UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering, Science and Mathematics

A group design project report submitted for the award of
Master of Engineering

Supervisor: Dr. Peter Reid Wilson
Examiner: Iain McNelly

Equine Horse Monitor

by Jose Cubero, Merve Oksar, Konke Radlow,
Michail Sidorov, Yaman Umuroglu

December 11, 2012

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS

A group design project report submitted for the award of Master of Engineering

by Jose Cubero, Merve Oksar, Konke Radlow, Michail Sidorov, Yaman Umuroglu

Diseases such as grass sickness may cause sudden death of horses and in some cases early diagnosis can be life saving. Therefore it is of critical importance to examine horses often. However this is not possible for horse owners who live far from their horses. Also, it is inconvenient for people who own a large number of horses. An automated solution can be used to check vital signs of horses from distance and forward collected data to a veterinary when there is a suspicious situation. A wireless equine health monitor is proposed to provide convenience to horse owners for monitoring vital signs of their animals.

The system consists of wireless monitoring devices attached to horses and a base station. Monitoring devices have several sensors which measure vital signs of the horses. The collected sensor data is then sent to the base station or recorded in the memory card on board. Data transferred to the base station is accessible via a web interface.

The system can be fine-tuned into a commercially exploitable device, and is flexible enough to be usable on humans or animals besides horses.

Contents

List of Figures

List of Tables

Listings

List of Symbols

w The weight vector

Acknowledgements

Thanks to Energy Micro and Nordic Semiconductor for their generous support of our project.

To ...

Chapter 1

Introduction

1.1 Project Brief

The goal of this project is to design a system consisting of a battery powered monitoring device that can be attached to a horse in order to monitor different vital signs, and an accompanying base station infrastructure to allow simultaneous monitoring of multiple horses. The proposed system collects data which could enable the users to detect health problems without physical examination. The data could be used to detect lameness and other diseases such as grass sickness, which has a 95% mortality rate without an early diagnosis `#TODO: citation`.

In order to keep the complexity and power consumption of the sensor devices at a minimum, they communicate over a low power wireless connection with a base station that collects and the data, making it available for the user via a web server.

The system could also be used on human subjects or other mammals, though at its current state it is not optimized for this purpose.

1.2 Objectives

The system is aimed at long-term monitoring which is why long battery life is of high importance for the monitoring devices. Therefore low power consumption is one of the main concerns for design of the monitoring device.

The monitoring device will be attached to horses for long periods of time. Hence it should be small in size, suitable for attaching onto a horse and encapsulated in a splash-proof case to be protected from possible water damage due to the outdoors environment.

Scalability is another goal of the project. It should be possible to use the system in scenarios where it is desirable to monitor multiple horses simultaneously. Taking into

account time and resource constraints for the project, a small amount of scalability (no more than ten horses simultaneously monitored) should be feasible RFC.

The collected data should be presented in a way that it would be easy to access for users. A web server is an ideal solution since a wide range of devices has web access.

Chapter 2

Research

2.1 Grass Sickness

A literature research and discussion sessions with experts were done to acquire information about which sensor data could be useful and how it could be used to diagnose grass sickness.

2.2 Literature Search

Grass sickness (equine dysautonomia) is defined as a disease of equidae characterised by damage to autonomic, enteric and somatic neurons which cause low gastrointestinal motility and paralysis of the gut as a result of this [citations]. It can be classified as acute, subacute and chronic grass sickness based on its severity. Acute grass sickness has a sudden onset while chronic grass sickness shows clinical signs gradually. Symptoms of acute grass sickness include severe colic, gastrointestinal stasis and increased heart rate (70-120 beats per minute). Horses with subacute grass sickness exhibit less severe gastrointestinal stasis and increased heart rate (60-80 beats per minute). Chronic grass sickness cause gradual weight loss, increased heart rate (50-60 beats per minute) and milder gastrointestinal stasis. Horses with acute grass sickness die within 48 hours after the onset of clinical signs while the ones with chronic grass sickness can be saved with intensive care. `#TODO: Citations`

Since low gastrointestinal motility is an indicator of grass sickness, borborygmi can be used for diagnosis. Borborygmi (gut sounds) is the noise caused by gas and fluid pushed in the gastrointestinal system. It shows the status of the gastrointestinal system [citations]. The frequency of borborygmi is 2 to 4 times a minute in normal conditions. Although borborygmi may be an indication of the problems with the gastrointestinal tract, it cannot not provide enough evidence to detect them. `#TODO: Citations`

2.3 Discussions with Bioscientists

The team had meetings with Dr John Chad and Dr Neil Smyth TODO from which dept? uni? about the project. A presentation about horse gastrointestinal system and gut sound monitoring was given by Dr Smyth. The outcomes of the meetings will be explained in this section. `#TODO: Cleanup`

For a successful abdominal examination, four different sites should be auscultated: left and right lower and upper parts of the abdomen [citation]. However, the most loud sound can be heard at the lower right part of the body where the large intestine is.

The proposed system contains mobile monitoring devices attached to the horses. It is not convenient to use a mobile device with four audio recording units to auscultate four different sites of the body since it would require additional wires for communication. Therefore the monitoring device which contains one single audio recording unit can be placed at the lower right part of the abdomen.

The situations which may cause too loud or too quiet gut sound vary [citation]. The complexity of diagnosing grass sickness makes an automated solution unfeasible in the scope of this project. Therefore, the goal of the project was shifted from diagnosis to profiling of vital signals. `#TODO: Citations`

Chapter 3

System Design

3.1 Design Challenges

In the early stages of project, three main challenges were identified: Gut sound monitoring, energy efficiency and wireless connectivity. The problem with these three challenges is, that they cannot be faced independently but have to be seen in a wider context as they are tightly coupled. A design decision that is made in one challenge on behalf of one of the challenges will affect both other fields.

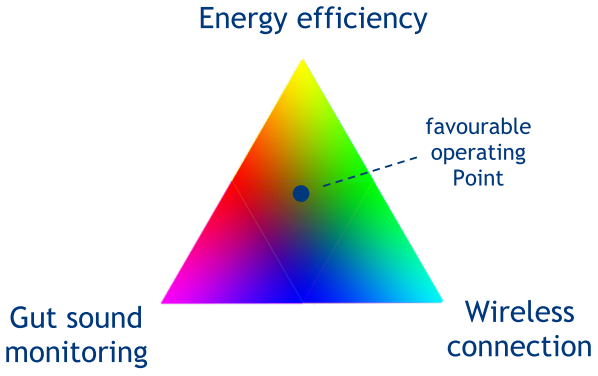


FIGURE 3.1: System design challenges

Figure 3.1 shows one way to visualize this three-way tradeoff - it is impossible to design a system with a very high energy efficiency that performs high quality gut sound monitoring and transmits data wirelessly at the same time, because the design space doesnt contain an operating point for this configuration.

An example for this is the our goal to be very energy efficient, because battery life is highly important for mobile devices. Aiming for a high energy efficiency puts a constraint on the available choices for a wireless connection, because many wireless protocols are rather energy hungry. At the same time it limits the amount of processing that can be done on the monitoring device, because a busy processor consumes a lot of energy and the aim is to keep the processor sleeping for as long as possible. The cases where sending a larger block of data over wireless is actually more energy-efficient than processing the data on the monitoring device and sending less data over wireless are good illustrations of this tradeoff. `#TODO: reference temperature sensor section`

In order to make design solutions for the main challenges a design space analysis was conducted that aimed at finding an operating point close to the center of the triangle. The trade-offs that were taken into consideration will be explained in more detail in the following paragraphs.

Wireless communication between the monitoring device and the base station is another important issue. A wireless protocol solution which satisfies the bandwidth requirements (audio transmission) while consuming only very little energy is desired.

Gut sound monitoring is the last of the main design challenges. To be able to do any kind of examination based on the audio data, the length of the recording has to be in the minute range. This requires a wireless protocol with a bandwidth that allows to transmit the data in a reasonable amount of time and limits the processing that can be done during recording because the processor should not be involved in the optimal case.

`#TODO: Rewrite/Extend next paragraph`

After discussions and research, a good compromise was found which offers a reasonable degree of fulfillment for each challenge.

3.2 Proposed System

The system that we propose to tackle the design challenges is a distributed system, consisting of wireless monitoring devices optimized for battery usage and a stationary base station that collects data from monitoring devices and presents it to the user over a web interface.

This stands in contrast to the early drafts of the system design in which there is no distinction between monitoring station and base station and the user interacts directly

FIGURE 3.2: Block Diagram: Distributed system

with the system over a wireless protocol commonly available in handheld devices and laptops (Bluetooth) or connects to the device via USB. We realized early that this design was suboptimal, as its neither scalable nor suited for running on battery power over an extended period of time as Bluetooth is an energy hog and the data processing cannot be offloaded but has to be performed on the device itself.

Distributing the system gives the possibility to design nodes that are optimized for their role in the overall system, which is especially important in this case. One of the key focus areas of this project is to build a battery powered device with a long battery runtime. This can only be achieved if the goal is kept in mind during all stages of the design, from the selection of the used components to the design of the control software.

3.2.1 Distributed system

Figure 3.2 shows block diagram of the proposed system consisting of multiple monitoring devices and a base station. The base station is a stationary device connected to a constant energy source. It provide the network for wireless communication between monitoring devices and the base station. Over this network it receives measurement data from monitoring devices which is stored in an internal database. It serves as a common access point for users to retrieve the collected data from multiple monitoring devices and provides easy access from a wide range of devices by presenting the collected data via a web interface.

In order to achieve low power consumption, the monitoring device does periodic sampling with configurable period. The monitor devices first sample and then push the collected data to the base station at periodic intervals. The wireless connection is inactive at all

other times. When the base station is out of reach, the data is stored and transmitted during the next successful connection.

3.2.2 Wireless connection between nodes

When the decision was made that a distributed system will be developed this also meant that the collected data had to be transmitted wirelessly to the base station, which is as not straightforward as it sounds, given the constraints imposed by the other main challenges that were discussed in section 3.1.

A wireless protocol that had very low energy consumption while offering enough bandwidth to transfer audio signals and supported a range of around 100m was what the team was looking for.

In the end the choice fell on the ZigBee protocol, which came closest to fulfilling our feature wishlist. The main factors that influenced our decision were

- Range: up to 100m
- Power consumption: very low, as its specifically designed for low-power applications
- Bandwidth: up to 250kB/s
- Complexity: easy to setup and use

The system is designed in such a way, that monitoring devices have to establish a connection with the base station if they want to transmit data. For energy saving reason, the monitoring devices send their ZigBee modules to sleep when they are not actively transmitting data. This means that the base station cannot send data (commands/requests) to the monitoring devices at arbitrary points of times. It has to wait until a monitoring device connects to the base station before it can dispatch messages.

Chapter 4

Hardware Subsystems

In this chapter building blocks of the proposed system, each of which were separately prototyped to allow a good degree of parallelisation, are described. The driver software that makes each unit functional is also mentioned. We discuss the top-level software that brings these blocks together into a functional system in the chapter 5.

4.1 Monitoring Device

The monitoring device is designed to be a simple device that acquires data from the sensors, stores the data and sends it to the base station when the ZigBee link is available. It contains a microcontroller, a set of sensors for data acquisition, a memory card to store data and a ZigBee module for wireless communications.

The following sub-chapters will discuss the implementation of each subcomponent of the monitoring device, shown in Figure 4.1, together with the properties of the hardware and how they map to the requirements of the project.

4.1.1 The Microcontroller (MCU)

As the monitoring device is required to be a battery-operated system that contain a number of diverse peripherals, choosing the right central processing unit was essential. Reviewing existing microcontrollers in the market we opted for an EFM32 Giant Gecko microcontroller from Energy Micro, which contains a 32-bit ARM Cortex M3 core and a set of on-chip peripheral interfaces. The EFM32 was considered to be a good match for the requirements, as shown in the table 4.1.

Some of the more specialized properties of the microcontroller which give it advantages for our solution are briefly explained below:

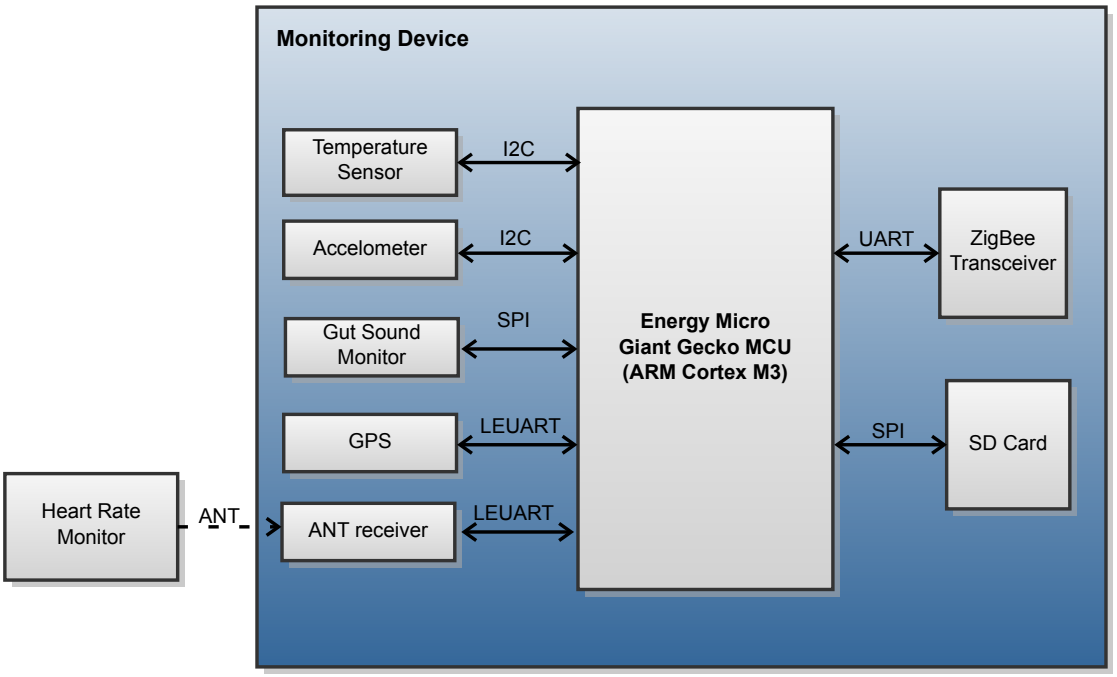


FIGURE 4.1: Block Diagram: Monitoring Device

Energy Modes and Low Energy Peripherals

The EFM32 microcontrollers offer four levels of sleep, and some special peripherals (called low energy or LE peripherals) that are able to keep functioning even in deep sleep modes. Of particular interest to us are the LEUART interface which while deeper sleep modes provide more energy savings, it also becomes more difficult to implement a reasonable level of active functionality. An overview of sleep modes and available peripherals in each is presented in table 4.2.

Peripheral Reflex System (PRS)

Using a signal producers - signal consumers concept, it is possible to make peripherals in the EFM32 microcontroller directly communicate with each other without involving the CPU. Thus, an action can be automatically triggered in peripheral A when an event occurs in peripheral B. This can be used to achieve more stable time-critical operations and less software overhead, as well as saving energy since the CPU does not have to intervene.

Direct Memory Access (DMA)

DMA allows blocks of data to be moved between RAM and peripherals without CPU intervention, thus freeing CPU resources and allowing longer periods of sleep, as well as offering stable high bandwidth memory transfers for operations like audio acquisition.

Device Properties	**Corresponding Requirements**
◇ up to 48 MHz high-frequency operation	◇ handle large data volumes from multiple sensors and audio
◇ different levels of sleep ◇ fast wake up time (2 μS) ◇ 32.768 kHz low-frequency operation for sleep modes ◇ low energy periphals operational even in deep sleep modes ◇ DMA and PRS (Peripheral Reflex System) for acquiring data without waking up the processor core	◇ long battery life ◇ long periods of sleep ◇ low energy periodic sampling
◇ 128 KB SRAM, 1MB flash memory	◇ ease of development ◇ suitable for overheads introduces by C++
◇ 3 x USART with I2S support ◇ 2 x low energy UART (LEUART) ◇ 2 x I2C interfaces	◇ various interfaces for peripherals and sensors
◇ 64-pin TQFP package	◇ ease of soldering (lack of BGA mounting equipment)

TABLE 4.1: Microcontroller properties

Further details of how the PRS, DMA and different sleep modes are used in the system is provided in chapter 5

4.1.2 Heart Rate Monitor (HRM)

The heart rate is an important vital sign for any health monitor, although designing a portable device to acquire heart rate is not a trivial task. The issues of electrode placement, filtering out changes due to subject movement and processing the resulting ECG signal are challenging and time consuming. Another challenge specific to our project is acquiring both gut sounds and the heart rate; it is not possible to reliably

Energy Mode	**Power consumption**	**Description**
EM0	$200\frac{\mu A}{MHz}$	All peripherals active, CPU core active
EM1	$50\frac{\mu A}{MHz}$	All peripherals active, CPU core sleep
EM2	$1.2\mu A$	LEUART, LETIMER, I2C, RTC, LCD, PCNT, LESENSE, ACMP, OPAMP, USB active
EM3	$0.9\mu A$	Full CPU and RAM retention, asynchronous external interrupts and I2C can wake up the device
EM4	$20nA$	All functionality off, GPIO pin retention and wake up from GPIO interrupt

TABLE 4.2: Microcontroller energy modes

Device Properties	**Corresponding Requirements**
low power	low energy periodic sampling
simple UART interface	ease of development
can pair with a particular ANT+ HRM chest strap	monitoring multiple horses

TABLE 4.3: Heart Rate Monitor properties

acquire both signals from the same physical location so a wired microphone or electrode would have to be attached to the horse.

To address both these problems, we chose an ANT wireless module that can be used with any ANT+ compatible heart rate chest strap, commercially available for both horses and humans. Attaching a separate heart rate monitor chest strap and transmitting heart rate information wirelessly solves the problems mentioned above and brings the additional advantage of making the system usable for humans.

Interface: `#TODO: rephrase` LEUART with configurable baud rate, not lots of data so high BR does not make sense

Driver: `#TODO: talk about ANT driver:` LEUART, configure & open network with wildcard id, establish connection & rcv heart rate. search timeout & disconnection & pairing.

4.1.3 Temperature Sensor

5.1.3 Temperature Sensor As with many other warm-blooded animals, body temperature can be a valuable tool for diagnosis in horses. While electronic temperature measurements are usually done by an element that comes into physical contact with the subject, doing temperature measurements in this manner on a moving horse is likely to cause

Device Properties	**Corresponding Requirements**
low power: ◇ 240-325 μA during active conversion ◇ 90 μA in sleep mode	low energy periodic sampling
contactless temperature measurement	portable, mobile monitoring device
passive (slave) device over I2C bus	advantageous for a system with lots of sensors, will not send and generate interrupts unless requested

TABLE 4.4: Temperature Sensor properties

fluctuations in the read values. This is the reason we preferred to use a contactless temperature sensor for our implementation.

`#TODO: Fix this` TMP006 [http://www.ti.com/lit/ds/symlink/tmp006.pdf] which is a contactless infrared temperature sensor from Texas Instruments was chosen for the project.

Interface: The TMP006 offers an SMBus-compatible interface, which is interoperable with I2C and was connected to the I2C interface on the microcontroller. Controlling device functionality and accessing measurement data is done via write to register and read from register commands.

Driver: Our TMP006 driver configures device sleep state and conversion rate by writing values to the appropriate registers (TODO insert reference to TMP006 datasheet). The temperature measurement is provided by reading the two registers which contain the voltage generated by the thermopile and the die temperature. The temperature of the object can be calculated from these two data. However this calculation involves heavy floating point arithmetic and it is rather inefficient on the microcontroller. Therefore, the raw temperature reading consisting of these two values is not processed on the monitoring device any further but sent to the base station, whose ARM11 core is much more efficient at handling this calculation.

4.1.4 GPS

A popular peripheral in many consumer devices today, GPS (Global Positioning System) allows tracking the position of a sensor in terms of global coordinates. While not immediately useful for clinical purposes, the ability to track the location of horses on this level can be beneficial if recording of movements or activity recognition [TODO ref High Classification Rates for Continuous Cow Activity Recognition using Low-cost GPS Positioning Sensors and Standard Machine Learning Techniques.] is desired over a longer period of time in a larger area (for free-roaming horses or other animals).

Commercial drop-in GPS modules are available and straightforward to use, but power consumption while searching limits the possibilities for a battery-powered system with energy efficiency focus. We chose a UP500 GPS module from Fastrax [1] for our project. UP500 is a GPS receiver module with embedded antenna offered in a very small package, and offers significant power savings by providing an option to save satellite data to RAM while still being able to wake up from this state and get a position fix within a few seconds (called a hot start).

Interface: The UP500 with the microcontroller via UART and uses a baud rate of 9600 (8 data bits, 1 stop bit, no parity). The Low Energy UART (LEUART) peripheral on the EFM32 was used for this connection, which can receive data at low baud rates even while in deep sleep mode (down to EM2).

Driver:

- ◇ Data acquisition: For maximum energy efficient operation, the GPS driver configures the GPS module to use LEUART interface with DMA. DMA transfers incoming NMEA messages into a fixed size buffer when new data is available. The LEUART module is configured to generate an interrupt whenever it receives the newline character (0x0A). Since every NMEA message ends with a newline, an interrupt is generated every time a complete message is received, and the contents of the DMA buffer are copied to another internal buffer for processing at a later time.
- ◇ Parsing NMEA messages: `#TODO:` simple string parsing, comma delimited fields,NMEA message types GPRMC and GGPA are handled
  - output: valid position fix, latitude and longitude
- ◇ Sleep mode: The UP500 does not have a dedicated sleep pin, but once satellite data has been acquired it is possible to put the device into a sleep-like mode where the power consumption becomes quite low. This is done by turning off the power to the V_{cc} supply pin, while keeping the V_{bat} supply pin active. By keeping ephemeris data in RAM, the GPS is thus able to establish the position quickly when the V_{cc} supply is reinstated. Switching the V_{cc} supply is done via a transistor attached to a GPIO pin, consult section `#TODO:` for more details.

4.1.5 Accelerometer

In addition to the location tracking capabilities provided by the GPS, it can be useful to measure smaller movements with higher precision. Accelerometer data collected in this manner can be used to detect lameness and injuries. The particular challenge

[1] http://www.fastraxgps.com/products/gpsantennamodules/500series/up500/

Device Properties	**Corresponding Requirements**
. ◇ Ultralow power: $45\mu A$ in measurement mode and $0.1\mu A$ in standby mode ◇ FIFO buffer can hold up to 32 samples	low energy periodic sampling

TABLE 4.5: Accelerometer Sensor properties

for using an accelerometer in our system was the high sampling rate necessity - the collected acceleration data will not be useful at low sampling rates[2]. While high sampling rate itself is not a problem for the EFM32 microcontroller, it conflicts with low energy periodic sampling - if the microcontroller has to poll the device for new data at a high frequency, it will not have a lot of time to sleep. An accelerometer that contains an on-chip buffer can remedy this problem; the MCU can simply enable sampling, go back to sleep, and wake up when the desired number of samples have been acquired to read them from the sensors own buffer.

We chose the 3-axis ADXL350 [3] digital accelerometer from Analog Devices for the project. It is capable of measuring acceleration in the ranges 1g, 2g, 4g or 8g and it has a FIFO buffer which allowed us to implement the scheme discussed in the previous paragraph.

Interface: Similar to the TMP006, the ADXL350 is interfaced through the I2C bus; controlling device functionality and accessing measurement data is done via write to register and read from register commands.

Driver: ADXL350 supports both SPI and I2C interfaces. In this project, it is used in I2C mode. The accelerometer offers a 32-level FIFO which has 4 modes of operation configured by control registers: Bypass, FIFO, Stream and Trigger Mode. The driver configures the FIFO in Bypass Mode which disables the FIFO. The 4-phase scheme (`#TODO: I dont know how to refer this`) is applied for power management.

4.1.6 Gut Sound Monitoring

`#TODO: Insert content from document`

[2] `#TODO: find a good citation for this`

[3] http://www.analog.com/en/mems-sensors/mems-accelerometers/adxl350/products/product.html

4.1.7 Data Storage

Relatively low throughput of ZigBee link between the base station and the monitoring devices makes gut sound streaming inconvenient considering the amount of audio data produced. `#TODO:` : Here we can quote amount of audio.... One possible solution was to stream a smaller amount of data and record a longer audio data into an on-board storage element. Therefore, a microSD card was incorporated into the system.

SD Card solution provides two other benefits to the system besides complementing the limits of wireless streaming. The SD Card can be used to save data collected from all sensors when the base station is out of reach. This way the sensor data is not lost when there is no connection between the base station and the monitoring device. Also, it can be removed by the user to copy the data when needed, since a wide range of devices are SD card compatible.

SD Cards are based on flash memory technology. They support 3 communication modes: SD 1-bit, SD 4-bit and SPI mode[4].

In SD 1-bit mode, the data is transferred over 1-bit wire in a synchronous serial fashion. SD 4-bit protocol is the same as SD 1-bit protocol except that bulk data transfer is done over 4-bit bus. SD 4-bit mode can provide speed benefits over SPI mode. However it is not recommended in software solutions since it requires CRC calculation for each of the four wires and may result in a big computational effort. Also, accessing the complete SD Card Specifications to build an SD standard device requires licenses from SD Card Association[5]. Therefore, in this project, the card is used in SPI mode.

`#TODO: Check this:` TODO: Check this (If your company is planning to manufacture or have manufactured SD standard devices (eg. cell phones, cameras or computers) or SD ancillary products (eg. adapters or SD I/O cards), your company is required to:, I think using SD bus is manufacturing an SD standard device)

To implement microSD prototype system, an existing FAT file system and a low-level disk interface module were used. The card was connected to one of the SPI interfaces of the microcontroller. MicroSD cards support up to 208 MHz clock frequency [TODO citation]. The clock system of the Energy Micro microcontroller allows maximum SPI bit rate to be at half speed of the source clock in master mode `#TODO: Citation: EFM32_GG Ref` . Theoretically, SPI clock can be up to 24 MHz when the source clock is set to 48 MHz. However, there is an upper speed limit caused by delays of inputs and outputs of the microcontroller which is not specified by the manufacturers[6]. Our microSD driver sets SPI clock speed at `#TODO: X` Mhz at which it operates safely.

[4] http://alumni.cs.ucr.edu/~amitra/sdcard/Additional/sdcard_appnote_foust.pdf
[5] https://www.sdcard.org/developers/howto/
[6] http://forum.energymicro.com/topic/288-microsd-card-spi-baudrate/

FIGURE 4.2: XBee prototyping interfaces

4.2 Wireless Interfaces

4.2.1 Data transmission - ZigBee

Because of the decision to design a distributed system that transmits data wirelessly between monitoring devices and the base station, it was necessary to come up with a customized stable wireless communication path. As discussed in section 3.2.2 the ZigBee protocol was chosen for the wireless communication between the base station and the monitoring devices.

As ZigBee is an open standard there are multiple vendors offering devices implementing the protocol. Because of the widespread use in homebrew electronic projects we decided to use devices by Digi which offer a whole ZigBee based product line called XBee.

The main advantage was that the prototyping could be sped up due to availability of breakout boards, and XBee to USB converters (figure 4.2). This allowed to implement, test and debug the software module for wireless communication on a normal computer before porting it to the embedded system.

The wireless software module uses an existing open-source software library to handle the low level communication with the XBee hardware, and adds an object oriented interface as well as utility classes that allow to use and control the wireless connection at a high abstraction level. More details on the design of the software module are given in section 5.2

4.2.2 Communication with Heart Rate Monitor - ANT

```
#TODO: Not yet written
```

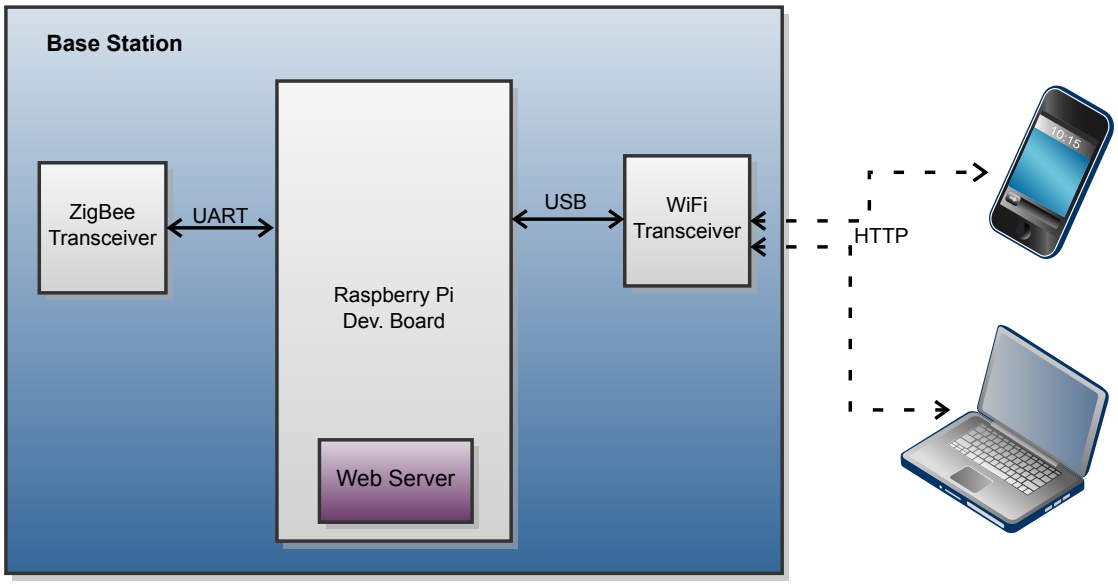


FIGURE 4.3: Block Diagram: Base Station

4.3 Base Station

The base station collects data from the monitoring devices, stores this data in a database and makes it available and via a webserver running on the device itself. The advantage of using a website to provide access to the collected data is the support for a variety of devices with web access. In addition, it compensates disadvantage of the short range low data-rate ZigBee connection making the data accessible from anywhere.

4.3.1 Hardware platform

The base station acts as a bridge between the proprietary monitoring stations and the end user, who wants an easy way to access the data that is collected by the system. The challenges for the base station lay mainly in the software domain. The only requirements that exist for the platform is that it has a to be able to interface an XBee device, a WLAN transceiver and provide means for mass data storage.

For these reasons the decision was made, that an existing hardware platform would be used to implement the base station. The choice fell on the Raspberry Pi which provides a number of positive implications that are listed in table 4.6.

4.3.2 Raspberry Pi

In summary it can be said that the Raspberry Pi is very well suited for rapid development of embedded systems, as long as they are not supposed to run on battery. It offers the

Price	35
Software	Raspberry can run a Debian based distribution, which gives access to a huge set of open-source programs and libraries
Connectivity	Features an ethernet port, USB host capabilities and HDMI output that could be used to attach a display to the base station
Extendability / GPIP	20 GPIO pins, support for I2C, SPI, Serial
Storage	SD-Card slot with support for cards up to 64GB
Power consumption	Very low, between 1.5 and 2 Watt
Size	Very small form factor (85.6mm x 56mm)

TABLE 4.6: Raspberry Pi capabilities

convenience of developing software in a Linux environment while giving direct access to low level I/O capabilities for interfacing custom hardware.

4.3.3 XBee

As mentioned in section 3.2.2 we use of XBee devices to implement the ZigBee network for data transmission between monitoring devices and the base station. These devices are widely-used in (homebrew) electronic projects, and they have the advantage that a Raspberry compatible XBee adapter board already exists. The adapter goes by the name of Slice of Pi and is shown in figure 4.2.

Chapter 5

System Software

Having discussed how the functionality of each subsystem is implemented in chapter 4 we now provide a description of the software that brings these subsystems together into a complete system.

5.1 Monitoring Device

For the reasons discussed in section 3.1, the functionality of the monitoring device is kept simple and can be summarized as gathering data from the sensors, storing the data offline in the SD card when the ZigBee wireless connection is not available, and sending the gathered data to the base station. Figure 5.1 illustrates the data flow on the monitoring device.

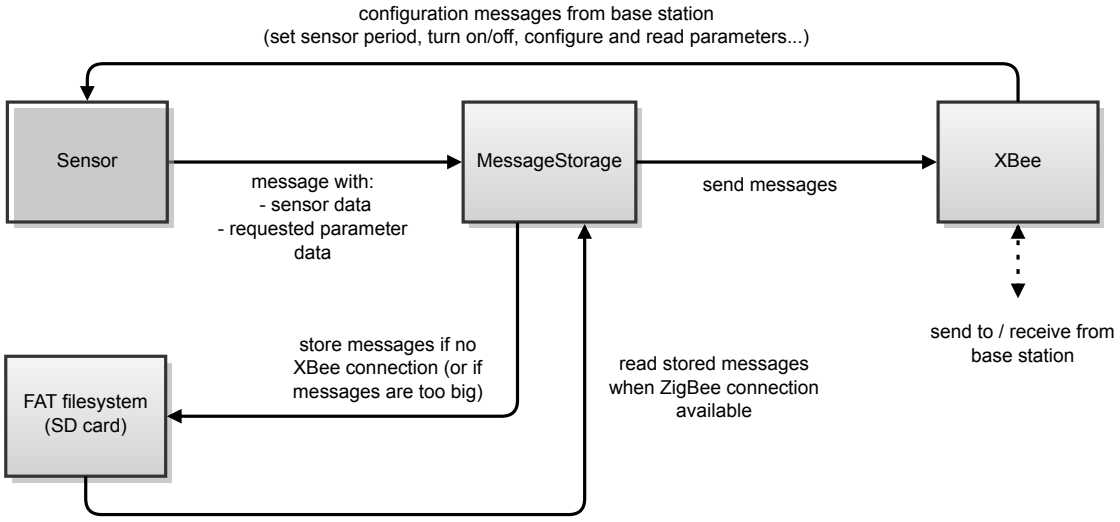


FIGURE 5.1: Monitoring Device: dataflow

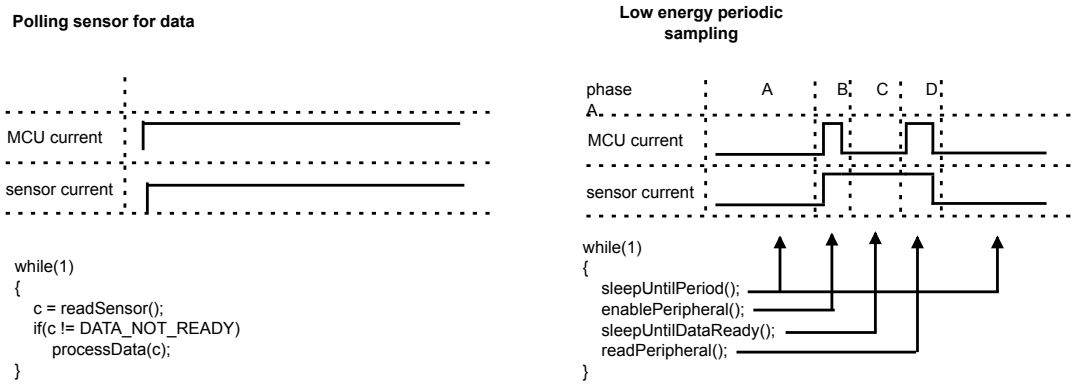


FIGURE 5.2: Sampling comparison

Furthermore, to keep the energy efficiency focus, this data flow has to be implemented using periodic sampling and automated data acquisition whenever possible. The following subsections will describe these implementation details for the system software on the monitoring device.

5.1.1 Development Paradigm and Environment

`#TODO: rephrase` - we actually do not use pure OOP methodology, we use a mixture of OOP and procedural to get the best of both worlds (encapsulation and polymorphism from C++, and C where small simple quick drivers/structures are needed). The monitoring device software was developed using C++ with IAR Embedded Workshop for ARM as the development environment. C++ classes were created to hide implementation details for most peripherals and subsystems, using the emlib peripheral library from Energy Micro was used to interact with the EFM32 peripheral.

The choice of object-oriented paradigm (OOP) and C++ may be considered unusual for a highly integrated embedded system, but we considered the advantages to outweigh the penalties for our project: `#TODO: justify C++`

5.1.2 Low Energy Periodic Sampling, Sensor Interface and Drivers

We now discuss our definition of low energy periodic sampling for the monitoring device, and the relevant OOP basis we constructed towards its implementation. The basic idea behind low energy periodic sampling is that a receiving a continuous stream of data from all the sensors all the time is not necessary for monitoring purposes is unnecessary. Depending on the sensor type, receiving a sample with a period of tens of seconds, perhaps minutes can be sufficient for analysis. In this case, since the sensor and the microcontroller are not required to be active all the time, significant energy savings can

Phase	**MCU state**	**Sensor state**	**Duration**	**Description**
Phase A	asleep	asleep	as desired	both sleepig and waiting for next sampling period to start
Phase B	awake	awake	MCU dependent (wakeup time)	microcontroller wakes up peripheral, peripheral starts sampling
Phase C	asleep	awake	sensor dependent (sampling rate)	microcontroller sleeps while waiting for peripheral to complete sampling
Phase D	awake	awake	bus dependent (data read speed)	data ready, microcontroller reads data from peripheral, then go back to Phase A

TABLE 5.1: Periodic sampling power phases

be achieved by putting them into sleep mode until the start of the next period. This scheme can be further extended by introducing an additional period of sleep for the microcontroller after it awakens the sensor and waits for data. An illustration of how this compares with the continuous polling approach can be found in figure 5.2. It can be seen that the low energy periodic sampling offers much less current consumption compared to polling.

This energy saving scheme can be applied to any sensor or peripheral that supports sleep mode. As we chose all only sleep-supporting sensors for our implementation, we were able to identify a set of properties and functions common to all sensors which forms the Sensor base class. The properties and methods defined in the class are listed in section `#TODO: Ref: appendix sensorinterface.h`.

To summarize, the setSleepState function is used to put the sensor to sleep and wake it up, the sampleSensorData acquires data from the sensor into internal buffers, and readSensorData gives access to the data in the internal buffers wrapped in a SensorMessage structure (described in section 5.2.3). Individual sensors drivers are derived from this base class and implement the common methods in their specific way, as described in chapter 4

5.1.3 Timekeeping and Alarm System

Timekeeping and Alarms System To implement the periodic sampling scheme discussed in the previous section, it is necessary to keep track of real time and trigger an alarm when an action (such as waking up the sensor from sleep) is needed. The same implementation can be used to provide small fixed-length blocking delays in code, which are useful in the driver implementations.

The EFM32 has a dedicated Real Time Counter (RTC) peripheral which makes this implementation easy, but the challenge is once again doing this in an energy efficient way. For example, it is possible to configure a system tick that wakes up the MCU every millisecond and updates an internal counter, but waking up every millisecond is very inefficient in terms of energy. Most of the time, we require periodic alarms that are generated in the range of seconds for the low energy periodic sampling scheme. This means timer tick interrupts that wake up the processor every second is enough for our purposes.

To our convenience, the EFM32 RTC peripheral can be kept active down to deep sleep (mode EM2) by using the low frequency oscillator (`#TODO: Ref` refer to section TODO section:pcboscillators for details) and be configured to generate interrupts only when the tick counter reaches a certain value. This allows us to do timekeeping at a very low energy cost and generate alarms with second-precision. Additionally, it is still possible to generate millisecond-accurate interrupts using the secondary compare register when needed.

This functionality is encapsulated inside the AlarmManager class, which offers easy energy-efficient timekeeping and periodic alarm generation functionality. It can be used to generate periodic or single-shot alarms that become triggered after a given number of internal ticks. The triggered alarm executes a callback function specified at the alarm creation time. The internal tick period itself is configurable from milliseconds to hours. The class also supports an energy efficient delay function that can be used to create a blocking delay with millisecond resolution while putting the MCU to sleep in the desired mode. Finally, the class exposes the number of seconds and/or milliseconds elapsed since the start of monitoring device operation, which is used to generate timestamps for the sensor messages that are sent to the base station `#TODO: :` connect this to appropriate section about relative timekeeping.

One final issue regarding timekeeping is persistence across reboots. The monitoring device may stop and restart execution after some time due to the battery running out. To keep the integrity of timestamps, the RTC second counter is regularly backed up to the SD card and re-read upon reboot.

5.1.3.1 Task Scheduling and Sleep Management

While it is intuitive to model the functionality that reads data from each sensor as a separate task and then use a real-time operating system to execute these tasks concurrently, we preferred not to use an RTOS as they typically use high-frequency system tick interrupts to do task scheduling and dispatching, and a high frequency system tick is not compliant with our low energy goals due to the reasons explained in the previous

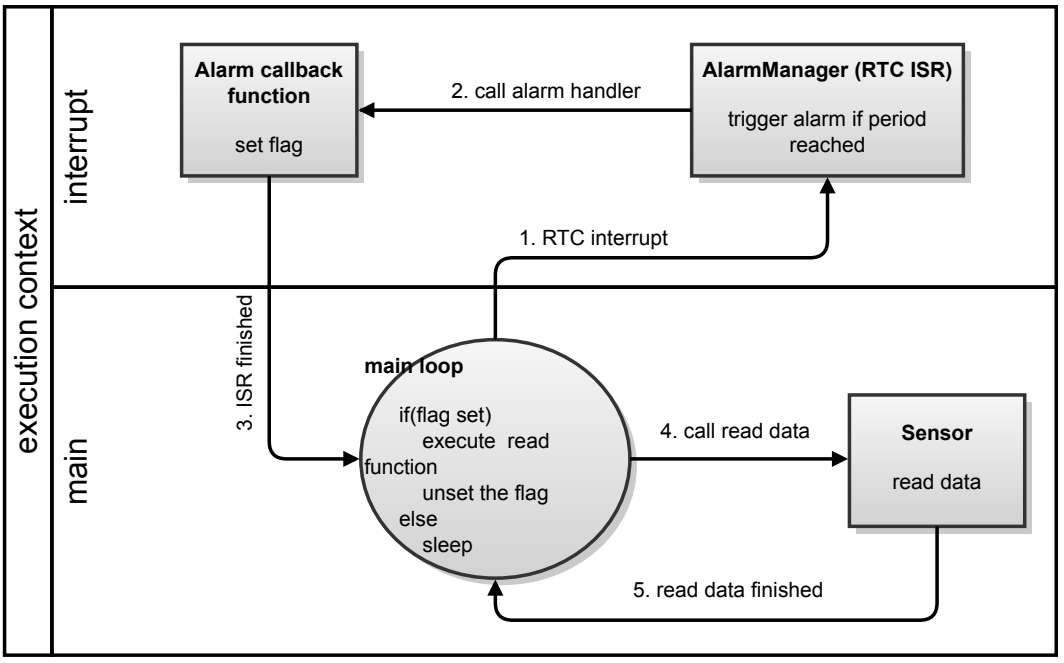


FIGURE 5.3: Deferred reading

section. There are RTOS ports for EFM32 that avoid this, Therefore, we rely on the periodic alarms generated by the AlarmManager to trigger sensor readings.

The problem with this approach is the fact that the alarm handler callback functions are executed inside the interrupt context. If the alarm handler takes a long time to execute (for example, read a large amount of data from the sensor) this will result in staying in the interrupt context for a long time. Even though the EFM32 supports nested interrupts, it is considered to be bad practice to have big interrupt service routines since it can introduce instabilities into the system. Our solution to the problem was to adopt the deferred procedure call (DPC) approach for periodic readings, illustrated in figure 5.3.

periodic alarms for waking up and reading sensor, set allowable sleep level in order not to miss interrupts, minimal ISRs and deferred data reading

5.1.3.2 Port Configuration and Access System

similar code reuse / C+++ model / developed across multiple boards (EFM32 GG and TG STK, GG DK, PCB) which had different configurations and ports. expose bus interfaces via singleton/factory manager classes. generic include file includes part-specific config file depending on defined part number in the project.

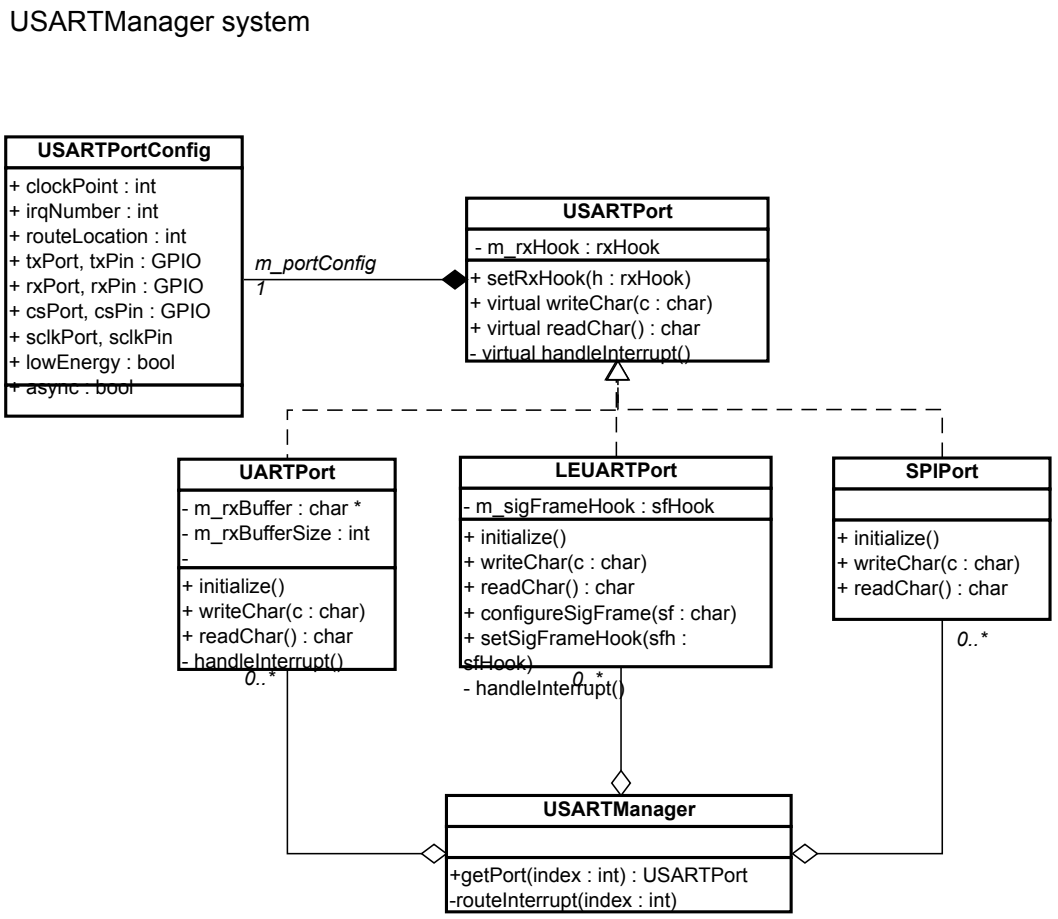


FIGURE 5.4: Class diagram: USART Manager

5.1.3.3 Automated Data Acquisition: DMA and PRS

#TODO: Jose: why? decreases load on CPU and allows more sleep, guarantees better timings for time-critical/BW-critical operations

5.2 Wireless Communications

5.2.1 XBee basics: Transparent vs. API

The XBee firmware can operate in two modes, transparent and API. Transparent mode is the simplest way to use the XBee devices and it can be seen as a serial line replacement between two nodes. The problem with this mode is that it is not suited for complex network configurations. Every time a network parameter has to be changed (e.g. destination address) the devices have to enter a configuration mode which alone takes two seconds. In addition the mode is not suited for transmitting larger message frames.

The second mode of operation is called API mode. This is a frame based approach, where all all data and command messages have to be packed into well defined frames where informations like destination are contained in the message header and device parameters can be changed on the fly. In addition this mode makes it possible to implement a more reliable data link, because successful transmission of each frame is acknowledged.

For these reasons the system uses the modules in API mode. This adds complexity to the implementation because it requires an intermediate layer for creating the frames according to the specifications, but the advantages we get from this mode outweigh the additional effort that has to be put into the implementation.

libgebee

Libgebee[1] is an open-source library written in C that provides a portable interface for accessing XBee devices. Its main task is creating XBee compatible frames, while hiding away low level procedures and base system dependent interactions with the serial port.

Because library has been written for the ZNet 2.5 XBee standard which has been replaced by the ZB standard over the last year it was necessary to update the library source code by adding newly introduced frame-types and updating existing ones to the new standard. These additions might flow back into the official repository if the author is interested in a cooperation.

The library also needed to be ported to the Energy Micro architecture before it could be used on the monitoring devices. Because the library aims at portability, it has a well defined interface between target system dependent functionality and the actual functionality of the library, which makes porting the library to a new architecture significantly easier. In our case, porting to the EFM32 architecture was almost trivial using the developed port/bus abstraction classes described in section 5.1.3.2 for the UART interface and the timekeeping class described in section 5.1.3 for operation timeouts.

5.2.2 XBee Interface

While the libgebee driver adds a nice layer of abstraction over the construction of API frames, it is not well suited for the direct use in an application with the complexity of the system we are building. Too many function calls are required to transmit or receive data and it would lead to a large amount of code repetition. Furthermore our core application is written in C++ which allows for a higher level of abstraction.

A C++ wrapper and a number of utility classes were developed to simplify data transmission and control of XBee devices (from the viewpoint of the core application). Figure

[1] http://sourceforge.net/projects/libgbee

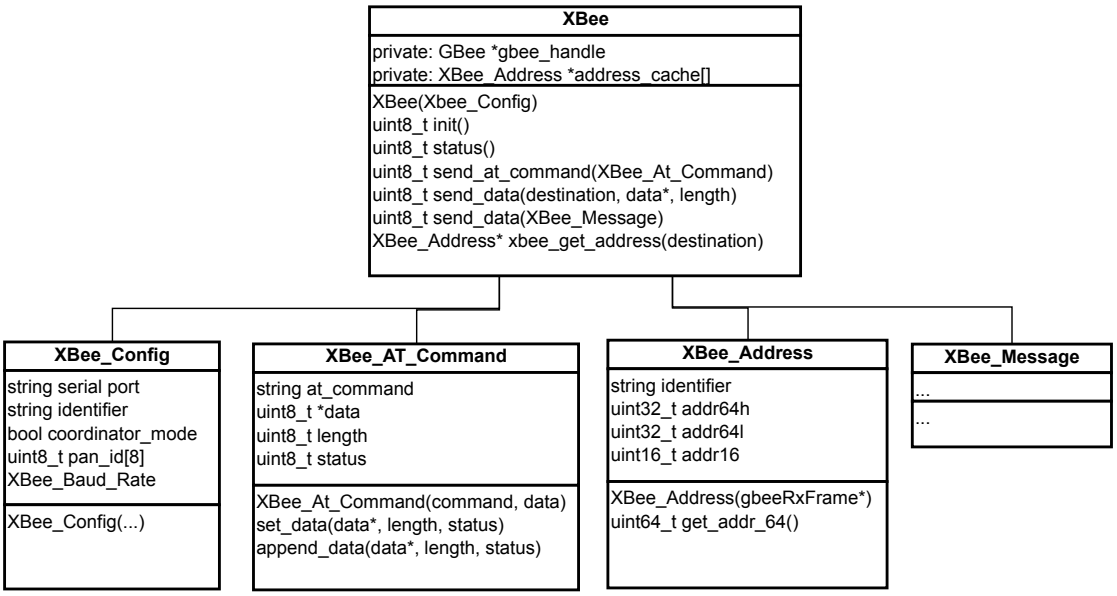


FIGURE 5.5: Class diagram: XBee Interface

5.5 shows the public interface functions of this wrapper and the utility classes will be discussed in the following sections

transmission of data

`#TODO: Insert figure:` use state machine / flow chart diagram to visualize the algorithm

5.2.3 Message handling

The size of data frames that can be transmitted over ZigBee networks in one transmission is limited to a payload size of 84 Bytes. Our system needs to be capable of sending and receiving messages that exceed this size limit by multiple factors. For this purpose a XBee_Message class was implemented that takes care of chopping arbitrary sized messages into frame sized pieces and re-assembling messages from received frames.

XBee_Message

The XBee_Message class expects a pointer to the data that should be packed into the message and the size of the data field. During instantiation the data is copied into a object internal buffer and the object takes over the responsibility for handling the allocated memory.

The XBee interface uses objects of this class internally for message handling and also accepts objects of this kind as parameters for the send function. When data is received

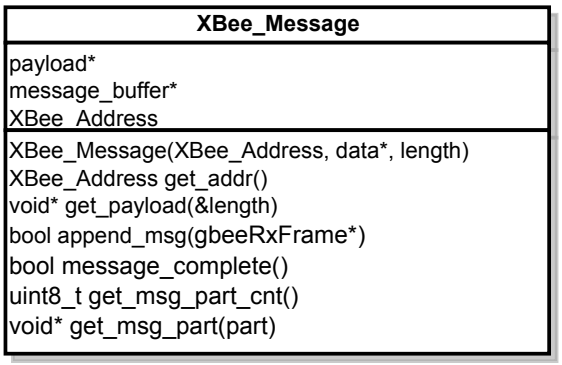


FIGURE 5.6: Class diagram: XBee Message

over the ZigBee network, the append_msg function of this class is used to reconstruct the messages consisting of multiple parts before passing a handle to a complete message object to the user.

The XBee_Message class is aimed at handling arbitrary data of arbitrary size and does neither know nor care about the type of data it contains. It treats all data in exactly the same way. The definition of the data that allows us to give a meaning to it is done independently.

Message Types

The previous paragraph explained how the XBee_Message class is used to transfer arbitrary data between network nodes. This paragraph deals with the way in which meaning is added to this data.

We use an inheritance based approach to define message types. Inheritance is particularly suited for this case as we can group messages into categories that have many common fields and only limited need for specialization.

This is implemented with structures instead of classes because structures guarantee that members are stored sequentially in memory which makes the job of serializing and deserializing the messages for transmission purposes a lot easier.

The design of the message type classes allows to pack an array of multiple measurements (from the same sensor) into one message, which can help to increase the data throughput by improving to data to overhead ratio.

Message de-/serialization

The inheritance based message type data structures make extensive use of references, which is why they cannot be used for transmission as they are, but have to be serialized

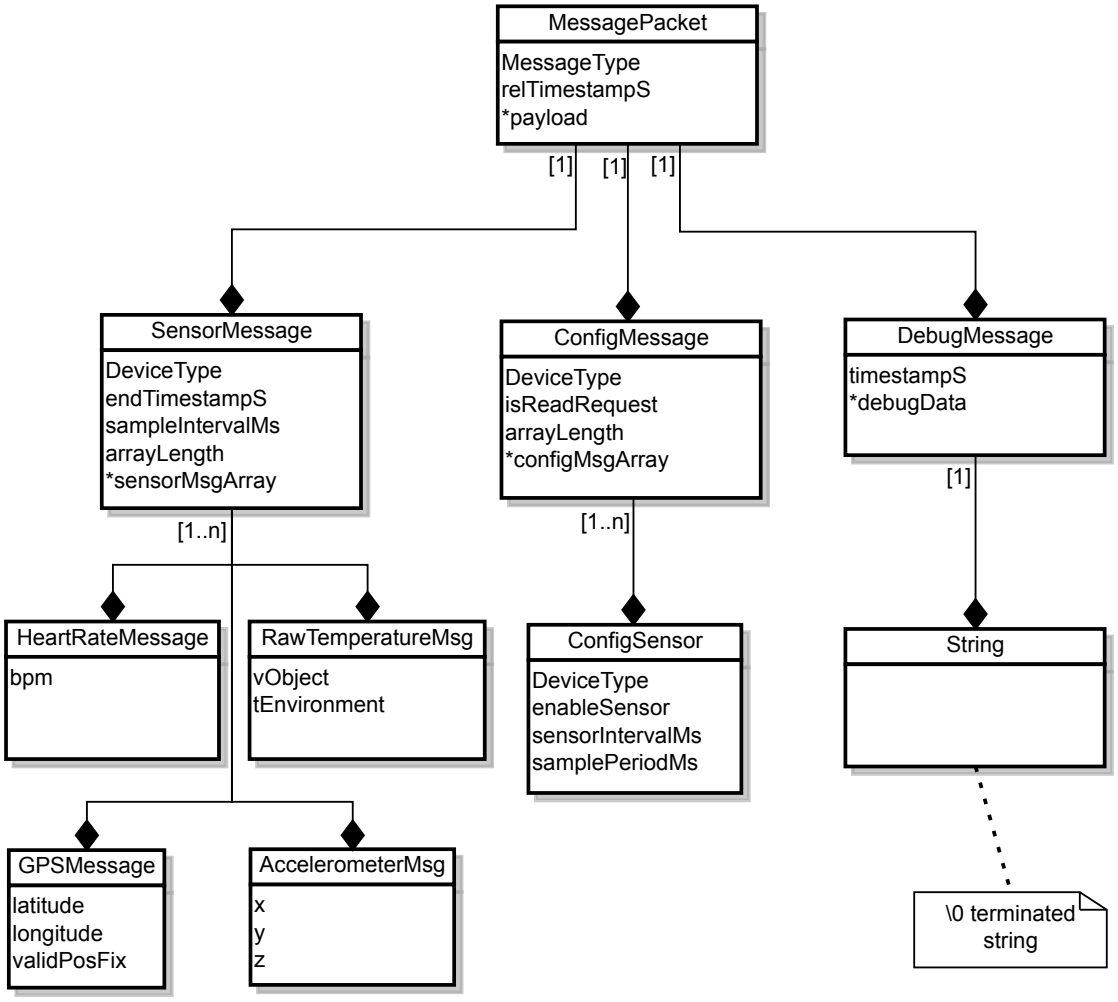


FIGURE 5.7: Class diagram: Message Types

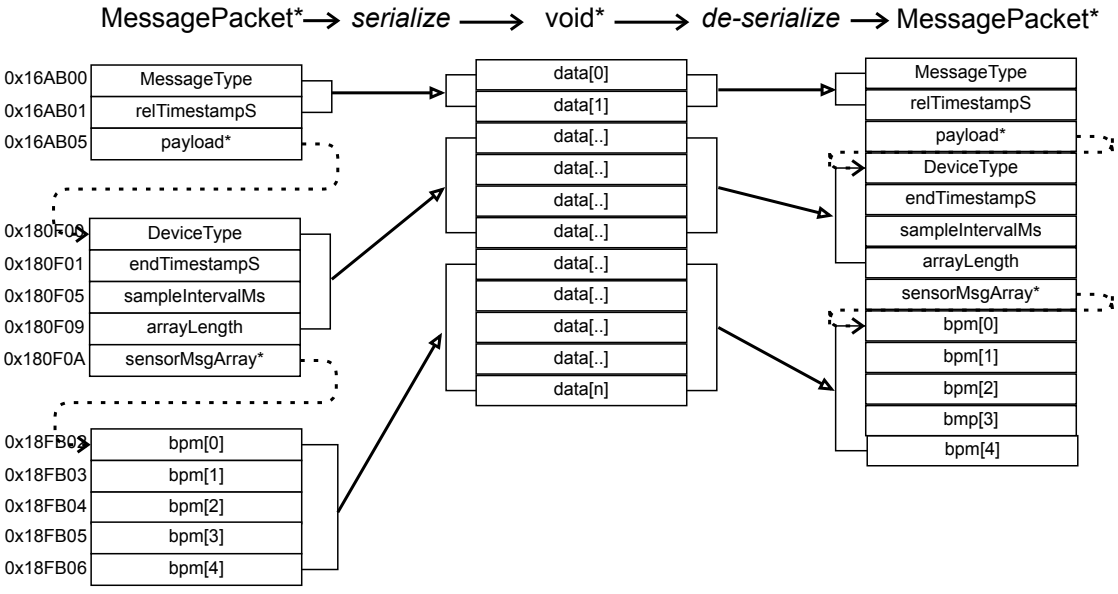


FIGURE 5.8: Message serialization

first. The process of serialization follows the references and copies all data of the message into a continuous memory area that can be transmitted across the ZigBee network, and deserialized into a message object on the other side. Figure 5.8 depicts the steps involved in the process.

Message de-/serialization is implemented by two free-standing functions. The one that performs serialization expect a MessagePacket (see 5.2.3) and a pointer to pre-allocated memory as arguments and copies the values from the MessagePacket into the continous memory. The deserialize function performs the inverse of this operation, and returns a MessagePacket that is an exact copy of the source MessagePacket, except for the destinations of the pointer members.

Timestamps - relative and absolute timekeeping

The timekeeping of the system is based on a the Unix style Real Time Clock (RTC), with the peculiarity that the monitoring devices do not count absolute time but time relative to the moment when they were first switched on. This decision is based on the fact that we cannot guarantee that the monitoring devices are able sync their internal RTC to the RTC of the base station when they are switched on (base station not in range, no GPS signal) which might lead to artifacts in the measurement data.

$$T_{abs} = RTC_{abs} - (T_{transmission} - T_{measurement})$$

To calculate the absolute time when a measurement was taken, the messages contain two relative timestamps - one that shows when the measurement was taken, and one that shows when the measurement was transmitted. The absolute time is calculated on the base station, right before the message is stored in the database.

5.3 Base station software

There are two independent software modules running on the base station software that use different Inter-process communication methods (IPC) to cooperate with each other. The receive and store module (R&S) is responsible for the communication with the monitoring devices and storing received data into a database. The data presentation module on the other hand provides a graphical user interface (GUI) for the collected data and allows the user to configure the base station and the monitoring devices connected to the network.

The decision to divide the base station software into two independent parts was based on the fact that is is possible to create a very clear and well defined interface between

the two modules. This allows us to develop the parts independently and use different programming languages that are best suited for the task. It also makes it possible to modify or replace the data presentation module without the need to adapt the rest of the system.

This is especially important in the case of the data presentation module, because we cannot foresee how the collected data is going to be used, and what is the optimal way of presenting the data to the user. In addition to that, the task of analysing and presenting the data was assigned to our sixth team member during the planning phase of the project, and had to be taken over by the rest of the team. For this reason we could not allocate as many man-hours to the presentation module as we would have liked, as it is a non-critical part of the system.

5.3.1 Inter-process communication

Database

A file based sql database (sqlite3) is used to share the collected data and information about network status and node configuration between the two modules. The direction of communication is one-sided. The receive and store module writes to the database and when the user accesses the web interface the data presentation module extracts the requested information from the database. It can be seen as a passive way of communication as the one side who writes to the database doesnt care about the other side.
`#TODO: more details about sqlite3?`

Message queues

POSIX message queues are used to implement reactive communication between the processes, e.g. if the user requested a change of configuration in the web interface. The R&S module will check the message queue periodically and react to them as promptly as possible. In case of messages that affect the base station the commands are executed almost immediately, in case of messages that concern monitoring devices, the messages are queued until the device connects to the base station, and the message can be dispatched.
`#TODO: more details about mqueues?`

5.3.2 Receive and store module

The receive and store module sets up and controls the ZigBee network and provides measurement data storage facilities in a sqlite3 database. It implements a simple state machine, waiting for incoming messages over the ZigBee network or the IPC message queue.

When there is incoming data from the ZigBee network, the module enters the receive state and tries to receive an XBee_Message. When a complete XBee_Message is successfully received, the message content is deserialized and stored into the database. When theres not pending data left, the message queue is checked for messages destined for the node that just transmitted data. In case there are pending messages, they are dispatched, else the module goes back into the idle state.

When there is incoming data over the message queue, the module checks if the request can be handled locally in which case it is executed immediately. If the request is directed at a monitoring device, the R&S has to defer dispatching the request until the next time the destined monitoring device connects to the base station. The reasons why the base station has to wait for the monitoring devices to open up a communication channel were explained in section 3.2.2

5.3.3 Data presentation module

The data presentation module is written in Python and uses flask[2], a lightweight Python web framework to generate the web interface dynamically.

Web Interface

The framework strictly divides display and logic parts of the website. The dynamic content of the pages is generated on the fly with the whole functionality of Python

[2] http://flask.pocoo.org

Chapter 6

PCB design

#TODO: introduction that mentions why we designed our own PCB (small device that can be contained in splashproof case + lots of peripherals + attaching to a moving horse requires extra stability).

6.1 Schematic

The schematic and layout were designed using CadSoft Eagle software version 6.1.0 It is not considered to be the most user friendly software in terms of the usage interface, but is fairly simple and straightforward. Furthermore, the user community is very large, therefore it is possible to find various component footprints already made by others to be used in the design. This decreases the overall time required to make a PCB design. Nevertheless, as we were using very specific components (ANT, GPS, etc.) a footprint for them had to be designed manually by consulting the datasheet.

FIGURE 6.1: Block Diagram: Parts

6.1.1 Power conditioning

The power conditioning circuitry consists of two stages. First stage incorporates a constant current constant voltage single cell Lithium Ion battery charger centered around a ADP2291 chip from Analog Devices. The second stage is a low dropout voltage (LDO) regulator based on a ADP1706 chip from the same manufacturer. For schematics see `#TODO: Ref` Appendix X.

6.1.2 Battery charger

The input voltage of the charger is in the range of 5 to 12V, therefore no power supply providing higher voltage than this is allowed to be used. D1 acts a reverse voltage protection diode in case the power supply is connected with a reversed polarity to the board. In case this ever happens it will not cause any damage to the components.

C4 and C16 capacitors are used to filter out the noise that might come from the power supply. The chosen values of 820uf and 0.68uf are sufficient for this purpose. Resistors R2 and R6 placed in series with a battery provide current measurements for the charger. Hence, the charging current is adjusted by changing the R2 and R6 resistor values, that is currently set to 750mA. Internal LED driver is used to indicate the charging status of the battery. Charge status LED1 is ON when the battery is charging. Transistor Q1 acts as a pass device which provides a charging current to the battery. It was chosen with a reserve in parameter specifications. The device can handle continuous collector current of 3A and has a V_{ceo} of 60V. This enables the device to function properly not only in the current setup but in a scenario when a higher current is required for faster charging. As a driver requirement the transistor has to have a certain minimum PNP beta (hfe coefficient), which for current configuration has to be no less than Imax / 40mA, where 40 mA is the base drive current and Imax is the charging current. This gives us the minimum beta of 18.75 that the transistor has to have. The chosen one has an hfe of 80 at collector current equal to 1A.

In order to keep it from overheating a thermal protection is used by utilizing the NTC1 thermistor. This enables the charger to monitor the temperature of a pass device and decrease the charging current accordingly. Shutdown temperature of the charger is set internally to 100C.

The charger works in step-by-step sequence illustrated by diagram 6.2:

- **Shut-down:** If the input voltage is lower than 3.8V the charger is in the shut-down mode. Power consumption from the battery is 1uA.
- **Pre-charge:** When a voltage higher than 3.8V is detected at the input of the charger the charger checks the voltage across the battery and enters a pre-charge

FIGURE 6.2: Battery charging sequence

state if the battery voltage is lower than 2.8V. In this stage the maximum supplied current is 75mA. In cases where battery is deeply discharged and measures less that 1.5V the supply current is set to 150mA.

- ◇ **Fast charge:** If the voltage is higher than 2.8V the charger enters a full current fast charge mode which continues until the battery voltage reaches 4.1V
- ◇ **End-of-charge:** The charging current is reduced as the battery reaches its full capacity and once this current is less than 75mA an end-of-charge mode is initialized.

Capacitor C6 determines the time of all operation modes. The ratio between the fast charge mode to pre-charge and end-of-charge is always 1/6. Given the capacity of our battery is 2200 mAh time required for it to fully charge with current settings is 2.9 hours. Therefore a timeout of 3 hours for fast charge mode is sufficient. C6 capacitance is calculated from the following formula $Ct = tchg * 1uf/1800$ and is equal to 0.1 uf. The pre-charge and end-of-charge modes timeout in 30 minutes. Time limit of the fast charge mode is required as a precaution in case the battery fails to reach end-of-state mode for any reason.

6.1.3 LDO regulator

The LDO regulator takes in the provided voltage by the battery and outputs a fixed value of 3.3V required to power all on-board devices. Decision to use this type of regulator was made from the fact that the difference between input and output voltage is very small. Therefore, using a traditional linear regulator in this case is not an option. Our chosen device has a maximum dropout voltage of 55mV at 100mA output current. This figure increases to 345mV at output current of 1A respectively. The latter dropout will never be achieved, as in the worst case scenario when all peripherals are operational

the maximum current consumption will be `#TODO: check:` datasheet GPS, ANT, Xbee SD card + MCU 9.6mA all peripherals combined will hardly ever pass the 100mA current consumption point.

This LDO regulator was designed for operation with small value ceramic capacitors for space saving applications. Nevertheless, using a larger value output capacitors improves the transient response. Hence, as space is not an issue a value of 150uF was chosen. The regulator has a built in accidental short circuit protection. If for any reason the output becomes shorted, the regulator will conduct the maximum amount of 1.5A current into the short, increasing the junction temperature to critical level and triggering the thermal protection as a result turning off the output.

A soft start function is implemented by connecting the C3 capacitor to ADJ pin of the regulator. This ensures a gradual output voltage ramp-up. The ramp up time is calculated using the following formula $T_{ramp-up} = 0.8V(C3/1.2uA)$ Tramp-up represents the time it takes for the output to reach 90% from 0%. In current configuration where the value of C3 is 10nf the ramp-up time is 6.67 ms.

To insure a stable power supply to all components organic solid capacitors with conductive polymer (OS-CON) are used. Their benefits are discussed in capacitorlab and capacitorplus

6.1.4 MCU

As was mentioned earlier the MCU used for the project is EFM32GG332 based on ARM Cortex M3 core with a 1024KB of Flash and capable of running at a 48MHz speed. The chip has several clock sources and different crystal oscillators are used to generate that clock. XT1 is a high speed oscillator (HFXTAL) rated at 48MHz and running in a fundamental mode. XT2 is a low frequency oscillator (LFXTAL) rated at 32.768KHz and is produced by MEMS processing technology. This oscillator is used for peripherals running in low energy modes and operates in fundamental mode as well. The MCU will not work with a high speed oscillator that is specified to operate in an overtone mode. To insure that the crystal operates correctly, it has to have a certain load capacitance, in our case for XT1 this is 18pF and for XT2 it is 7pF.

In case there is a need to reset the MCU it can be done by using the S1 global reset button. Capacitor C14 is used in parallel with the switch to implement a hardware de-bounce function.

A standard approach for decoupling is used. One large capacitor C2 rated at 150uF is used for the whole system with 0.1uF C7-C12 capacitors used to decouple separate power pins. In our design we are not using ADC of the MCU; therefore it was not

necessary to separate the digital and analog power domains. Separation is usually done to ensure better isolation and noise suppression between the power domains.

For programming and debugging the custom made PCB a debug interface consisting of clock input (SWCLK), data in/out lines (SWDIO) and serial wire output (SWO) had to be exposed. An external development kit was used to program/debug the PCB. By supplying the power to the PCB from the development kit via VMCU pin it was possible to measure the current consumption of a target with an Energy Aware profiler.

`#TODO: Integrate with Monit Devs`

6.1.5 I2C devices

The TMP006 MEMS sensor from Texas Instruments was chosen for its ability to measure the temperature without the need of making a contact with an object. The measurement is done by allowing the thermopile to absorb the IR energy emitted from an object and based on the thermopiles voltage change determine its temperature.

Temperature sensor uses SMBus for transmitting transmit the data. The address of the device on a bus is set to 1000000 by grounding ADR0 and ADR1 pins. DRDY pin indicates whether the data is available to be read by the MCU and requires a pull-up resistor in order to operate correctly. Decoupling of the sensor is performed using a 22uF OS-CON and a 0.68uF plastic film capacitors.

The I2C bus is shared with an accelerometer. As this is not the only mode the accelerometer can operate the I2C mode had to be set by pulling the CS pin high. By setting the SDO/ ALT ADDRESS pin to high the accelerometers address becomes 0x1D on the bus. No pins are left floating in order to prevent leakage currents. Hence the Reserved_3 pin is connected to +3.3V and Reserved_2 together with Reserved_1 are grounded. Optionally they can be left floating. The device can operate in several power supply modes. In this particular design it operates in a single supply mode, where VS is connected together with VDD/IO. Decoupling is performed for both of the pins using two 0.1uF ceramic capacitors. In addition to that a 22uF OS-CON is used at the VS pin to minimise the digital clocking noise.

ZigBee

`#TODO:`

ANT

`#TODO:`

GPS

This module requires two power supplies, the first one VCC is used for digital parts and I/O and the second one VBAT is used for non-volatile back-up block. By default GPS does not have low energy mode that can be switched on demand, it does go into low energy mode automatically once it has acquired the satellite positions and collected the Almanac data. In normal operation mode GPS typically consumes 115mW of power which translates to 35mA of current draw from the battery. Nevertheless, this figure can rise up to 40mA. Power consumption drops to 75mW (22mA current draw from battery) once the GPS has finished with the cold start. This figure is still high. Power supply can be switched off for the VDD block if the navigation is not needed. This will enforce the GPS to enter the backup mode. The power then will be drawn from VBAT which is used for keep the collected satellite data in RAM. This will ensure a fast TTFF (time to first fix) once power on VDD is re-applied again. Power draw in this mode is in the region of 15uW (4.54uA).

The PPS (pulse-per-second) output is not used in the design as it is only required for timing purposes.

For decoupling purposes a 0.1uF ceramic capacitors are used for both power pins with an addition of 22uF OS-CON for the VCC.

MEMS Microphone

`#TODO:`

6.2 Layout

`#TODO:`

When positioning the components on the PCB some considerations had to taken into account. These are going to be discussed step by step starting with the power conditioning circuitry.

The pass transistor Q1 used in the Charger generates a lot of heat, according to the calculation it dissipates `#TODO: calculate` of power. In order to keep a low profile board the PCB itself acts as a heatsink. The pad of the transistor that draws away the heat from the crystal cannot be connected to the largest copper area of the board - GND, as the pad is internally connected to Vdd. Therefore a dedicated area was left for that purpose and thermally isolated from the neighbouring components. This also prevents the shortening of their lifespan. Multiple vias are used to connect the upper layer of copper with the bottom one making the heat dissipation more efficient. In case

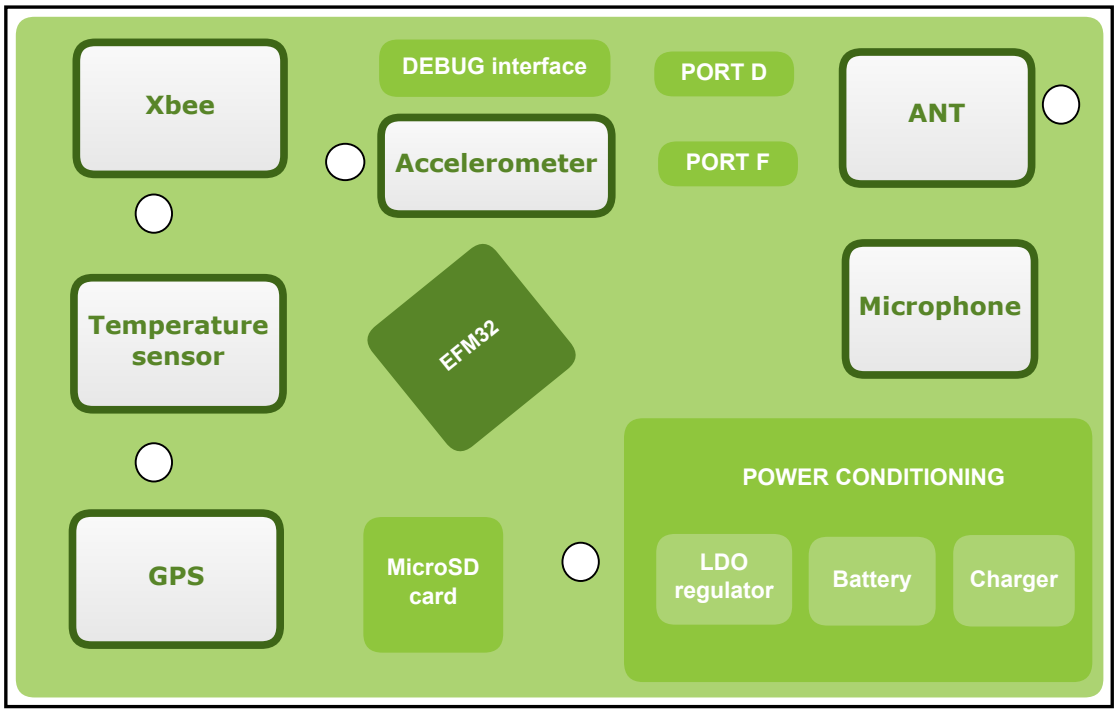


FIGURE 6.3: PCB board layout

of overheating the NTC1 should tell the charger to decrease the current passing through the transistor lowering the power dissipation. Hence, the NTC1 was placed as close as possible to the transistor and was surrounded by the copper heatsink.

The LDO regulator has a SOIC package with an exposed pad underneath it. This is done to dissipate the generated heat directly into the board. The pad is internally connected to GND and in this case it was unnecessary to make dedicated heatsink area for this chip. As with the Multiple vias connect the upper and bottom layers of the PCB to promote better heat transfer.

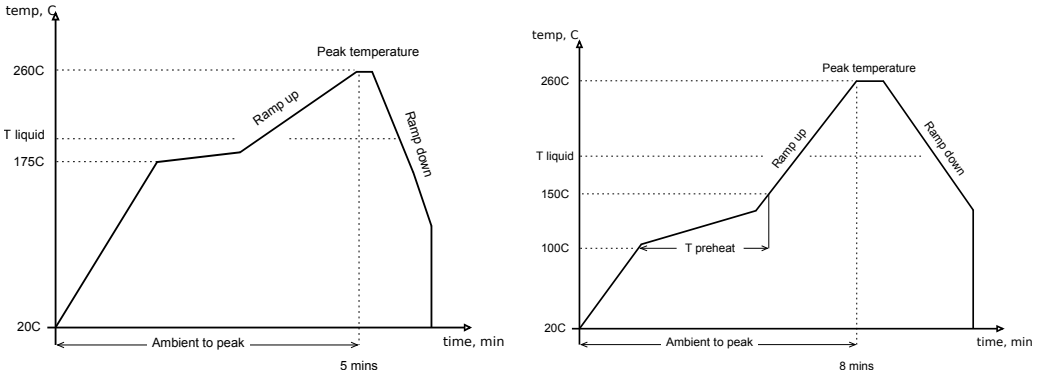
Temperature sensor used in the project is one of the kind `#TODO:`

The most practical way of placing the MCU is to put it in the center of the PCB. This enables the usage of short trace connections to the peripherals located on the sides of the board.

Decoupling capacitors are placed as close to the component power pins as possible. This applies to all ICs and modules used on the board without any exceptions.

6.2.1 Soldering

Soldering of components was done using several tools. This included a hot plate for reflow process and conventional tools for SMD components. Temperature sensor and an accelerometer were attached to the PCB using solder reflow as it was impossible to do



(a) (b)

FIGURE 6.4: Optimal temperature profiles for soldering

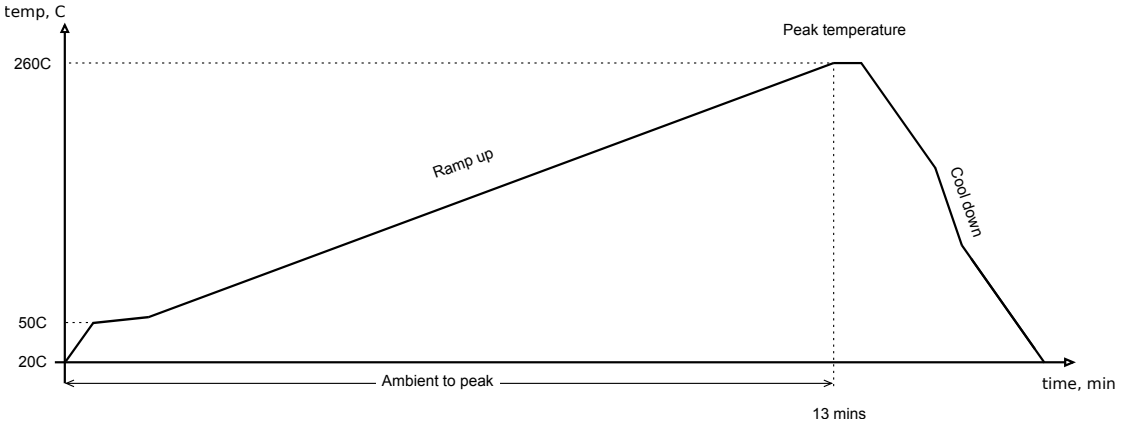


FIGURE 6.5: Used temperature profile for soldering

solder them using conventional tools. An appropriate thermal profile had to be followed to prevent damage to the components. Both ICs have a peak soldering temperature of 260C but a slightly different thermal profiles, figure 6.4(a) and figure 6.4(b) show the thermal profiles for temperature sensor and accelerometer respectively.

Ideally It should take no longer than 7 minutes to achieve the peak point from ambient temperature. This way the BGA package balls of temperature sensor would melt and produce a reliable contact to the PCB traces and accelerometer without any risk of damaging the device. Nevertheless, the suggested profile for both ICs was impossible to achieve with the used hot plate as it only had an option of setting the end temperature point which took considerable time to reach. On average this time was 13 minutes as seen from the plotted thermal profile provided by the hot plate, figure 6.5.

It did increase the risk of damaging both of the components. Nevertheless, after the reflow they functioned as expected.

Chapter 7

Future Work

#TODO: write a nice intro paragraph for future work: huge project - first prototype - realizations during prototyping/integration - unforeseen limitations during planning stage - proposed solutions and fixes - project scope can be extended

In this section further development possibilities for the current prototype will be discussed.

The device is designed as an equine health monitor, however the need for a health monitor is not only limited to horses. As there are almost no horse specific parts used in the system, the device could be adapted for the use on human subjects or agents of other mammal species. The only adaptation that has to be made is the usage of a target specific heart rate monitor.

The scope of the project is rather large. It is possible improve some of the features to evolve it into a more specialized system. One possible solution could be a tracking system using GPS and accelerometer data. These data can also be used for health-related applications such as tracking the movement of the animals and detecting extraordinary situations.

Another field of application could be professional horse training, to monitor the heart rate in relation to the covered distance over an average of the velocity which allows to analyze the improvements of a horses stamina over time.

As the system works as a health profiler, it is possible extend it by adding signal analysis features to perform autonomous diagnosis or issue warnings when there is a problem with the vital signs. Further research has to be undertaken to determine relations of parameters and indicators for certain diseases.

In the current implementation, the gut sound monitor works periodically. The problem with this approach is introduced noise when the horse is moving. One way to improve the behavior is to disabling the audio recording conditionally, if the horse is moving and

delay it until the horse stays at a stable position for a period of time. The (already available) accelerometer and GPS data can be used to implement this feature.

- ◇ TODO: Improve web interface to behave more like a standalone application (ajax)
- ◇ TODO: Improve the web interface to deliver a mobile device version of the website
- ◇ TODO: Replace XBee with a faster 802.11 based wireless connection - energy cost / bit is too high for ZigBee - we want to transmit relatively large files (Audio) - low duty cycle
- ◇ TODO: Use XBee pin sleep with cyclic sleep mode - this way the base station doesnt have to wait for a monitoring node to connect before a request can be dispatched
- ◇ TODO: Get sample recordings from horses that will make it possible to perform an acoustic profile. This can be used to improve the design of the audio acquisition module.
- ◇ TODO: Add other possible improvements

Chapter 8

Project Management

8.1 Development Plan

The team initially consisted of 6 people. However one member, Ahsan Baig, had to drop the project after the sixth week since he had visa issues. The Gantt chart was created considering six project members. As it was unclear from the beginning if Ahsan would manage to arrive in time for the project he was given tasks that were extensions for the overall system and could be integrated in the later stages of the project, such as display and analysis of collected sensor data. Therefore he could not contribute to project much in the first weeks. His tasks were shared by other people or, as in the case of data analysis, discarded.

The meetings with Dr John Chad and Dr Neil Smyth about horse grass sickness were effective on excluding data analysis and diagnosis from the project scope. The reasons for this decision will be discussed in more detail in the `#TODO: RESEARCH CHAPTER`.

The project was assigned to the team in the second week of the semester. Therefore the team had to start working on the project later than the official release date. This resulted in a tighter schedule and delayed the placement of component orders. `#TODO:` However, the team managed to build a working prototype.

8.2 Task Distribution

Task distribution was made considering skills and interests of all group members. A high degree of parallelisation of tasks was needed to comply with the tight schedule. A shared document was used to record the tasks and their status which can be found in APPENDIX-TASKS. All members attended discussions about system design and did research on component selection. All members with their primary responsibilities are listed below.

`#TODO: List each persons major tasks`

Jose Cubero:

- Audio acquisition
- Internal flash storage
- Background research

Merve Oksar:

- Background research
- Data storage
- Debug interface
- Report management

Konke Radlow:

- Wireless communication over ZigBee
- Base Station
- Web Interface

Michail Sidorov:

- Breakout boards
- PCB
- Budget planning and tracking

Yaman Umuroglu:

- Sensor system implementation
- Integration
- Testing
- Project management

8.3 Budget Planning

The team had X budget for this project. The team chose components based on their price, quick availability and technical features. The components used in the project and their costs are listed in the `#TODO: APPENDIX-BUDGET`. Samples were requested for some of the components. Therefore both the actual cost the team spent on the components and their actual prices are listed in the appendix