entireLog	*acm_poll*	*acm_write*
bc_specifiedLog	*acm_poll*	*acm_write*
bc_lastLogFile	*acm_poll*	*acm_write*
bc_fromLast	*acm_poll*	*acm_write*
bc_config	*acm_poll*	*acm_write*
bc_ack	*acm_response*	
bc_nak	*acm_response*	
bc_identify	*acm_poll*	*acm_update*
bc_transFinished	*acm_update*	
bc_sessionFinished	*acm_update*	

In addition to the *bmp_parser* module, the protocol is also implemented by the *bmp_com* module. The implementation defines what actions to take on the different commands and access modes. Some of the BMP commands have not been implemented in the *bmp_com* module, but the module allows extensions to accommodate these commands as well, if need be. Below, the actions and expected messages are listed:

- When a message with access mode *acm_write* is transferred, a *bc_transFinished* command will be issued when all of the data has been transferred.

- The sender of the *acm_write* messages expects either a *bc_ack* or a *bc_nak* to confirm the data transfer.

A.1.2 Structure

Table A.2: The Buoy Message Protocol

Byte number	Description
0 - 1	Start of header (SOH) characters
2 - 5	4 byte sender address
6 - 9	4 byte receiver address
10	BMP Access mode byte
11	BMP Command byte
12	Device log to access
13 - 25	File name of file to access (if file transfer)
26	End of file name (EOFN) character (if file transfer)
27 (13)	Start of text (STX) character
28 (14) - n	Payload
n + 1	End of transmission (EOT) character

A.2 GSM Commands

Several GSM/GPRS commands is implemented to enable simplified communication with the buoy controller. Some of the commands are only available on either SMS or TCP, while some are available on both interfaces. The "Info" command is for example a universal command implemented to provide information about the available commands on the current interface. All of the currently available commands are listed in table A.3.

The implemented log system make use of dates for file names, and is designed to create new log files at midnight for the coming day. To access specific log files via a TCP connection, the specific date must be supplied in the following format: "DDMMYY". The device log number is derived from the *Device_dev* enumeration found in the *shared.h* file. As for the implementation in this thesis, three log devices have been implemented: "TBR700"(0), "GPS"(3), and "Gas gauge"(5).

If a "Not OK" -or "CMD not recognized" response is received when trying to access data, it may be that the format of the request is not correct. Another reason can also be that the log for the specified date does not exist, or have been deleted (not implemented in the design).

Table A.3: Available GSM commands

Command	Description	Available on
"Reg"	Register phone number to enable SMS updates (if implemented)	SMS
"Unreg"	Removes the phone number from memory	SMS
"Connect:<"IP addr(:port)">"	Starts a TCP client and tries to connect to the specified IP address and port, or the standard port if "port" is omitted	SMS
"Start"	Establishes a TCP server and returns an SMS with IP address and port to connect to	SMS
"Close"	Ends the current TCP session and disconnects from the client or server	Both
"Info"	Returns the available commands on the current interface	Both
"Status"	Returns the system info for all of the registered nodes	Both
"Get last<device log>"	Get the last device log data since last time data was polled	TCP
"Get all<device log><"date">"	Get the specified device log for the specified date	TCP
"Ok"	Acknowledge on received data	TCP
"Not Ok"	Not acknowledge on received data. Starts a re-transmit of the transferred data	TCP

A.3 Configuration Files

The *log_manager* module implements two types of configuration files: the user configuration file, and the operational parameters file. The user configuration allows the user to set, or change different parameters in the system without having to compile and upload new software to the MCU. This enables custom configuration of each buoy controller, which for example is needed when setting the unique ID for the system and the radio module. It is also easy to extend the configuration files to encompass other parameters.

The operational parameters file is not meant for user configuration, instead it is a file created and maintained by the system itself. It contains parameters such as "last read file", "last read file position", and "number of TBR700 tag detections". These parameters are implemented as a

part of the fault tolerant design, where the parameters are saved to file when it is safe to assume that the receiver of the data from the specified file have received it without data loss. This is implemented in the design by the use of "ack" and "nak" messages, as described in section A.1.1.

The TBR700 flash code number is also stored in the file. This number represents the last read tag detection from the specific memory position in the TBR700 memory. The number is extracted from each tag detection or temperature reading that is received from the TBR700. In case of power loss, or if the communication to the TBR700 is temporary shut off, the tag detections that has been registered since the communication was last active, can be collected from the TBR700. This way, no detections are "lost", as far as the buoy controller knows.

Table A.4: Implemented user configuration parameters

Parameter	Description
"Port:"	Provide the preferred standard port to use for TCP communication with the buoy controller. This port is used when establishing a TCP server, and may also be used if the buoy controller is configured as a TCP client. Must be set, or the buoy controller will not start!
"First time:"	Set to "1" first time the buoy controller is powered on / employed. Enables first time configuration. The buoy controller will reset the value.
"APN:"	Set the access point name for the service provider. Must be set, or the buoy controller will not start!
"PIN:"	Set SIM card pin if the SIM card is locked.
"SID:"	Sets the system ID to be configured in the radio module. All buoy controllers supposed to be on the same network must have the same SID.
"UID:"	Sets the unique ID for both the buoy controller itself, and the radio module. Must be unique, else the network will not function properly!
"ResetTBR:"	Set to "1" to enable a full reset and memory wipe of the connected TBR700 memory. The buoy controller will reset the value to avoid a memory wipe on next system reset.
"<Device>:<enable(0,1)>,<always on(0,1)>,<interval(0->)>,<on period(0->)>"	Enables custom configuration of each device. If "always on" is set, the device will never be powered off. The interval -and on periods are defined in seconds.

In table A.4, the implemented user configuration parameters have been listed. The allowed "Device" names can be found in the *shared.h* file. An example of the user configuration file has also been added to the enclosed files. As mentioned above, the software implementation allows for easy extension of the configuration files, adding more parameters if need be. Table A.5 lists

the implemented operational parameters used by the system itself.

Table A.5: Implemented operational parameters

Parameter	Description
"<Device_local>:<last read file>,<last read pos>"	Last read file and file position for the specified local device log, since last data was collected by gateway node.
"<Device_comb>:<last read file>,<last read pos>"	Last read file and file position for the specified combined device log since last data was collected by user / through communication interface.
"TagDetections:"	Stores the total number of TBR700 tag detections since last data was collected by the gateway node
"TbrFlashCode:"	Stores the TBR700 memory flash code for the last received tag detection or sensor reading

A.4 Operation

A.4.1 System Info String

The System info structure defined in the *System* module provides basic information about a buoy controller, such as battery status, last position, temperatures, flags, etc. This information is collected by the buoy gateway controller, and can also be collected by a remote host/user through the communication interface. A description of the formatted System info string will therefore be provided here, to provide a full understanding of its content.

Table A.6: Formatted System info string description

Parameter	Description
UID	The unique ID of the buoy controller.
Gateway	Either "0", or "1", defining if it is a gateway controller or not.
Last contact	Provides the UTC time stamp that the buoy gateway controller last had contact with the buoy controller node. If the specified controller is a gateway, the time stamp is approx. the current time.
System flags	Status flags for each buoy controller. Defines if any of the parameters in the system info structure has exceeded set threshold values (more info found in the *system.h* file).
Last position	The last registered position of the buoy controller.
At time	The UTC time stamp at which the position was registered.
Battery voltage	The last measured battery voltage.
Avg. current	The last measured average current (net flow from the battery)
SOC	The last calculated state of charge in percent, of the calculated total battery capacity.
OnBoardTemp	The last measured on-board temperature, which is provided by the internal temperature sensor of the MCU.
SeaTemp	The last measured sea temperature measured by the TBR700 (if configured to).
EnDev	A bit field providing information about the currently enabled devices (devices that are working).
FauDev	A bit field providing information about the devices that did not respond when the MCU tried to contact them, and are considered faulty, or not connected.
Total tagDet	Provides the total number of tag detections registered by the buoy controller. The value is never reset, and is also registered on the SD-card.

A.4.2 LED Indicators

Five LED indicators were implemented in the hardware design to provide simple feedback from the system. These have been implemented in the software as well. Two of the indicators are connected to the radio module, which provides information about the network connectivity status, RSSI etc. These are controlled by the radio module itself, but can be configured to be shut off after a number of minutes (configurable through the *radio_config.h* file). The other three indicators are connected directly to the MCU, and are used to provide certain information about the current state of the buoy controller. These LED indicators can also be switched off after a configurable time, defined in the *time_manager.h* file.

Table A.7: Implemented LED indicator functionality

Color	Description	Detailed Description
Green	Steady	The green LED lights constant during start-up, after configuration of all of the devices has been successful. It will continue to light until a valid UTC time stamp has been received from the GPS.
Green	Steady or blinking	When the green LED is active or blinks during operation, it means that the buoy controller is active and may be executing a task, receiving data, collecting data etc.
Yellow	Steady or blinking	When the yellow LED is active, it means that the buoy controller currently is in a low power mode (sleeping).
Red	Blinking slowly	The SD-card was not detected when trying to access it. It may be missing, or faulty.
Red	Blinking fast	A major failure was detected during start-up, which means that the buoy cannot continue before the problem has been resolved (missing parameters in the user configuration file).

A.5 Network Scaling

Since no actual testing of buoy controller network scalability has been possible, due to the limitation of available buoy controller nodes, a simple timing test was performed to get an overview over the time it took for the gateway controller to handle -and receive incoming data packets through the TinyMesh network. The *SysTick* timer was utilized for this task, which was enabled when the first byte was received on the UART interface. When the data had been handled, or written to the log file, and the MCU had left the receive function, the timer was stopped.

The time it took for receiving and handling a System info packet was about 2.68ms. Receiving the first full log data packet took 4.82ms, while receiving the following packets took 3.66ms. The reason for the difference between the first packet and the following, is that the FAT file system and the specified log file is accessed. When receiving the following packets, the file is already open.

At the highest transfer rate setting, the gateway controller will at most receive 9.6kB/s. The time it takes to receive the corresponding number of packets can therefore be calculated:

$$
\begin{aligned}
T_{total} &= ((\text{Transfer rate}/\text{Bytes pr.packet}) - 1) * 3.66ms + 4.82ms \\
&= ((9600/116) - 1) * 3.66 + 4.82 \\
&= 304.06ms = 0.3s
\end{aligned}
$$

This shows that the buoy controller can handle 9.6kB of incoming data in about 0.3s, or 32kB/s. This means that it easily can handle other tasks in between as well, such as collecting data from any of the log devices, or sending data to the GSM/GPRS module. More time demanding tasks, such as sending and receiving SMS, should however not interrupt with the transfer, as it may block the execution for up to several seconds.

Appendix B

Pin Maps

B.1 MCU Pin Map

All of the connected pins of the MCU are described. Each table contains the connections to the specified component. "Pin number", "pin name" and "function" refers to the MCU. "Connected to" refers to the connected peripherals' pins. Some of the pins have been changed in the revised hardware design, which have been included in the pin maps as well.

Table B.1: Miscellaneous

Pin number	Pin name	Function	Connected to
4	PA3	GPIO	5V $\overline{ENABLE}$
61	PE1	GPIO	12V ENABLE
62	PE2	GPIO	WS/UART interface ENABLE
48	PD2	GPIO	LED100
49	PD3	GPIO	LED101
50	PD4	GPIO	LED102

Table B.2: Radiocrafts TinyMesh RC1180HP

Pin number	Pin name	Function	Connected to
5	PA4	GPIO	GPIO 0
9	PB0	GPIO	$\overline{CONFIG}$
10	PB1	GPIO	$\overline{RTS/SLEEP}$
11	PB2	GPIO	CTS/RXTX
18	PC0	GPIO	$\overline{RESET}$
12	PB3	US2 TX	RX
13	PB4	US2 RX	TX

Table B.3: U-blox NEO-7P

Pin number	Pin name	Function	Connected to
74	PC14	GPIO	ON/OFF
73	PC13	GPIO	TIMEPULSE
72	PC12	GPIO	EXTINT
71	PC11	GPIO	$\overline{RESET}$
54	PD8	GPIO	$\overline{CS}$
65	PE5	US0 CLK	CLK
66	PE6	US0 RX	MISO
67	PE7	US0 TX	MOSI

Table B.4: InvenSense MPU-9250

Pin number	Pin name	Function	Connected to
51	PD5	GPIO	INT
52	PD6	GPIO	FSYNC
70	PC10	GPIO	$\overline{CS}$
65	PE5	US0 CLK	SCLK
66	PE6	US0 RX	SDO
67	PE7	US0 TX	SDI

Table B.5: MAXIM MAX3483:1

Pin number	Pin name	Function	Connected to
26	PA7	GPIO	ON/OFF 12V
6	PA5	LE U1 TX	DI
7	PA6	LE U1 RX	RO
23	PC5	GPIO	$\overline{RE}$
20	PC2	GPIO	DE
19	PC1	GPIO	RS485 Bias ON/OFF

Table B.6: MAXIM MAX3483:2

Pin number	Pin name	Function	Connected to
27	PA8	GPIO	ON/OFF 12V
42	PB13	LE U0 TX	DI
43	PB14	LE U0 RX	RO
30	PA11	GPIO	$\overline{RE}$
29	PA10	GPIO	DE
28	PA9	GPIO	RS485 Bias ON/OFF

Table B.7: SD card

Pin number	Pin name	Function	Connected to
75	PC15	GPIO	ON/OFF
53	PD7	GPIO	$\overline{CS}$
65	PE5	US0 CLK	CLK
66	PE6	US0 RX	MISO
67	PE7	US0 TX	MOSI

B.2 Pin Map of Connectors

All of the interfaces connecting the PCBs to other modules, instruments and power sources are described here. Each interface has its own table where each pin function is specified.

B.2.1 Communication Interface

Pin number	Function	MCU pin name	MCU pin number
1	3.3V		
2	GND		
3	5V		
4	GND		
5	RX	PF6 (U0 TX)	84
6	TX	PF7 (U0 RX)	85
7	GPIO 0	PD11	90
8	GPIO 1	PD12	91
9	GPIO 2	PE8	92
10	GPIO 3	PE9	93

B.2.2 Harvester Module Interface

Pin number	Function	MCU pin name	MCU pin number
1	Harvested energy/battery		
2	GND		
3	Battery		
4	GND		
5	I2C SDA	PA0 (I2C 0 SDA)	1
6	I2C SCL	PA1 (I2C 0 SCL)	2
7	GPIO 0	PA15	100
8	GPIO 1	PE15	99
9	GPIO 2	PE14	98
10	GPIO 3	PE13	97
11	GPIO 4	PE12	96
12	GPIO 5	PA2	3

B.2.3 RS485 Interfaces

Pin number	Function
1	12V
2	RS-485 A
3	RS-485 B
4	GND

B.2.4 Weather Station / UART Interface

Pin number	Function	MCU pin name	MCU pin number
1	Selectable 3.3V/5V/12V		
2	RX	PD0 (US1 TX)	46
3	TX	PD1 (US1 RX)	47
4	GND		

B.2.5 Battery Interface

Pin number	Function
1	PACK +
2	PACK -
3	NTC Thermistor

B.2.6 Solar Interface

Pin number	Function
1	SOLAR +
2	GND
3	NC

Appendix C

Hardware

C.1 Hardware Components

In the tables below, the implemented main components and interfaces on the PCBs designed in the preliminary project are listed. All of the components used in the project are listed in the BOM.

C.1.1 Main Buoy Controller PCB

- Microcontroller: **Silicon Labs EFM32G880 "Gecko"**
- IMU (Inertia Measuring Unit): **InvenSense MPU-9250**
- Wireless network communication: **Radiocrafts TinyMesh RC1180HP-TM**
- GPS localization chip: **U-blox NEO-7P**
- 5V Boost DC-DC converter: **MAXIM MAX1708**
- 3.3V(adj) Buck/Boost DC-DC converter: **Texas Instruments TPS63000**
- 12V Boost DC-DC converter: **Pololu U3V12F12**
- RS485 transceivers: **MAXIM MAX3483**
- RS485 interfaces: **4 pin wire-to-board connector**

- Harvester module interface: **12 pin wire-to-board connector**
- Communication module interface: **10 pin wire-to-board connector**
- Debug interface: **20 pin ARM header connector**
- Weather station / UART interface: **4 pin wire-to-board connector**
- SD Card slot

C.1.2 Harvester Module PCB

- MPPT Li-ion battery charger: **Linear Technology LTC 4121-4.2**
- Battery gas gauge measurement chip: **Texas Instruments BQ27510-G3**
- Solar input interface: **3 pin wire-to-board connector**
- Battery connection interface: **3 pin wire-to-board connector**
- Main PCB interface: **12 pin wire-to-board connector**

C.1.3 Communication Module

A communication module was bought off-the-shelf to be able to easily enable GSM/GPRS functionality without having to design a completely new module. The module was bought from http://www.rhydolabz.com, and is based on a quad band chip called **SIM900A**.

C.2 Current Consumption

Table C.1: Current consumption as stated in the data sheets

Component	Deep sleep current	Sleep current	Standby current	"On" current
Radio	5 uA	-	25 mA	560 mA
IMU	1 uA	-	-	3.5 mA
SD-card	-	-	-	50 mA
MCU	2 uA	1.7 mA	5 mA	8 mA
GPS	9 uA	-	9 mA	28 mA
GPS antenna	-	-	-	8.5 mA
GSM/GPRS	30 uA	1.5 mA	22 mA	300 mA
Gas Gauge	4 uA	0.02 mA	0.06 mA	0.1 mA
RS485 bias	-	-	-	1.8 mA
RS485 transceiver	0.01 uA	-	-	2.1 mA

C.3 Modification, Adjustment and Operation

C.3.1 Programmable Options

Some of the components in the buoy controller design lets the designer program certain features using resistors. This features will be elaborated here.

Charge current

The charge current of the battery charger is resistor-programmable, which means that by selecting the value of the resistor connected to the PROG pin, the charge current is selected. The max charge current is 400mA, which is selected in this design. If, for some reason the charge current needs to be lowered, the resistor must be changed.

$$I_{CHG} = \frac{1212V}{R_{PROG}} \tag{C.1}$$

MPPT Voltage

The MPPT voltage was during the hardware design programmed for a specific solar cell (CT 30 SolarMarine), where the V_{OC}(Open Circuit)= 22.5V and the V_{MP}(Maximum Power)= 18.5V. The relation between the V_{OC} and V_{MP} are in many cases almost the same for many solar cells,

independent of the manufacturer and voltages. If other solar cells than the one specified will be applied, one will probably want to calculate the MPPT resistor divider to find out if they should be replaced. The following relation between the two MPPT resistor are found in the data sheet of the battery charger:

$$R_{MPPT2} = \frac{0.1}{\frac{V_{MP}}{V_{OC}} - 0.1} \cdot R_{MPPT1} \tag{C.2}$$

C.3.2 Operation

The operation of the hardware should be possible without further information. However, there are two header connectors that allows some adjustments. One of them, P402, is situated on the buoy main PCB. By shorting two of the pins with a jumper, the weather station / UART interface voltage can be selected. One of the accessible voltages can be selected, either 3.3V, 5V or 12V.
The second one, P3, is situated on the harvester module. By shorting two of the pins with a jumper, one can select whether a battery with an internal NTC thermistor is used or not. If a battery with a thermistor is required, it is required that it is a type with $R25 = 10.0k\Omega \pm 1\%$ and $B25/85 = 3435k\Omega \pm 1\%$ (such as Semitec 103AT). All the necessary circuitry is already implemented on the PCB. By using a NTC thermistor the battery temperature can be monitored by the battery gas gauge, and if necessary, it can shut down the charger. All of this can be programmed via the I2C interface.