GDP Project 2012/2013

UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering, Science and Mathematics

Equine Health Monitor

A Group Design Project report submitted as part of the European Masters on Embedded Computing Systems (EMECS)

Authors:
Jose Cubero
Merve Oksar
Konke Radlow
Michail Sidorov
Yaman Umuroglu

Supervisors:
Dr. Peter Reid Wilson
Iain McNelly

December 13, 2012

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

Faculty of Engineering, Science and Mathematics

A Group Design Project report submitted as part of
the European Masters on Embedded Computing Systems (EMECS)

Equine Health Monitor

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. Constant manual monitoring of horses is often not feasible for horse owners who live far from their horses, or 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 thus proposed to provide convenience to horse owners for monitoring vital signs of their animals. An alternative usage is data collection for scientific purposes such as bioprofiling.

The system whose design and implementation is covered in this report 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 on 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

Acknowledgments

We would like to thank Dr. Peter Reid Wilson for letting us undertake this challenging project, for supporting it during the development phase and organizing a field trip that allowed us to test the system under real conditions.

We also owe gratitude to Dr. John Chad and Dr. Neil Smyth for sharing their knowledge about horses with us and helping us to define a realistic scope for the project.

Energy Micro and Nordic Semiconductor supported of our project generously hardware parts, for which we are very thankful because it sped up the development process a lot.

Chapter 1

Introduction

1.1 Project Brief

The goal of this project is to design a scalable system for remote monitoring of equine vital signs. The system consists of battery powered monitoring devices that can be attached to a horse in order to monitor different vital signs, and an accompanying base station to allow simultaneous monitoring of multiple horses.

The proposed system acts as a data collection device that can be used to build a database with horse's vital signs and gut sounds that can serve as a basis for causal research in the field of grass sickness disease, which has more than 95% mortality rate without an early diagnosis [Robinson and Sprayberry(2009)]. The data can also be used to diagnose other horse related problems such as lameness.

In the future the system can be extended to detect health problems autonomously without physical examination.

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. It is inconvenient to change or recharge batteries on the monitoring device often, especially if a large number of animals will be monitored. Therefore low power consumption is one of the main concerns for the design of the monitoring device.

The monitoring device is intended to 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 splashproof 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.

Finally, 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

Since an important goal for the project was to determine the feasibility of automatically diagnosing grass sickness, literature research and a discussion session with experts were conducted to acquire more theoretical and practical information about grass sickness disease. This information was very important to decide which sensor data could be useful and how it could be used to provide indicators for a diagnosis of grass sickness disease.

2.2 Literature Search

Information about normal vital signs of an adult horse and the symptoms of grass sickness were collected through literature research. This gave us an idea about how the collected data could be used either for automation or by the medical experts to make a diagnosis. Here we attempt to summarize the knowledge we obtained on the definition, symptoms and potential results of the disease, as well as the relevant vital signs from healthy horses for comparison.

Grass sickness (equine dysautonomia) is defined as a disease of equids characterised by damage to autonomic, enteric and somatic neurons which cause low gastrointestinal motility and paralysis of the gut as a result of this. 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. 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. [Robinson and Sprayberry(2009)], [Edwards et al.(2010)Edwards, Martz, Rogge, and Heinrich].

Symptoms of grass sickness include colic, gastrointestinal stasis and increased heart rate. Therefore heart rate data could be useful to detect grass sickness. Heart rate of a horse with acute grass sickness is typically 70-120 bpm (beats per minute), whereas horses with subacute grass sickness have a heart rate of 60-80 bpm and the ones with chronic grass

sickness have that of 50-60 bpm. Heart rate of an adult horse changes between 30-40 bpm under normal conditions. [Corley and Stephen(2009)], [Robinson and Sprayberry(2009)]

Since low gastrointestinal motility is also an indicator of grass sickness, borborygmi (gut sounds) can be used for diagnosis. Borborygmi is the noise caused by gas and fluid pushed in the gastrointestinal system. It shows the status of the gastrointestinal system. 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 itself enough evidence to detect them. [Corley and Stephen(2009)]

2.3 Discussions with Bioscientists

The team had meetings with Dr. John Chad and Dr. Neil Smyth 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.

Dr. Smyth explained that for performing a successful abdominal examination, four different sites should be auscultated: left and right lower and upper parts of the abdomen. However, the loudest can be heard at the lower right part of the abdomen where the large intestine is.

The proposed system contains a single mobile monitoring device attached to each horse. 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 a single audio recording unit can be placed at the lower right part of the abdomen.

Dr. Smyth stated that the situations which may cause too loud or too quiet gut sound can vary. There is not a enough data available on the web regarding the properties or signal characteristics of gut sounds and the relation of those with common horse illnesses and these tasks are normally performed by specialists based on empirical knowledge. The complexity of diagnosing grass sickness makes an automated solution unfeasible in the scope of this project.

Therefore, the main goal of the project was shifted from diagnosis to health data collection and profiling of vital signals. The scalability and wireless communication features of the Equine Health Monitor make it suitable for building biological profiles by collecting the vital signals of groups of horses. This constitutes an innovative and practical tool for veterinary and biological research as well as clinical practice.

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 common context, as they are tightly coupled with each other. A design decision that is made in one area on behalf of one of the challenges will affect both other fields.

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 doesn't contain an operating point for this configuration.

An example for this is our goal to be very energy efficient, because battery life is highly important for mobile devices. Aiming for 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.

In order to come up with a design solutions for the main challenges a design space analysis was conducted, where the team attempted to find an operating point close to the center of the design space triangle. The tradeoffs that were taken into consideration will be explained in more detail in the following paragraphs.

Energy efficiency and the inconveniences caused by the lack of it were mentioned before. One major design goal is a long battery runtime which can only be achieved if every component of the monitoring device is optimized for low power usage. This limits the available choices for the wireless communication protocol and also puts a constraint on the amount of data processing that can be done on the monitoring device.

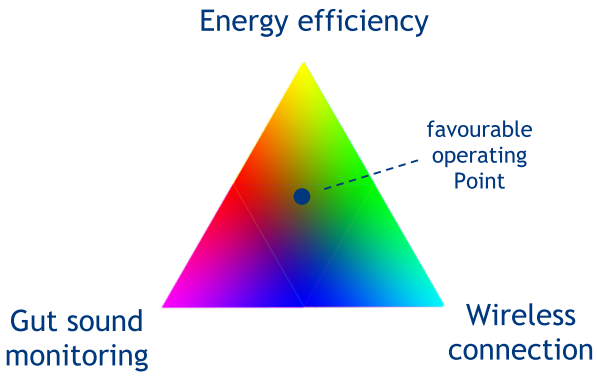

Figure 3.1: System design challenges

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.

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 low energy consumption and stationary base station that collects data from monitoring devices and presents it to the user over a web interface.

This design stands in contrast to the first proposal for the system design in which there is no distinction between monitoring station and base station and the user interacts directly 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 it's 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.

Separating the roles in the system into distinct devices gives the possibility to design nodes that are optimized for their role in the overall system, which is especially important in this case, as 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 system software.

3.2.1 Distributed system

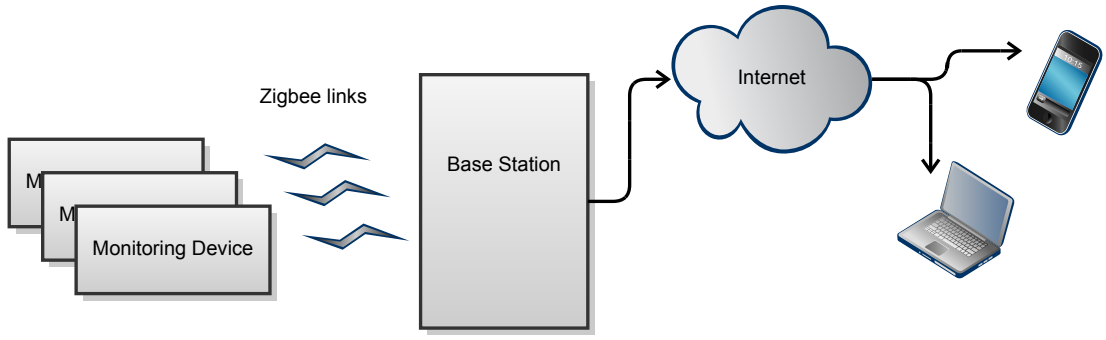

Figure 3.2: Block Diagram: Distributed system

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 provides the network for wireless communication between monitoring devices and the base station. Measurement data from monitoring devices is received over this network, and 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.

The monitoring device is a portable battery powered device that can be attached to horses with a strap, and features a range of sensors, a digital stethoscope and means for temporal data storage. The collected data is transferred periodically to the base station, or preserved for later transmission in the local data storage, if the base station is out of reach.

Data access is provided to the user by the base station over a web-interface that displays the collected data and allows the user to download a database for detailed analysis. The audio recordings are stored on an flash memory card in the monitoring devices, and only transferred to the base station of the user requests a data transfer.

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 has 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 required to adhere the project goals.

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

- Range: up to 100m
- Power consumption: very low, as it is 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 put 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:

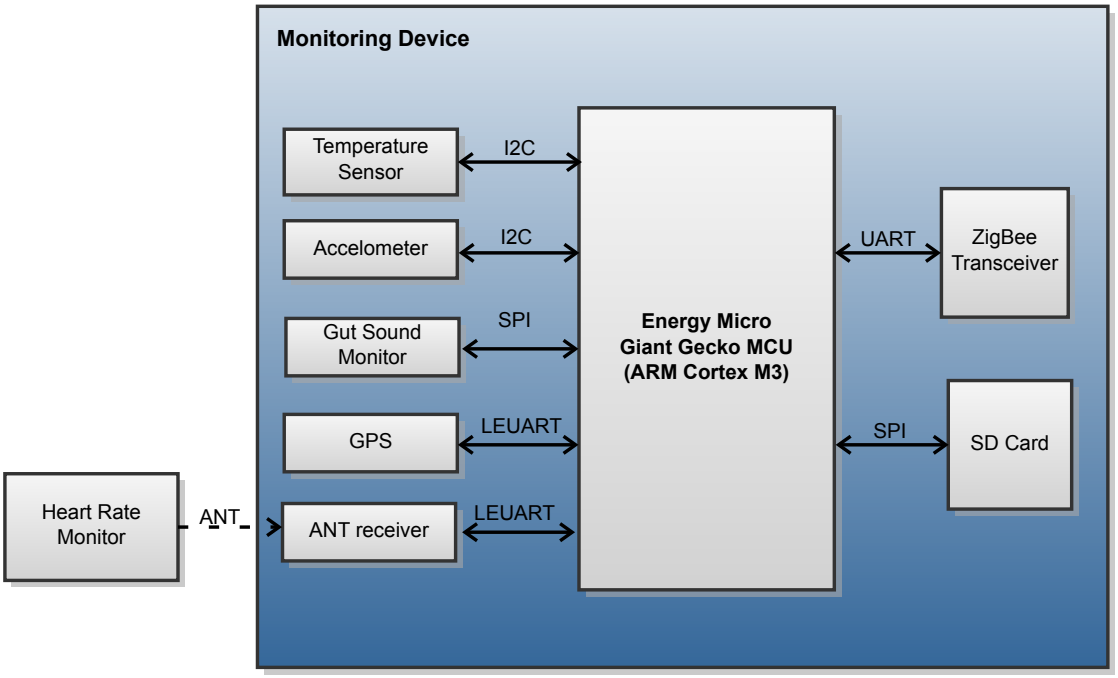

Figure 4.1: Block Diagram: Monitoring Device

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.

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.

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

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

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 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 (ANTAP281M4IB from Nordic Semiconductor / Dynastream) 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

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

wirelessly solves the problems mentioned above and brings the additional advantage of making the system usable for humans.

Device Properties	Corresponding Requirements
low power ◇ peak rx current 17mA ◇ peak tx current 15mA ◇ idle (no RF activity) current $2\mu A$ ◇ search current 2.8mA automatically enter idle mode if no activity detected	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: ANT HRM interface properties

Interface: The ANT module handles most RF processing internally and offers a simple UART interface to the microcontroller with configurable baud rate. Since there message volume passed between the ANT and the EFM32 will be relatively small (less than a hundred bytes per message, and a message rate of 1-4 Hz while active) and because the messages can arrive asynchronously while the monitoring device is in sleep mode, a LEUART port with a baud rate of 4800 was used for the ANT module connection.

Driver: The ANT driver, whose implementation is based on the ANT+ HRM receiver example from Dynastream, can be described as consisting of three layers:

Message reception and format validation layer: The LEUART port, which is able to receive data even while the EFM32 is sleeping in EM2, sends the characters to the ANT driver which validates the correctness of the received ANT messages according to the ANT message structure[1]. Validated messages are passed to the ANT protocol layer for handling. Similarly, data received from the ANT protocol layer is packaged according to the ANT message format and put through the LEUART port.

ANT Protocol layer: This layer handles details of the ANT protocol such as configuring the RF channels and network key, and is largely based on the Dynastream standard implementation. One important function is handling disconnections and device pairing. The current behaviour is to pair with the particular HRM chest strap as soon as a connection is made, and attempt to reconnect to the same device if disconnected. ANT+ specific messages are passed to the ANT+ layer and actual message sending/receiving happens through the interfaces to the message layer.

ANT+ HRM protocol handling: This layer handles the ANT+ data pages received from the heart rate monitor chest strap, which contain a variety of information about the monitored heart rate. Our current implementation only reads the computed heart rate in beats per minute and makes this data available through the standard Sensor interface. Please consult the ANT+ Heart Rate Monitor device profile document[2] for more information.

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

TMP006[3] 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. The temperature measurement is provided by

[1] http://www.thisisant.com/resources/ant-message-protocol-and-usage/
[2] http://www.thisisant.com/resources/heart-rate-monitor/
[3] http://www.ti.com/lit/ds/symlink/tmp006.pdf

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

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 [4] 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).

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

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**??** 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 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[5].

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 sensor's own buffer.

We chose the 3-axis ADXL350 [6] digital accelerometer from Analog Devices for the project. It is capable of measuring acceleration in the ranges ±1g, ±2g, ±4g or ±8g

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

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

and it has a FIFO buffer which allowed us to implement the scheme discussed in the previous paragraph.

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

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 is used in Stream mode, meaning that samples will be continuously accumulated in the FIFO buffer at the desired sampling rate, and old samples will be discarded. Sample rate configuration, sleep and wakeup are all supported through writing to the appropriate registers, which have their own wrapper functions in the driver.

4.1.6 Gut Sound Monitoring

One key feature of the desired Equine Health Monitoring System is the possibility to acquire and record sounds coming from horses guts which can come very useful for diagnosis of multiple health disorders in the digestive tract of the animal under observation.

As it was mentioned in chapter 2 listening the gut is a common task performed by a specialist during a regular physical examination. This is done by placing a stethoscope in several parts of the animal's trunk. Intermittent noises that will repeat every 15-30 seconds will be detected in a healthy animal. The specialist will monitor this sounds for an interval of 4-5 minutes. The aim of the designed system is to record and store those sounds which will eliminate the need for a physical examination in situ by a specialist.

Audio acquisition requirements

As part of the background research phase it was found that digital recordings of gut noises are regrettably not easily available on the web and despite of different attempts it was not possible to obtain an audio sample of good quality that could be used as a reference. Hence a proper signal profiling and characterization could not be performed.

Because of that the audio acquisition system was designed so that it preserves the mayor amount of information present in the original signal and the parameters were selected to fulfill the hearing capabilities of the final human user (a veterinary specialist).

The recording of audio samples implies two major challenges that are not shared by the other sensors in the system.

Real time constraints: In order to acquire an audio stream without distortion accurate periodic sampling must be ensured. That means that during a recording task delays or failures in fetching the samples are inadmissible.

High data rate and memory usage: While in the case of other sensors like the temperature sensor and GPS for example fetching a few bytes every 2-3 seconds is enough to provide reliable real-time information, storing an audio stream requires receiving thousands of samples per second. The system capabilities like memory size and bus bandwidth will impose physical barriers to the parameters use for audio sampling.

Sampling parameters

Taking in consideration the requirements as well as the capabilities of the selected system microcontroller platform the audio sampling parameters were set as shown in table 4.6. Other audio related projects in this platform were also used as a reference for setting this parameters.

Parameter	Value	Justification
Sampling frequency	8000Hz	The microcontroller is not able to handle the recording of higher frequencies. Most of the information present in gut sounds is presumed to be in the low frequency range.
Sample word size	16bits	High dynamic range. Available in many AD-converters.
Recording time	240 sec	To mimic the examination done by a specialist

Table 4.6: Audio sampling paramters

According to that the memory space in bytes required for one single recording can be calculated as follows:

$$
\begin{aligned}
Size_{audio} &= f_{sampling} * \frac{Bits}{Sample} * t_{recording} \\
Size_{audio} &= 8000Hz * 16\frac{Bits}{Sample} * 240s \\
&= 128kbit/s * 240s \\
&= 3.75MByte
\end{aligned}
$$

As it can be noted this is a very high amount of data and thus the use of a external memory device for storage is required as it is explained in section 4.1.7.

Component selection justification

The component ADMP441 from Analog Devices[7] was chosen for the audio acquisition peripheral. This single-chip MEMS (Micro Electro Mechanical System) microphone provides very convenient characteristics which are summarized in the following table.

Device Properties	**Corresponding Requirements**
Single chip solution: MEMS sensor, signal conditioning, analog-to-digital converter, antialiasing filters, power management	Ease of integration. No need for extra components (codecs/amplifiers)
I2S digital interface	Ease of integration, industry standard
Low voltage supply and current consumption: ◇ 3.3 Vdd ◇ Normal mode: 2.5 mA ◇ Standby mode: 0.8 mA ◇ Power down mode: 4.5 uA	Low energy consumption
16-24 bit samples High SNR: 61 dBA High sensitivity: −26 dBFS Flat frequency response from 60 -15000 Hz	Audio quality suitable for the project scope, high intelligibility
Small size: 4.72mm × 3.76mm × 1mm smd package	Low weight, portability

Table 4.7: MEMS microphone properties

Interface: The ADMP44 chip uses I2S protocol as interface. I2S, also known as Inter-IC Sound, Integrated Interchip Sound, or IIS, is an serial bus interface standard used for connecting digital audio devices together. It is used to transmit PCM (Pulse Code Modulation) audio data. PCM simply consists on the serial transmission of the binary encoded samples. The Universal Synchronous Asynchronous Receiver Transmitter (USART) hardware modules present in the EFM32 microcontroller include support for the I2S protocol.

Driver: One of the available USART ports was configured to work as I2S master providing the clock signals needed for the synchronous data transmission. The sampling

[7] http://www.analog.com/static/imported-files/data_sheets/ADMP441.pdf

frequency is set as a function of the baudrate set for the port interface. After having configured properly the reception of samples from the peripheral is triggered by using the AUTO_TX feature of the USART port. In this mode the microcontroller will always generate the clock signals for the slave device as long as the port is enabled.

To support the continuous data flow of audio samples coming from the microphone double-buffered DMA transactions are used. This method is explained in detail the section 5.1.5.

4.1.7 Data Storage

A storage element is included in the monitoring device. It provides a significant benefit because it 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, relatively low throughput of ZigBee link between the base station and the monitoring devices makes gut sound streaming difficult considering the amount of audio data produced. Having a storage device makes it possible to complement the limits of wireless communication bandwidth by streaming a smaller amount of data and record a larger amount of audio data in it.

We had three possibilities to store data: SD card, internal flash memory and external flash memory chip. The internal flash memory of the MCU has 1024 kilobytes capacity half of which can be used to save data[8]. Also, the maximum number of erase cycles of the internal flash is limited to 20000. Therefore after writing and erasing roughly 10 gigabytes of data, the internal flash memory would not be usable and reduces lifetime of the monitoring device. The external flash memory and the SD card have higher capacities than internal flash memory, therefore they are more preferable.

We decided to use a microSD card in this project. The SD Card can be removed from the monitoring device by the user to copy the data when needed, since a wide range of devices are SD card compatible. This advantage overcomes the external flash chip soldered on PCB.

Interface: SD Cards are based on flash memory technology. They support 3 communication modes: SD 1-bit, SD 4-bit and SPI mode[9]. The former two are similar. The main difference is that in SD 1-bit mode, the data is transferred over 1-bit wire in a synchronous serial fashion while in SD 4-bit mode, 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 [10]

[8] http://cdn.energymicro.com/dl/devices/pdf/d0126_efm32gg332_datasheet.pdf
[9] http://bit.ly/VCKHYR
[10] http://alumni.cs.ucr.edu/~amitra/sdcard/Additional/sdcard_appnote_foust.pdf

Also, accessing the complete SD Card Specifications to build an SD standard device requires licenses from SD Card Association [11]. Therefore, in this project, the card is used in SPI mode.

Driver: 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 depending on the SD Card class they belong to.

On the other hand, 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. Theoretically, SPI clock can be up to 24 MHz when the source clock is set to 48 MHz. However, this is not true in practice because there is an upper speed limit caused by delays of inputs and outputs of the microcontroller which is not specified by the manufacturers[12].

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

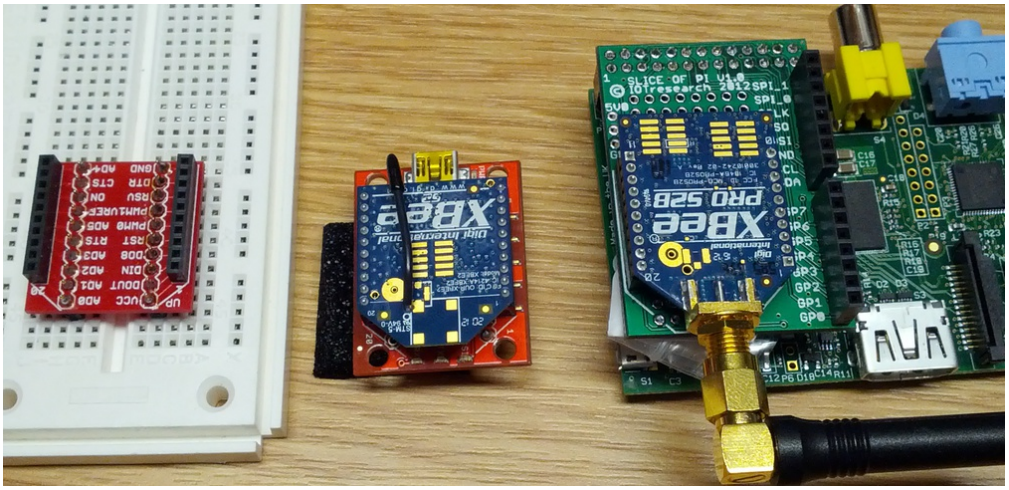

Figure 4.2: XBee prototyping interfaces

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 4.8.

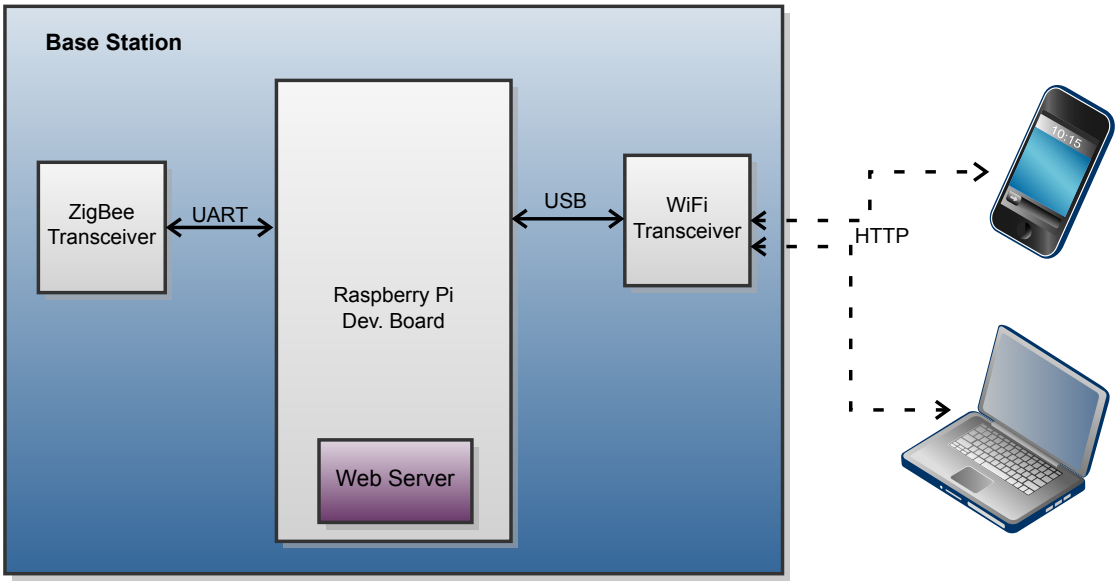

Figure 4.3: Block Diagram: Base Station

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 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.

[11] https://www.sdcard.org/developers/howto/
[12] http://forum.energymicro.com/topic/288-microsd-card-spi-baudrate/

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 4.8: Raspberry Pi capabilities

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.

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

The monitoring device software was developed using a mix of C++ and C, with IAR Embedded Workshop for ARM as the development environment. Using this mix of object-oriented and procedural programming paradigms, we aimed to achieve the best of both worlds - C++ classes were created to take advantage of polymorphism and encapsulation where possible, while pure C was used in places where small and efficient functions (e.g for handling DMA callbacks or audio-related tasks) were needed.

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. The 128 KB of RAM is enough to compensate for overheads introduced by C++, and sensor drivers involve handling tightly coupled data and functionality, which is suitable for modelling them as objects. Additionally, deriving

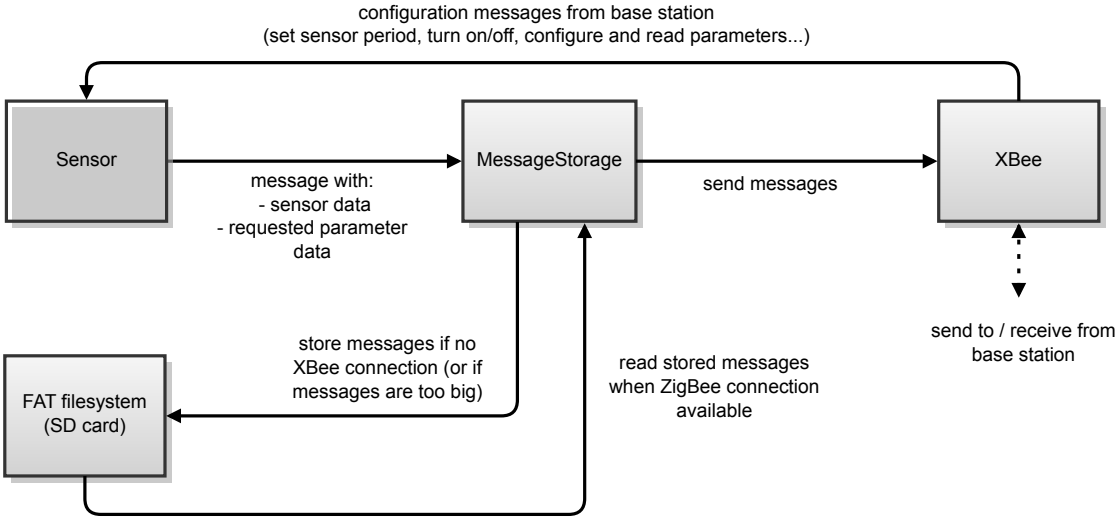

Figure 5.1: Monitoring Device: dataflow

all sensor implementations from the same Sensor base class allowed the integration of sensors into the final system in a plug-and-play manner, since they all share the same interface.

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. 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 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, with a more detailed explanation of the power management phases in table 5.1. 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 A.6.1.

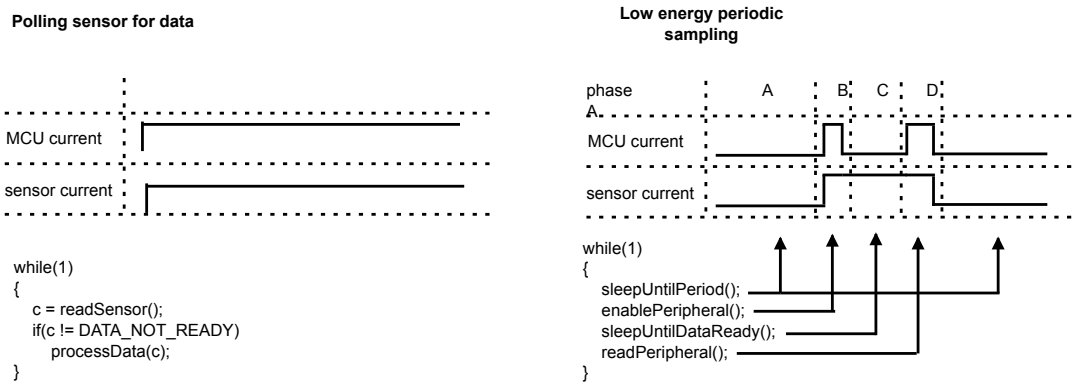

Figure 5.2: Sampling comparison

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

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 a low frequency oscillator 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.

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.

5.1.4 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 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 a DCP shifts the execution of time consuming data acquisition functions to the main context instead of doing them inside the interrupt context..

In this scheme, the interrupt handlers do not do any processing beyond taking actions so that received data is not lost but buffered in memory. The ISR then sets a flag indicating that the corresponding handler task should be executed. When the control returns to the execution loop, it checks each flag in order of decreasing task priority, executes the task if its flag is set and clears the flag afterwards. The processor then goes to sleep until the next interrupt.

Since the order of checking the task flags determine the execution priority and because the scheme is not preemptive, the tasks must be designed to be short (i.e not block execution for long periods). Not observing this principle can cause a big backlog of tasks to be executed and leave the processor no time to sleep.

A final consideration is the handling of asynchronous interrupts during sleep. This scheme is dependent on interrupts from hardware waking up the MCU from sleep, so care must be taken to ensure that the sleep level chosen will not disable the hardware that will generate the interrupts.

5.1.5 Message Storage System

In order not to lose any acquired samples when the wireless connection to the base station is not available, it is necessary to save the samples on persistent storage. The

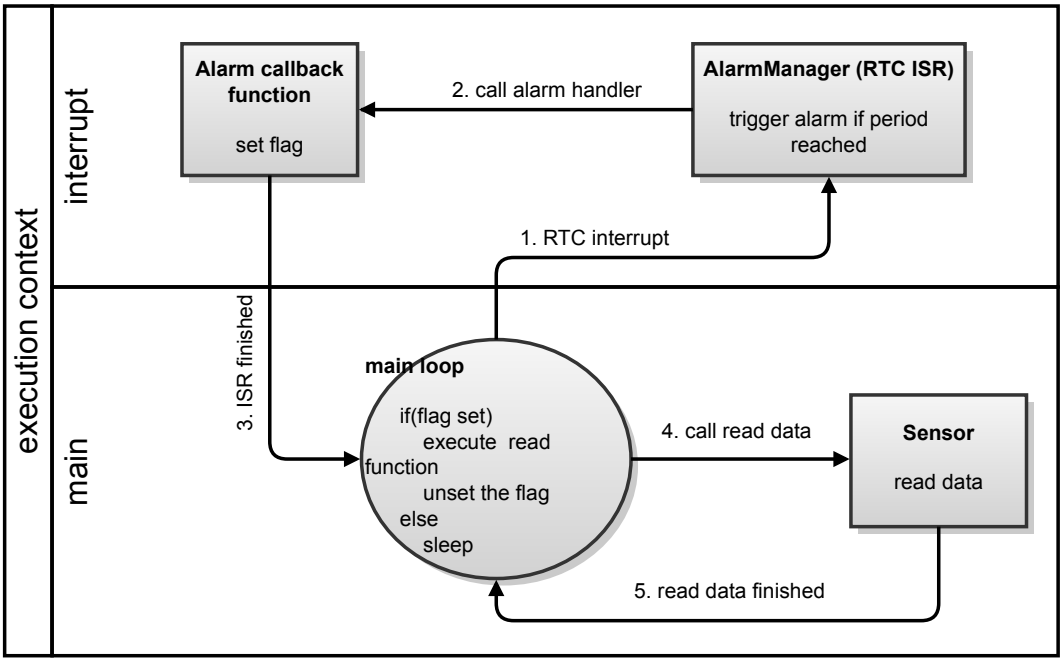

Figure 5.3: Deferred reading

project utilizes an SD card for this purpose, as described in section 4.1.7. This chapter focuses on the software component which abstracts the data storage for the convenience of sensor systems.

It is possible to model the message storage requirements of the monitoring device on two levels: in main memory and in SD card. Sensor readings are placed in the main memory when retrieved and stay there until they either get sent to the base station or become saved to the SD card. When the base station connection becomes available, the messages stored in the SD card are read out and sent first.Therefore, if we view the sampled data as a queue of messages, the way the rest of the system interacts with this queue is actually irrelevant whether the data is stored in memory or on disk.

Our *MessageStorage* class takes advantage of this and offers a simple enqueue-dequeue interface to the rest of the system. Aside from having the ability to flush all messages to SD when desired, the rest of the system does not actually know if elements in the message queue are stored in-memory or on-disk. MessageStorage maintains internal structures to keep track of each message in the queue, and where it is stored (memory or disk). While our current implementation relies on the flushAllToDisk function being externally called to save data to the SD card, it would be simple to implement a simple mechanism that checks of much memory is in use by the in-memory storage and flush this to disk to free up system memory.

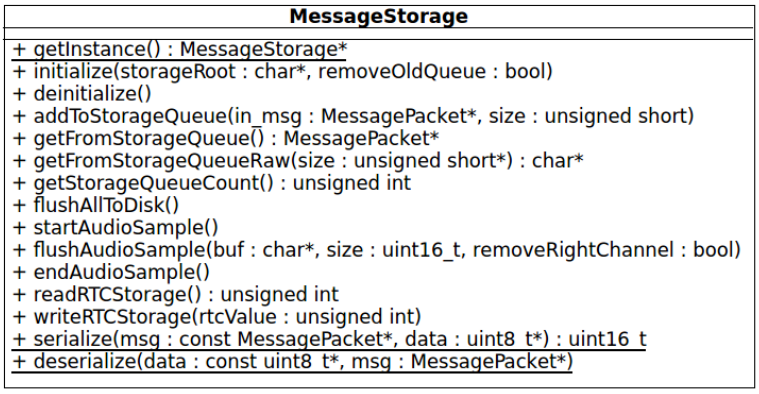

Figure 5.4: Class Diagram: MessageStorage

Messages that will be stored on disk or in memory are serialized in the same manner as they are serialized when they are prepared for transmission over the wireless connection. Details of the message structure and the serialization process can be found in section 5.2.3.

Figure 5.5 illustrates the lifetime of a message inside the MessageStorage class. The data is passed to MessageStore in the form of a MessagePacket pointer, which is first serialized into an internal dynamically allocated buffer and enqueued at the end of the message queue. If requested, this queue entry is flushed to disk and the allocated memory is freed. When the time comes to dequeue this entry for sending, the entry is first read back into memory if it was flushed to disk. The data in memory is then made available for sending via the ZigBee interface or otherwise processing the message data. Once the processing is complete, all resources held by the entry (files on disk or memory) can be freed or deleted.

Audio samples are treated differently since they are too large to keep in memory - they are simply written to SD card as audio samples arrive. These entries are not tracked in a queue since audio samples are not currently being dequeued and sent to the base station. A sequence counter is kept and a new file is created for each new audio recording to avoid overwrites.

If the system is rebooted, the class traverses the SD card storage directories to fill the queue with the unsent messages, and these become the first to be dequeued and sent to the base station.

Automated Data Acquisition: DMA and Double Buffering

Among the different on-chip peripherals included in the EFM32GG microcontroller is a highly configurable channel DMA controller. This device is able to take control of the

system bus and perform data block transfers between different memory locations and between RAM and peripherals without the intervention of the CPU.

The software must initialize a DMA channel with the DMA transfer attributes, the source and destination memory address, the number of bytes to be moved. The trigger to start the transfer can be provided by the software but in many cases it will be a peripheral like the USART which will generate a request.

Advantages: Direct Memory Access allows for a better utilization of the system data bus and lower energy consumption. After being configured the DMA works without intervention from the CPU. This has the benefit of reducing the energy consumption and the workload of the CPU, and enables the system to stay in low energy modes when moving for instance data from the USART to the RAM, thus freeing CPU resources and allowing longer periods of sleep.

DMA transmissions also provide stable high bandwidth memory transfers for operations like audio acquisition, which was particularly suited for our project.

Double-buffered DMA (Ping pong) In some cases it is necessary to provide continuous data flow from or to a peripheral. This is the case of audio acquisition in which samples arrive periodically during all the time while a recording is executed. The system must fetch all the samples before the sample buffer is overwritten by the peripheral.

A single normal DMA transfer is not sufficient to cope with those requirements since the number of transactions is limited to a maximum of 1024 and there is not enough time to reconfigure the channel before the next sample has to be read out. A double-buffered strategy is used instead. A representation of the double-buffered DMA scheme, also called ”ping-pong transfers”, is given in figure 5.6. Two separate buffers A and B are reserved in the RAM memory and one DMA channel is programmed in ping pong mode to use those channels alternatively. The primary descriptor of the channel will have the necessary configuration to perform transfers from the RX_section_DATA register to the buffer A while the alternate descriptor will configure the it to use the buffer B as destination.

For the case of our system, the EFM32 USART peripheral receives one audio sample coming from the microphone through I2S protocol, saves it in its RX_DATA register and then signals that to the DMA engine. The DMA engine will in turn move that sample to currently configured RAM buffer (initially buffer A). That transaction repeats for a number of cycles that corresponds to the size of the buffer. When the buffer is full the engine will switch automatically to the alternate descriptor so that it is immediately ready for the fetch the new sample. At the moment it will also trigger an interrupt request to inform the system that a new buffer full of samples is ready to be read. The message storage implementation will then move the new buffer to the final location (in the microSD card) and concatenate all buffers into a single audio file.

In order for this system to work correctly it must guaranteed that the time that the processor takes to read and process the data from the buffer is less than the time in which the DMA will fill a buffer. Using a quality microSD card, the write throughput to storage is about ten times what is required for the audio samples, so

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

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

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

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 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 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 5.10 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 does not care about the other side.

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.

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 there's 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 a lightweight Python web framework (flask[2]) to generate the web interface dynamically.

The use of a dynamic programming language to generate the content that the user can see dynamically allows us to build a dynamic webinterface with a backend that's communicating with the controller software.

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

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. This has the advantage that the menus of the website automatically get extended with sub-menus if a new monitoring device connects to the base station.

The sensor data is currently displayed in dynamically generated static html tables. This allows a good quick overview over the collected data, but is suboptimal to analyze the data in more detail. In future versions this could be improved by using Ajax to display the tables, to offer better access to the collected data.

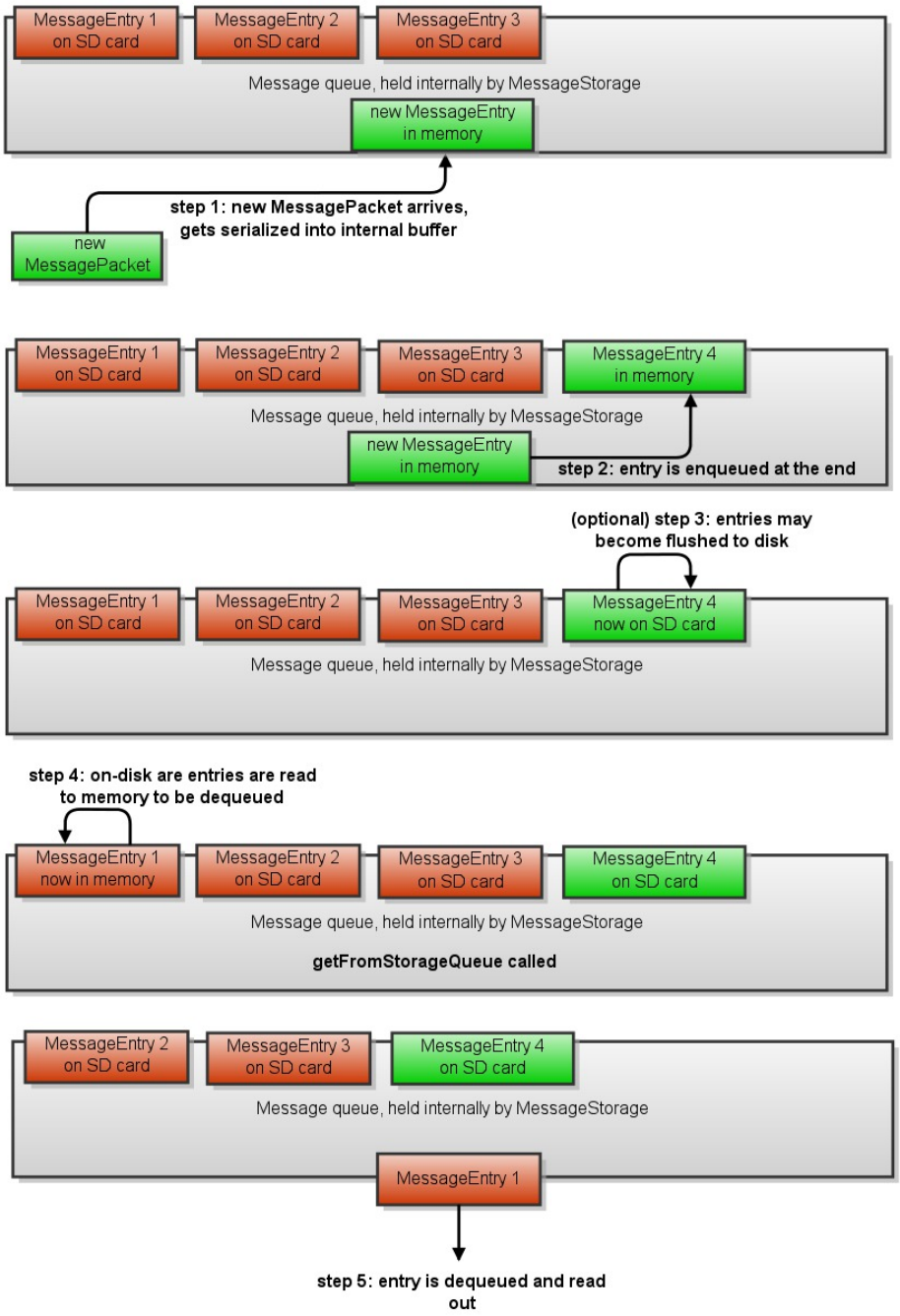

Figure 5.5: messages inside MessageStorage

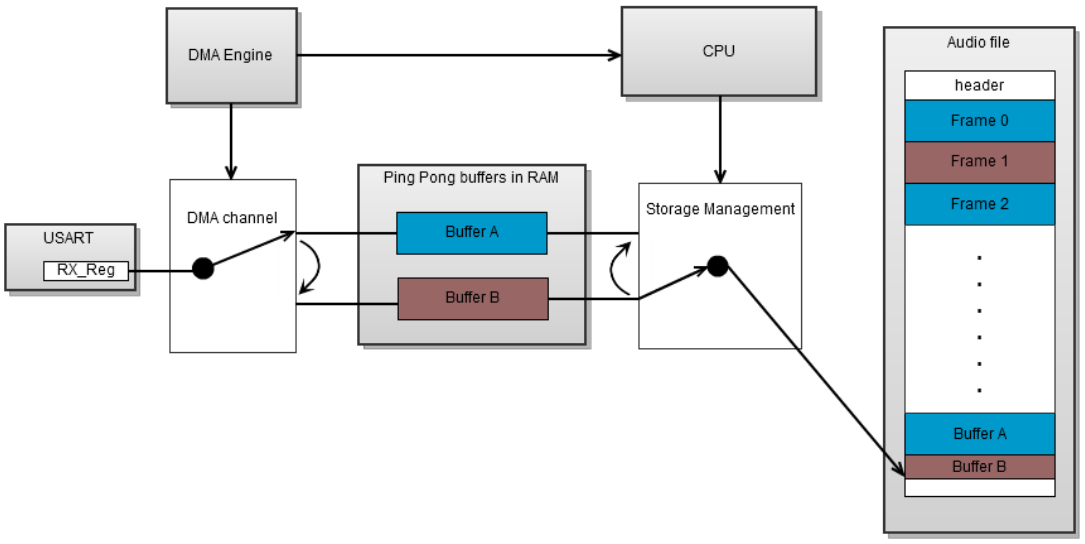

Figure 5.6: Ping Pong diagram

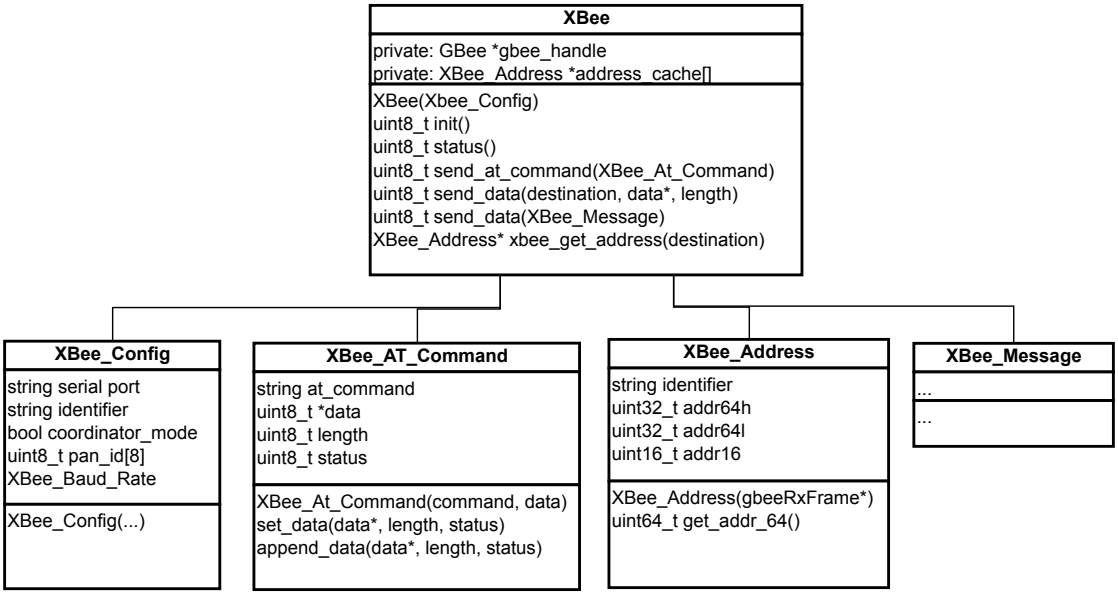

Figure 5.7: Class diagram: XBee Interface

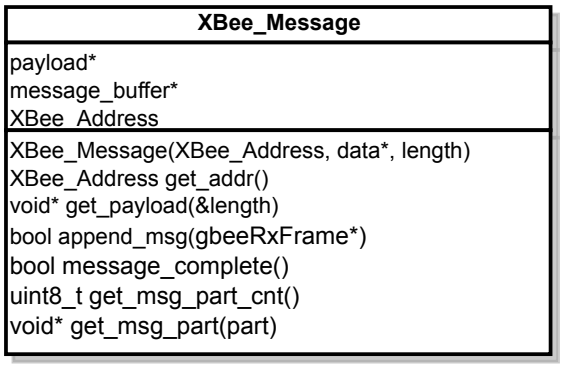

Figure 5.8: Class diagram: XBee Message

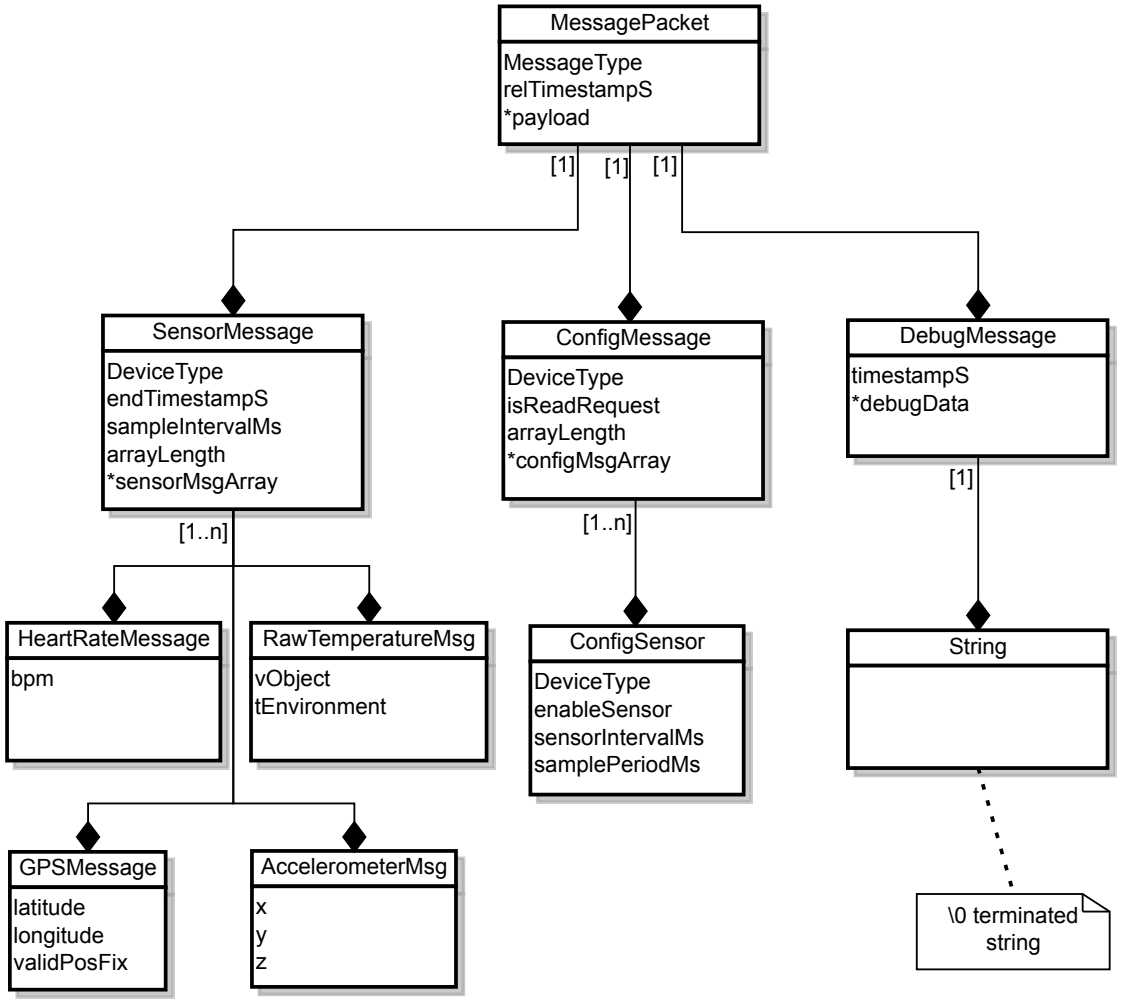

Figure 5.9: Class diagram: Message Types

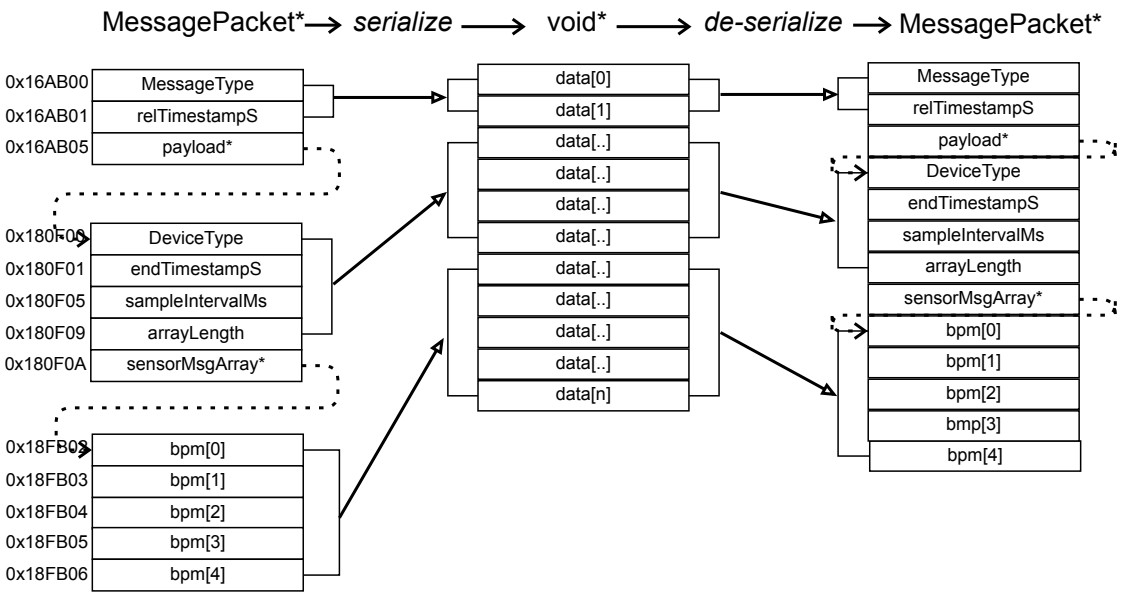

Figure 5.10: Message serialization

Please stand by.
Coming soon!

Figure 5.11: Web Interface

Chapter 6

Hardware design

In order to make a complete system and test it on adequate subjects without any harm to the circuitry it was decided to make a custom PCB and encapsulate it in a splash proof casing. PCB not only eliminates the accidental connection losses between the components as it usually happens on breadboards and protoboards but also keeps the traces short and ensures the components function according to their specification.

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 datasheets (TMP006[1], GPS[2], AD441 Microphone[3], ADXL350 Accelerometer[4], ANT[5]).

6.1 Schematic

This section will describe the connections between different components of the system, their intended use and operational principles. Full schematic of the system can be seen in section A.2 of the Appendix. It is separated into different sheets for ease of reading.

6.1.1 Power conditioning

This part is responsible for providing energy to the rest of the board. The circuitry consists of two stages: the first stage incorporates a constant current constant voltage single cell Lithium Ion battery charger centered around a ADP2291 chip from Analog

[1] http://www.ti.com/lit/ds/symlink/tmp006.pdf
[2] http://www.fastraxgps.com/showfile.cfm?guid=17fc51dd-0618-49aa-b62e-020234006b29
[3] http://www.analog.com/static/imported-files/data_sheets/ADMP441.pdf
[4] http://www.analog.com/static/imported-files/data_sheets/ADXL350.pdf
[5] www.thisisant.com

Devices. The second stage is a low dropout voltage (LDO) regulator based on a ADP1706 chip from the same manufacturer.

Battery charger

The input voltage of the charger is in the range of 4.5 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 $I_{max}/40mA$, 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 6.1:

- **Power-down:** If the input voltage is lower than 3.8V the charger is in the shut-down mode. Power consumption from the battery is $1\mu AA$.
- **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 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

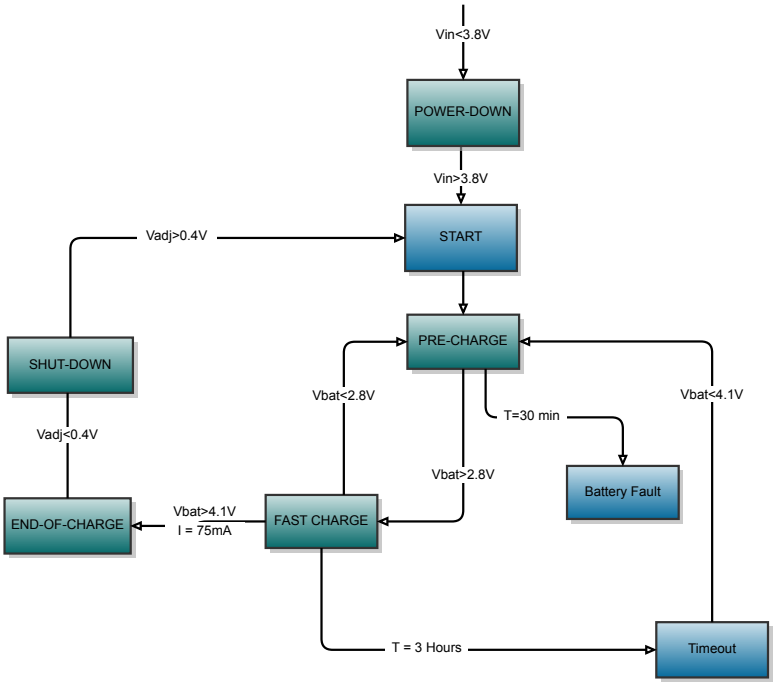

Figure 6.1: Battery charging sequence

- ◇ **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.
- ◇ **Shut-down:** This mode will be initialized if the voltage on pin Vadjust drops down below 0.4V.

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.

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 150mA.

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 $T_{ramp-up} = 0.8V \times (C3/1.2uA)$ 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

6.1.2 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 see figure A.5.

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, see figure A.8. 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.

6.1.3 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.

6.1.4 RF devices

Despite the fact that Xbee has a lot of pins, the only connections that are required for it to be operational are VCC, GND, DOUT and DIN. Others are not used in this case. Xbee is connected to USART 0 of the MCU as it requires high baud rate communication.

The GPS module on the other hand is connected to low energy UART (LEU0). By default its communication rate is set to 9600 baud. The GPS module requires two power supplies. VCC is the first one and is used for digital parts together with I/O. VBAT is the second one and 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.

ANT module is connected to low energy UART 1 (LEU1). Baud rate selection is made by setting the dedicated BR1-BR3 pins either low or high. In the system the pins are pulled down which gives us the speed of 4800 baud.

When not in use Xbee, GPS and ANT modules can be completely shut off to save more energy. This is done by means of P-channel mosfet. N-channel mosfet is not acceptable in this case, because the devices are ground referenced. By default these modules are ON and setting the correct mosfet high will shut-off the power to the device. Shutting down both power supplies for the GPS will trigger the loss of all stored satellite data in the non-volatile block. Therefore, once the GPS becomes operational it will have to go through the cold start again and use a lot of energy.

6.1.5 Microphone

MEMS microphone is connected to the USART1 of the MCU. There is a possibility of using two microphones in a stereo configuration. Nevertheless, in this design only one channel was used. The L/R pin that is responsible for channel selection was grounded indicating that Left channel is used only. SD pin requires an external pull-up resistor. To our advantage MCU has an integrated one, therefore this one was used instead of an external one.

6.2 Layout

The PCB layout was made not only for the final board, but for several breakout boards as well. The accelerometer and MEMS microphone both have a specific pad. Therefore, the breakout boards were custom made for them. This enabled us to speedup the prototyping process and use these devices prior to the assembled PCB. The accelerometer breakout board is not used for the final product, but the microphone board is as it is mounted to the head of a stethoscope. PCB layout for the microphone can be seen in Appendix

A.3. The layout of the final board is shown in Appendix A.2 and Appendix A.1. It is a 2-layer board, therefore a layout for both top and bottom is provided. The final component placement can be seen in figure 6.2.

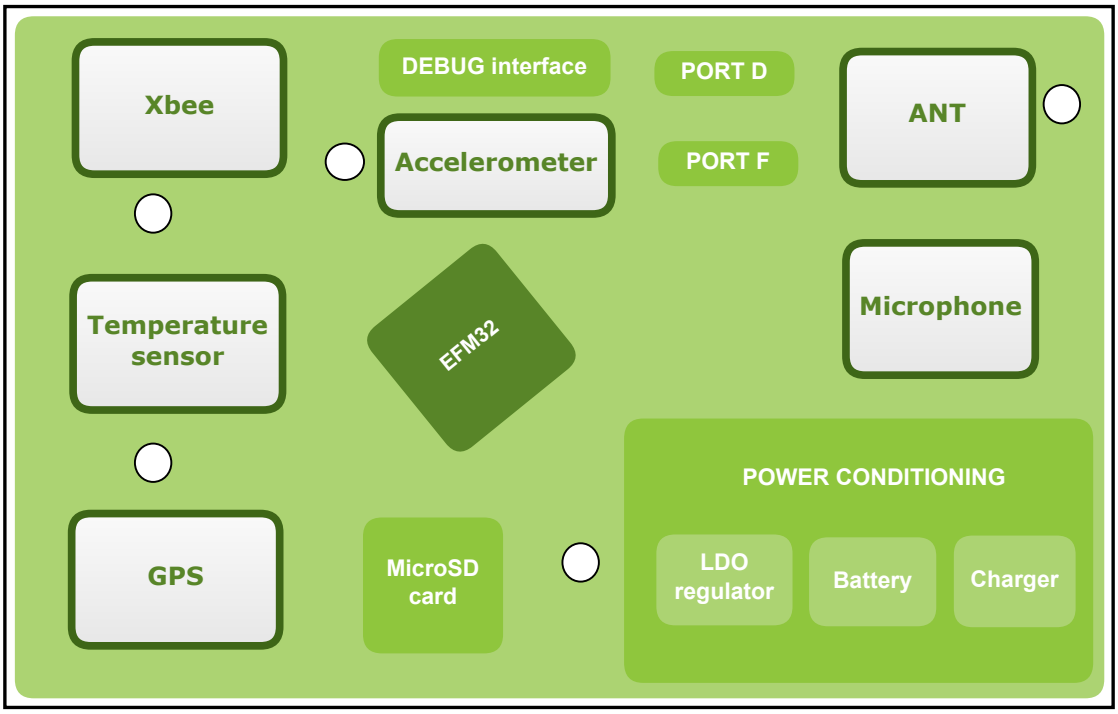

Figure 6.2: Main component placement on a PCB

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:

$$P_{Diss}(W) = I_{MAX}x(V_{ADAPTER(MAX)} - V_{PROTECT} - V_{RS} - V_{BAT}) \qquad (6.1)$$

When 750mA of current (I_{MAX}) is flowing through it and the input voltage (VADAPTER(MAX)) can take the maximum value of 9V Q1 dissipates 4.38W 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 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.

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 previous heatsink design multiple vias connect the upper and bottom layers of the PCB to promote better heat transfer.

TMP006 temperature sensor used in the project is one of the kind and has to have a specific board layout in order to function correctly. The TMP006 is affected by the IR energy from below the sensor as it is affected by the IR energy coming from objects that are in its field of view. In case the temperature of the sensor is the same as the PCB there will be no energy transfer between them and the sensor will function without errors. Nevertheless, if there is a difference between the PCB temperature and the die temperature it will cause the heat energy to be conducted through the thin layer of air and this will introduce offset to the voltage readings making the temperature calculations incorrect. In order to avoid this several factors had to be addressed:

- ◇ Thermal time constant matching of the PCB to the temperature sensor
- ◇ Thermal isolation of the TMP006 from any heat sources

Difference in thermal time constants can generate temperature mismatches between the PCB and TMP006 die until the temperatures stabilize. In order to avoid this and bring the constants close to each other a small copper fill under the temperature sensor is placed and connected to GND (see figure 6.3). An island of copper surrounds the the sensor and helps with the constant matching as well. Without the copper fill the temperature between the sensor and PCB will never achieve equilibrium, unless the system was in equilibrium for an extended period already.

The second factor addresses the thermal isolation problem. The sensor has to be isolated from devices that act as a heat source, this includes any active component placed near it. When the device becomes operational it dissipates a certain amount of heat into the ground plane, this in turn raises the temperature of the TMP006 as GND is common and could cause the same problems as with the thermal time constant mismatch. To address this issue the sensor and the copper island was isolated by low k material (see figure 6.4. In this case the PCB material is used as an insulator.

This isolation prevents the unwanted thermal interaction with other devices. Moving the device away from any source of heat would be the best solution. This way any drastic temperature changes would be invisible the temperature sensor.

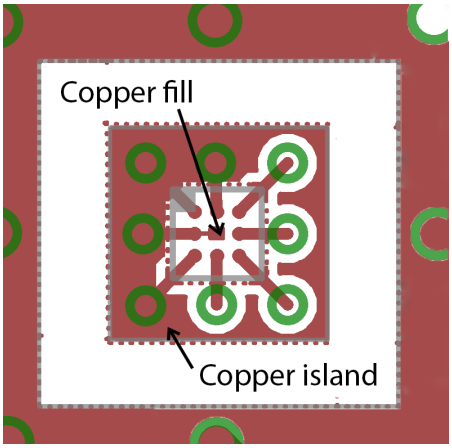

Figure 6.3: Constant matching

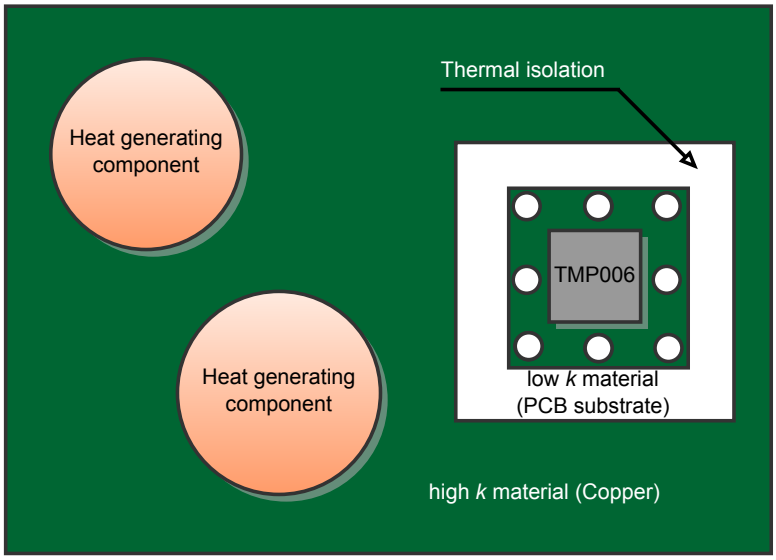

Figure 6.4: Thermal barrier

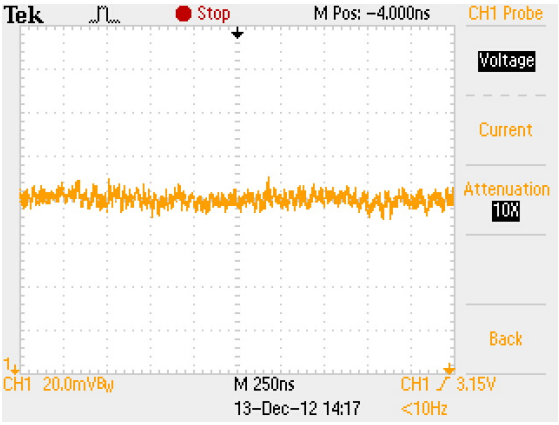

Figure 6.5: Measured voltage noise

Accelerometer has a mechanical issue that has to be addressed. In order to prevent any PCB vibrations being visible to the accelerometer it had to be mounted close to the PCB hard mounting point.

Several modules have integrated antennas in them, these are ANT, Xbee and GPS. No special PCB placement is required for them other than taking mechanical considerations into account. Any device containing metal should be kept away from the field of view of the integrated antenna to avoid signal scattering and reflections. Otherwise this could lead to an effective loss of signal radiation resulting in the reduction of the transmission range. Hence, modules were placed on the edges of the PCB and the GND plane was removed from the antennas field of view.

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. By doing this the voltage noise can be dropped to a very low figure. In the system the measured noise was 4mV, as it can be seen in figure 6.5

Oscillators are placed as close to the MCU as possible to avoid any additional trace capacitance and delay, which would change the generated frequency.

Taking all these considerations in mind the board was routed manually as this ensures the correct placement of the tracks and special zones needed for components. A good layout will always result in a proper functionality of the PCB and will reduce the probability of redesign.

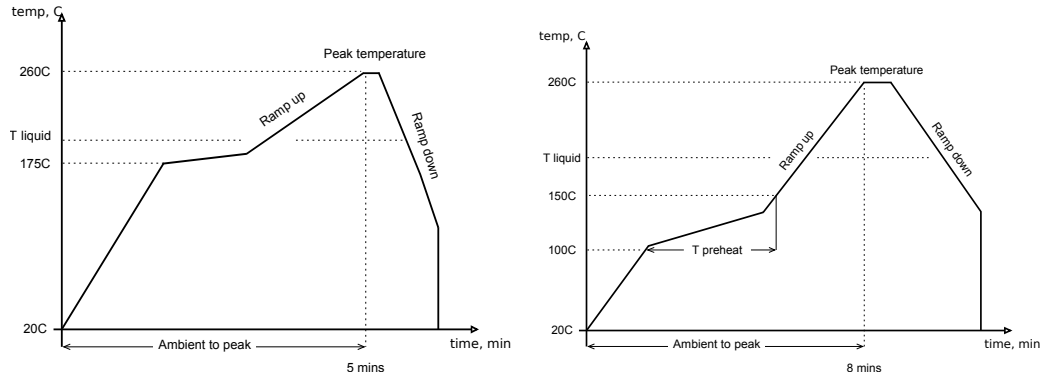

Figure 6.6: Optimal temperature profiles for soldering

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 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 6.6(a) and figure 6.6(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 6.7.

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

6.3 Case design

Several options for the case have been considered, including the usage of rapid prototyping facilities to make a custom case to fit the PCB. Due to high cost of the material used in production and limitations set on our budget it was decided to exclude this option. Case for the PCB was ordered through Maplin online store and rapid prototyping facilities were used to make a custom part that would attach the MEMS microphone board

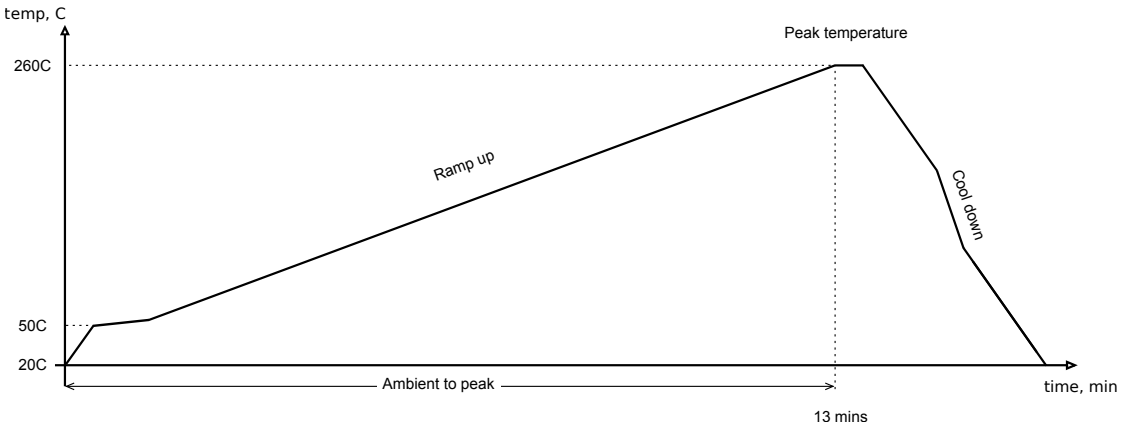

Figure 6.7: Used temperature profile for soldering

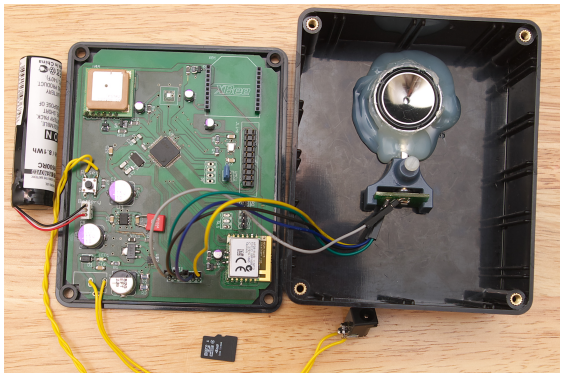

Figure 6.8: Exposed System in a case

to the head of a stethoscope instead. Case is made of plastic, which is not the best solution, considering the horse might damage it while rolling around. The optimal solution would be to use a metal case, this would prevent it from being damaged in the harsh exploitation environment. Nevertheless, this is not an option, as metal would act as a shield for RF signals either completely blocking them or diminishing to an unacceptable level.

The head of the stethoscope was then integrated into the case and isolated to ensure a decent level of splash proofness, as depicted in figure 6.8.

The Complete encapsulated system is shown in figure 6.9. The stethoscope head protrudes slightly from the case enabling a better contact with the horse under test. The case is certified to be splash proof according to IP54. This way the PCB might be protected even in rainy weather.

Figure 6.9: Completed encapsulated system

Chapter 7

Testing

In order to verify the correct behavior of our system, independent unit tests of each each subsystem were performed, before integrating them into the rest of the system. The verification of the test results was done manually and not by using a set of autonomous testbenches, as the test-driven methodology is still new and uncommon in the field of embedded systems.

7.1 Wireless communication

The functionality of our system depends on a wireless connection, which is why we needed to ensure that we can guarantee a stable connection and reliable transmission of data over the wireless link, while being able to transmit data at a reasonable data rate.

To verify that the connection fulfils these requirements we ran a stress test, where two devices, one configured as network coordinator, the other configured as a network node, exchanged small data packets over a long period of time (¿4h).

The reliability of the transmitted data was tested by sending a predictable set of values between the network nodes and aborting the test when receiving a value that differs from the expected value.

The stability of the connection was tested by imposing a timeout limit on both sides of the connection. If no data was received within a defined timeframe or no acknowledgement was received the test was also aborted.

When the system passed this stress test and we could rely on a stable and reliable connection we had to verify the last of the three requirements - the data throughput.

To test the data throughput we used the XBee interface we had developed to transmit messages larger than the maximal ZigBee frame size. The throughput was obtained by sending a defined amount of data and measuring the delay between sending and receiving the acknowledgment for the last part of the message. The results can be seen in figure 7.1.

Figure 7.1: XBee throughput

The throughput test revealed that the chosen wireless protocol is not able to transmit data nearly as fast as we expected. The maximal throughput that can be reached by our system is 13kbits/s, which is about 20 times slower than what we had expected (ZigBee data rate is often quoted as 250kbit/s).

It turned out that the firmware for the devices we are using (XBee) imposes a large additional overhead on the protocol and that even using a pure ZigBee implementation the data rate of 250kbit/s can never be reached, because the ZigBee standard itself uses part of the data rate to implement its functionality and vendor specific ZigBee implementations add even more overhead. The XBee device manual states this in a subsection, that we didn't take into account during the design phase. The maximal data rate that is specified for network using our configuration is 16kbit/s.

This result has a strong influence on our system, because it renders wireless transmission of audio data infeasible. A small calculation shows why

$$t_{transmission} = \frac{datarate * t_{duration}}{f_{transmission}} \tag{7.1}$$

$$= \frac{128kbit/s * 240s}{13kbit/s} \tag{7.2}$$

$$= 39min \tag{7.3}$$

It takes 39 minutes to transmit a 4 minute gut sound recording, which conflicts with our goal of a battery powered energy efficient system, because the processor has to be awake while transmitting the data over ZigBee and the battery runtime would decrease drastically.

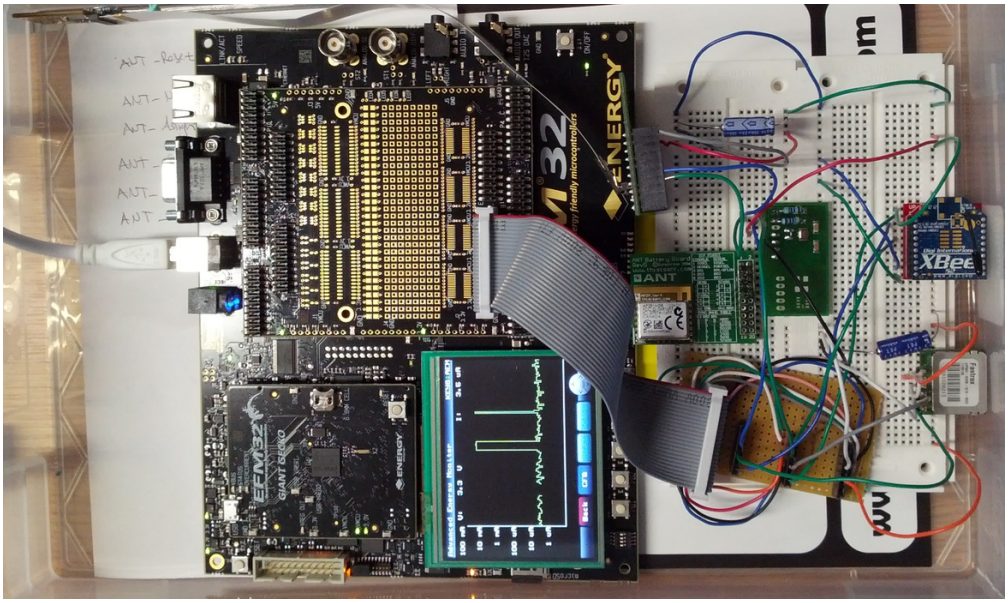

Figure 7.2: System Integration on Energy Micro DK

The current solution for this problem, is to store audio recording on a removable SD card in the monitoring device. Proper solutions for this problem are discussed in the Future Work Chapter (10.3)

7.2 System Integration testing

After the individual hardware subsystem prototypes had been verified, the units were connected to build a prototype of the complete system. Gradually all the sensors were integrated to the final monitoring device PCB prototype. Integration testing was done on the Energy Micro DK and integrated software debugging was done before the PCB was received. The prototype can be seen in figure 7.2.

In this stage, the monitoring device with its subsystems and the base station were tested. Measurement data was successfully transferred from the monitoring device over the wireless link and stored in the database in the base station. After the prototype of the complete system was tested and verified the modules were soldered on the PCB.

The final stage of integration on the PCB went without problems, or required changes to the software of the system.

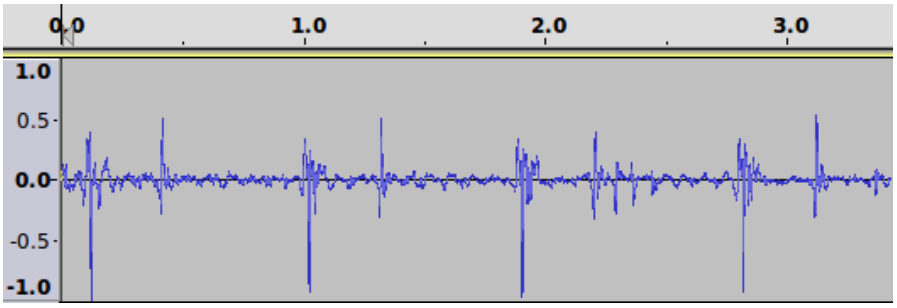

Figure 7.3: Testing results: heartbeat

7.3 Field Testing

Throughout the development process, the team did not have access to a horse and there is almost no sample gut sound recordings available online. After the integration and PCB production phases were complete, the whole system #TODO: was tested on human subjects since the subsystems of the monitoring device are usable on humans as well. The only horse-specific part of the system was the gut sound monitor - although it is also usable on humans, the design was not made with humans or species other than horses in mind.

In the laboratory environment, the team could only test the gut sound monitor on human subjects to record heart beats since human gut sounds are not as periodic and loud as those of a horse. Human heart beat recordings were made without problems, and four beats of an approximately 65 BPM heartbeat with the S_1 and S_2 (respectively the ”lub” and ”dub” sounds) can be clearly seen in figure 7.3

However, this is not the real usage of the module. Thus, the team made a field trip to New Forest to perform gut sound monitor testing on equines. Gut sounds of three horses were recorded from different sites of the abdomen. Each recording was done one minute long and saved in the SD card. The result was satisfactory although the environment was noisy because of the horses' movements and conversations. Therefore, noise cancellation filter is needed to improve the performance. The recordings can be found in the Appendix A.9.1.

- ◇ no gut sound recording was available on web
- ◇ we could test the integrated system (temperature etc on humans) but gut sound is horse-specific so we arranged a field trip. (TODO not sure if we should add this info: + testing the whole components on gut requires more time, components and people) and it was not crucial, because complete system testing was successful.
- ◇ we did gut sound recording on 3 horses from different parts of the abdomen, each 1 minute long. although the environment was noisy (horse was moving, conversations

etc) the recordings were successful. They can be found in appendix. A noise filtering needed to get better results.

Chapter 8

Results

#TODO:

Chapter 9

Project Management

9.1 Development Phases

To accomplish the objectives in the limited timeframe, development was planned in a way that would allow maximum parallelization of developer resources and tolerate unforeseen delays by leaving some unallocated time at the end. The development consisted of the following main stages:

1. **Project planning and system design:** The first step was to define the scope of the project, identify the main tasks and come up with a Gantt chart to schedule the tasks. Since the project was assigned to the team in the second week of the semester, the team had to start working on the project later than the official start date, which resulted in a tighter schedule. The top-level system design was also completed in this stage. The approach to design a distributed system with multiple energy-efficient wireless monitors and a base station was accepted. Functional requirements from the hardware were also determined according to the given specification.

2. **Background research:** The team researched the existing solutions for equine health monitoring, identified the applications of the resulting system and familiarized themselves with energy efficient implementation methods. As stated in section 2.3, discussions with field experts resulted in narrowing the scope of the implementation to a data collection tool for general monitoring and research since diagnosis of grass sickness was not considered feasible with our approach.

3. **Component research and ordering:** This stage consisted of searching for the hardware components that fulfilled the requirements identified in the previous stage. Decisions were made on the basis of energy efficiency, ease of development and stock availability of each component. Some components had to be ordered from non-ECS suppliers as they were not available from school suppliers. There were some delays in ordering and receiving the components which had a negative impact on the schedule because they prevented the team from starting with the next phase and pushed the work on hardware-dependent tasks in the system further back into the schedule.

4. **Hardware subsystem prototyping:** In this stage the team members worked in parallel on getting each hardware subsystem functional, writing drivers for them as described in chapter 4 and making the implementations energy efficient and stable.

5. **Integrating subsystems, system software:** This stage involved connecting together the implemented hardware subsystems into a single functional system, and creating the software frameworks on both the monitoring device and the base station to implement the desired data flow. A completely integrated prototype of the monitoring station on a development board allowed the team to test and verify the distributed nature of the system architecture before the custom PCB was ready.

6. **PCB design, production and case:** Ran in parallel with the other tasks, this stage involved designing the final printed circuit board (PCB) that would later be placed inside a stable case and became the core of the monitoring device. The experience gained during the hardware subsystem prototyping stage was infused into the PCB design process, which meant that the design of each sub-module on the PCB was tested and verified before the final PCB was ordered.

7. **Testing the integrated system:** Once the integration and PCB production stages were complete, the system was integrated on the PCB and tested with separate unit tests for each hardware subsystem. Due to careful individual prototyping and the previous successful integration on the development board of a complete prototype the final stage went relatively smooth and did not cause any delays in the schedule. The resulting completed prototype was also tested on human and horse subjects.

The time slots allocated for each task can be found in section A.3, and a detailed allocation of tasks and subtasks to team members can be found in section A.4.

Within the given timeframe the team managed to design and build a working prototype of the distributed system, which was almost feature complete according to the specifications. Please consult the chapter 8 for details.

9.2 Resources

9.2.1 Hardware Development Tools

he core of the monitoring device system is based on an Energy Micro microcontroller. Because some of the team members had worked with Energy Micro devices in the past

we had access to three Energy Micro starter kits (STK), and were supported by Energy Micro who supplied us with an additional starter kit[1] and a development kit (DVK).

Having access to four STKs enabled us to build build the subsystems in parallel on a very similar hardware platform that is also used for the final product. The more powerful DVK was used to integrate them before the custom PCB arrived. The custom PCB layout had to be sent for manufacturing as there are no facilities to produce the board in-house. Typical turnaround can be up to 7 days. Nevertheless, this is a very crucial component to the project and delay would result in significant setbacks. Hence, it had to be ordered with a faster turnaround.

PCB assembly was done in the GDP lab using the following conventional tools: soldering and hot air stations, microscope, tweezers, etc. Hot plate was used for solder reflow of wafer chip scale components. Breadboards and wires were provided from the start of the project and aided in the prototyping stage. Oscilloscopes and multimeters were used for testing and fine-tuning the PCB modules.

9.2.2 Team

The development team initially consisted of 6 people. However one member, Ahsan Baig, had to drop the project after the sixth week since he could not enter the UK due to visa issues.

The initial project planning was made considering six project members. As it was unclear from the beginning if Ahsan would manage to arrive in time for the project, he was mostly assigned 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. After it became certain he would not be working with the team his tasks were distributed among the rest of the team or, as in the case of data analysis, discarded.

9.2.3 Research

To gain a deeper insight into the physiology of horses and horse related health issues, contact was established with experts in the field: Dr. John Chad, lecturer at Biological Sciences and Dr. Neil Smyth, lecturer at Development and Cell Biology, both from the University of Southampton.

The initial contact led to a meeting where Dr. Neil Smyth took the time to give an enlightening presentation about the digestive system of horses.

The discussion after the presentation helped us to have a more realistic view on the goals of our project and excluding data analysis and diagnosis from the project scope,

[1] The team joined the Energy Micro EFM32 Design Contest 2012 with this project and qualified as one of the best design ideas. One of the starter kits was received as an award from this competition. The team will also compete at the final stage of the contest with the final product.

because even expert veterinarians have difficulties diagnosing grass sickness disease. A discussion of the reasons for this decision are provided in the chapter 2.

9.3 Task Distribution

Project tasks were distributed 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 Google document was used to record the tasks and their status, which can be found in section A.5. All members attended discussions about system design and did research on component selection. Apart from these, the primary responsibilities of the team members are listed below.

Jose Cubero:

- ◇ Audio acquisition
- ◇ DMA data transfers
- ◇ Internal flash storage
- ◇ Background research
- ◇ Field testing

Merve Oksar:

- ◇ Background research
- ◇ Data storage and accelerometer subsystems
- ◇ Debug interface of the final product
- ◇ Report management
- ◇ Testing

Konke Radlow:

- ◇ Wireless communication over ZigBee
- ◇ Base Station
- ◇ Web Interface
- ◇ Testing

Michail Sidorov:

- ◇ Schematic design
- ◇ Breakout board and PCB layout design
- ◇ Hardware assembly

- ◇ Budget planning and tracking
- ◇ Testing

Yaman Umuroglu:

- ◇ System software
- ◇ ANT HRM, GPS, temperature sensor subsystems
- ◇ Integration
- ◇ Project management

9.4 Budget Planning

The team had 200GBP of official budget for this project. The components were chosen based on their price, quick availability and technical features. The components used in the project and their costs are listed in section A.7. It was possible to obtain free samples for some of the components. To give an idea of both the team budget and how much it would cost to build a commercial device similar to ours, both the cost the team spent on the components and their actual prices are listed in the Appendix A.7.

9.5 Communication

The team was aware that good communication was essential to leverage to full potential of a team of five. It was set as a central principle that each group member should be aware of the tasks the other group members are working on, have knowledge about progress that was made and the troubles that were encountered. For this reason weekly group meetings were held to inform all members about each person's progress, and to distribute outstanding and new tasks to the group members. This allowed iterative refinements to the initial development plan stated in the Gantt chart.

The meetings also served as a ground for important or critical decisions that required views from the entire group.

An e-mail group and online document sharing over google docs was used to share information between the group members and the supervisor. The code was managed in a git code repository [2].

[2] urlhttp://code.google.com/p/equine-health-monitor-gdp12/

Chapter 10

Conclusions

10.1 Success

A working prototype for an equine health monitor has been built. The project can be further developed into a commercial product. It also provides many research possibilities in the related fields.

- ◇ it has a potential for further research, lots of interesting possibilities
- ◇ well documented, structured for future developers
- ◇ commercial success potential

Objectives specified for the project are met. Scalability was a desired feature to be able to monitor multiple horses simultaneously. A distributed system with one base station and multiple monitoring devices was designed meet this requirement. Power consumption was critical for the battery-powered monitoring devices. Long battery life is achieved through power-aware design with the help of periodic-sampling strategy and minimal processing on the monitoring device. The collected data is accessible via web and this enables users to retrieve data easily.

- ◇ Long battery life is achieved through the power-aware design
  - – Periodic sampling
  - – sleep modes
  - – minimal processing on the monitoring device
- ◇ distributed system addresses the scalability problem
- ◇ web server done for easy accessibility
- ◇ a low-cost, splash proof case for pcb. Stethoscope head from mechanical workshop for the microphone. But a special design can be made, e.g. with straps etc. PCB can be made smaller.

The team had to face some problems and managed to solve most of them in a quick and efficient way. The situation of the sixth member was not clear, therefore the team did

the task distribution in a way that it would minimize the problems in case of late arrival or quitting. After he quitted, the team discarded or reallocated his tasks.

Late assignment of the project to the group delayed the development stage however the team managed to meet the deadlines although it affected some of the design decisions...

The PCB production was not on time, therefore we ordered more PCB's from other manufacturers when the deadline we set ourselves for PCB production approached.

Since none of the team members has a background in bioscience or a related field, consultation to an expert was needed and the team used the information gathered from consultation sessions to make a more realistic approach to the problem.

The team managed its human and hardware resources well by parallelizing tasks. Testing each unit enabled a faster integration. Integration on DK before the PCB arrived made PCB integration and debugging easier.

The team did a good budget management to reduce costs. We requested sample ICs from the manufacturers/suppliers. This was especially important for ANT development kit which is quite expensive. Also, we joined a design competition and won a starter kit containing MCU we are using.

Things we managed well:

- sixth member being far and then quitting
- late start of the project
- taking quick action against pcb production delay
- consulting the experts for a more realistic approach
- parallelization of tasks, using developers and the devices efficiently
- testing each unit and integration on DK before PCB arrived made the PCB integration stage faster
- ordering samples, ANT module was very expensive and we managed to borrow one

10.2 Criticism

The team did not have much time to perform an in depth research to choose the optimum tools and components for the project. During the development stage some lessons learned: `#TODO:`

- Zigbee was chosen because... the throughput of X can be achieved with zigbee. another wireless protocol maybe better (Bluetooth?)..

- ◇ some components arrived much later than expected. that delayed prototyping subsystems. what could be done for that? backup plans? but we learned our lesson and didn't do the same mistake when pcb manufacturing was delayed
- ◇ Gantt chart was kind of unrealistic, integration and building subsystems are assumed to be finished at the same time

10.3 Future Work

The scope of this project is rather broad and the size of the project is quite large given the limited timeframe. For this and other reasons mentioned throughout the report the state of the system is far from being perfect, yet it is a functional prototype of the envisioned Equine Health Monitor.

We would like to use this chapter to discuss solutions to problems in the system design and implementation that we discovered, and propose a number of improvements that can be made to the system to increase its usability and extend the possible applications.

10.3.1 Solutions for design problems

During the testing phase of the wireless communication module of our system we realized, that the chosen solution is not able to deliver the bandwidth that we require to transfer audio recording from the monitoring devices to the base station in a reasonable amount of time.

The problem is that the ZigBee protocol is designed for low bandwidth requirements, for even it is a low power protocol, the energy cost per transmitted byte is a lot higher than for other wireless solutions, such as 802.11 or Bluetooth.

Because we do not require a constant connection, but are able transmit data periodically in a burst transmit style, it is feasible to use an alternative solution and run it at a low duty cycle.

To be more specific in terms of our design, there are pin compatible replacements for XBee devices that implement the 802.11 standard and the manufacturer of the XBee devices is working on a 802.11 based XBee version as well (currently available as a development kit, not retail)

Replacing the XBees with alternative devices doesn't require modifications to the system design, and because of the Object Oriented design of the software system and well defined interfaces the changes that are required in the software stack are limited to a clearly defined scope.

An audio compression scheme is then recommended if the system is required to perform audio recordings often and/or send this data over the wireless connection. Such a

scheme was not implemented in the project mainly the short amount of time available for development and that a deeper knowledge of the signals is required but could be implemented in an improved version of the system.

10.3.2 Possible improvements

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 horse's 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.

In the current state of the system the webinterface is not very user friendly when it comes to analyzing large amounts of collected data. This could be improved by using advances web interface technologies like Ajax to build dynamic version of the website that behaves more like a desktop application than a website.

As accessibility is an important goal of the project it would be a useful addition to extend the website so that a handheld optimized version of the website is delivered when the user accesses the base station with a smartphone.

The limitations of the used XBee devices was discussed in section 7.1. To be able to transmit audio recordings the ZigBee network should be replaced with a faster wireless connection. Pin compatible XBee replacement ICs based on the 802.11 standard exist and require only a modification on the software part of the system.

Even though the system was designed with energy efficiency in mind from the very beginning, there are subsystems that require further optimization for low energy consumption. The XBee devices are not sent to sleep in the current system, because the delay before they manage to reestablish a connection is too long.

Appendix A

Appendix

A.1 PCB layout

Figure A.1: PCB layout top

Please stand by.
Coming soon!

Figure A.2: PCB layout bottom

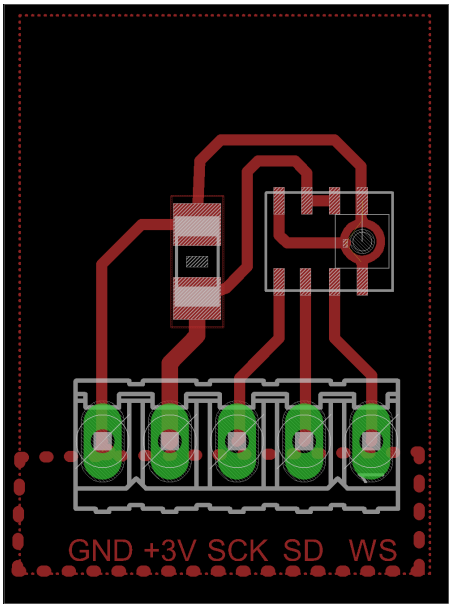

Figure A.3: PCB layout microphone

A.2 PCB schematics

- ◇ PCB schematics: Power and LDO A.4
- ◇ PCB schematics: MCU, decoupling, oscillators and reset A.5
- ◇ PCB Schematics: ANT, XBee and SD A.6
- ◇ PCB Schematics: I2C and GPS A.7
- ◇ PCB Schematics: Debug headers and microphone A.8
- ◇ PCB schematics: Microphone A.9

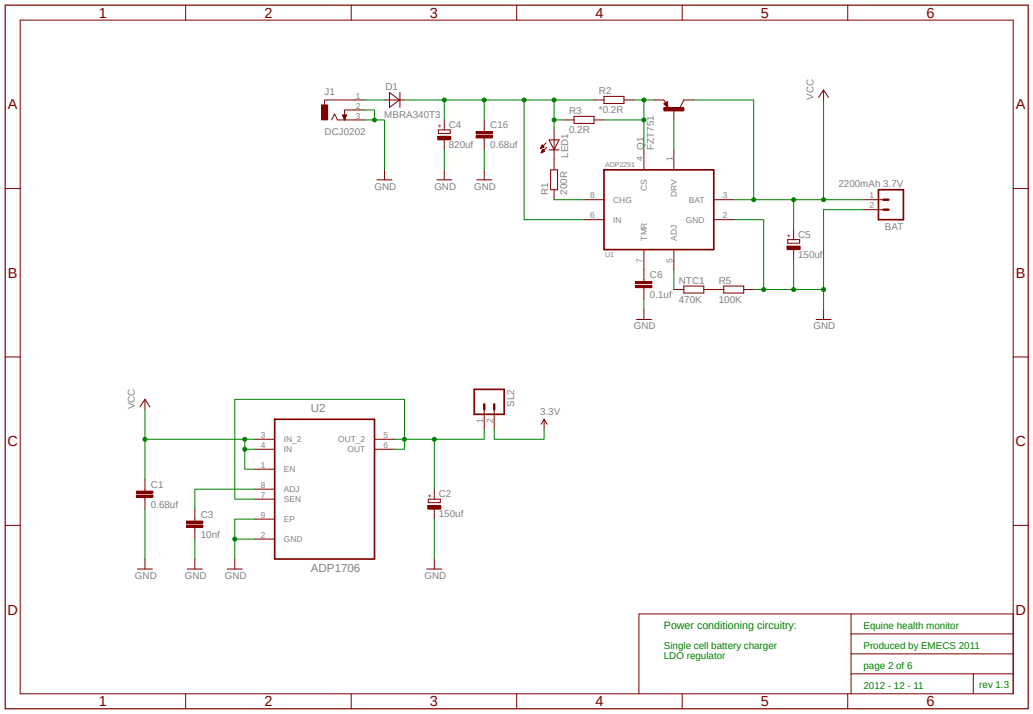

Figure A.4: PCB schematics: Power and LDO

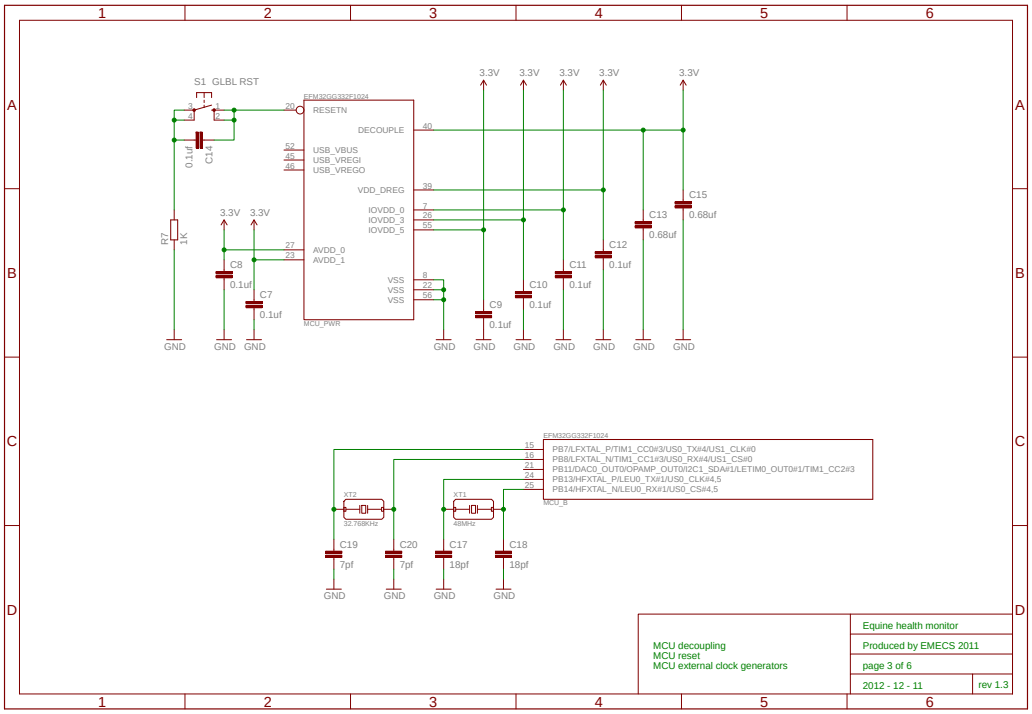

Figure A.5: PCB schematics: MCU, decoupling, oscillators and reset

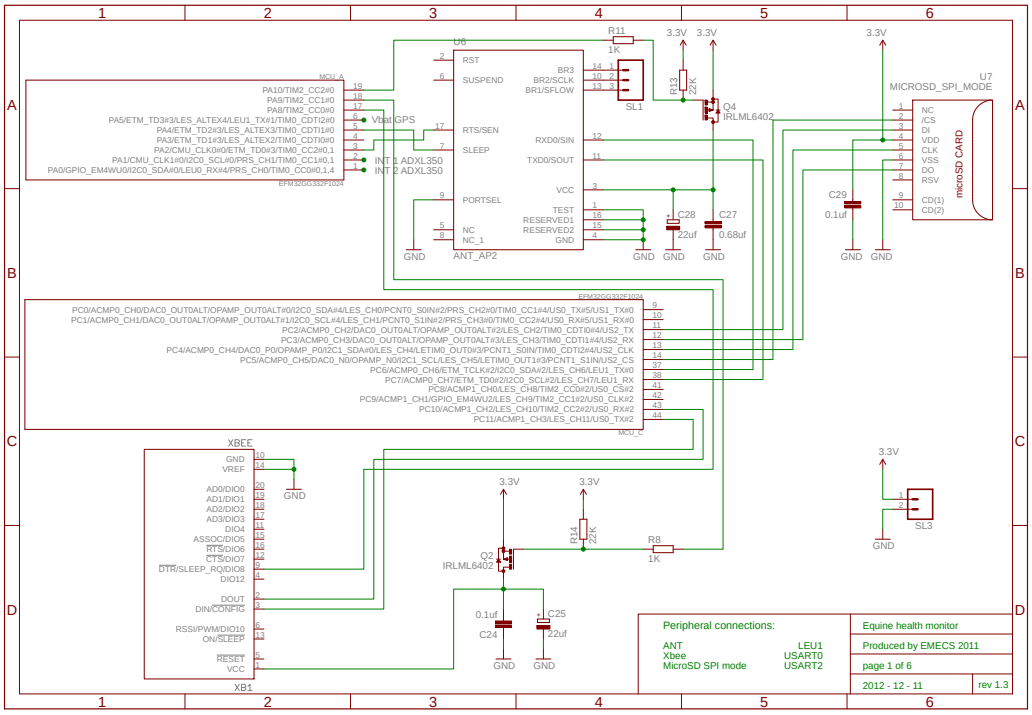

Figure A.6: PCB schematics: ANT, XBee and SD

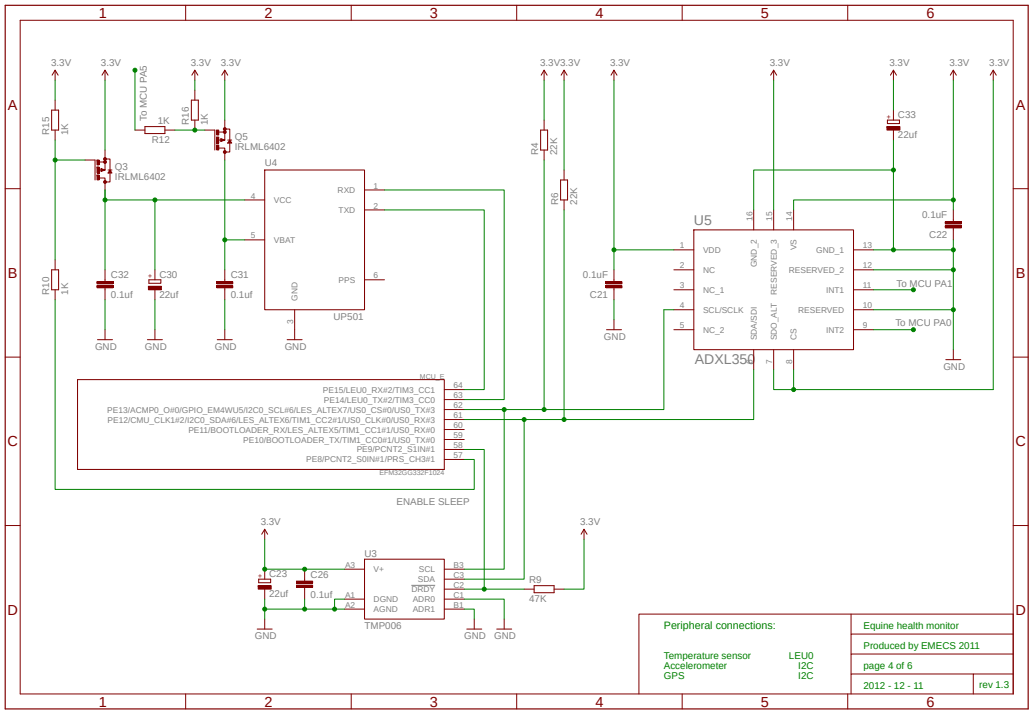

Figure A.7: PCB schematics: I2C and GPS

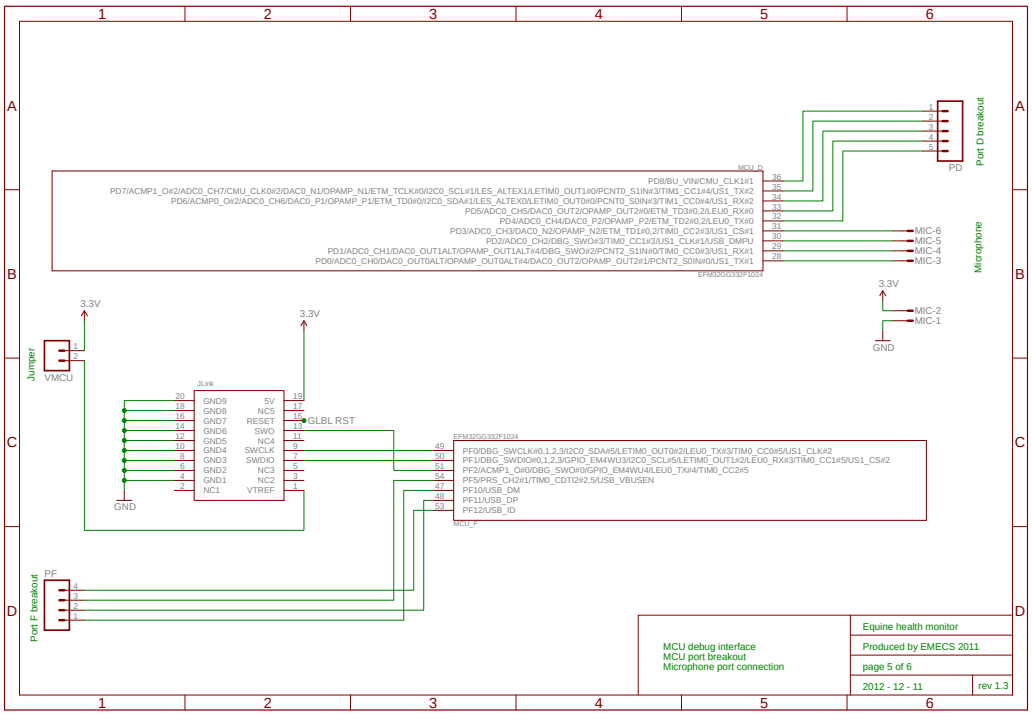

Figure A.8: PCB schematics: Debug headers and microphone

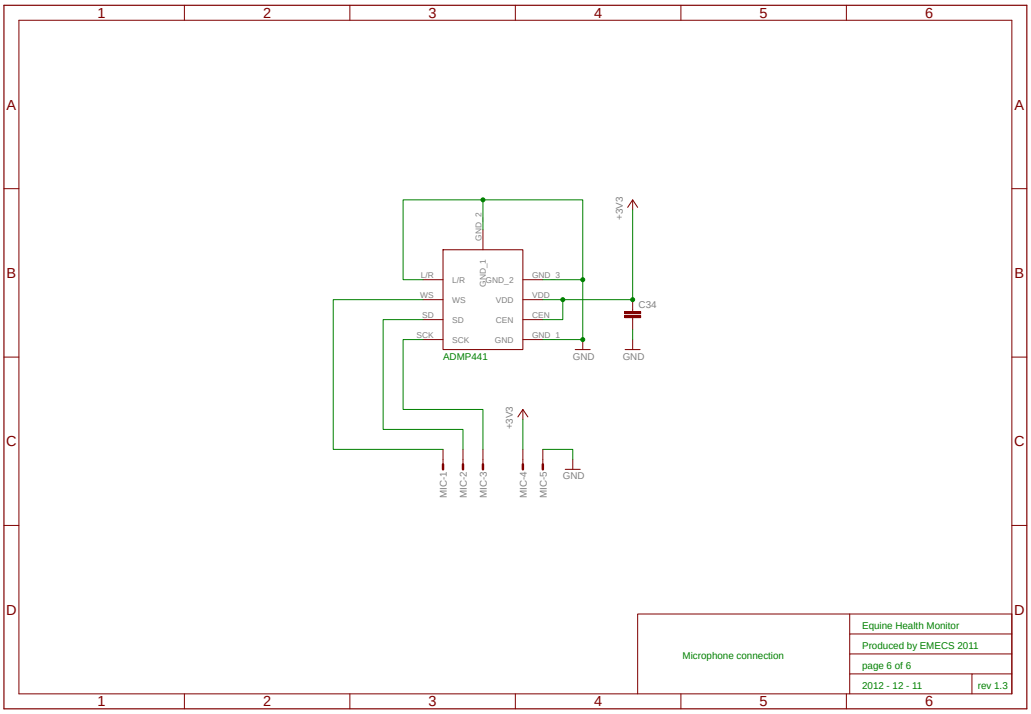

Figure A.9: PCB schematics: Microphone

A.3 Gantt Chart

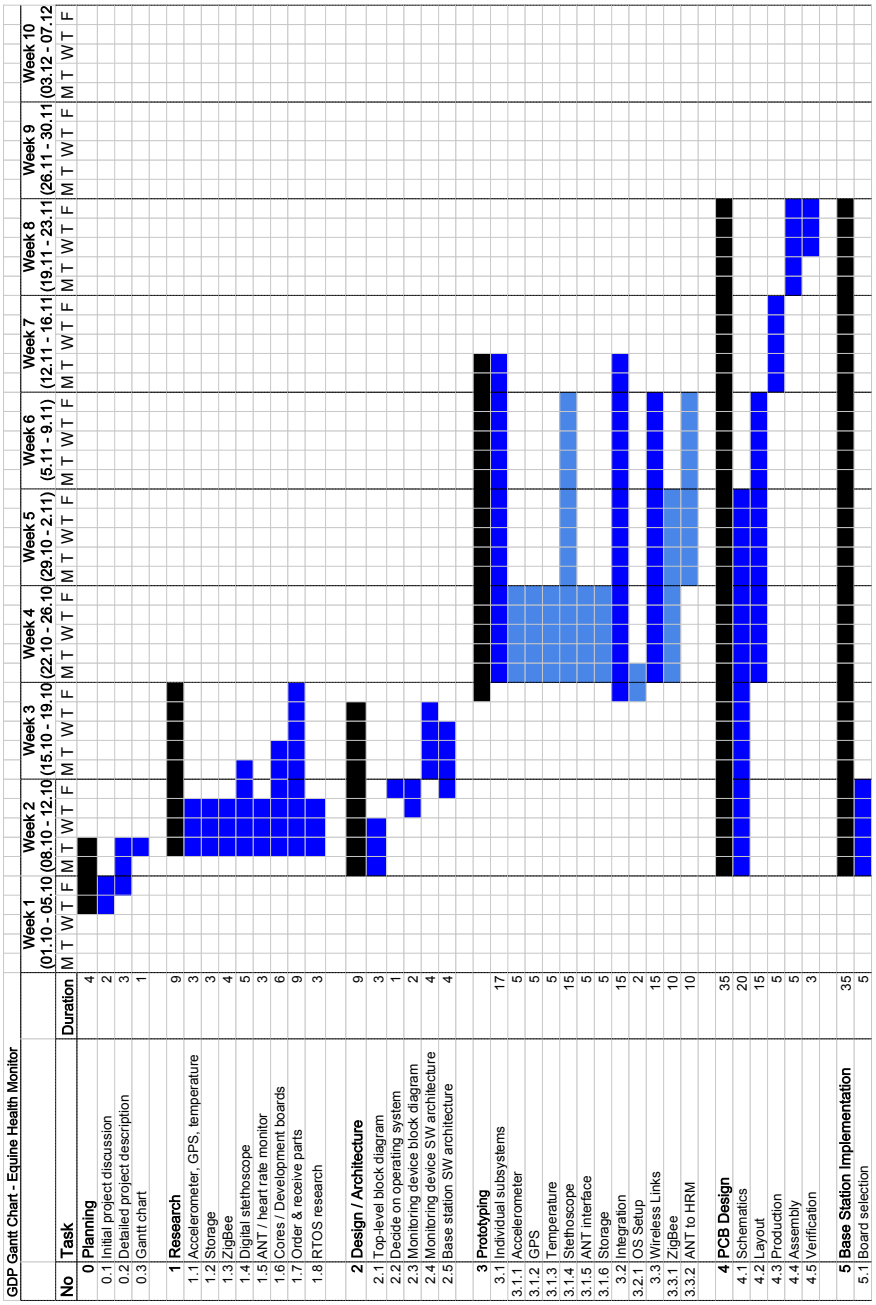

Figure A.10: Gantt Chart

A.4 Task distribution

#TODO:

A.5 Task break-down

#TODO:

A.6 Code Listings

A.6.1 sensorinterface.h

A.7 Price List

Please refer to table A.1.

Component	**Price (GBP)**	**Retail Price (GBP)**
Ceramic capacitors	0.24	0.24
Flastic Film Capacitors	2.45	2.45
Polymer Capacitors	5.52	5.52
Diodes	1.2	1.2
Resistors	0.17	0.17
Transistors	2.25	2.25
Charger IC	Sample	0.64
LDO	Sample	0.64
Crystal Oscillators	3.18	3.18
Temperature Sensor	Sample	4.09
MEMS microphone	Sample	0.62
GPS module	33.89	33.89
ANT module	14.52	14.52
Accelerometer	Sample	2.84
EMF32GG332 MCU	Sample	4.67
Battery	6.89	6.89
Misc sockets - connectors etc.	20	20
Xbee 2	22.8	22.8
Xbee PRO	44.32	44.32
Xbee Explorer USB	19.94	19.94
2.4 GHz antenna	6.61	6.61
MicroSD card	2	2
Raspberry Pi	26.99	26.99
Heart rate monitor	20	20
Device cases	12.27	12.27
Total:	212.97 GBP	226.47 GPB

Table A.1: Component Price List

A.8 Part List

Please refer to table A.2 and A.3 for the part list.

Part	Value	Device	Description	Package	Manufacturer	Man. part nr.
BAT	2200mAh 3.7V	Battery	Li-Ion - 2200mAh - 8.1Wh	18650	Cameron Sino	CS-PSU9600RC
C1	0.68uf 25V	Capacitor	ECPU(A)- Plastic Film	C1206	Panasonic	ECPU1C684MA5
C2	150uf 10V	Capacitor	OS-CON SVP	153CLV-1010	Sanyo	10SVP150M
C3	10nf 50V	Capacitor	X7R MLCC	C1206	AVX	12065C103KAT2A
C4	820uf 25V	Capacitor	Electrolytic	153CLV-1010	Panasonic	EEEFT1E821AP
C5	150uf 10V	Capacitor	OS-CON SVP	153CLV-1010	Sanyo	10SVP150M
C6	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000478
C7	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000479
C8	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000480
C9	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000481
C10	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000482
C11	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000483
C12	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000484
C14	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000484
C17	18pf	Capacitor	NPO MLCC	C1206	Kemet	C1206C180J5GACTU
C18	18pf	Capacitor	NPO MLCC	C1206	Kemet	C1206C180J5GACTU
C19	7pf	Capacitor	NPO MLCC	C0805	Multicomp	MCCA000315
C20	7pf	Capacitor	NPO MLCC	C0805	Multicomp	MCCA000315
C21	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000478
C22	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000478
C23	22uf	Capacitor	OS-CON SVP	PANASONIC A	Sanyo	6SVP22M
C24	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000478
C25	22uf	Capacitor	OS-CON SVP	PANASONIC A	Sanyo	6SVP22M
C26	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000478
C28	22uf	Capacitor	OS-CON SVP	PANASONIC A	Sanyo	6SVP22M
C29	0.1uf 50V	Capacitor	X7R ceramic	C1206	Multicomp	MCCA000478
C30	22uf	Capacitor	OS-CON SVP	PANASONIC A	Sanyo	6SVP22M
C31	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000478
C32	0.1uf 50V	Capacitor	X7R MLCC	C1206	Multicomp	MCCA000478
C33	22uf	Capacitor	OS-CON SVP	PANASONIC A	Sanyo	6SVP22M
CON1	JLINK SWO	Connector	2X10 way	DIP20	AMP	1-1241050-0
D1	MBRA340T3	MBRA340T3	Schottky Pwr Rect. 3A 40V	SMA	ON Semicond.	MBRA340T3G
J1		Socket	DC Power Socket 2.1mm	DCJ0202	SwitchCraft	RAPC722
LED1	3.2V 20mA	LED	True green 300mcd	CHIPLED 1206	Multicomp	OVS-0804
MCU	EFM32GG332F1024	MCU	Cortex M3 based MCU	TQFP64	Energy Micro	EFM32GG332F1024
NTC1	470K	Thermistor	0.25W 10%	R0805	AVX	NB20Q00474KBA
PD	-	Header	5 way	05P	AMP	
PF	-	Header	4 way	04P	AMP	

Table A.2: Part list part 1

Part	Value	Device	Description	Package	Manufacturer	Man. part Nr.
R1	200R	Resistor	0.25W 5%	R1206	Panasonic	ERJ8GEYJ201V
R2	0.2R	Resistor	0.5W 1%	R1210	WELWYN	LR1206-R20FI
R3	0.2R	Resistor	0.5W 1%	R1210	WELWYN	LR1206-R20FI
R4	22K	Resistor	0.25W 5%	R1206	Multicomp	CRCW120622K0FKEA
R5	100K	Resistor	0.25W 5%	R0805	Multicomp	MCSR12X1003FTL
R6	22K	Resistor	0.25W 5%	R1206	Multicomp	CRCW120622K0FKEA
R7	1K	Resistor	0.25W 5%	R1206	Multicomp	MCMR12X102
R8	1K	Resistor	0.25W 5%	R1206	Multicomp	MCMR12X102
R9	47K	Resistor	0.25W 5%	R1206	Multicomp	MC 0.125W 1206 5% 47K
R10	1K	Resistor	0.25W 5%	R1206	Multicomp	MCMR12X102
R11	1K	Resistor	0.25W 5%	R1206	Multicomp	MCMR12X102
R12	1K	Resistor	0.25W 5%	R1206	Multicomp	MCMR12X102
R13	22K	Resistor	0.25W 1%	R1206	VISHAY DRALORIC	CRCW120622K0FKEA
R14	22K	Resistor	0.25W 1%	R1206	VISHAY DRALORIC	CRCW120622K0FKEA
R15	22K	Resistor	0.25W 1%	R1206	VISHAY DRALORIC	CRCW120622K0FKEA
R16	22K	Resistor	0.25W 1%	R1206	VISHAY DRALORIC	CRCW120622K0FKEA
S1		Push Button		B3F-10XX	TE CONNECTIVITY	5-1437565-0
SD1		MicroSD Slot		MSD CON	MOLEX	49225-0821
SL1		Header	3 way	03P	AMP	
SL2		Jumper	2 pin	02P	AMP	
SL3		Header	2 way	02P	AMP	
U1	ADP2291	Charger	Sgl Cell Batt. charger	MSOP8	Analog Devices	ADP2291ARMZ-R7
U2	ADP1706	LDO Regulator	3.3V fixed	SOIC8	Analog Devices	ADP1708ARDZ-3.3-R7
U3	TMP006	Temperature Sensor	IR MEMS thermopile sensor	DSBGA8	Texas Instruments	TMP006AIYZFR
U4	UP501	GPS	GPS Receiver	GPS UP501	Fastrax	UP501
U5	ADXL350	Accelerometer	3-axis - 13-bit - SPI I2C output	LGA CAV 16	Analog Devices	ADXL350BCEZ-RL7
U6	ANT AP2	ANT transceiver	RF module	ANT	Dynastream Innovations	ANT AP2
USART1		Header	6 way	06P	AMP	
VMCU		Jumper	2 pin	02P	AMP	
XB1	XBEE	XBEE	RF module	XBEE	Digi International	XBEE2
XT1	48MHz	Oscillator	Fundamental AT cut	6X3.5	Abracon	ABM7-48.000MHZ-D2Y-F-T
XT2	32.768KHz	Oscillator	QMEMS	3.2X1.5	EPSON TOYOCOM	FC-13F 32.768KHZ 20PPM - 7PF
MIC	PmodMIC	Microphone	Mic w/ dig. if	6 way header	Digilent	PmodMIC

Table A.3: Part list part 2

#	Description	Part of the abdomen	Link
1	15 - male	lower right	https://www.dropbox.com/s/crzdd7bwobuvlvo/1.wav
1	15 - male	upper right	https://www.dropbox.com/s/0ov4a1cwo0c5psg/2.wav
1	15 - male	mid right	https://www.dropbox.com/s/eyu9nwihmvw2iux/3.wav
1	15 - male	lower left	https://www.dropbox.com/s/oyqwoq4t5sx9os2/4.wav
1	15 - male	upper left	https://www.dropbox.com/s/80hywu98uce7dpu/6.wav
1	15 - male	mid left	https://www.dropbox.com/s/mcgfwh4gkfd99g6/7.wav
2	4 - female	lower right	https://www.dropbox.com/s/3g4wemajjnaneka/1.wav
2	4 - female	upper right	https://www.dropbox.com/s/xqmel1pp2wikxm1/2.wav
2	4 - female	mid right	https://www.dropbox.com/s/ljvuwyrmnzeoty5/3.wav
2	4 - female	lower left	https://www.dropbox.com/s/kejdiwvqdn7sv77/4.wav
2	4 - female	upper left	https://www.dropbox.com/s/sc0i685pfisuik3/5.wav
2	4 - female	mid left	https://www.dropbox.com/s/eowf9oj7zl8wn4u/6.wav
3	8 - female	lower left	https://www.dropbox.com/s/36hx3r2hwatueoe/7.wav
3	8 - female	upper left	https://www.dropbox.com/s/c2mceq21zuokni1/8.wav
3	8 - female	lower right	https://www.dropbox.com/s/49uhkhl2cgfu3xr/9.wav
3	8 - female	upper right	https://www.dropbox.com/s/vusnz42go74w1xy/10.wav
3	8 - female	mid right	https://www.dropbox.com/s/j0okmi9rzn0vl7k/11.wav

Table A.4: Gut sound recordings

A.9 Fieldtrip

A.9.1 Gut Sound Recordings

Please refer to table A.4

A.9.2 Pictures

Please refer to figure A.11 and A.12.

Figure A.11: Photo: Field trip (a)

Figure A.12: Photo: Field trip (b)

Bibliography

[Corley and Stephen(2009)] K. Corley and J. Stephen. *The Equine Hospital Manual*. Wiley, 2009. ISBN 9781444309379.

[Edwards et al.(2010)Edwards, Martz, Rogge, and Heinrich] S.E. Edwards, K.E. Martz, A. Rogge, and M. Heinrich. Edaphic and phytochemical factors as predictors of equine grass sickness cases in the uk. *Frontiers in Pharmacology*, 1, 2010.

[Fund()] Equine Grass Sickness Fund.

[Robinson and Sprayberry(2009)] N.E. Robinson and K.A. Sprayberry. *Current Therapy in Equine Medicine 6*. Current Veterinary Therapy Series. Elsevier Limited, Oxford, 2009. ISBN 9781416054757.