NTNU – Trondheim
Norwegian University of
Science and Technology

A Sensor Module for the WiVind Project

Optimizing energy consumption in wireless
sensors through conditional
data-transmission

Audun Lønmo Knudsrød

Master of Science in Cybernetics

Submission date:	June 2014
Supervisor:	Amund Skavhaug, ITK
Co-supervisor:	Oddbjørn Malmo, Kongsberg Maritime
	Stig Petersen, SINTEF ICT

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Statement

Insufficient battery lifetimes impedes the utilization of wireless sensors, and a majority of the energy is spent on wireless communication with the sensor's network. This project will investigate the potential of reducing transmitted data as a means of reducing energy consumption. This reduction will be attempted achieved by increasing the computational capabilities of the wireless sensor.

Preface

This report documents the work done on a technical project on low-energy embedded sensor systems. It was written at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology in the spring semester of 2014.

The project is auxiliary to a research project conducted by SINTEF and Kongsberg Maritime, WiVind. WiVind investigates the possibility of mounting small sensor modules on the surface of wind-turbine blades (the ”wings”) to monitor the condition of the blade. Such a module will have limited energy available, and investigating how to use the available energy more efficiently was suggested as a master thesis problem by Oddbjørn Malmo of Kongsberg Maritime.

Personnel both at SINTEF and Kongsberg Maritime have been consulted and given input to the project.

Trondheim, 2014-06-26

Audun Lønmo Knudsrød

Acknowledgment

I would like to thank my supervisor for the project, Amund Skavhaug at the Dept. of Engineering Cybernetics, NTNU, for his guidance during the project. He has provided significant advice on the project work itself and counsel on the writing of a technical report.

Further I would like to thank the assistance I have been given by SINTEF, particularly my assistant supervisor Stig Petersen. His knowledge on wireless instrumentation helped shape the path of the project work. I would also like to thank all the help I have received from Kongsberg Maritime's Sensor & Instrumentation department, especially so my assistant supervisor Oddbjorn Malmo for suggesting the project.

Key parts of the hardware used in this project has contributed by *Texas Instruments* and *Silicon Labs*. These companies have also been very helpful in answering questions and giving assistance related to their equipment. This help greatly accelerated my work.

Finally, I continue to be inspired by the Internet's ability do distribute human knowledge never ceases to amaze me. The forums and websites where enthusiasts and experts provide each other with assistance for no obvious personal gain is inspiring.

A.L.K.

Summary

This report examines the energy consumption in a wireless sensor module, a module which is part of the WiVind research project. The sensor module is to be surface-mounted on wind-turbine blades and alert a host system if a fault is detected in the blade structure. It must operate without an external power supply for several years at a time and thus reducing energy consumption is important to the project success.

The report's results indicates that the most effective way to reduce energy consumption is to reduce the amount of data transmitted from the sensor module to its receiving host system.

This reduction is made possible by using computing power placed in the sensor module to analyse the sensor data and transmit only the results of that analysis. The extra energy cost of that computing power is shown to be outweighed by energy reduction in reduced data transmission, thus resulting in lower system-wide energy consumption.

The quantitative results in the report show that energy savings may first be reduced by approximately three quarters by using the module to analyse data, and that once this is in place, further energy savings are achieved by shutting down hardware circuits when they are not in use. Specifically, the average power level of the sensor module is reduced to 23 % by utilizing local data analysis and then to less than 3 % by switching off inactive hardware.

Additional savings can be made by making the sensor module capable of transmitting data conditionally, only spending energy on communication when detecting a fault in the blade.

These findings are of such a magnitude that they might be relevant for sensor modules which differs in configuration or tasks.

In addition to the quantitative results, the design and operation of the sensor module prototype that was built to aquire the results is shown.

Sammendrag

(Norwegian Summary)

Denne rapporten undersøker energiforbruket i en trådløs sensormodul som er en del av forskningsprosjektet WiVind. Sensormodulen skal overflatemonteres på en vindmøllevinge med den hensikt å detektere feil på vingen. Den skal kunne fungere i flere år uten ekstern strømforsyning, og dermed er reduseret energiforbruk viktig for prosjektets suksess.

Resultatene i denne rapporten tilsier at den mest effektive måten å redusere energiforbruket på er å redusere mengden av data sendt fra sensormodulen til mottakersystemet.

Denne reduksjonen er gjort mulig ved hjelp av datakraft plassert i sensormodulen som analyserer sensordata og kun overfører resultatene av denne analysen. Den ekstra energikostnaden som analysen medfører er vist å ha en langt mindre størrelse en den mengden som spares ved å sende mindre data, og dermed er resultatet et lavere totalt energiforbruk for sensormodulen.

De kvantitative resultater viser til at energibesparelser først kan reduseres med omtrent tre fjerdedeler ved hjelp av at modulen analyserer data, og at når dette er på plass, kan ytterligere energibesparelser oppnås ved å slå av maskinvarekretser når de ikke er i bruk. Spesifikt er det gjennomsnittlige effektnivå av sensormodulen redusert til 23 % ved hjelp av data-analyse, og deretter til mindre enn 3 % ved de ytterligere forbedringene. Disse funnene er så påfallende at de kan være relevante også for sensormoduler med andre konfigurasjoner eller oppgaver.

I tillegg til disse kvantitative resultatene er også utformingen av prototypen som ble bygget for å finne disse resultatene beskrevet.

Contents

Chapter 1

Introduction

1.1 Background

This report describes the development and examination of a prototype for a wireless sensor module for offshore wind-turbines. The module is part of a research project called WiVind, and a more elaborate explanation of the WiVind project can be found in chapter 2. For reasons discussed in the same chapter, the work done here is not an integrated part of the WiVind project, but is instead auxiliary to it. The results in this report are expected to be of interest to the WiVind project, but is not part of the project plan.

The work which is done in this project is related to the energy consumption of such a wireless sensor module. The module is to be surface-mounted on wind-turbine blades and problems related to the mechanical and aerodynamic properties of the module are outside this project's scope. A sensor module prototype is developed to evaluate the energy consumption.

In technical terms, the prototype uses a triple-axis acceleration sensor, a 32-bit microcontroller and a sub-GHz digital radio as its hardware components. This provides all the parts needed to make the necessary evaluations; a data source (sensor), a control unit (microcontroller) and wireless communication (radio). The setup is shown in figure 1.1.

It is expected that initially, the most significant energy savings can be made by reducing the amount of data sent from the module to its host system, as wireless radio-links often consume significant amounts of power. Such a reduction will be attempted by utilizing signal-processing equipment within the sensor module which will process the sensor data and only communicate with the host system on the condition that the process results are important to the host system. The exact nature of that process, and what results that may be deemed important is part of the WiVind project.

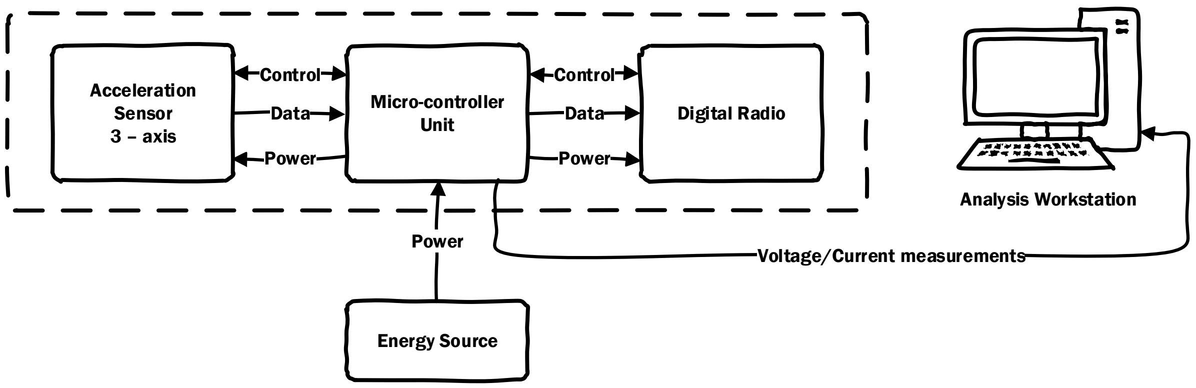


Figure 1.1: *A simple overview of the project work. Details regarding the energy source is not part of the master thesis.*

1.2 Problem Formulation

The problem is that the sensor module draws too much power from its energy source and thereby the system runs too quickly out of power to be a practical solution.

Rather than achieving incremental gains in energy-efficiency through the use of hardware with slightly better performance statistics, major gains are sought by principally modifying the operation of the sensor module.

1.3 Literature Survey

The primary foundation for the work done in this master thesis was an explicit interest from the WiVind project owners to commit research on this topic. Literature was explored to understand similar work and underpinning technologies. Most research on low energy in wireless networks examine the efficiency of the network standard or its radio frequency properties, which is not the focus of this report. The idea that low-energy performance can be achieved by using local processing instead of transmitting data is not new[5], but ideas that depend on using FPGAs to do the processing work will quickly end up in the wrong price range in terms of unit cost.

1.4 Objectives

The main objectives of this Master's thesis are:

1. Design and implement a sensor prototype for energy analysis.

2. Evaluate the impact of controlling the power-consumption of the components.

3. Evaluate the energy consumption of different *transmission schemes*.

1.5 Limitations

Due to the limitation of time and resources that is imposed on the writing of a master's thesis, the project must constrain its scope.

Therefore it will not compare different hardware solutions, even though the hardware properties is deemed to be significant to energy consumption. But as this project aims to explore order-of-magnitude energy gains the results are expected to be of interest despite the undesired shortcoming in hardware setup cross-referencing.

1.6 Approach

The design and implementation of a wireless sensor module must be done first as this creates the test-system or prototype for the rest of the project. This will consist of identifying and acquiring suitable, existing, hardware and connecting these such that the system becomes operative and can communicate with a host system.

A quantitative comparison of the energy consumed in a local-processing scheme and a raw-data scheme will be done by measuring the energy consumption in the system over time. A local-processing scheme will transmit only the result of the processed sensor data, but it will still make a transmission for every data set. To evaluate the scheme where communication between the sensor module and host system is done conditionally, a no-transmit scheme is also used. Chapter 6 elaborates the use of these *transmission schemes* and their purpose.

Controlling component-wise power-consumption will be achieved by developing a software framework for the system that can decide which necessary activity level of each component at a given time, so that components which have no work may be put into a low-power mode.

1.7 Structure of the Report

This report is structured so that chapter 2 details the WiVind project as a background and this project's relation to WiVind. Chapter 3 then presents the theoretical foundation of the project, aiming to explain both the environment in which the sensor most operate and the technologies which makes the solution possible.

The parts comprising the prototype, the tools used to develop it and the methods applied to evaluate it can be found in chapter 4, details regarding the design of the system and particularly its sofware is presented in chapter 5.

While the setup of system tests are shown and discussed in chapter 6. Chapter 7 contains the quantitative results of these tests and also contains an evaluation of the results. This evaluation discusses the test results specifically, while chapter 8 discusses the project as a whole before chapter 9 concludes the project work and presents some suggestions on further work. The appendices trailing the report contain additional information prototype hardware and software as well as a complete set of test results.

Chapter 2

Background

2.1 The Wivind Project

The practical and motivational background for this master thesis is a development project called WiVind. The goal of WiVind is to develop technology and products needed to conduct system monitoring of offshore wind-turbines. Particularly, it aims to develop a mathematical model of the turbine blades and use MEMS devices[1] to monitor the condition of the blades. This is intended to automate the process of identifying wind-turbine blades that are in need of maintenance. It is a joint project between Kongsberg Maritime and SINTEF, with Sensonor AS providing the MEMS devices. The MEMS (sensor) device will need to be equipped with adequate protec-

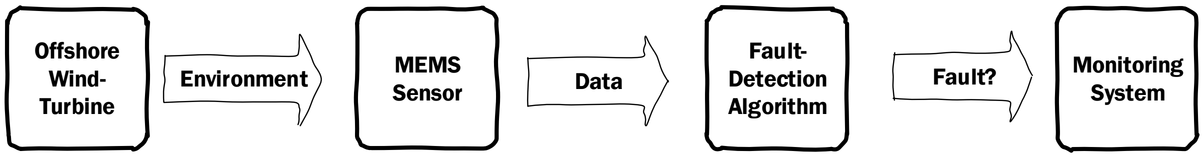


Figure 2.1: *The WiVind project at a highly conceptual level. A sensor gathers data about the surrounding environment and feeds the data into an algorithm. The algorithm allows a monitoring system to know whether the blade is faulty or not.*

tion from the environment, power supply, a control system and a digital radio to transmit data from the sensor to the monitoring system for the rest of the wind-turbine.

[1]A MEMS device is a miniaturized measuring device, with physical dimensions down to a few hundred micrometers.

The power supply is expected to include hardware for energy harvesting, as the system lifetime is planned to be several years.

The project is comprised of primarily two parts: The first part is to develop a turbine blade model which describes the blade condition. This forms the basis for a fault-detection algorithm, capable of automatically detecting structural faults in the blade. The other part is the development of a small sensor module which can provide the measurement input to the model. Figure 2.1 shows the conceptual theory of operation for the project goal. As a final commercial product, the sensor module must be easy to install, be able endure tough weather conditions, have a competitive price and reliably measure one or more physical phenomena as a basis for blade condition analysis. It must also have a lifespan of several years in which very little or no maintenance at all of the sensor is needed.

For the scope of this report, most of the practical considerations regarding ruggedness, design and manufacturing cost forms the backdrop of the report, but are not immediately relevant. Importantly, the exact details of what to measure, and how these measurements are to be analysed remains to be decided. The same counts for many of the practical aspects of the WiVind project as it is still in an early phase.

2.2 This project

The primary idea behind this master thesis is that the energy cost of data computation is lower than the energy cost of transmission.

The interpretation of that statement is that over time, total energy consumption is reduced by utilizing signal processing to reduce the amount of data it is necessary to transmit.

The WiVind project specifies that a suitable sensor module must be developed and a fault-detecting algorithm to be found. This project extends that specification by implementing parts of or all of said algorithm in the sensor module. This requires the sensor module to have more powerful computation circuits but the module can also expect to spend less energy on data transmission.

The work here does not aim to present a complete sensor module prototype, nor to develop the fault-detection algorithm. These are tasks of the WiVind project. The scope of this project is

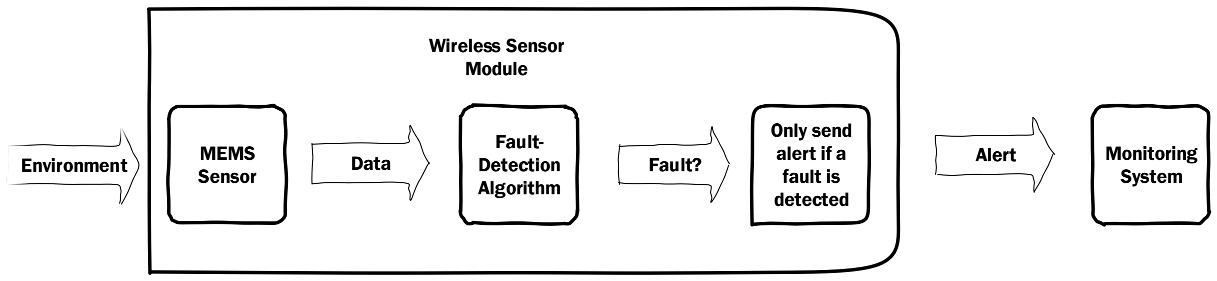


Figure 2.2: *This project focuses on the energy consumption of the wireless sensor module, with the biggest energy-saving coming from limiting the data sent to the monitoring system.*

to use the WiVind project not as exact boundaries but as guidelines inside which to explore the energy-saving potential of smart sensors and energy-driven system design.

An important principal difference between these two projects is that the while WiVind project simply aims to solve the problem in *some* way, this project is explicitly based on the idea that part of the solution is to run the fault-detection algorithm locally in the wireless sensor module. Figure 2.2 shows the setup of this project.

This project operates with much a closer deadline than the WiVind project and it is not possible to wait for exact specifications from WiVind.

Therefore, specifications and properties which would have been set by the WiVind project but are not yet available are therefore approximated such that any findings in this report will be relevant to the WiVind project. One such example is the properties of the actual sensor. The type of sensor to use, its power consumption, it's precision and the required quality of the digital converter are properties that are all important to the WiVind project but are not fully known. The fault-detection algorithm itself is also incomplete, and for test purposes this must be replaced by some other computational work.

The principles regarding local data processing and an energy-driven design should be of interest to readers not involved in the WiVind project.

It is expected that this approach allows this master thesis to be completed in time while still providing results and insight that is of relevance to the WiVind project.

Chapter 3

Theory

The work of this thesis is related to the design and construction of a low-energy embedded sensor system for offshore wind-turbines. It makes use of pre-existing parts and technology and integrates them into a functional concept. This chapter presents the environment in which the sensor module will operate, the principal challenges of the task it will do and the technology making such a sensor possible.

3.1 Offshore Wind-turbines

3.1.1 As an engineering environment

In order to present the environment in which the sensor module will operate, some background on offshore wind-turbines is appropriate. Offshore Wind-Turbines (*OWT*s) is an evolution of traditional land-based wind-turbines. They are typically bigger, both in physical size and in the power generated. A wind-turbine's size is usually a referral to the theoretical maximum amount of electrical power it can generate, which for an offshore wind-turbine is around 6 MW. Sizes up to 8 MW also exist and some as large as 10 MW are being planned.

The length of the blades ranges from 40 to 60 meters and the height from the sea surface to the top of the mast (the hub height) is typically around 20 meters more than the blade length.

The rationale for increasing turbine sizes is that the installation costs per offshore wind-turbine is quite high and does not grow as fast as turbine size. Therefore fewer and larger turbines can produce the same power output as smaller and more numerous ones at a lower total installation cost. In addition, wind conditions are more stable further up from the sea surface, which results in more stable power production.

The rotation speed of the turbine is on the order of 6-12 rotations per minute, and the area swept by blades of 60 meters length is over 11310 square meters, which is roughly one and a half times as large as a conventional football field. Even though the blades would be held at a stand-still for a manual inspection, such a task would still imply traversing a large area to assure a complete inspection. For the largest wind-turbines today, the swept area is over three and a half times that of such a football field.

Offshore wind-turbine technology is mainly based on the technology for land-based turbines which has been adapted for offshore use. This has allowed installations in shallow waters up to a depth of 30 meters, where the turbine is fixed to the bottom by driving the mast 30 meters down into the seabed. For depths beyond 30 meters, more refined approaches will need to be developed and are expected to allowed bottom-fixed installations for depths up to 60 meters. For greater depths, floating wind-turbines might be used.

This represents the physical environment in which a sensor module will exist. It necessitates a module which is mechanically robust enough to endure the weather (The wind-turbine owner will want to place the turbine where there is as much wind as possible) and the g-forces exerted on it by the rotation of the rotor and wind forces.

Maintenance work on the sensor must be kept to a minimum after turbine installation, meaning the sensor module should last as long the blade. This has implications for the reliability of both hardware and software, as well as duration of the energy source. The fact that it will be mounted on the surface of a wind-turbine blade whose aerodynamic properties is vital to the performance of the entire turbine leaves restrictions on the weight, size and form factor of the sensor module.

The sensor module may also be installed on onshore turbines, whose principle of operation is identical but with significantly easier access due to their placement on dry ground.

Thus, placing a sensor on an offshore wind-turbine blade requires it to endure harsh weather conditions, it must be virtually maintenance-free, have low weight and a low aerodynamic profile.

3.1.2 Fault-detection algorithm

The primary principle behind the Wivind project is that instead of performing manned, visual inspections of the windmill blades, instruments can be used to measure specific blade properties which in turn can be used to deduce information about the state of the blade.

Regardless of which properties are measured and what kind of computing power is needed to find this information, this necessitates that there exists an algorithm which can take some instrumentation data as input and give information about the blade state as its output.

Further, measuring these properties must be practically and industrially possible, meaning that the resulting sensor must adhere to the summarizing requirements of the previous section. The computing power need to run the algorithm must be reasonable (That is, a scientific supercomputer must not be needed).

These are the necessary conditions which must be satisfied for the Wivind project to succeed.

3.1.3 Mathematical Blade Model

The development of the fault-detection algorithm will be based on the understanding of a wind-turbine and its blades as mechanical systems.

A mathematical model must be developed which connects the flaws and damages in the physical structures to measurable properties. It is, for example, expected that such damage will cause changes in how a blade vibrates, and such vibrations will be possible to measure. The model is then used to predict what kind of vibrations that will be caused by structural damage or to detect deviance from a "healthy" vibration.

If the model can be used in reverse order, so that knowledge about how the blade vibrates can be used to induce knowledge about structure damage, one has the basis for a fault-detection algorithm.

3.1.4 Summary

Successful detection of a fault in the blade depends on a developing a mathematical model of the blades, which in turn may be used to develop a fault-detection algorithm. The sensor mod-

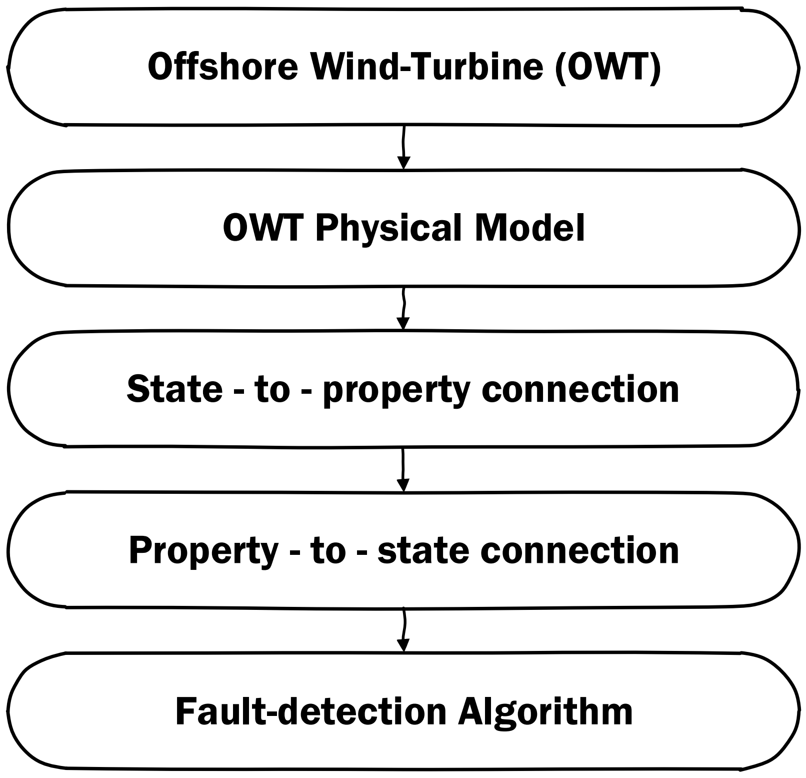

Offshore Wind-Turbine (OWT)
OWT Physical Model
State - to - property connection
Property - to - state connection
Fault-detection Algorithm

Figure 3.1: *The process from analysing the wind-turbine as a mechanical construction to the creation of an algorithm. Once a model exists to describe its system state, this state will predict the value of some of the blade's properties. Successfully reversing this connection allows the system's (the blade) state to be predicted by knowing the value of these properties. This information can then be condensed into an algorithm which detects either faulty system states or deviances from a "healthy" system state.*

ule must provide the input data for that algorithm and must be designed so that it endures the environment in which it will operate. Figure 3.1 shows the process from a physical structure to a monitoring algorithm.

3.2 Embedded Systems as Measuring Devices

Embedded systems technology is prerequisite to the WiVind project. The term "embedded systems" usually refers to a small computer system dedicated to a specific task.

Three technologies has been especially important to today's application of embedded systems. First is the emergence of the integrated circuit, allowing complex control logic to take up only a few square millimeters of space. Second is the development of digital wireless communication, potentially allowing such control systems to communicate with a larger, central control system without installing any additional cables. Finally, the commercial breakthrough of MEMS devices has allowed an embedded system to monitor one or more of a system state's variable with equipment that is cheap and small. The application, functionality and consequence of these three technologies will each be reviewed in order to document how a small sensor module (such as the one for the WiVind project) is able to operate and why such modules are not commonplace already.

3.2.1 Integrated Circuits

The integrated circuit, *IC*, refers to electronic circuits where multiple electronic components are integrated into the same physical component, with the integrated transistor the most significant part. This miniaturization of electronics allowed complex logic to be embedded in devices only a few millimeters in size, reducing the cost, heat dissipation and volume of computing power. Essentially, it made it possible to build computers of sufficient computing power so small they could be embedded into other devices.

3.2.2 Processors

The terminology by which processors are referred vary. In conventional desktop computers and laptops, they are referred to as CPUs, Central Processing Units. In smartphones the processor is commonly integrated into an even greater piece of circuitry named a System-on-Chip. For this report it is only relevant to focus on processors as they are used in embedded systems, where they are referred to as *Micro-Controller Units*, MCUs. In an MCU the processor is the primary circuit, with additional circuitry providing auxiliary services to the processor. Examples of such auxiliary services is communicating externally with other circuits, converting analog signals to digital signals and vice versa, storing data and digital signal processing.

Until recent times, such MCUs has been 8-bit controllers such as the 68HC family of MCUs, introduced by Motorola and now produced by Freescale. These are 8-bit systems operating at a few MHz. This is sufficient for simple logical and mathematical operations, but working with 16-

bit and 32-bit values significantly increases the clock cycles required for each operation, which is a limited resource. Such an MCU can successfully do simply monitoring jobs but performing digital signal processing is typically beyond what can be expected.

While such MCUs are still valuable for simple tasks, far more powerful MCUs are now available. These feature 32-bit word length, clock frequencies in the tens (and even hundreds) of MHzs and complex circuitry dedicated to digital signal processing has found its way from desktop computer graphic cards to these MCUs. Computer graphics is computationally expensive work, which in principle is similar to that of digital signal processing. Key features of such circuitry is saturated overflow, Multiply-and-Accumulate and Single-Instruction-Multiple-Data.

Saturated overflow operates as a graceful degradation of data computation. Should the result of a computation be larger than the targeted data type can store, conventional computer logic would store a value far smaller than the actual result. In such cases, saturated overflow instead stores the highest values the data type can represent.
Multiply-and-Accumulate – MAC – refers to a process in which two vectors of equal length are multiplied element-wise before all elements of the resulting vector is summarized into a single scalar value. This operation is very common in DSP and dedicated hardware can perform the operation in just a few clock cycles.
Single-Instruction-Multiple-Data – SIMD – instructions perform an instruction on vectors rather than single values.

For example, calculating the energy consumption of a component can be achieved by measuring both current and voltage levels through the component, which a controller will receive as binary data values corresponding to the voltage level read by an ADC, which stores the readings in a vector. SIMD instructions are valuable for scaling these binary ADC signals to values in amperes and volts. Then a MAC operation on these two vectors will find find the energy consumed in the given period. Saturated overflow prevents exceptionally high or low power transients as being interpreted as their inverses (without it, power surges would appear as power drops and vice versa) without increasing computational complexity.

The increased data power of the modern MCU – both in word-length and clock rate – and the possibility of DSP circuitry completely changes the tasks that can be solved by an embedded system. It enables the possibility of running more demanding algorithms, such as the fault-detection algorithm, locally on the MCU rather than running it on a more centralized computer with raw data provided over a wireless link.

The modern processor is a key part of any embedded system and makes possible a wide range of modern life's devices. This is primarily due to its ability to execute virtually any task, delivered in a compact form. Because this means that the same approach (writing software) and knowledge can be applied to a wide range of problems, the same hardware products are used over and over again, driving the prices down. In many cases this makes them so cheap that they are preferable to dedicated solutions, as such solutions require dedicated design and manufacturing resources. This is turn accelerates the use of processors, as tools and knowledge already aquired by a business or society can be applied to new tasks, reducing the development costs of each new product.

Consequently, the concept of the processor and the software utilizing it now dominates the digital world and thus the modern world.

3.2.3 MEMS devices

Another, more recent type of integrated circuit is the MEMS sensor. MEMS is a short-hand for Micro-Electro-Mechanical-System and it integrates an entire measuring instrument into a single component. The first MEMS devices appeared as early as the 1980s[1] and the 1990s saw the full-scale development of the market. At this point the market was primarily the industrial sector, producing such devices as manifold absolute-pressure sensors and blood-pressure sensors. The successful commercialization of the smart-phone by Apple's iPhone in 2007 opened up a new market for MEMS devices as accelerometers found their use in these smart-phones. In many situations the alternative has historically been far larger devices and although practical at an industrial production facility, large sizes prohibits embedding the sensor in a product. The MEMS allows the same measurements to be made with a device that is a few square millimeters big, weighs a few grams and can be produced for as little as a few dollars per piece.

MEMS technology allows accurate measurements to be made with devices that are physically small and with low voltage and current levels. Today's global mass markets for such products has driven the price down to a few dollars for a simple implementation.

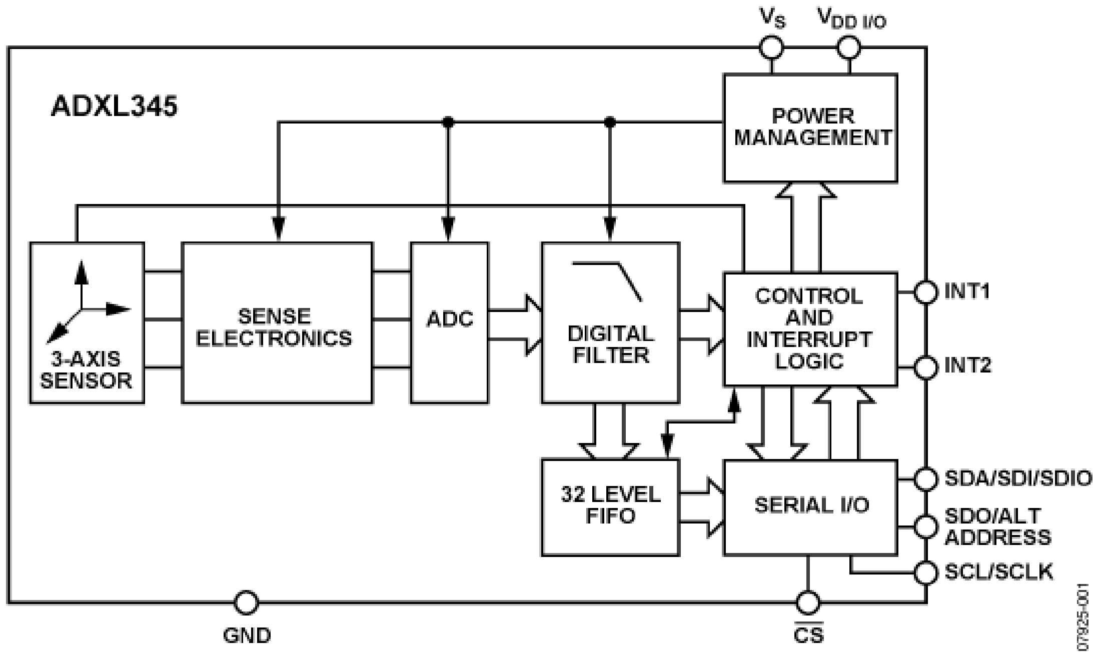


Figure 3.2: *A block diagram of a MEMS sensor built into an IC. The module as a whole contains a MEMS sensor, analog-digital conversion, a FIFO buffer for storing data and an I/O controller for communicating digitally with other ICs. Picture: Analog Devices*

3.2.4 Radio Communication

All wireless transmission of information makes use of one or more radio frequencies (RF). Different frequencies have different advantages and disadvantages in terms of range and energy-efficiency for different conditions. For a digital link, the most useful interpretation of energy-efficiency is the amount of energy it takes to transmit one bit of data in such a way that it can be correctly read by the receiver under the expected operating conditions.

Most RF equipment will transmit not only at its designated frequency but also at its neighbouring frequencies. This range of frequencies is a *band*. Referrals to a sending frequency is usually a referral to this band.

Naturally, the sender and receiver must operate at the same frequencies in order to function and hardware must be chosen that supports these frequencies. Equipment that is not intended to directly communicate but sending in the same frequency band at the same time will interfere with each other. Further, radio frequency signals interfere with equipment such as flight radars and medical equipment. Because of this, the use of which frequencies that are available to use for which purpose and what kind of signal strengths that are allowed is subject to regulation by the authorities. In addition, some areas may impose additional restrictions on RF use, where the most common example today are the airlines' mandatory shut-down of electronic equipment during an airplane's take-off and landing. The chosen frequency must therefore adhere to regulations, have support in available hardware and preferably be one which experiences a low amount of interference.

The GSM technology used by cellphones around the world as its own dedicated frequency bands which are only allowed to use for this purpose. Most other types of day-to-day solutions take place in the ISM bands, ISM being shorthand for Industrial Scientific Medical. These bands are subjects to some power limits but utilizing these bands is open for everyone. This means that interference is a problem which each user must find a way to resolve, the regulations concerning the ISM band contains no rules on interference. Most networking standards which operate in these bands are designed to manage interference in some way. One major issue is that exactly which frequencies that are part of the ISM band varies from country to country.

The most common frequency band is that around 2.4 GHz, which is used for WiFi[1].

[1]The most common kind of wireless network connection for phones and laptops today

In a commercial perspective, using the frequency band around 2.4 GHz is highly appealing as this is allowed to be used on a global basis, allowing one single product to be sold in all markets. The company which provided the radio hardware for this project recommended the use of sub-GHz equipment on the grounds that this would have better energy-to-bit ratios than the 2.4 GHz. Using sub-GHz frequencies will complicate the sensor module as a commercial product as different versions will have to be made and maintained, but as this project is a master thesis at a university it seems acceptable to give priority to the technical aspects on the choice of frequencies rather than the commercial ones.

The ITU-R is the institution which manages the available radio frequency spectrums worldwide. ITU-R, the International Telecommunications Union – Radiocommunication Sector is a sector of the ITU which is an agency of the United Nations. A summary of unlicensed ISM frequencies can be found in table 3.1[4]. Note that the 868 frequency is unlicensed on the Electronic Communications Committee of Europe [2], and is as such not part of the ISM bands itself.

Frequency Range	Bandwidth	Center Frequency	Availability
6.765 MHz – 6.795 MHz	30 kHz	6.780 MHz	Subject to local acceptance
3.553 MHz – 13.567 MHz	14 kHz	13.560 MHz	Worldwide
26.957 MHz – 27.283 MHz	326 kHz	27.120 MHz	Worldwide
40.660 MHz – 40.700 MHz	40 kHz	40.680 MHz	Worldwide
433.050 MHz – 434.790 MHz	1.74 MHz	433.920 MHz	Region 1 only and subject to local acceptance
863 Mhz – 870 MHz	7 Mhz	866.5 Mhz	Europe only. (*As regulated by the ECC in SRD860*)
902.000 MHz – 928.000 MHz	26 MHz	915.000 MHz	Region 2 only
2.400 GHz – 2.500 GHz	100 MHz	2.450 GHz	Worldwide
5.725 GHz – 5.875 GHz	150 MHz	5.800 GHz	Worldwide
24.000 GHz – 24.250 GHz	250 MHz	24.125 GHz	Worldwide
61.000 GHz – 61.500 GHz	500 MHz	61.250 GHz	Subject to local acceptance
122.000 GHz – 123.000 GHz	1 GHz	122.500 GHz	Subject to local acceptance
244.000 GHz – 246.000 GHz	2 GHz	245.000 GHz	Subject to local acceptance

Table 3.1: *The available, unlicensed, frequencies for radio communication as specified by the ITU-R and ECC.*

3.2.5 Energy consumption

Power is lost (dissipated into heat) in a digital integrated circuit in two ways. There is a static power consumption present as long as the circuit has a voltage supplied (effectively being 'on'), called a *leakage*. The other is a dynamic power consumption which takes place whenever a transistor switches state and is roughly proportional to the frequency of the circuit. The ratio between these two factors vary between technologies and designs, but they are both present and both can to some extent be managed.

Because dynamic power is proportional to switching frequency it is also proportional to the work done by the circuit. Given a fixed voltage and and a processor that is stopped between task executions, a processor that completes a task quickly at a high frequency incur the same active power loss as that of a processor that executes a task slowly at a low frequency. The static loss however, will be independent of the frequency and can only be reduced by shutting the circuit down between tasks. If shutting the circuit down is unfeasible, the low-frequency task execution may be more energy-efficient, doing more work per joule.

If the task load of a processor is periodic and deterministic in task size, such that the amount of work necessary per task period never changes and is known at design-time, the entire system design can be adjusted such that voltage level and processor frequency are at optimal levels. This requires the system designer to fine-tune these parameters for the chosen processor and a hardware change will mandate a revision of these parameters. This approach couples purpose and implementation tightly, but allows the theoretical optimal energy consumption and actual energy consumption to be closely matched. Further, it is not necessary for the processor to be neither stopped nor shut down. It exists in one constant state of operation.

For a more dynamic approach, a more complex set of processor power states is needed. If the processor can be powered down and powered up with ease, a high-frequency processor can process its tasks long before the task deadline, and go into a power-saving state when idle. This is highly suitable when the scheduling of tasks is indeterministic, for example when the processor must respond to aperiodic interrupts. The processor still has an upper limit on task processing capability and can still be overloaded by work, but by executing tasks quickly, an expected run of events (executing all periodic tasks) can be done with time to spare, which means that handling a limited amount of aperiodic events becomes trivial, both to the hardware and to the system designer. For a complex real-time system which must handle multiple deadlines, the problem of deciding whether the system can achieve deterministic behaviour or not remains unsolved.

The impact of the power switching is that processor hardware design and the set of tasks can be loosely coupled while still achieving some matching between theoretical optimal energy consumption and actual energy consumption. Essentially, for a low-energy system, it increases hardware complexity to simplify system design.

3.2.6 Energy Harvesting

Energy harvesting refers to the idea of producing (electrical) energy for a small system by using available energy in the environment. Examples are small coin-sized solar panels and piezo-electric devices harnessing mechanical vibrations. The WiVind project expects that some form of energy harvesting will be possible and necessary to reach the target lifetime of the system. The presence of an energy-producing unit in the system, rather than just working with a drain-only energy supply such as a battery, changes some aspects as to what the most efficient energy strategy is. If the harvesting system can cover the normal operation and more, there will be spare energy available for doing extra work such as increasing the data communicated with the host system. However, the scope of this project is to investigate how to solve the same task with less energy, and the entire energy harvesting issue is outside the project scope. The sensor control system developed here should be flexible enough to be adapted to the presence of an energy harvester.

3.2.7 Summary

Although the operating principles behind the technologies presented here has existed for some time, recent years has seen the emergence of these solutions as commercial, off-the-shelf components, allowing them to be easily incorporated into commercial products. Much of the development of MEMS devices, energy-efficient processors and radios is currently driven by the personal computing market, with key product categories such as smart-phones and tablets making up most of the demand. The designer of an embedded system today has access to a multitude of technologies and products that are ideal to the development of the WiVind sensor module, whereas only a decade ago the project would be infeasible.

Chapter 4

Parts, Tools and Methods

This chapter provides detail on the equipment used for the project. Section 4.1 lists the hardware parts used to develop the project and shows their conceptual interconnection. The hardware and software tools used throughout the project is described in section 4.2. This includes everything from software development to energy analysis and documentation. Finally, section 4.3 indicates the methods used to measure energy and the inter-connection of all the tools used.

4.1 Prototype Parts

To attain a rapid prototyping progress pre-existing hardware kits were used, mainly in the form of manufacturer development and evaluation kits. Table 4.1 shows a list of these kits and table 4.2 provides technical details on the key hardware components such as the micro-controller units. Their interconnection is shown in figure 4.1, displaying both the parts and the hardware used for development.

4.1.1 Module MCU

The MCU used in the sensor module is a Wonder Gecko chip from Silicon Labs. The exact technical specification is *EFM32WG990F256-BGA112* and it is a 32-bit ARM Cortex-M4f controller with a standard clock speed of 48 MHz. The most significant features with regard to this project is its ability to quickly change power-state, and its DSP capabilities. As explained in 3.2.2, convenient power-switching allows low-energy systems to be designed with reduced effort. The target

Prototype Parts List				
	Name	Producer	Serial No.	Revision
Sensor Module				
Controller	EFM32WG-STK3800	Silicon Labs	134601483	A00
Radio	CC1200EM-868-930	Texas Instruments	0000526	1.1.1
Sensor	ADXL345 Breakout Board	Sparkfun	N\A	1.4
Battery-Board	CC120X Breakout Board	Texas Instruments	N\A	1.3
Host				
Controller	SmartRF TRxEB	Texas Instruments	0x2133	1.7.0.4
Radio	CC1200EM-868-930	Texas Instruments	000054C	1.1.1

Table 4.1: *The parts used to create the prototype setup.*

of this project is to run a fault-detection algorithm locally in sensor module to save energy. This algorithm is expected to involve digital signal processing, and DSP support is therefore likely to reduce the execution time of said algorithm, which in turn may reduce energy consumption.

The MCU also has two separate USART controllers which has hardware support for SPI, such as automatic enabling/disabling of the *Chip Select* signal. This automates a lot of the register-level work needed to consistently operate a SPI bus.

The MCU has two different high-frequency clocks that can function as the main clock signal, an RC oscillator (HFRCO) and a crystal oscillator (HFXO). The HFXO can drive the system at 4-48 MHz and the HFRCO can drive frequencies from 1-28 MHz. When waking up from a sleep mode, the HFRCO will always be utilized as this clock is a low energy oscillator with a short wake-up time. It defaults to run at 14 MHz. Changing frequency and/or clock source must then be done by software routine calls. Because the MCU defaults to use the HFRCO at 14 MHz, changing this is only meaningful when it can save energy or there are deadlines to reach.

Some tests were done to inspect the impact of using the HFXO as the clock source when doing the DSP work (locally processing the sensor data). But as long as the MCU was hibernated when inactive, this made no difference, the processing was done at a faster rate and a higher power level, resulting in the same energy consumed. The MCU was therefore configured to always use the HFRCO at 14 MHz to simplify system design.

Primary Components List		
Component	Power consumption	Features
Module MCU Silicon Labs *EFM32WG990F256-BGA112*	Run mode: 180 uA/MHz Sleep mode: 45uA/MHz Deep Sleep: 0.9 uA Stop Mode: 0.6 uA Shut-off Mode: 20 nA	ARM Cortex-M4f 48 MHz (**set to 14 MHz**) DSP: Saturated overflow, MAC, SIMD, hardware 32-bit floating-point support. 2x USART controllers, hardware SPI support
Sensor Analog Devices *ADXL345*	40 – 145 uA, scales with measuring frequency. At Vs = 2.5V	3-axis accelerometer. 13-bit resolution, up to 16g range and 1600 Hz bandwidth. SPI interface, data output buffering
Radio Texas Instruments *CC1200*	45 mA at 14 dBm tx. 0.3 uA in sleep mode	868-, 915-, 920-, 950-MHz ISM/SRD Band Scalable output power up to 16 dBm 0-200 kbps data rate 128-byte transmit and receive-buffers
Host MCU Texas Instruments *MSP430F5438A*	N\A	16-Bit RISC Architecture, SPI

Table 4.2: *The most important hardware components of the prototype parts. The table indicates their specific part numbers and the parameters that are most relevant to the project work.*

4.1.2 Sensor

The *ADXL345* 3-axis MEMS accelerometer used as a sensor for this project is easily accessible by means of a digital interface. It can communicate via I2C and both 3- and 4-wire SPI, out of which the latter is the chosen interface for the project. It has a built-in 13-bit ADC whose data output is fed into a 32-level FIFO buffer, with each level holding data from all three axes. The buffer data is represented as 16-bit twos-complement values, with one value for each axis. The sensor operates exclusively as a slave device, using two interrupt channels as a means of actively notifying a master device of events. Exactly which events that are to trigger an interrupt is configurable via the SPI bus and for this project it was set up to trigger an interrupt to the MCU whenever the FIFO buffer was filled up to a certain level (a ”watermark”) with data. During normal operation, this increased the intervals between the MCU-sensor operations, allowing longer hibernation times for the MCU. Setting the watermark somewhat lower than the maximum buffer level allows the sensor interrupt to be treated as a low-priority interrupt. This imposes firm, rather than hard, real-time constraints on the MCU's reading of sensor data; an interrupt means there is limited time to act before sensor data is lost but with a sufficiently high SPI clock speed it does not require immediate attention. Please see table 4.3 for general sensor details and table A.1 in the appendix for details on the configuration registers setup.

Parameter	Value
Sampling Rate	50 Hz
Watermark	16 Samples
MCU Interval	320 ms
Range	±16g
Interrupt	On watermark reached

Table 4.3: *Sensor configurations*

4.1.3 Radio

The *CC1200* is a radio chip designed by Texas Instruments to handle sub-GHz digital two-way communications. It is operated by transmitting data to a TX FIFO buffer with a 128 byte length. The radio adds a minimal amount of frame header data: A small preamble, an (optional) sync word, an (also optional) address byte and one byte indicating frame length. This allows for very simple operations as it can be used without any communication or network protocol such as ZigBee or Bluetooth, using only the described header. Any networking protocol must be handled in software but this project operates on the basis of one transmitter (sensor module) and one receiver (host system) and as such there is no need for any addressing or routing data to be sent or maintained. Instead, a focus is kept on minimal header data to reduce energy consumption. The two devices are set up to operate on the 868 MHz band. The radio can be configured for different bit-rates and transmission power levels, see table 4.4 for general radio details and table A.2 in the appendix for details on the configuration registers setup.

Parameter	Value
Bit rate	1200 bps
TX amplitude (at 3.3v)	14 dbm
TX/RX frequency	868 MHz
Packet length	Dynamic
Interrupt	On TX complete or error

Table 4.4: *Radio configurations*

4.1.4 Host MCU

The emphasis of this project is the energy consumption of the sensor module. The host system will be able to draw power from a major power supply so its energy characteristics are of effectively no interest. Further, it's role in the project is simply to verify correct reception of transmitted data. Because of this, its details as a whole are of little interest and are omitted from the report, other than to say that it is an *MSP430F5438A* from Texas Instruments, reading data from a CC1200 radio, also from Texas Instruments. It's source code is included in the files submitted as part of the digital part of this report.

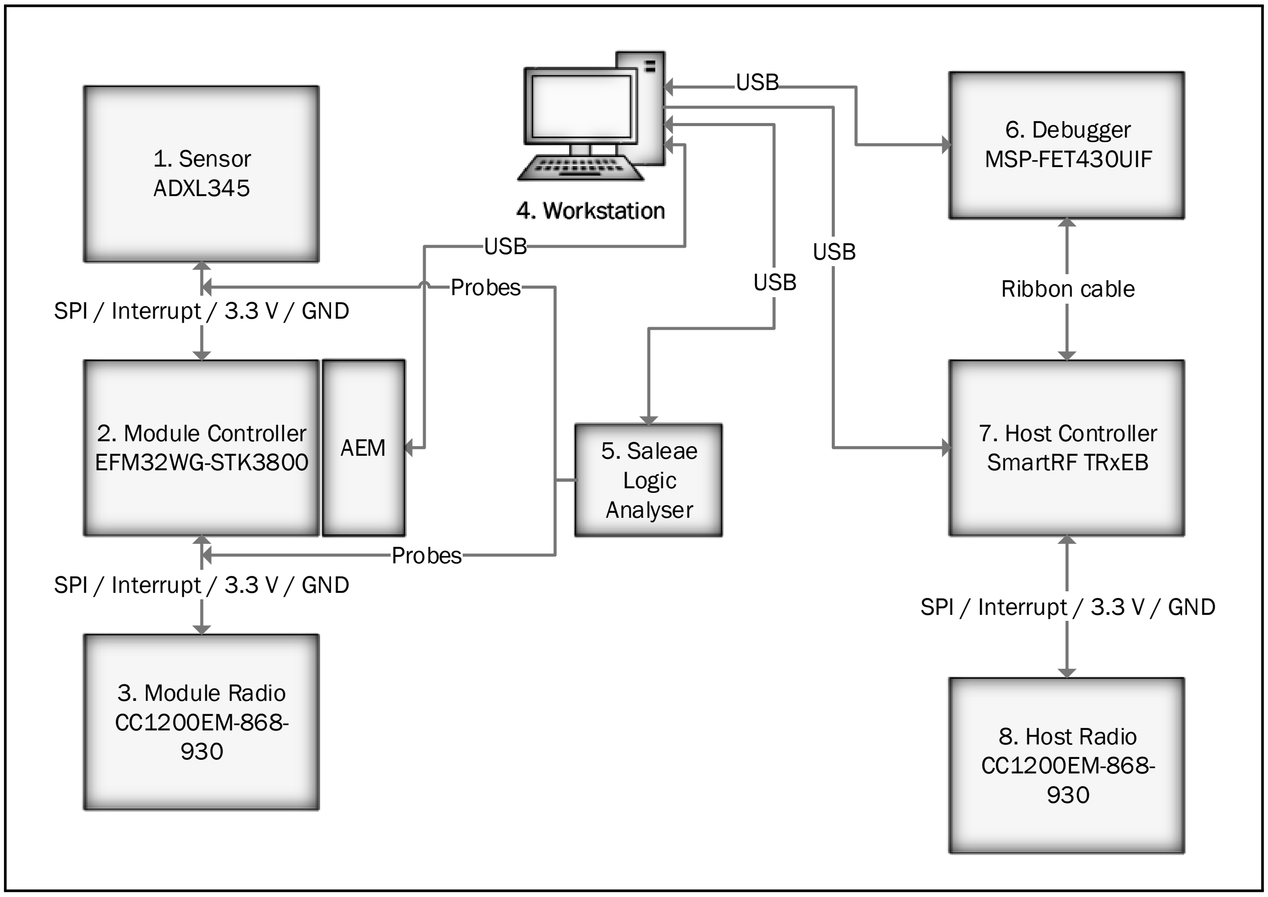


Figure 4.1: *A diagram of all the hardware involved in the test-setup. Energy measurements are done via the AEM hardware on the Module Controller (2) which are sent to the workstation (4) for storage and analysis.*

4.2 Development Tools

4.2.1 Hardware

MSP430 USB Debugging Interface

Interfacing the TRxEB host controller was done with the MSP430-FET430UIF debugger from Texas Instruments. This tool provided both debugging capability and allowed hex-files to be flashed to the MSP430 on the TRxEB.

Logic Analyser

An 8-channel Saleae Logic Analyser was used to debug the digital buses and interrupt signals of the system. With a sampling rate of 24 MHz it was more than fast enough for the bus clock rates

Development Tools list				
Hardware				
Name	Producer	Purpose	Version number	Comment
Advanced Energy Monitor	Silicon Labs	Energy measuring	N\A	AEM, part of Wonder Gecko Starter Kit
Saleae Logic Analyser	Saleae	Digital signals debugging	N\A	Combined hardware and software tool from Saleae
Software				
MSP430 Debugger	Texas Instruments.	Programming and debugging TrxEB	V1.4a	
energyAware Profiler	Silicon Labs	Display and store energy info	1.14	Software interface for AEM
Matlab 64-bit	Mathworks Inc.	Energy analysis	R2014a (8.3.0.532)	
Visio 2013	Microsoft	Software and project documentation	15.0.4420. 1017	
Simplicity IDE	Silicon Labs	Software development for Wonder Gecko	v2	
GNU ARM Toolchain	GNU	Toolchain for the module MCU	4.7.3 20130312	
IAR Embedded Workbench	IAR	Software development for TrxEB	5.60.7	Version number includes toolchain components
SmartRF Flash Programmer	Texas Instruments.	Flashing programs to TrxEB	1.12.7.0	

Table 4.5: *List of development tools.*

of this project.

AEM, Advanced Energy Monitor

The Wonder Gecko Starter Kit has an on-board unit for measuring energy consumption in the hardware, called an AEM. It measures and samples not only voltage level and current flow but also the value of the program counter of the MCU and can send this to a computer by means of a USB cable. It samples at 6250 samples per second and has a dynamic range spanning from 100 nA to 50 mA. The Starter Kit also provides output voltage pins which are supplied through the AEM, so that the power consumption of external components may be included in the measurements. See figure 4.2 for details.

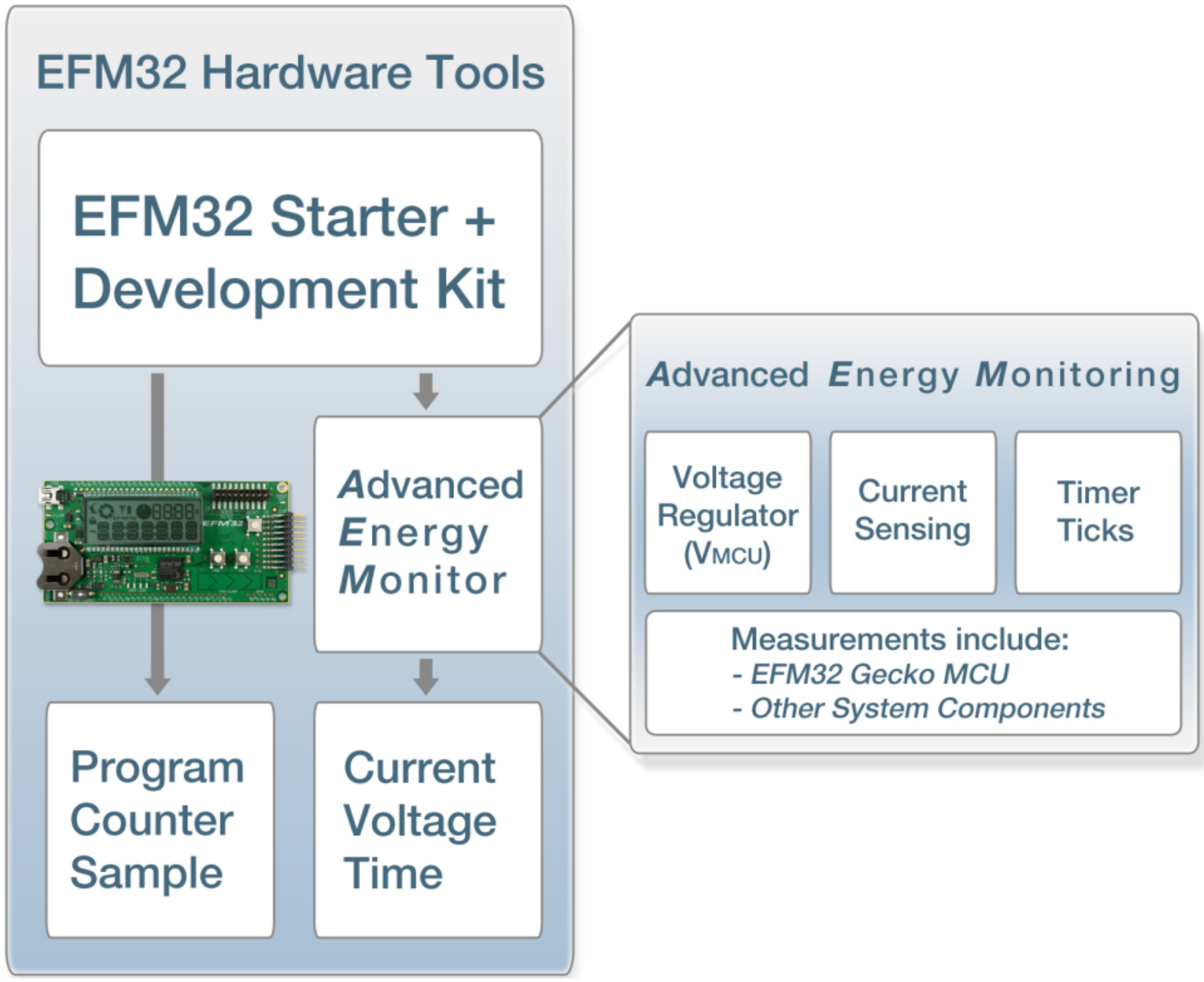


Figure 4.2: *The Advanced Energy Monitor. Picture: Silicon Labs.*

4.2.2 Energy Analysis

energeAware Profiler

The AEM communicates via USB with a program called energyAware Profiler (Profiler) from Silicon Labs. The primary function of the profiler is to show voltage level and current flow as a graph. Significant secondary features are based on the ability to load the object file of the executed program in to the Profiler. By referring the sampled values of the program counter to the object code, the power consumption of any single moment can be linked to the code being executed at that precise moment. By integrating this over time, the energy cost of the software running on the MCU can be decided. Silicon Labs refers to this as *energy debugging*, and allows the developer to search for pieces of code that consume extraordinary amounts of energy. However, note that this technique has some pitfalls. Any power drawn at a given point

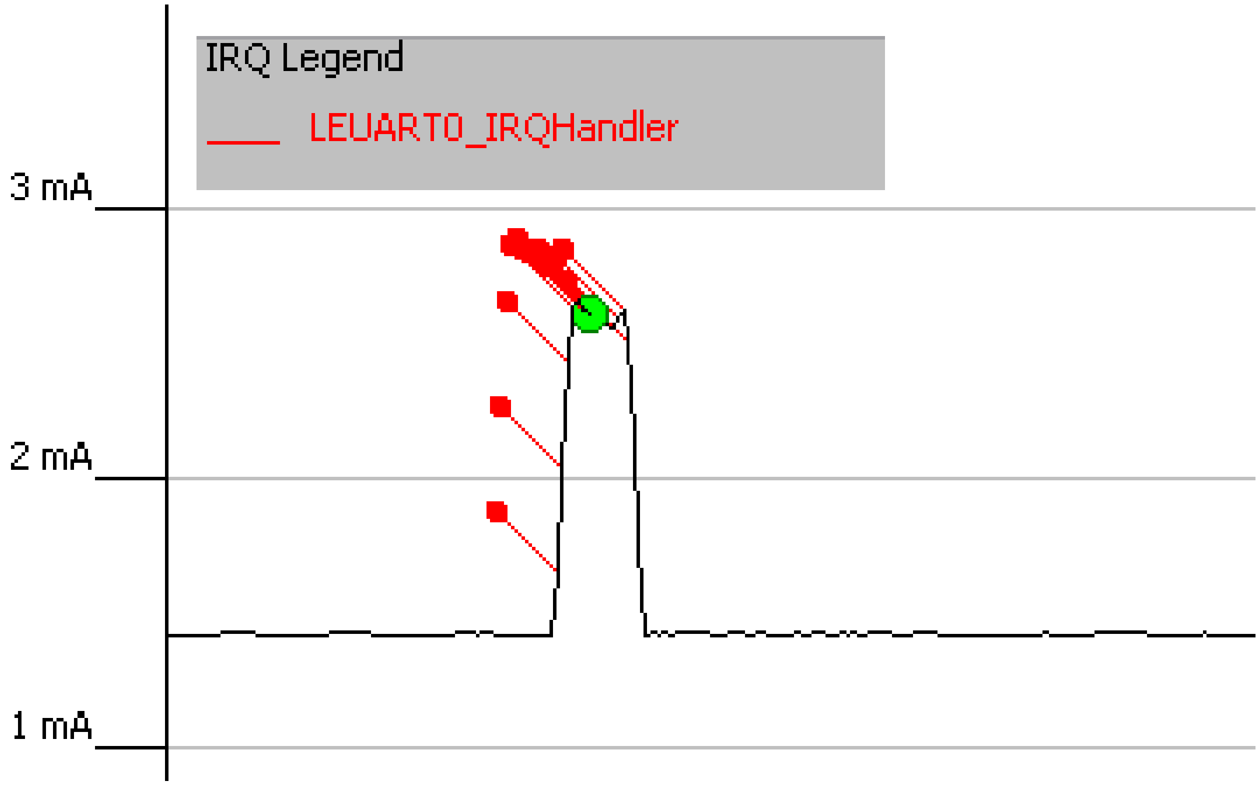


Figure 4.3: *Part of the energyAware Profiler interface, used for energy analysis. Picture: Silicon Labs.*

of time will be associated with the function executed at that point of time. The consequence of this is that a function which usually happens to be executed simultaneously with the radio being active will be associated with very high energy levels, as the power level of the radio is easily ten times that of the MCU. This will happen regardless of whether the function in question actually *is* associated with use of the radio or not.

The profiler also allows for the Interrupt Requests (IRQs) to be mapped on to the energy graph as small pins as shown in figure 4.3. A typical setup will be for a system to remain in a low-power state until an interrupt occurs, which makes the linking of interrupts and energy consumption a valuable feature. However, the exact placement of these pins appeared somewhat erroneous (bug confirmed by Silicon Labs). In addition, the drawing the graph displaying the current flow also seemed somewhat inconsistent and in some cases outright wrong.

The Profiler can output its results both directly to the screen but also to files. The files are comma-separated value files (*.csv-files) with a format as shown in list 4.1. Such files can then be re-opened in the Profiler or other programs at a later point. Because of the mentioned bugs in the Profiler program, analysis in a more stable, well-known program was of interest. Sampling for 5 minutes led to files on the order of hundreds of megabytes.

```
#Time (us) ; Current (mA) ; Voltage (V) ; PC ; IRQ ; Function Name ; File Name ; Line
    number
427729373 ; 45.9514 ; 3.02111 ; 6180 ; 0 ; event_fetch ; ../src/sys_ctrl/evt_sys.c ; 31
427729533 ; 45.8426 ; 3.02111 ; 6180 ; 0 ; event_fetch ; ../src/sys_ctrl/evt_sys.c ; 31
427729693 ; 45.6907 ; 3.02111 ; 6180 ; 0 ; event_fetch ; ../src/sys_ctrl/evt_sys.c ; 31
427729853 ; 45.6885 ; 3.02124 ; 1414 ; 15 ; INT_Enable ; /emlib/inc/em_int.h ; 97
427730013 ; 45.7221 ; 3.02124 ; 1414 ; 15 ; INT_Enable ; /emlib/inc/em_int.h ; 97
```

Listing 4.1: *Example of Profiler file output format. File paths are shortened for display convenience.*

Matlab

While the Profiler program was sufficient for simple, qualitative analysis, a more powerful program was necessary to do in-depth analysis and generation of better graphical representations of the data. Matlab offers a graphical user interface tool for importing data sets, but it had several problems. One was that the header line in the file is prepended by a 'comment sign' (see the first line of list 4.1.). This distorted the import tool's ability to automatically parse the header line for metadata. Far more serious was the fact that it took Matlab up to an hour to import a file containing five minutes of data samples, which made taking new samples very time-consuming. A small set of scripts was developed to import data from csv-files and store it in Matlab's own mat-files. This reduced the one-hour job to about 15 seconds. Further, the outputted mat-files were approximately 90 % smaller in size and far faster to import data from.

With this in place, Matlab could easily be used to analyse big data sets and to produce data plots to present the results of this report in a meaningful manner.

Matlab Scripts

Two sets of scripts were developed. The first set was made to easily import data sets from eAProfiler into Matlab and consists of four scripts, *convertdir.m*, *eAfileconvert.m*, *eAimport.m* and *eAmatsave.m. convertdir.m* is the top-most script, using the three other scripts to process the data gathered for the entire project into a far more convenient file format, occupying the computer's resources for a few minutes rather than a full day. It does this by invoking *eAfileconvert.m* on each file in a given directory, storing each of the mat-files outputted in a destination directory. *eAfileconvert.m* in turn invokes *eAimport.m* and *eAmatsave.m* to process each csv-file.

The second set automates the process of producing and storing appropriate data plots for this report. It too can work on entire directories, processing all mat-files it can find in a target directory. It outputs plots of the raw data it finds, to make easier the task of visually verifying that the data sets used are valid and plots of the data to be included in this report. While not as critical to the work progress as the first set, and subject to far more tweaking, these scripts still simplify both analysis work and report-writing.

4.2.3 Documentation

Microsoft Office Visio

Visio was used extensively to draw class diagrams, flow charts and other figures for the latter stages of software design and for documentation. Most of the design work was done with pencil and paper.

4.2.4 Software development

Simplicity Studio

Silicon Labs offers its own development suite for its products, called the *Simplicity Studio*. Simplicity Studio offers a complete development environment for working with the Silicon Labs hardware. It contains documentation, examples and the tool-chain needed for compilation, debugging and flash programming.

Simplicity Studio is based on the open-source IDE *Eclipse* and therefore allows many of plugins available for Eclipse. The only one used for this project was EGit, which integrates git functionality into Eclipse.

Source Version Control

For source version control, the open-source solution called *git* was used to store project code in as GitHub repository.

IAR Embedded Workbench

The code for the TRxEB, which was used as a host system controller, the Embedded Workbench from IAR was used. IAR does not offer student licenses and as such a free trial license was used which limits the size of the output hex-file. Because the host system was designed to do little else than verify the reception of sent messages, this was not a problem.

SmartRF Flash Programmer

The hex-files outputted by IAR Embedded Workbench were loaded onto the TRxEB by Texas Instruments' own SmartRF Flash Programmer and MSP-FET430UIF.

4.2.5 Vibration Jig

Kongsberg Maritime offered use of a vibration jig which was used to verify correct readings by the accelerometer.

4.3 Methods

The most important tests made in the project were the energy tests. As described earlier in this chapter, this was achieved by means of using the Advanced Energy Monitor of the Wonder Gecko Starter kit, allowing input voltage level, current flow and time to be measured and stored at a desktop computer. The sensor module was set up with a sensor sampling rate of 50 samples per second, and the MCU was set to collect 512 samples into a data set before handling this data set (either locally processing it or transmitting it directly). This leads to a data set period of 10.24 seconds, meaning that several minutes of testing was necessary to create a test sample of more than just a few periods. Each test was therefore set to last for 5 minutes in order to assure that the test sample would contain more than 30 periods while still keeping the test files at a manageable size.

Each test was set up to test a particular configuration in order to examine the impact of a parameter on energy consumption. The exact configurations that were tested and why they were chosen is the subject of chapter 6, Test Setup.

The energy tests presented in this report are the concluding tests made in this project. For the development process of the entire project, multiple other tests were done, some energy tests but also numerous tests with respect to sensor functionality, algorithm accuracy and software stability. To avoid unnecessary cluttering these tests are not part of the report, instead the interconnection of all the test- and development tools used in the project are shown in figure 4.4. The development process was done iteratively, with each successful step either integrating one more piece of software or hardware, and each failed step requiring a revision of that step. This iterative process does not show in the figure as it aims to describe the tools and not the work process.

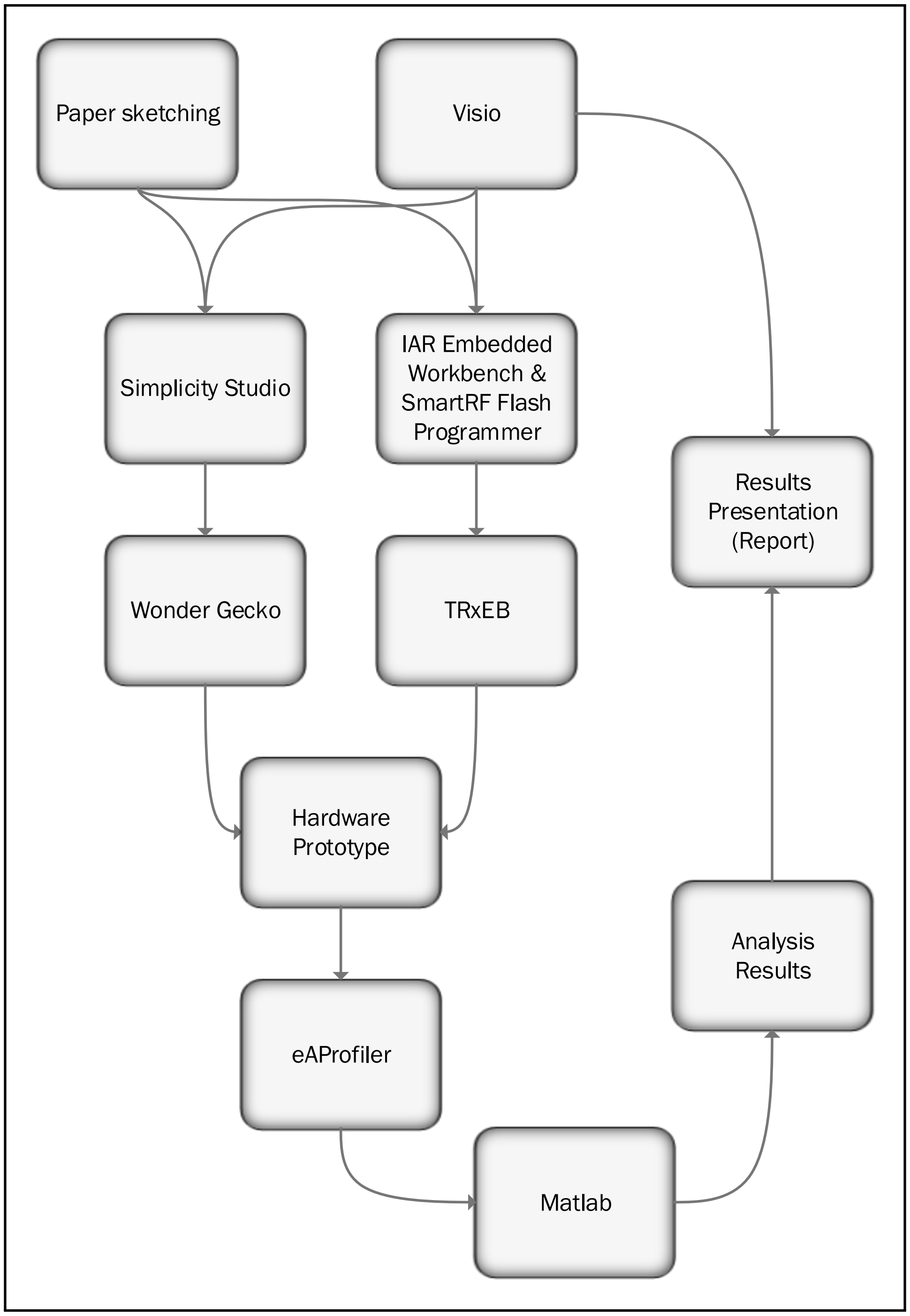


Figure 4.4: *The interconnection of the tools and parts used in test setup. The energy measurer, AEM, is embedded in the Wonder Gecko Kit.*

Chapter 5

System Design

When this projected was started in february 2014, no previous work had been done, neither by NTNU, SINTEF nor KM. Therefore the design and construction of a sensor module prototype, as a well as host system, was a necessary part of the thesis work.

This chapter explains the basic structure of this prototype and shows how the design and construction choices plays a key part in reducing the energy consumption of the sensor module. The hardware components and their specifications can be found in chapter 4, Parts, Tools and Methods. As the same chapter explains, off-the-shelf hardware was chosen to speed up the development process. Consequently, the remainder of all design choices must be implemented in software, and the emphasis of this chapter is therefore on software design and implementation.

5.1 Objectives

The complete prototype has a sensor module and a host system. These communicate via a wireless link. The sensor module produces data by means of a MEMS sensor and transmits data via the link. The host system receives the data sent via the link. Because it is the sensor module that has a limited energy supply, it is the design of this module's software that is emphasized in this report. Figure 5.1 shows a conceptual drawing of the two system units, the host system and the sensor module.

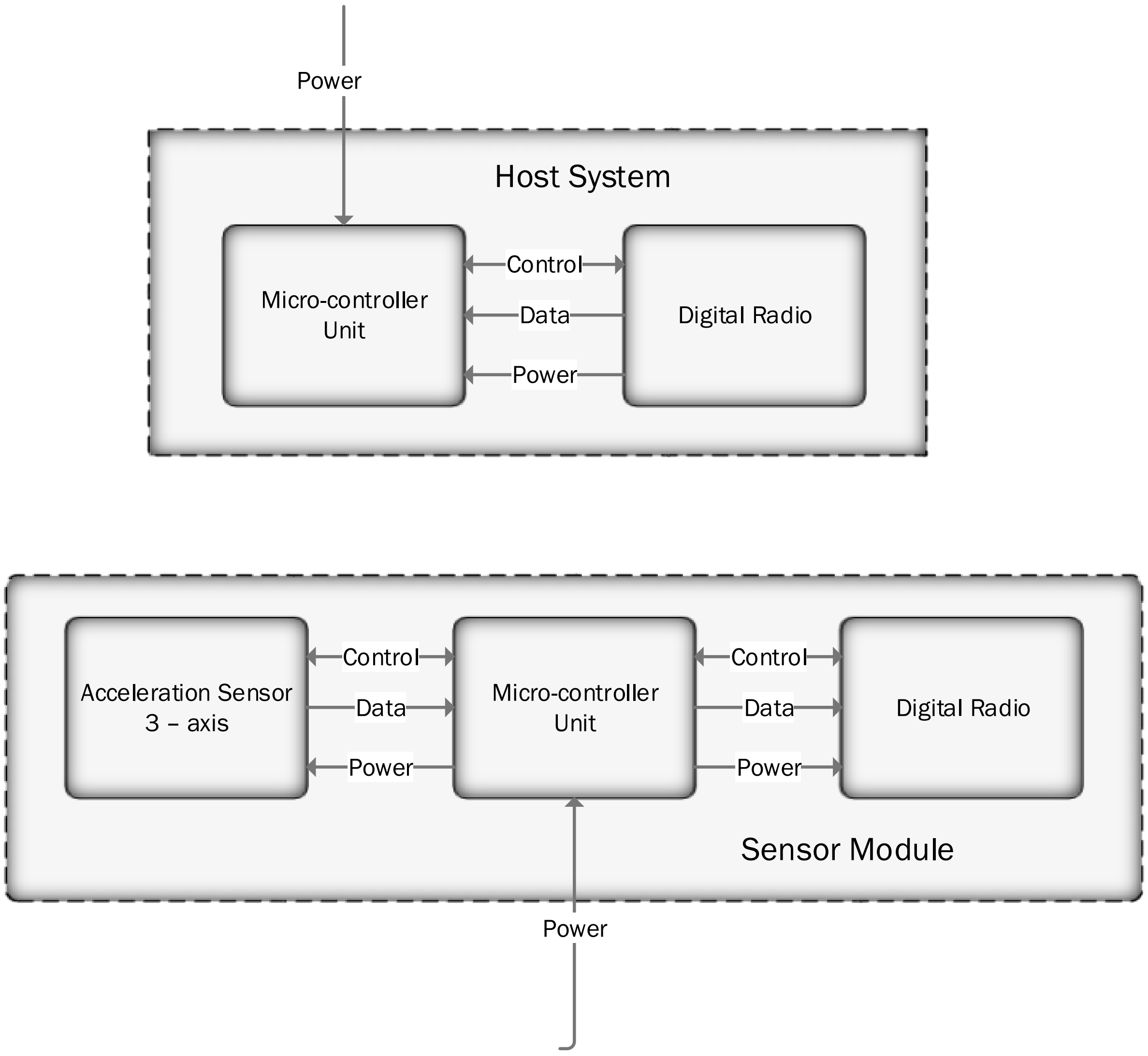


Figure 5.1: *A schematic view of the sensor module and host system, showing their principal part hardware parts.*

5.2 Requirements

The module controller must handle the following tasks:

MCU Control	Necessary run-time configurations of the MCU and its hardware.
Radio control	Configure the radio, load data into its buffers and initiate transmissions.
Sensor control	Configure the sensor and retrieve sensor data.
Data storage	Data received from the sensor must be stored in system memory and retrieved when a complete data set is ready.
Communication scheme	The controller must know whether to transmit the entire data set or transmit an analysis.
Data processing	If data is processed locally the controller must retrieve the data set from memory, run the appropriate algorithm(s) and store the result.
Message fragmenting	Transmitting data is done by transmitting packets each holding 64 bytes of payload data. Data sizes exceeding this must be fragmented and sent as multiple packets. The controller must organize the transmission of this set of packets.
Power-state Control	Energy can be saved by switching idle units into low-power sleep modes. The controller must have a correct and consistent overview of which hardware components that are idle so that these can be switched to such modes.

Acting correctly according to the current communication scheme and correct switching between power-states are the tasks which have a potential of saving energy. The other tasks are administrative tasks that are necessary to successfully gather data from the sensor and transmit information to the host system, or they are tasks which may themselves use energy but contribute to a lower overall energy consumption. Data storage is an example of the former and data processing is an example of the latter.

5.3 Structure

The system requirements listed in section 5.2 on page 39 are met by structuring the software as shown in figure 5.2; A module controller which accesses four software sub-modules. A quick description of each block in the figure follows.

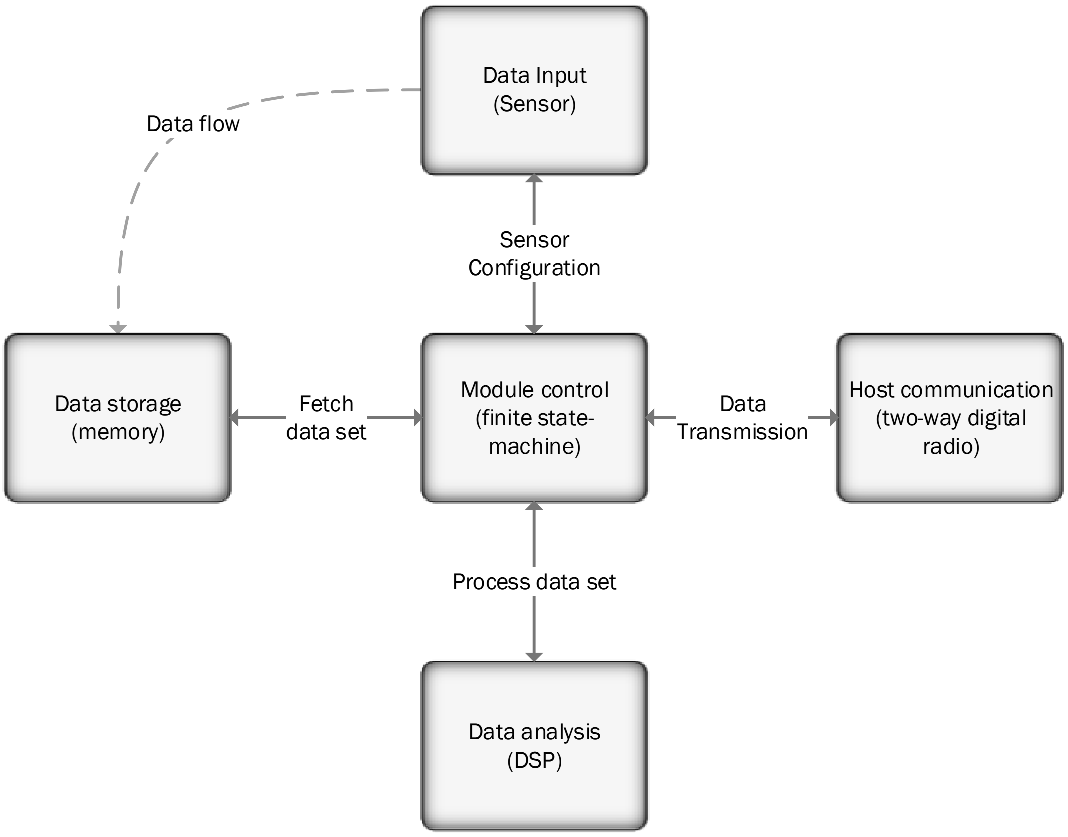


Figure 5.2: *The software modules in use. The system as a whole is controlled by the Module Control, with the exception of data input. This bypasses the MCU so that it may, if feasible, hibernate during data storage.*

5.3.1 Sub-Modules

The data input module provides an interface for configuring the sensor hardware, but the actual retrieval of data is performed by an interrupt service routine which stores data directly in the storage module. The sensor hardware contains a FIFO-buffer so that an interrupt only occurs when a set amount of samples have been made.

The data storage module is set up to contain two data sets, such that the sensor module writes its data into one data set while the other set is available to the system. Once a data set is filled up to a set number of samples, the storage module switches availability, so that the sensor can continue to write data into the other set and the freshly written one is made available to the system. This assures that data can be continuously written and read without interference.

The data analysis module contains a set of digital signal processing algorithms to be used on the data sets provided by the storage module. The framework is designed in such a way so that different algorithms can easily be chosen and evaluated without any changes to the rest of the system. This is intended to simplify the development process.

The host communication module provides access to the radio module. It provides an abstraction layer between what the system needs to send and how the radio needs to send it. It also keeps track of the progress of data transmission, feeding back this status data to the controller so that they controller may know when all data is sent and the radio may be shut down. This is vital to correct power control of the system.

This provides the control system with access to all the services and information it depends on for proper operation, with the exception of power-state control and utilizing the correct communication scheme. These two tasks were implemented in the control system itself.

5.3.2 Module Control

The operational framework of the software is an event-driven state-machine. Events are fed into an event queue when there are tasks to be done, and the state-machine processes these tasks by reading events from the queue. Thus, an empty queue indicates there are no tasks to process and thus the state-machine may set the MCU into a hibernating mode. New tasks are initiated by processes outside the state-machine, such as the sensor having new data, which, if the MCU is hibernated, wakes up the MCU through the use of interrupts. The event-cycle is shown in figure 5.3.

Another part of the wireless sensor module which can be hibernated is the radio. As the prototype is transmission only, the radio may be hibernated whenever there is no data to send. For the cases where the module only transmits a small set of analysis data, this will occur quite often

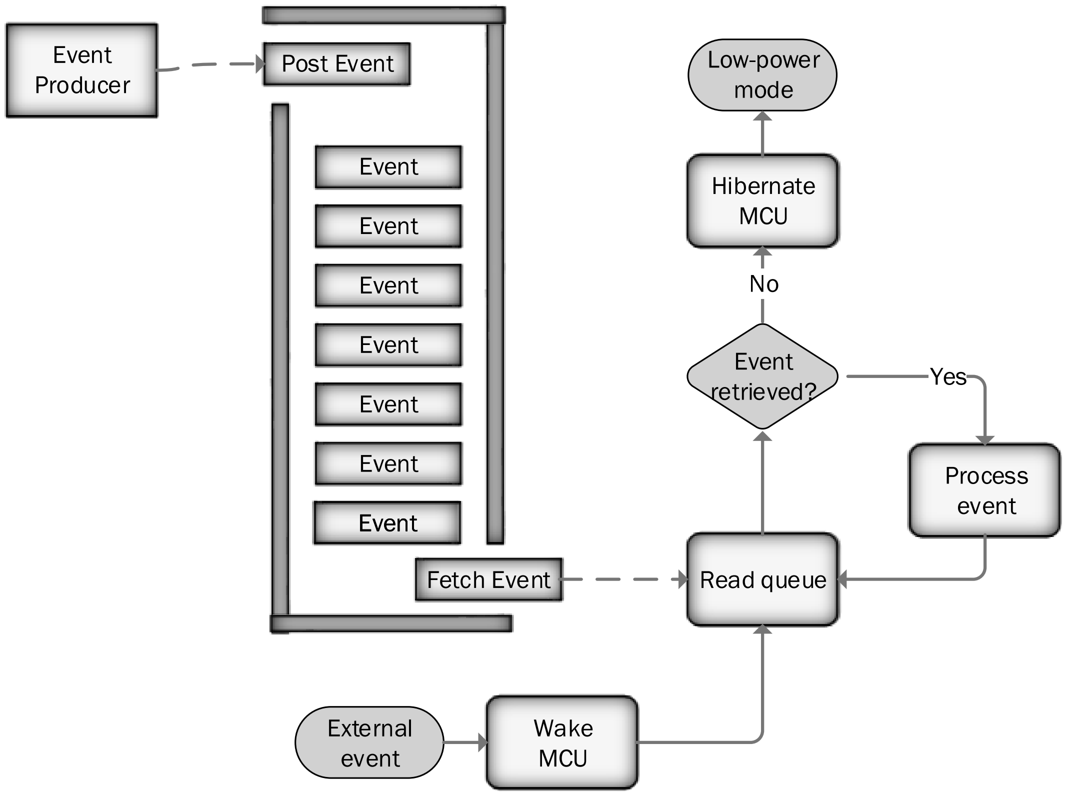


Figure 5.3: *Events are produced and sent into the queue, which in turn is processed by the state-machine. If there are no events to process, then the MCU may be powered down.*

and the data set may be small enough to be sent in one single transmission. For the cases where the module must transmit entire sets of raw data, the data set must be fragmented into multiple packets. The host communication module handles the process of fragmenting a *message* into separate *packets* and updating the message with meta-data on the transmission progress. The state-machine then uses this meta-data to detect if a message has been completely sent, as this means that the radio is idle and can be sent into a hibernating mode. The state-machine can power up the radio when a new message is to be sent. This design allows the data transmission to be handled in exactly the same way regardless of which communication scheme that is being tested. The power-state switching of the radio is also handled as event-driven tasks.

Figure 5.4 shows the design of the state-machine, using two states: Idle and Transmission. The Idle state is tellingly empty, as could be expected from an idle state. The state may seem unnecessary but it makes it explicit that the radio is currently not needed and can be kept powered

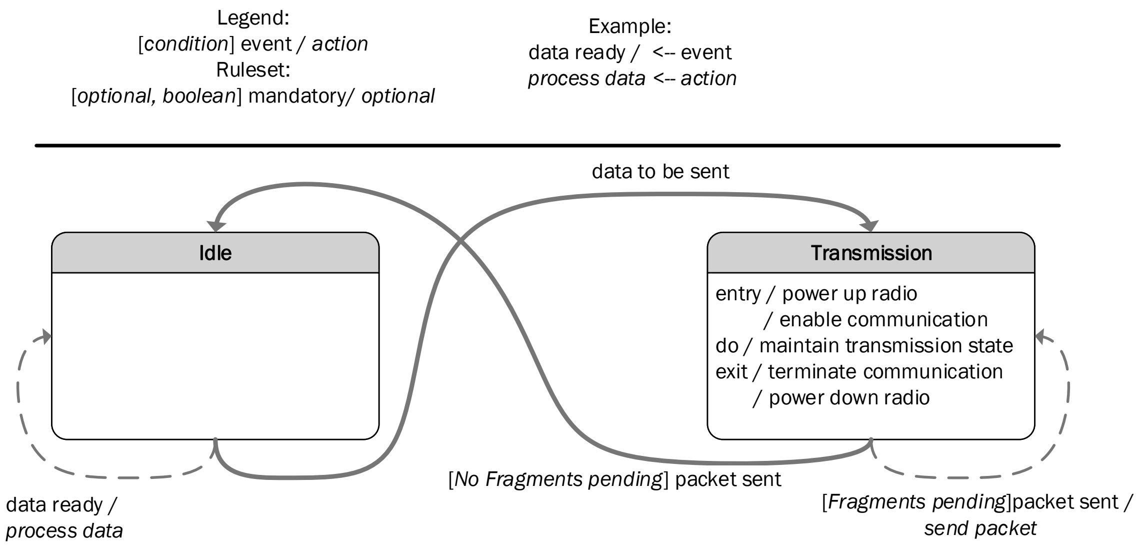

Legend:
[*condition*] event / *action*
Ruleset:
[*optional, boolean*] mandatory/ *optional*
Example:
data ready / <-- event
process data <-- *action*
data to be sent
Idle
Transmission
entry / power up radio
/ enable communication
do / maintain transmission state
exit / terminate communication
/ power down radio
data ready /
process data
[*No Fragments pending*] packet sent
[*Fragments pending*]packet sent /
send packet

Figure 5.4: *The state-machine controlling the system. The purpose of using the transmission as a state is not based on the importance of the radio but on the ability to power down the radio when possible.*

down. The design is also well suited to be extended with additional states (such as a Reception state) or additional tasks.

One exception was made to the organization of the event-system. The sensor module is allowed to autonomously call the functions necessary to move data from the MEMS sensor hardware buffer to the data storage. This decision was taken with energy consumption in mind: The hardware of the MCU includes a Direct Memory Access unit, which allows data to be moved in and out of memory without intervention of the MCU program counter. The motivation for the exception is thus that setting up the sensor to bypass the MCU is that with the correct configurations, it will be possible to leave the MCU idle (and hibernated) until a complete data set has been stored in memory.

5.3.3 Host Communication

The host communication module is by far the most complex of the sub-modules and is therefore given further explanation. It must handle the transmission of data from the sensor module to the host system. Because the prototype is to evaluate different types of transmission schemes, the communication module must be adaptable to different types of data transmission. Routing and networking capabilities are not necessary to be able to perform the tasks of the communication module and are therefore omitted to save energy. This results in a simple communication link rather than a networking solution.

The module uses a protocol stack to handle data transmission. The highest-level data structure is a *Message* and is stored in the module control. The next level uses *Packets* and Messages that can not be fit into a single packet must be fragmented into multiple packets. The logic which partitions a message into packets is located inside the host module. Packets are sent to the radio hardware where they are encapsulated in *Frames*, which is the hardware-level data structure. Each packet can always be fitted within a single frame.

The communication module uses a small buffer of constant memory size to store up to 8 packets in MCU memory. On input from the control system, it transmits the next packet to the radio hardware buffer, initiates packet transmission, inserts new packets into the buffer and updates the message meta-data on the state of the current message transmission. This process is shown in figure 5.5. The protocol stack can be seen in 5.6.

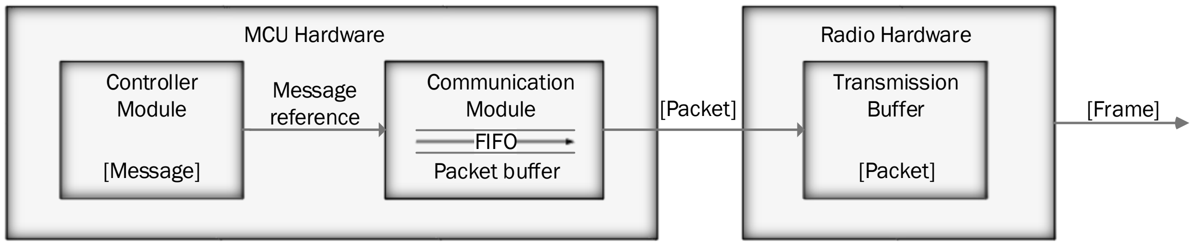


Figure 5.5: *The flow of communication. The message itself is stored in the system controller, passing a reference to it to the communication module which picks out the next bytes of data to transfer as a packet. This packet data is stored in FIFO buffer which in turn transfers a packet to the radio hardware. When the radio hardware is instructed to transmit its buffer data, it transmits the packet data as a frame.*

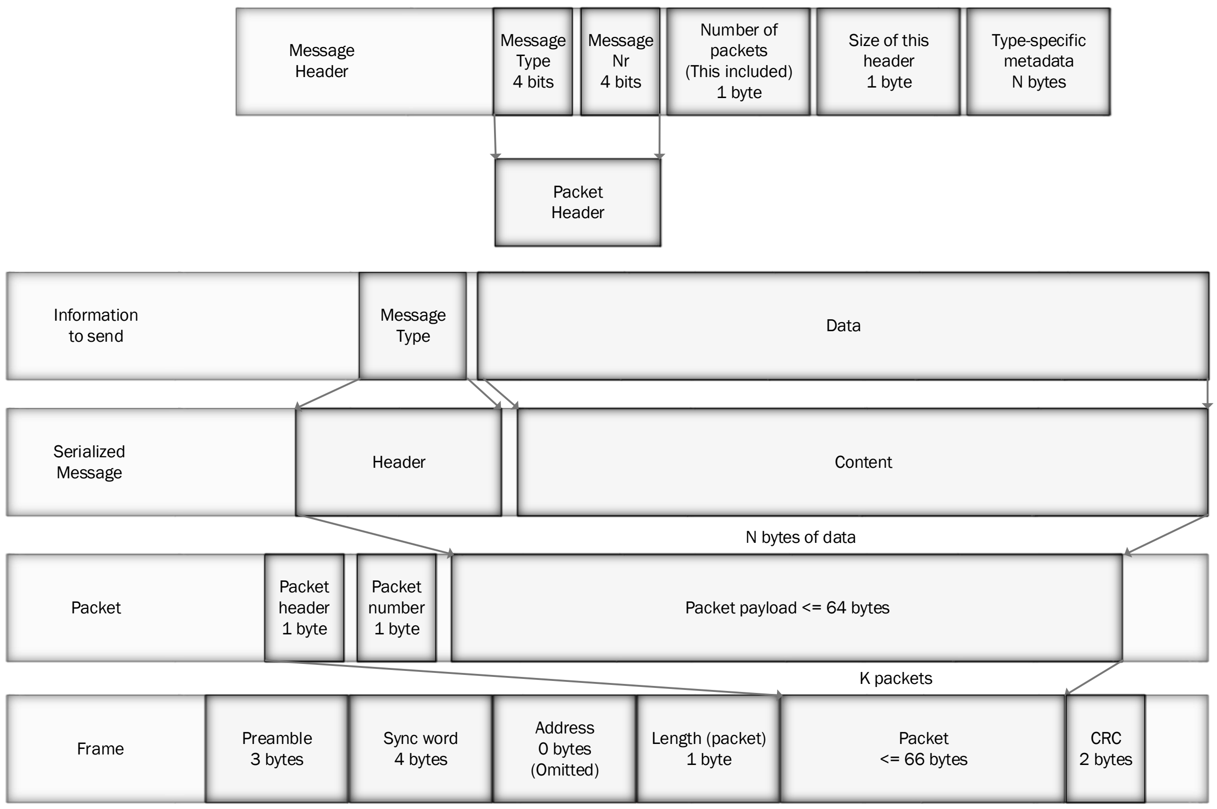


Figure 5.6: *The protocol stack. Each message contains a small header, and each packet also contains a minimized message header to verify which message it is part of. This prevents packets from being associated with the wrong messages.*

5.4 Implementation

5.4.1 State-Machine

The state-machine uses a set of enumerated values as an event pool and a ring-buffer for an event-queue. The use of a ring-buffer means that if the production of events becomes far greater than the consumption of events for a period of time, some events in the queue may be overwritten and thus never processed. However, the alternative scenario is that the memory used by the event queue grows dynamically until it consumes all free memory in the system, crashing the computer. The ring-buffer solution is chosen because it guarantees that a fixed size of memory, known at compile-time, is the only memory used for the event-queue. The state-machine itself is implemented as set a of nested switch-statements, which is a conventional and quite simple design, suitable for systems with low state complexity. If there are no events to process the system will enter a low-power hibernation mode. Interrupts will wake up the system.

5.4.2 Communication-Link

The communication module was developed from ground up in order to better control and understand the operation of the communication system and thereby its energy consumption. The result is a communication link with very little overhead, but without two-way communication. The sensor may only send data and the host may only receive.

The packet structure has a maximum payload of 64 bytes and messages exceeding this size is fragmented into multiple packets, see figure 5.7. The packet structure is designed so that one packet will always fit into a single frame.

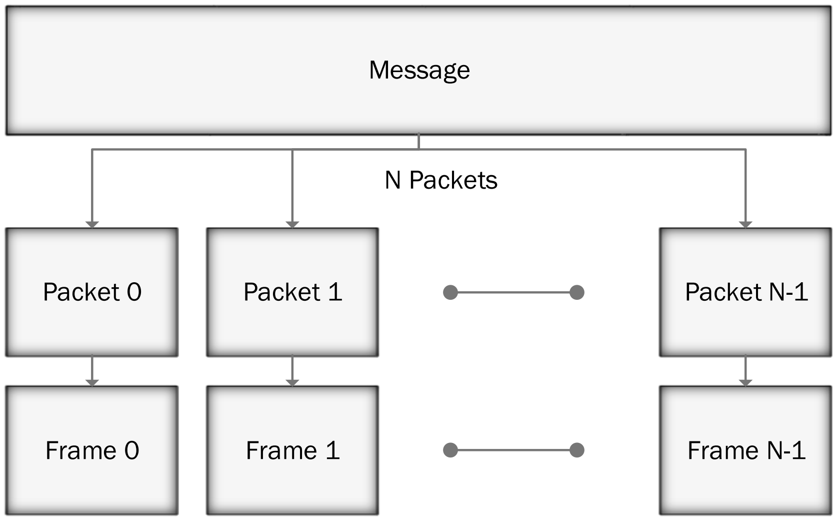


Figure 5.7: *The relation between a single message and several packets and frames. A message which amounts to 64 bytes of data or less can be sent as a single packet.*

The *[Message]*, *[Packet]* and *[Frame]* structures, as shown in figure 5.5, forms the communication protocol stack. The message consists of a small header and an amount of byte data. A message is split up into packets, and as only a single byte contains the packet number, each message must be small enough to fit within 128 packets. In turn, each packet only permits 64 bytes of payload data, constraining the message size to $128 * 64 = 8192 = 8kB$ of data, including the message header which size is a minimum of 3 bytes.

Equations (5.1) through (5.4) are valuable for calculating the necessary amount of packets and total bytes transmitted for a given amount of message data.

$$\text{message\_size}_{\text{bytes}} = \text{message\_header}_{\text{bytes}} + \text{message\_data}_{\text{bytes}} \tag{5.1}$$

$$\text{number of packets} = \frac{\text{message\_size}}{\text{packet payload}} = \left\lceil \frac{m_{\text{bytes}}}{64} \right\rceil = \text{n packets} \tag{5.2}$$

$$\text{frame header} + \text{packet header} = 3 + 4 + 1 + 1 + 1 + 2 = 12 \text{ bytes} \tag{5.3}$$

Equation (5.3) uses the numbers from figure 5.6 to calculate the necessary header data for each frame.

$$\text{transmitted bytes} = (\text{frame header} + \text{packet header}) * \text{n packets} + m_{\text{bytes}} \tag{5.4}$$

Using these equations, one can compute that sending 1024 bytes of information and minimal message header data (3 bytes), one needs to send 1027 bytes which requires 17 packets and a total of 1231 bytes must be transmitted. Comparing the size of the data to be sent at the top level with the data transmitted at the frame level, one gets $1024/1231 = 83.18\%$ network utilization. Sending far less data, for example 4 bytes, only requires 7 bytes message in a single frame of total size 19 bytes. This results in a utilization percentage of $4/19 = 21.5\%$, which is far worse. It is important to keep the main goal in mind: To reduce overall energy consumption, and the total amount of transmitted data is reduced, transmitting 19 bytes instead of 1231, which is a reduction of **98.5 %**.

Thus if processing data locally (which should result in sending less data) can effectively reduce data transmission, energy efficiency of the sensor module overall can increase despite the drop in network utilization.

Even though energy-optimization of the protocol stack will affect energy efficiency (Goldsmith[3] makes an argument for *cross-layer optimization* with respect to energy), these equations show that the impact of the protocols and the impact of the data work in tandem; improving one only has an effect when it is a dominant part of bytes to be sent. For the 1024 bytes example, protocol optimizations would make a small difference, for the 4 bytes example, protocol optimization could be a natural next step. As the optimization of the network is outside this thesis' scope, there is no further work on this topic to be found here.

Error messages are handled within the communication module. This implementation results in a communication module which can handle messages of dynamic content and size without using dynamic memory allocation. Further it hides away all this complexity from the control system, but instead only provides the control system with the information necessary to adapt to changing priorities. The packet buffer allows packets to be prepared while a preceding packet is in transmission, reducing waiting time between packet transmissions. It also makes it easier to implement re-transmission schemes.

5.5 An Energy-driven System Design

Typical objectives for an embedded system design is a solution which is reliable, compact and robust at an affordable cost. The design for this system adheres to these objectives but the paramount motive of the design is to control and reduce energy consumption.

The framework of the control system is an event-driven state-machine, and throughout the work with this project it became apparent that the primary states of the system would need indicate whether a transmission was taking place or not, as this made it easy to appropriately control the power-state of the radio unit. Likewise, the event-system is set up so that any work necessary is triggered by an interrupt which wakes up the MCU from a sleep mode.

Thus both the event-system and the state-machine is designed with power control in mind which over time translates into energy control. Because the framework is designed so that features can be added, changed and removed without disrupting this power control model, it seems reasonable to argue that the final solution is the product of an *energy-driven system design*. The objective of conserving energy affects the entire design and system integration, as opposed to an approach where energy efficiency is achieved primarily by choosing low-energy components or tuning chosen system parameters.

5.6 Summary

To conclude the description of the system design, a more elaborate diagram of the system is shown in figure 5.8. This can be compared with higher-level schematic found in figure 5.2, where the diagram in this section primarily expands the structure of the communication module, as this is the most complex sub-module, and the operation of the state-machine is already well covered.

Chapter 2 showed a conceptual diagram of the sensor module relation to its surroundings, which is repeated here in figure 5.9 for convenience. The arrow reading ”Environment” in this figure is analogous to the arrow reading X” Y” Z” in the system drawing, and the arrow reading ”Alert” corresponds to the radio hardware’s arrow to the host system (radio tower).

Table 5.1: *The pin-to-pin wiring between the MCU and two peripheral components; the radio and the sensor. The VMCU pins outputs a 3.3V power supply which is measured by the AEM. The* RESET_N *signal must be wired to the same supply source as the radio’s VDD. Wiring these to different sources creates circuits that confuses the MCU board’s startup routines.*

MCU Board		Purpose	Device PIN
Function	Pin		Radio
		Radio	
USART1_TX	PD0	SPI MOSI	MOSI
USART1_RX	PD1	SPI MISO	MISO
USART1_CLK	PD2	SPI Clock	SCLK
USART1_CS	PD3	SPI Chip Select	CSN
3V3	VMCU	Chip Reset	RESET_N
GPIO Interrupt	PD5	Interrupt	GPIO2
3V3	VMCU	Power	VDD
Ground	GND	Ground	GND
		Sensor	
USART2_CLK	PC4	SPI Clock	SCL
USART2_CS	PC5	SPI Chip Select	CS
USART2_RX	PC3	SPI MISO	SDO
USART2_TX	PC2	SPI MOSI	SDA
GPIO Interrupt	PB11	Interrupt	INT1
3V3	VMCU	Power	VCC
Ground	GND	Ground	GND

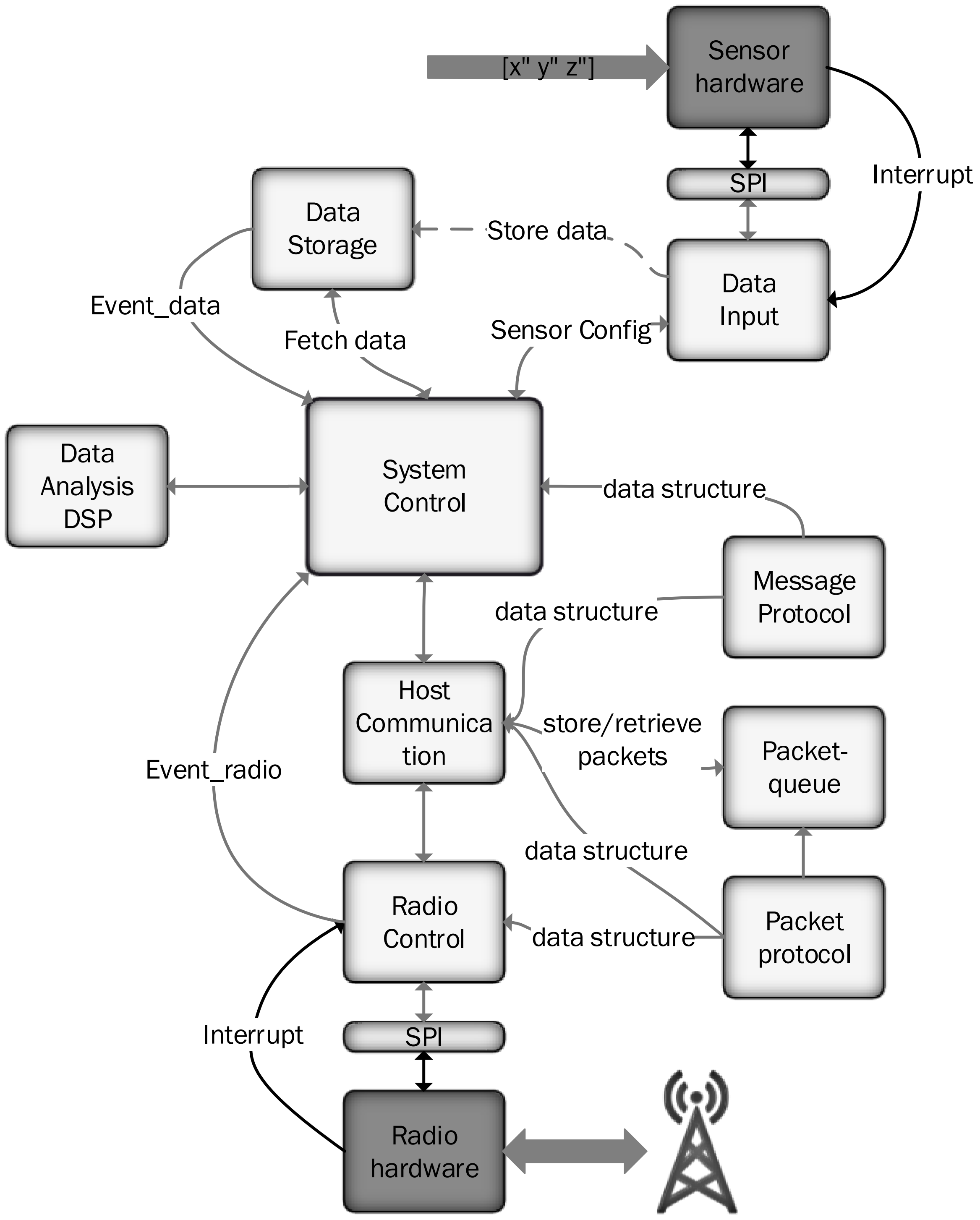


Figure 5.8: *An overview of the sensor module prototype. The light grey boxes are software modules within the MCU, the darker grey boxes are external hardware components.*

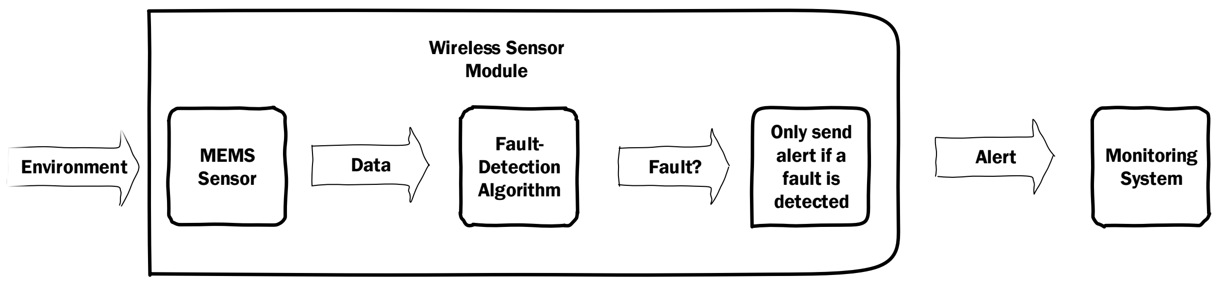


Figure 5.9: *The sensor module reviewed, as shown in chapter 2.*

Chapter 6

Test Setup

6.1 Objectives

The tests to be run are to investigate the impact which specific parameters of the system has on energy consumption. These parameters are as follows:

- Communication scheme.
- MCU Energy modes.
- Radio energy mode.

The communication scheme is the approach the system will use to data transmission: These are explained in a separate section. The MCU can switch between multiple energy modes. The top-level mode, EM0, is used when the system must execute tasks and the 3rd-order reduced-energy mode, EM3, is used when the system is idle. When the radio is not transmitting, it can be powered down into a sleep mode.

By running tests where all valid combinations of these parameters are evaluated, their impact on energy consumption can be exposed.

The rationale of this approach to energy-optimization is that it reveals where there is the most to gain. If 90 % of the energy is spent on radio transmission, there is very little to gain on improving MCU energy-performance. If 50 % of the energy is spent on the sensor, it will be a good start to investigate if the measuring can be done with less energy. The fundamental idea of this thesis is that data processing is far cheaper than external communication, and this approach is a good starting point for testing that idea.

As will be shown in the next chapter, such a trade-off between processing and transmission does exist; the use of local processing dramatically reduces the total energy consumption. Once such a discovery is made and verified, the model can be applied again to discover the next target of the optimization strategy. If the energy reduction in one component is reduced to a point where it is either on par with the either components or it is unlikely to be possible to reduce much further, some other component or trade-off must be explored.

6.2 Transmission Schemes

The two most important schemes to test are the raw-data schemes and the local-processing schemes. This is because these two combined perform all the possible tasks of the system. The no-transmit scheme represents a theoretical situation, and the scheme with the most promising realistic potential, the fault-dependent scheme, is constructed from the evaluation of the local-processing and no-transmit schemes.

6.2.1 Raw-Data Scheme

The raw-data scheme will transmit a data-set of the Z-axis without any processing done. A data set of one axis consists of 512 samples of 2 bytes each. As was calculated in chapter 5, this requires 17 frames and a total of 1231 bytes sent. The first 16 frames will have a length of 76 bytes and the final frame only 15 bytes. With a bitrate of 1200 bps, sending all frames with little or no delay between them should take 8.2 seconds plus delays.

6.2.2 Local-Processing Scheme

As the WiVind fault-detection algorithm is incomplete, a replacement algorithm must be used. The chosen algorithm is a fast fourier-transform of the Z-axis data, whose output is then scanned for the frequency with the highest amplitude. This frequency is the algorithm output. The algorithm takes 512 samples of data, converted to 32-bit floating point values. The output is also on the format of a 32-bit floating point value.

This requires 4 bytes of data to be transmitted, requiring a single frame of 19 to be sent across the link. At a 1200 bps bit-rate, this should take 0.127 seconds.

6.2.3 Fault-Dependent Scheme

As explained in chapter 2, the project background is to only send data when a structural fault in the blade has been detected. This means that whenever the blade is intact, no data should need to be transmitted at all, with the exception of a possible heart-beat signal[1]. Another possible transmission is that of erroneous fault-detections, which would occur if the sensor module erroneously detects a blade fault.

A heart-beat signal was not implemented in this project, neither was the rate for such a signal decided. As for erroneous transmissions, these could cause the wind-turbine owner to launch purposeless maintenance expeditions and must either be reduced to a minimum. One could also require fault-alerts to be transmitted for some time to verify a continous detection of the fault.

Testing a fault-dependent scheme would therefore require a far more complex testing jig which can produce accurate sensor input both for non-faulty and faulty situations. This is not done, instead the no-transmit scheme is used to investigate the potential of a fault-dependent scheme.

6.2.4 No-Transmit Scheme

A scheme in which no data is transmitted at any point is useful as a reference point for a fault-dependent transmission as it represents the lower power level asymptote of the sensor module. It spends energy only on controlling the sensor module and processing data locally.

[1]A periodic signal sent to indicate the sensor module itself is functioning.

6.3 Procedure

Each test setup is run for 5 minutes. Voltage and current will be measured at constant timing intervals to develop a power profile and from this an average power level can be calculated. This allows the energy-efficiency of each setup to be represented as a single scalar value, with the power profiles providing the information needed for a more detailed evaluation of each setup. Table 6.1 shows the different system configurations which will be tested. The left-hand and right-hand side differs on which communication scheme that will be used: The local-processing scheme, or the raw-data scheme. It is expected that this will have the biggest impact on the energy consumption and the table is therefore formatted accordingly.

Local-processing		Raw-Data	
MCU_Managed	Radio_Controlled	MCU_Managed	Radio_Controlled
MCU_Unmanaged	Radio_Controlled	MCU_Unmanaged	Radio_Controlled
MCU_Managed	Radio_Uncontrolled	MCU_Managed	Radio_Uncontrolled
MCU_Unmanaged	Radio_Uncontrolled	MCU_Unmanaged	Radio_Uncontrolled

Table 6.1: *The different system configurations which are to be evaluated. Each cell in the table represents a unique combination of the set of test parameters.*

Chapter 7

Results

This chapter presents the results of the test setup in the previous chapter. It emphasizes the test results of significance as well as important properties of the results. This formatting is chosen so that the content of the results is more easily understood. Should the reader wish to inspect the full set of test results, a graph compendium can be found in appendix B.

An evaluation of the test results can also be found in this chapter, discussing the meaning of the results and its implications for the energy consumption in the sensor module. This discussion is made here so that chapter 8 can be condensed to a discussion on the project overall.

7.1 Summary

Table 7.1 summarizes the most significant numerical results of the tests. The background for the numbers in the table can be found in this chapter. For each of the three schemes, the setups utilizing full power-control are shown, as these had the lowest overall power levels.

The total energy consumption for each scheme was computed directly from the test data and can be assumed to hold as high a level of accuracy as the involved tools allow. The numbers for the process and transmission costs were derived partially by means of visual inspection of the graphs shown in this chapter, and are therefore subject to a larger degree of error than the total energy cost values.

However, the results still hold relevance at a quantitative level and their differences are of such a scale that their qualitative value remains intact.

The no-transmit results are included as a reference point.

Transmission Scheme	Total Energy	Processing & Transmitting as % of total energy	Processing & Transmitting	Processing	Transmitting
No-Transmit	6.0 mJ	22.50 %	1.35 mJ	1.35 mJ	–
Local-Processing	24.3 mJ	80.58 %	19.58 mJ	1.35 mJ	18.23 mJ
Raw-Data	1056.5 mJ	99.95 %	1051.00 mJ	–	1051.00 mJ

Table 7.1: *A summary of energy consumption for the most important setups.*

7.2 Transmission Scheme Comparisons

7.2.1 Numerical Results

Table 7.2 contains the average power levels and energy consumption of each test setup, shown in the same formatting as the test setup in table 6.1. It indicates a major shift in average power level when switching from local processing to raw data transmission, with the application of power-control techniques only bearing significance once a local-processing scheme is chosen. This observation shapes the proceding result presentations; it indicates that the test parameters

Local-processing	Raw-Data
MCU_Managed Radio_Controlled 2.3726 mW 24.3 mJ	MCU_Managed Radio_Controlled 103.1722 mW 1056.5 mJ
MCU_Unmanaged Radio_Controlled 17.0987 mW 175.1 mJ	MCU_Unmanaged Radio_Controlled 115.8353 mW 1186.2 mJ
MCU_Managed Radio_Uncontrolled 10.8247 mW 110.8 mJ	MCU_Managed Radio_Uncontrolled 104.6583 mW 1071.7 mJ
MCU_Unmanaged Radio_Uncontrolled 25.2849 mW 258.9 mJ	MCU_Unmanaged Radio_Uncontrolled 117.7004 mW 1205.3 mJ

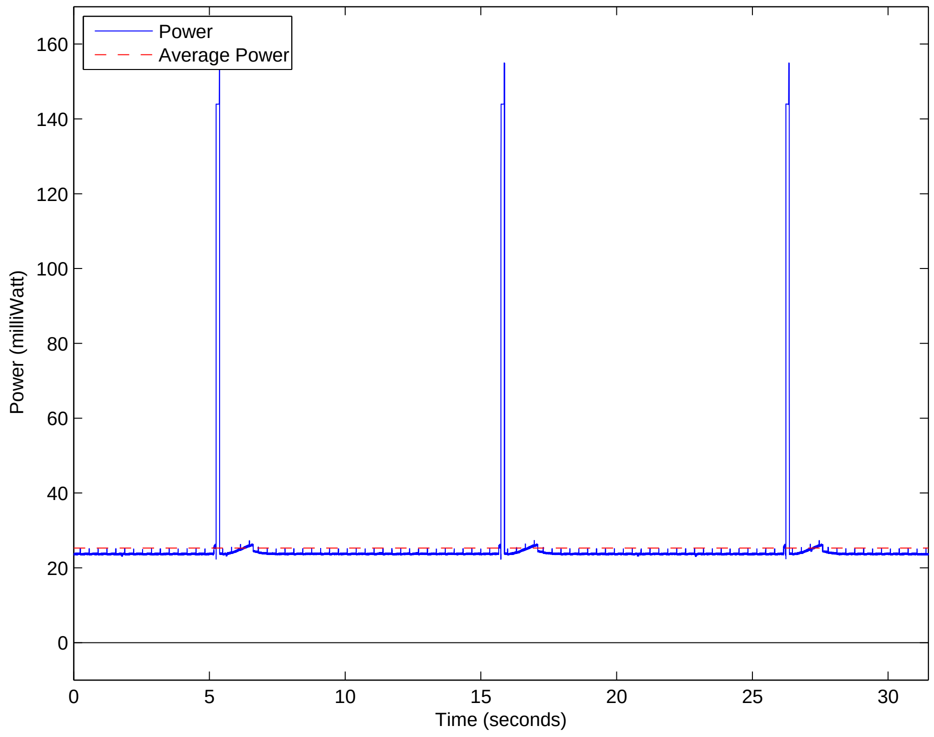
Table 7.2: *The different test setups and their numerical values.*

are tiered in terms of energy impact, with the transmission scheme as the first, top, tier, and the two power-controlling techniques at a second tier. Power-controlling the MCU and radio is therefore only shown for the local-processing scheme, as this is the only situation in which they have significant impact.

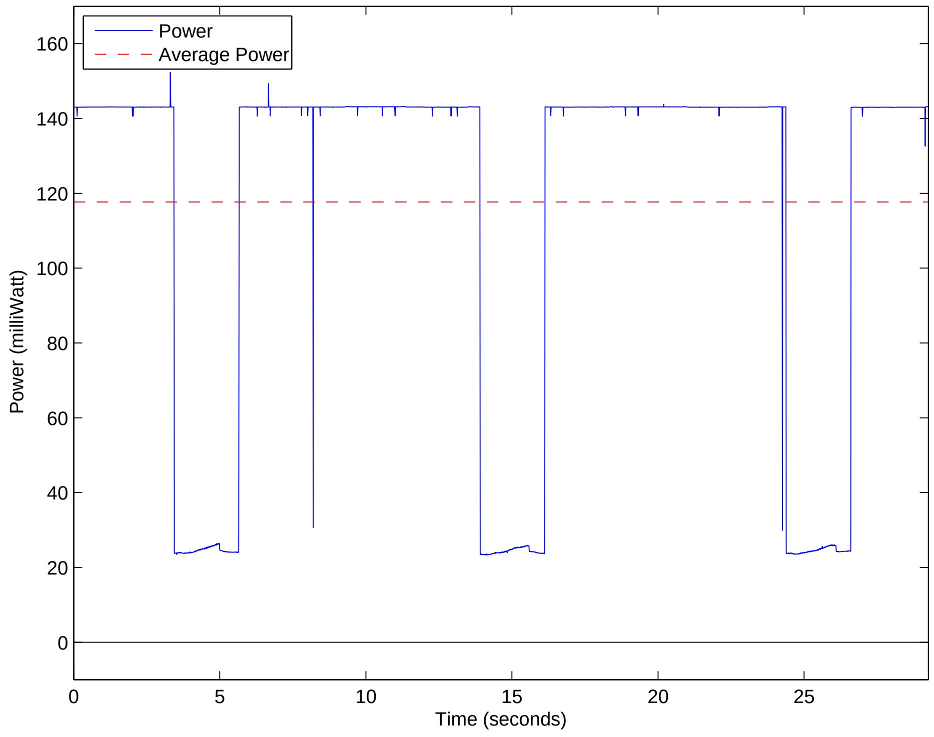
The average power value is found by averaging the power level of a 5 minute test run. This is done to make the result more robust to intermittent noise. The energy is the integral of that average power over a time period of one data set sampling, which is 10.24 seconds (At a sampling rate of 50 Hz it takes 10.24 seconds to gather a data set of 512 samples.). The zero-line of the y-axis is marked explicittly on most of the graphs to make that some, otherwise similar-looking graphs, deviates in their minimal power level.

7.2.2 Local-Processing and Raw-Data

Without power-controlling the hardware, the energy-performance of the local-processing and raw-data scheme is shown in figure 7.1. There is, as expected a significant difference, and when idle (not transmitting), both setups fall to the same power level, indicating they use the same amount of power in that state. Allowing the hardware to remain in active state at all times prevents the average power level from falling below 20 mW.



(a) *Local processing*



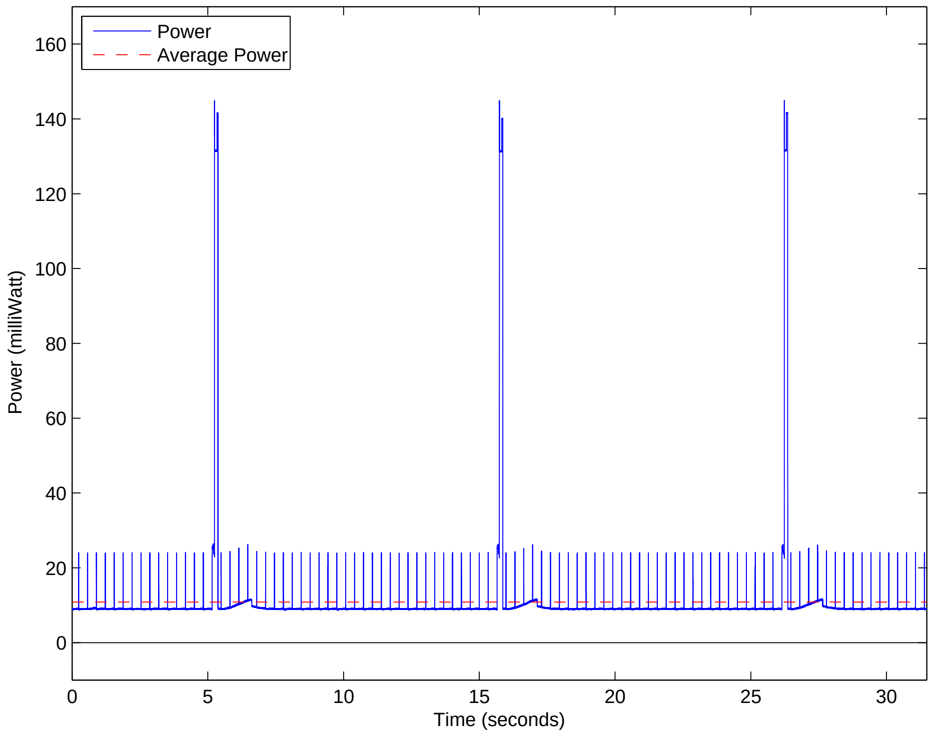
(b) *Raw data*

Figure 7.1: *The power impact of the communication schemes. Notice the significant difference in average power.*

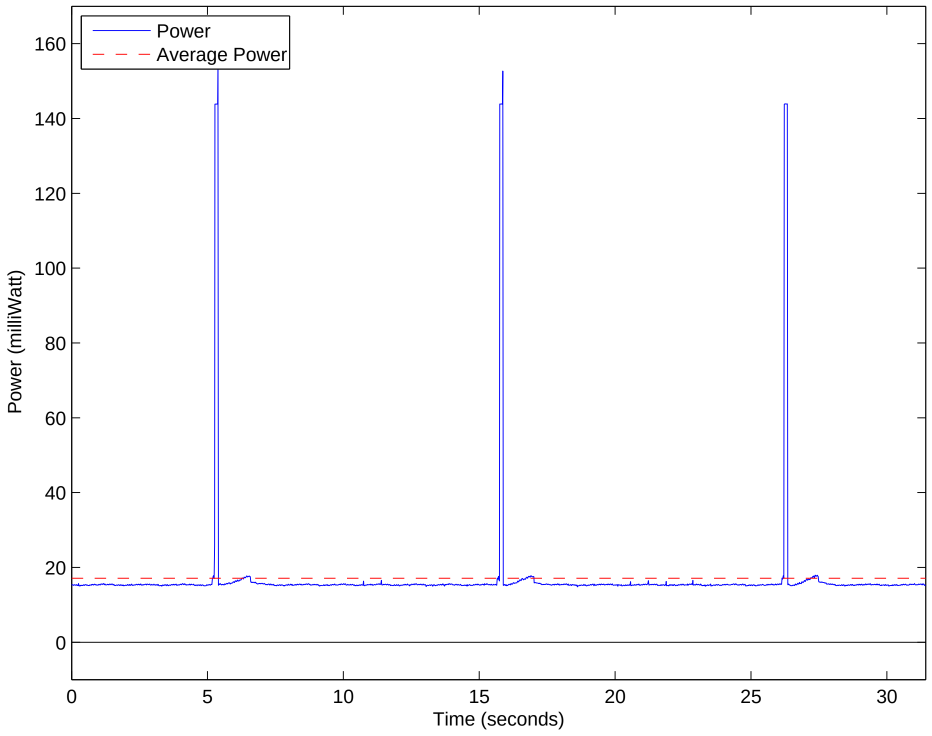
7.2.3 Power-control

Once a local-processing scheme is chosen, further gains are evaluated by power-controlling the MCU or the radio in figure 7.2. Both has an impact, but it is clear that the MCU is the more important energy sink in terms of shutting down idle hardware.

For reference, figure 7.3 shows the impact power-controlling has on a raw-data scheme, illustrating its minimal effect on the average power level.

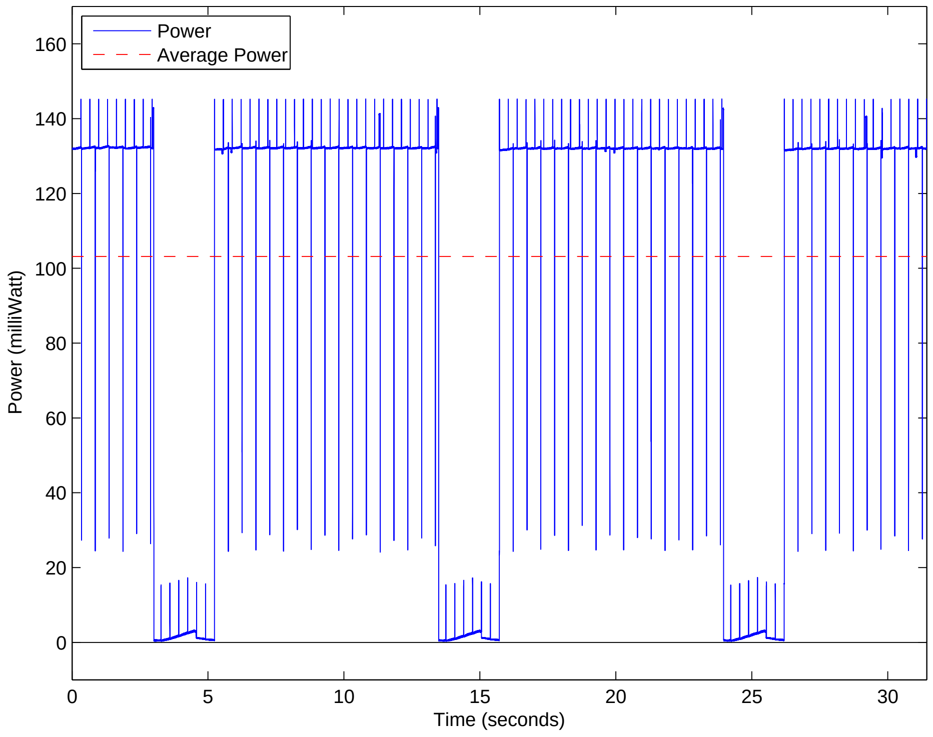


***(a)** The MCU is power-controlled, the radio is not*

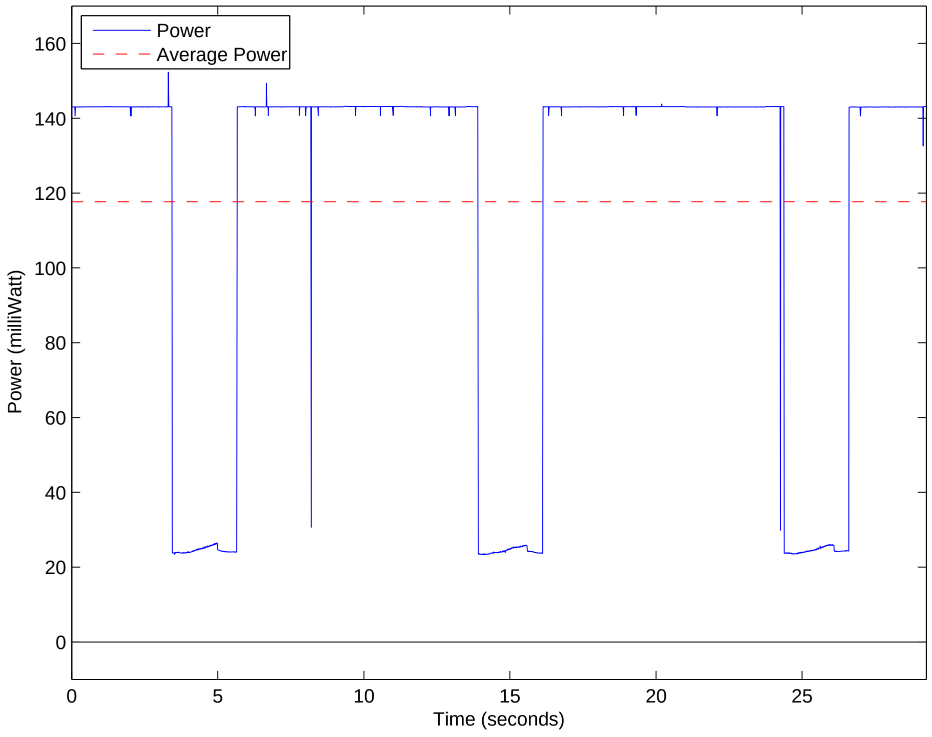


***(b)** The radio is power-controlled, the MCU is not.*

***Figure 7.2:** Settling for a local-processing scheme, the difference between power-controlling either the mcu or the radio is examined.*



***(a)** Power control of both MCU and radio enabled*



***(b)** No power control*

***Figure 7.3:** The raw data scheme, with and without power control.*

7.2.4 Local-processing and Power-control

Finally, a larger picture of the optimal setup is shown in figure reffig:poweroptimal. This setup utilizes a local-processing scheme and power-control of both MCU and radio to bring the average power level down to 2.37 milliWatts. In terms of comparing the local-processing scheme and the raw-data scheme, this is the optimal configuration.

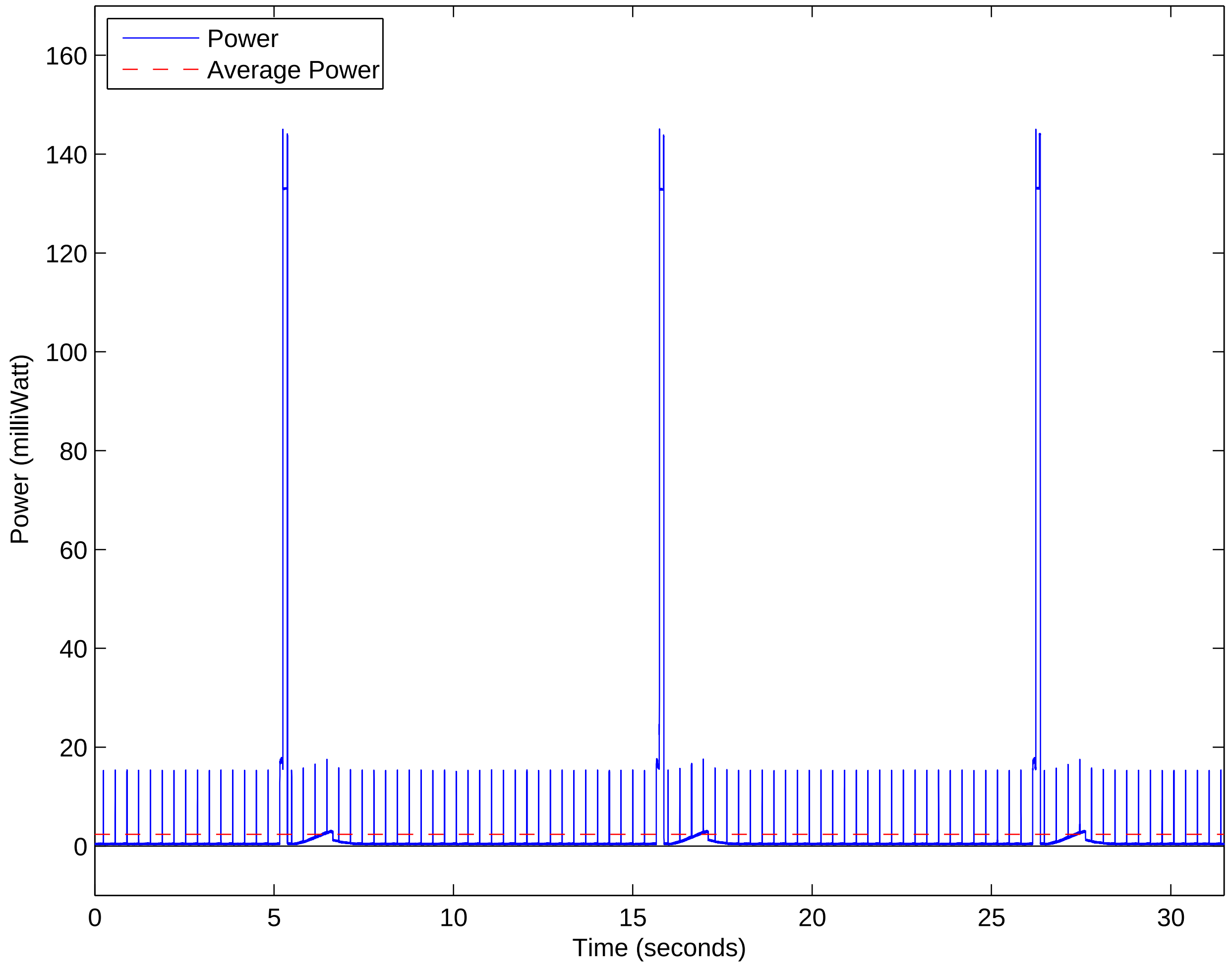


Figure 7.4: *With a local-processing scheme and power control on both the MCU and radio, average power is at 2.37 milliWatts.*

7.2.5 No-Transmit Scheme

The power consumption of the no-transmit scheme is driven only by the data retrieval, data analysis and system control tasks. There is, as the name implies, no transmission of any kind. It is therefore far lower than any of the other configurations, with an average power level of 0.58 milliWatts. As was described in chapter 6, this is not a realistic transmission scheme, but it shows a theoretical minimum of power consumption in the circuit. Comparisons between this scheme and the local-processing scheme is a useful starting point for discussing the impact of fault-dependent transmissions. The energy cost is of the data processing is shown in section 7.3 and is calculated to be 1.35 mJ.

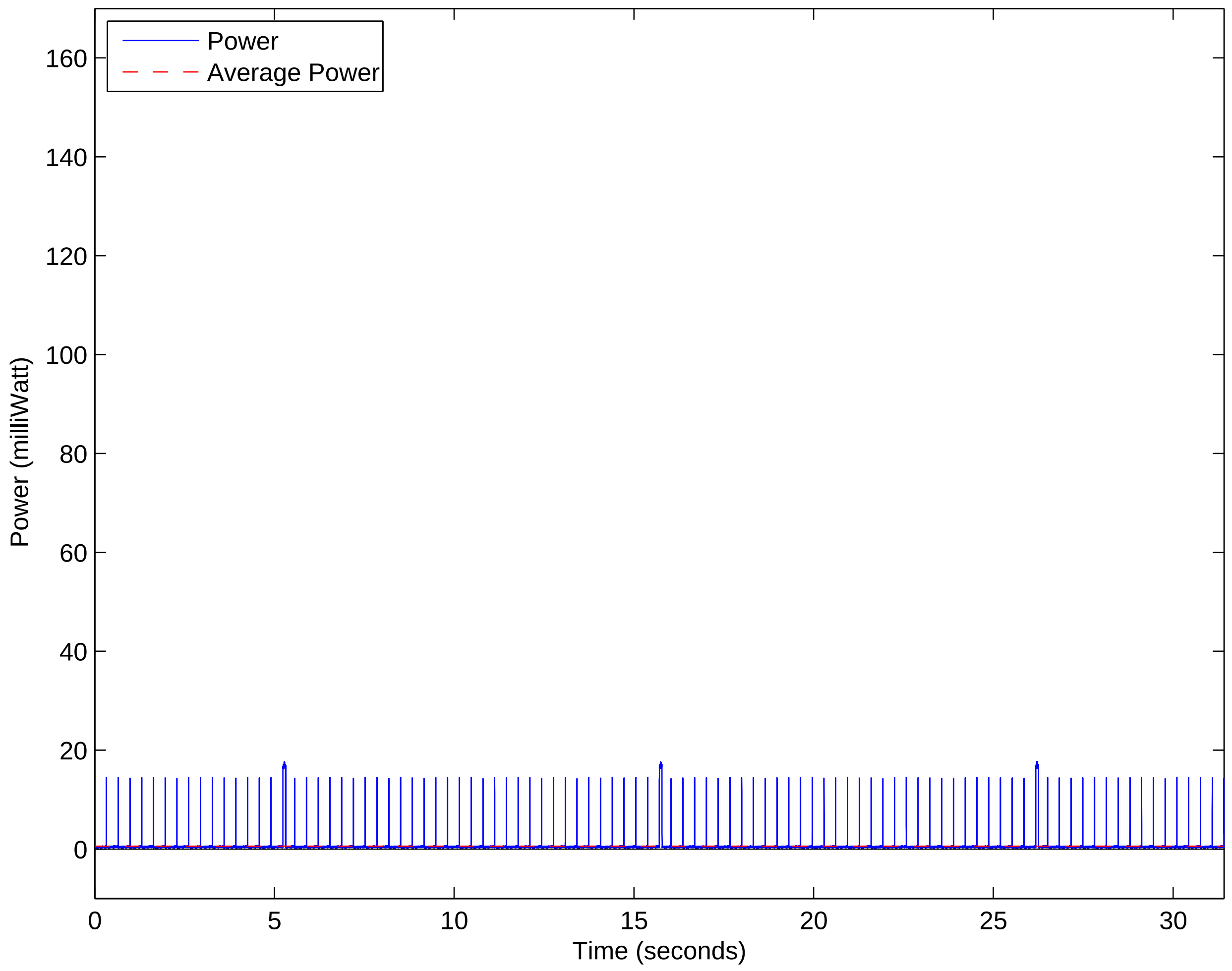


Figure 7.5: *By power-controlling both the MCU and radio and omitting any transmission, average power is at 0.58 milliWatts.*

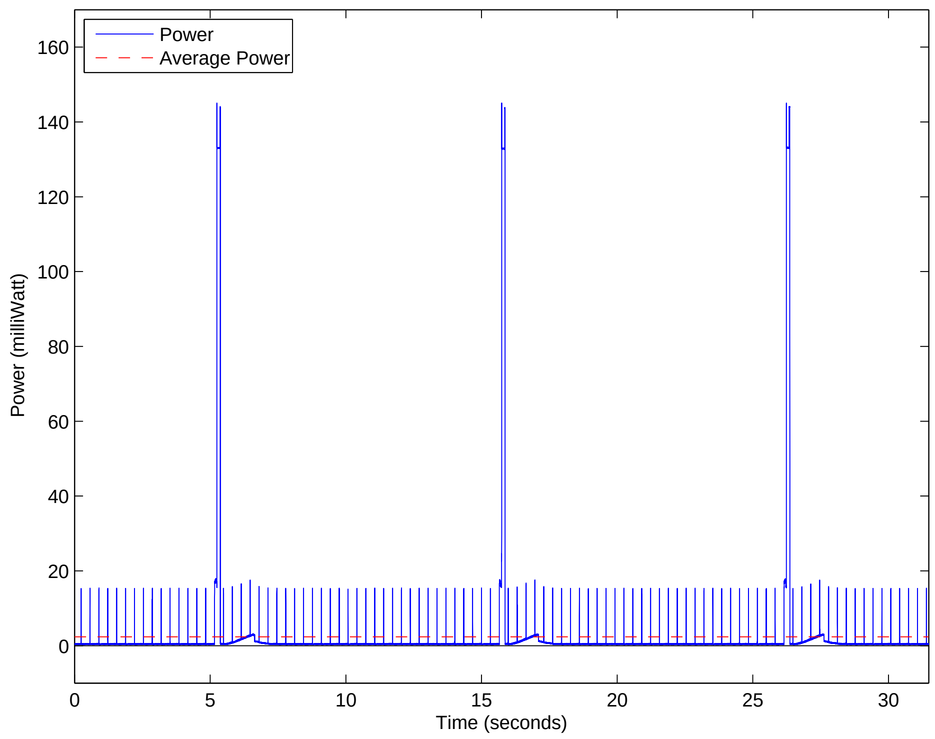
7.3 Power-Profile Analysis

The preceding sections has inspected the power-profiles on a macro-level. This section will have a micro-level view, investigating the details of the power-profile of the sensor module.

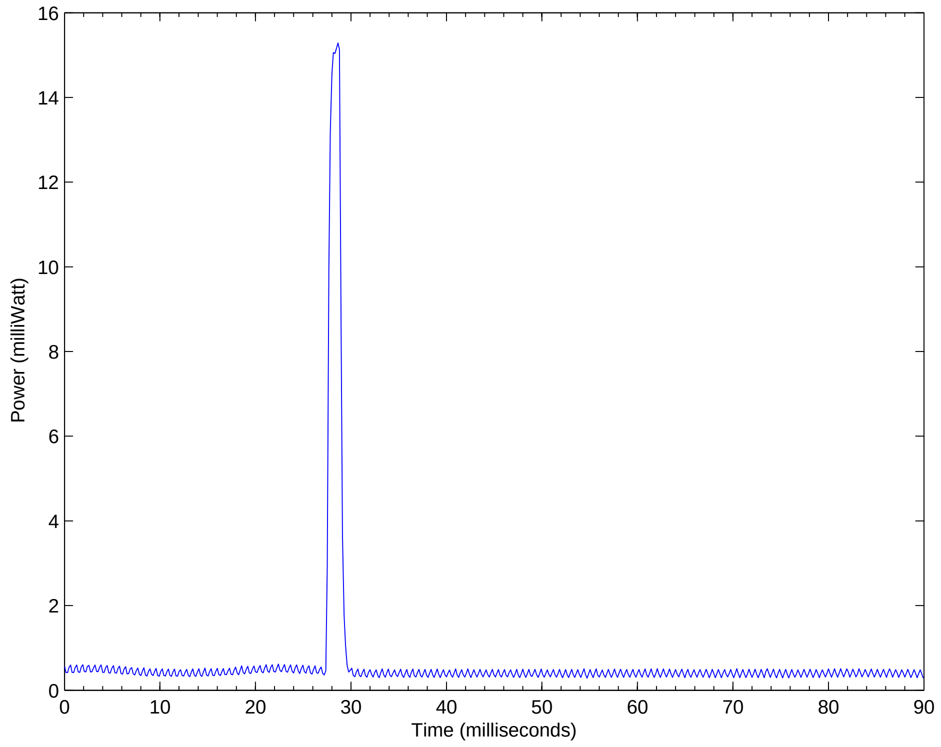
7.3.1 MCU Wake-ups

The graphs in figure 7.6 allows inspection of the power consumed when the MCU wakes up to retrieve data from the sensor's buffer. It is set up be alerted when 16 samples per axis are ready and each sample is 2 bytes. Including a command byte to initiate the retrieval, this requires the SPI-bus to transfer 97 bytes each time. The SPI-bus has a baud-rate of 2 MHz, requiring at least 0.388 milliseconds to transfer all 97 bytes. The spike on the graph lasts for approximately 2 milliseconds, indicating that only a small fraction of the awake-time is spent on actual data retrieval. As no other actual tasks are executed, the remainder of time can be assumed to be spent on changing the MCU's energy mode (waking up and hibernating).

These ”spikes” can be seen in all of the power graphs which power-switch the MCU.



***(a)** A plot of the power level in the sensor module. The small, tight, 'spikes' just below 20 mW is a result of the MCU waking up to retrieve data from the sensor.*



***(b)** A detailed display of one such 'spike'.*

***Figure 7.6:** The power profile of the MCU retrieving data from the sensor. Notice the difference in axis scaling between the two graphs.*

7.3.2 Local-processing Profile

The local-processing scheme transmits 1 frame of data of 19 bytes. This was calculated to last for 127 milliseconds and the graph in figure 7.7 indicates that the radio is in transmit mode for a total of 135 milliseconds. The data transmission energy is the tall block that rises above 125 mW. The same figure shows the impact of processing the data as a smaller block of energy prior to the transmission. This block lasts for approximately 90 milliseconds at 15 mWs. This suggests a processing energy cost of 1.35 mJ and a transmission energy cost of 18.23 mJ. Thus, in total, the local-processing scheme uses 19.58 mJ to transmit its data.

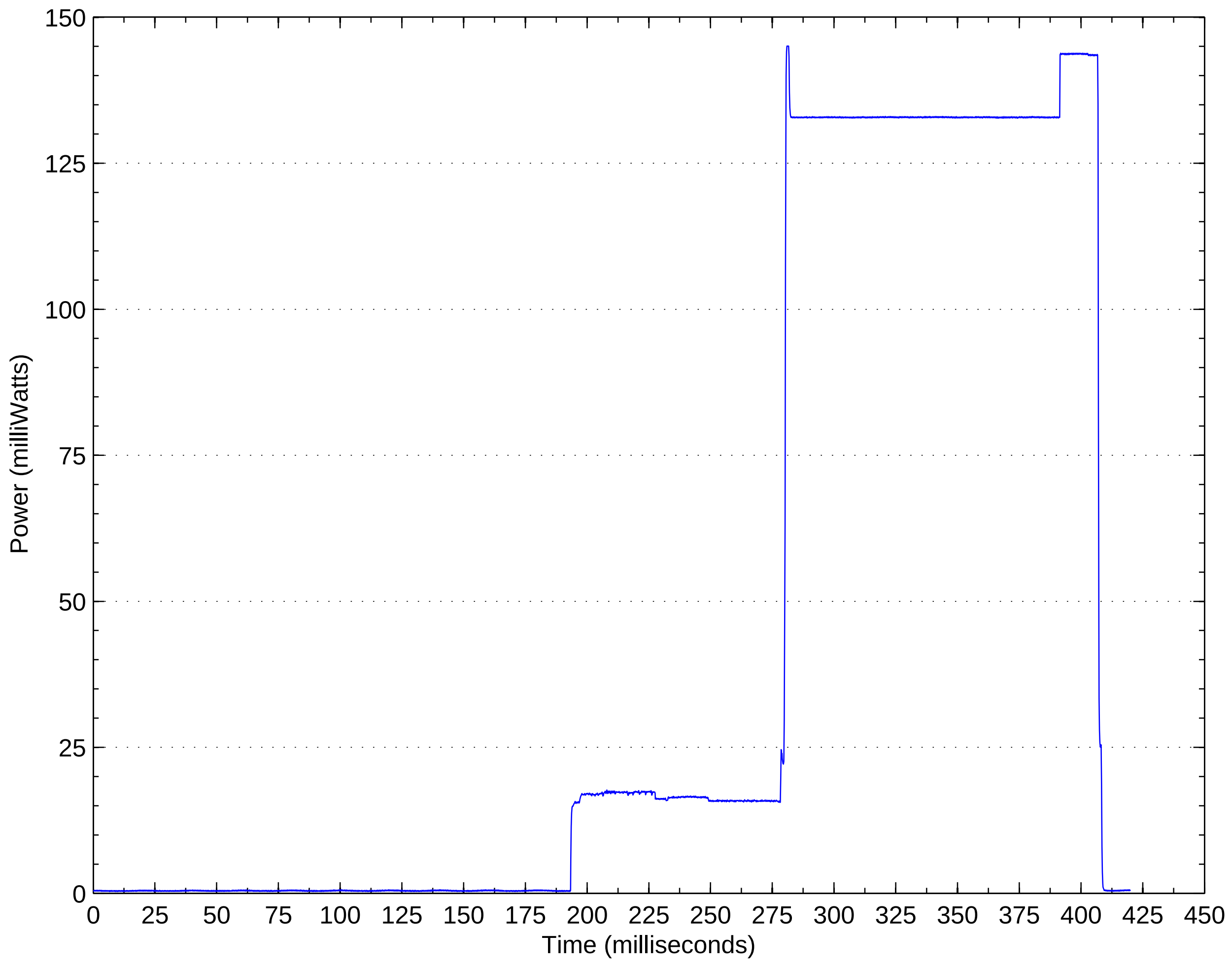


Figure 7.7: *A detailed look on the algorithm and transmission cost of an optimal energy setup. The first 'bump' is the algorithm, keeping the MCU active for 62 ms and then the large block of energy is the transmission, lasting 135 ms.*

7.3.3 Raw-data Profile

The raw-data scheme should transmit 17 frames of data, of which the last one is significantly smaller than the others. Inspection of the test data sets showed time duration to be approximately 8.2 seconds. The calculated minimum was 8.2 seconds, so there is very little overhead involved. Figure 7.8 shows these 17 frames as individual ”blocks” of power. Between the frames, power drops down to approximately 20 mW, which is because at this point the MCU is active, moving a packet into the radio hardware and initiating a transmission. Thus the power level does not drop down to zero. The ”spikes” at the top of the block is a result of the MCU waking up to retrieve sensor data.

The overall energy requires to transmit an entire set was found to be 1051.0 mJ.

An important note is that the sensor module gathers data at a sampling rate of 50 Hz. Outputting its results as 16-bit values for each of the axes, this causes the sensor to produce data at a bit rate of 800 bits per second per axis, or 2400 bits per second for all axes. As the radio transfers data at a 1200 bits per second, transmitting data from a single axis strains the radio-link to its maximum, and transmitting all of the data the sensor produces requires the bitrate to be more than doubled. This can be seen clearly in figure 7.8: When the last frame is sent, there is very little time left before the transmission of the next data set begins.

Overall the behaviour of the raw-data scheme is as expected, high network utilization but costly both in time and power.

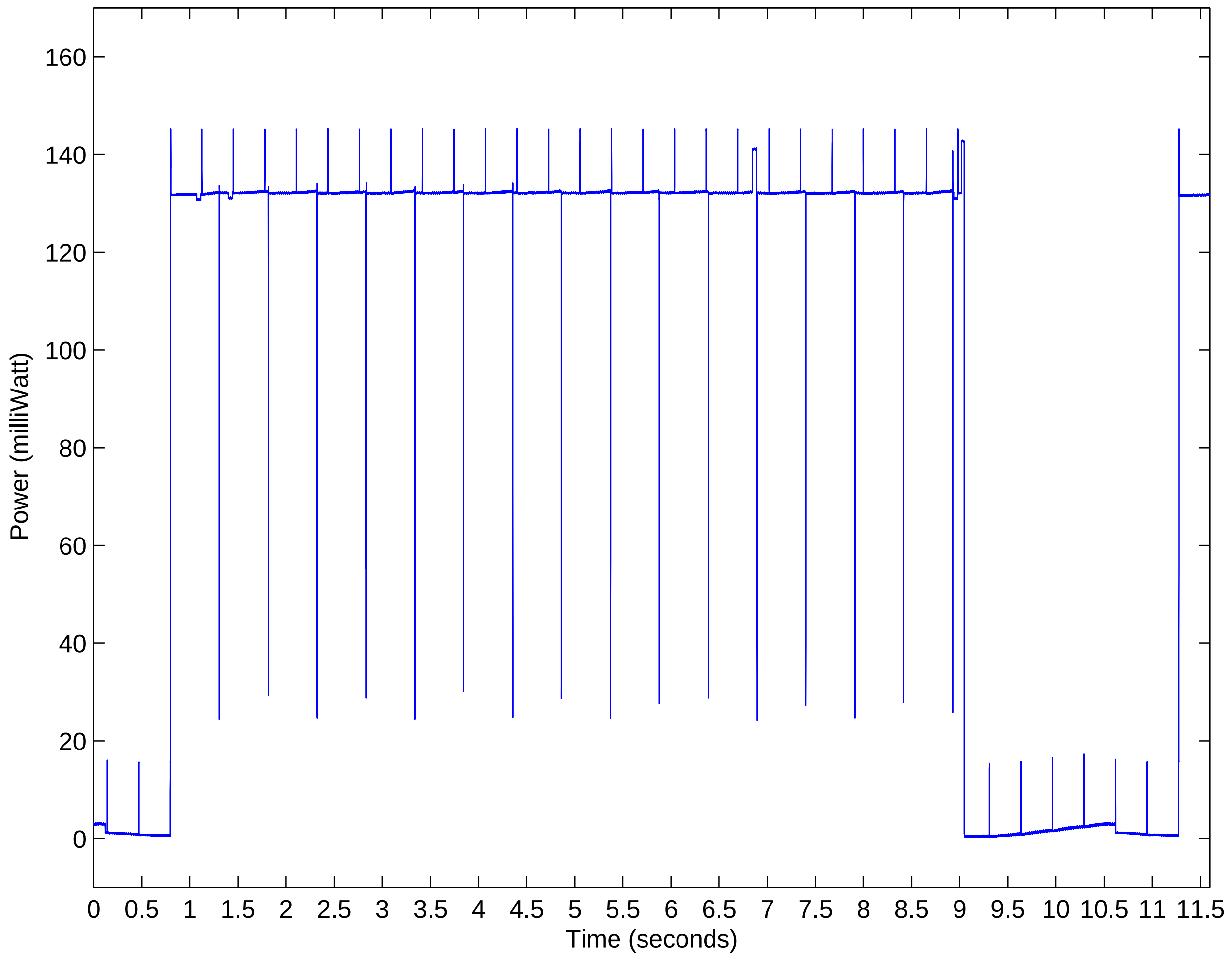


Figure 7.8: *Data transmission for the raw-data scheme. The vertical lines that drop down to ~20 milliWatts indicate the start of a new frame transmission. This can be used to distinguish the 17 frames sent. The start of the next data-set transmission can be seen as a vertical line to the right in the graph.*

.

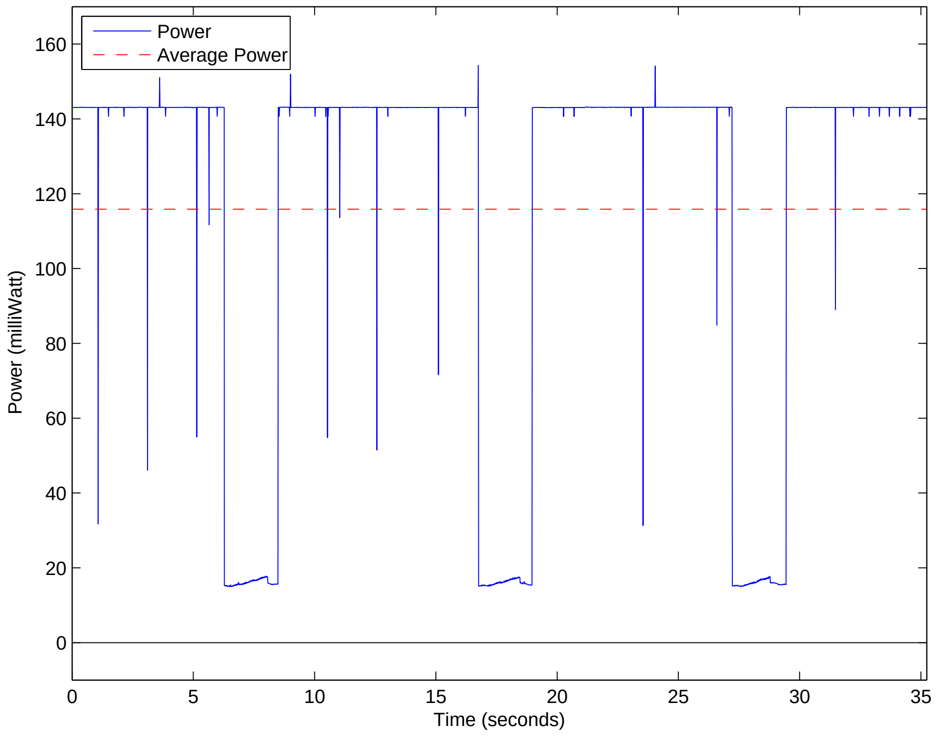
7.4 Test Deviances

Two particular test results are worth inspecting not for their qualities but for their deviances. Because of some difficultues with the program used to measure power (energyAware Profiler, see chapter 4), these two tests had to be made with an older version of the program. Ideally, all the tests would have been run with that version to achieve test consistency, but the sampling rate of the old version is lower than the new one. This difference is big enough to cause the small power drops between individual frames to be sampled only occassionally, decreasing the value of the test results. Because of this, the tests that were made before the measuring program failed are kept as they are.

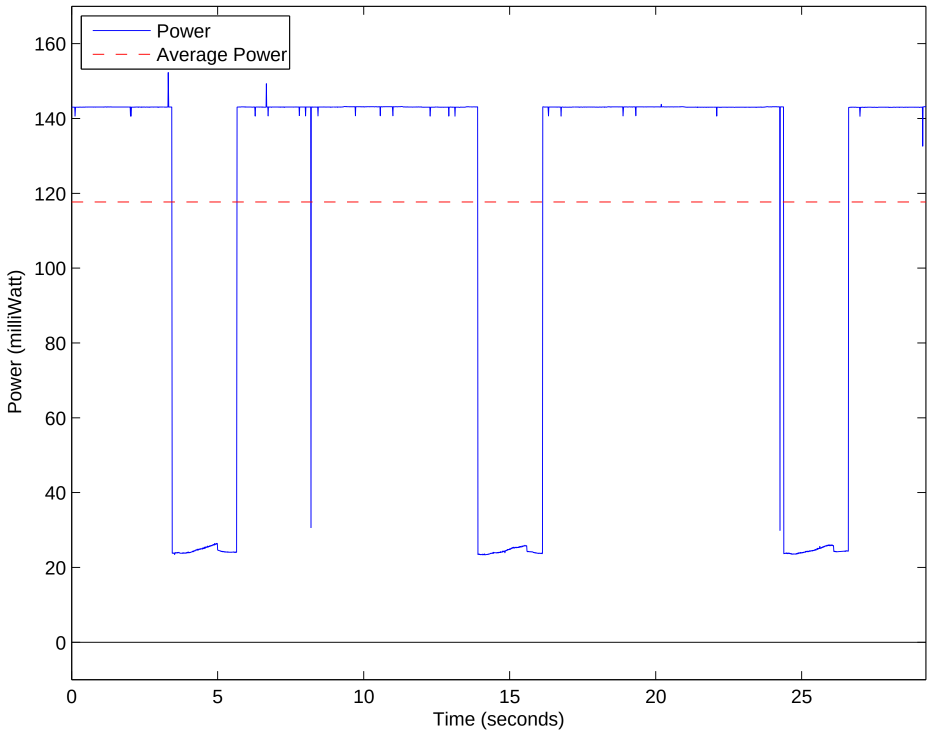
The two tests in question are almost identical in configuration: Both are set up to test the raw-data scheme without power-controlling the MCU, so that they differ only in whether the radio is power-controlled or not.

Because none of these tests are in any way an optimal solution, they are not used to do any further analysis and the author of this report reasons that this deviance does in itself not affect the quality of the test results overall.

The two tests were made with energyAware Profiler v0.995.



***(a)** Radio is power-controlled.*



***(b)** Radio is not power-controlled.*

***Figure 7.9:** The two test results that deviates from the test norm. The lower time resolution prevents the transmission frames from being clearly shown.*

7.5 Evaluation

7.5.1 Energy Savings

Transmission Scheme	Total Energy	Processing & Transmitting as % of total energy	Processing & Transmitting	Processing	Transmitting
No-Transmit	6.0 mJ	22.50 %	1.35 mJ	1.35 mJ	–
Local-Processing	24.3 mJ	80.58 %	19.58 mJ	1.35 mJ	18.23 mJ
Raw-Data	1056.5 mJ	99.95 %	1051.00 mJ	–	1051.00 mJ

Table 7.3: *A summary of energy consumption for the most important setups.*

The numerical results clearly indicate the energy saved by using the local-processing scheme.

An energy consumption change from 1056.5 mJ to 24.3 mJ is a 97.7 % reduction. Such a decrease is equivalent to an increase in battery lifetime of over 40 times, meaning that a battery that can drive a raw-data scheme for six months can drive a local-processing scheme for 20 years. This performance gain is primarily achieved through reductions in the data transmitted and secondarily through efficient power-control of the sensor module hardware. The raw-data scheme's power-profile is dominated by the large amount of data which needs to be sent, and without addressing this energy sink, other energy-optimizations appear insignificant.

7.5.2 Processing Trade-Off

The application of local-processing depends on two conditions. The first is that the receiving host system does not explicittly require the raw data of the sensor. If there is a need to store all sampled data for future analysis, or the data is only one of multiple inputs for some system-wide algorithm, the concept of local processing can not be applied. Second is the condition that the combination of the data processing and results transmission does indeed reduce energy consumption. If the energy consumption in the sensor module always consists of an administration part e_a (samling data and controlling the module), a processing part e_p and a transmission part e_{tx}, it is a condition that

$$e_{a,lp} + e_{p,lp} + e_{tx,lp} < e_{a,rd} + e_{p,rd} + e_{tx,rd} \tag{7.1}$$

Where *lp* denotes the local-processing scheme and *rd* denotes the raw-data scheme. If the administration part is assumed to be equal for both schemes, and one takes into account that the raw-data scheme performs no analysis, this is reduced to

$$e_{p,lp} + e_{tx,lp} < e_{tx,rd} \tag{7.2}$$

This is crucial to the application of the local-processing scheme. For this to happen, the data processing must be sufficiently conservative in energy consumption and it must reduce the amount of data transmitted such that (7.2) is satisfied. If the data processing consumes very little energy but does not reduce the amount of data which must be sent, the equation does not hold and neither does the conditions for using the local-processing scheme.

As the results presented here show, data processing has a much smaller impact on energy than transmission. As the data processing was found to use 15 mW, it can run for over a minute before reaching the energy sizes of the raw-data scheme.

The raw-data transmission cost 1051 mJ and the local-processing transmission cost 19.58 mJ. The data processing scheme thus has a spare 1031 mJ to consume before it becomes less efficient than the raw-data scheme. The algorithm used to represent data processing in this project consumed only 15 mW for 90 milliseconds when the MCU is ran at 14 MHz. The same algorithm would have to run for over 60 seconds to consume the available 1031 mJ. If the MCU was switched to run at 48 MHz, its top speed, and constantly process data, the power level of the MCU would rise by a factor of $48/14 = 3.42$ and thus the processing cost would approach $15mW * 3.42 * 10.24s = 527mJ$.

This suggests that the hardware used for this project is incapable of using more energy for processing than for transmitting data, as long as the processing causes sufficient reductions in the amount of data to be transmitted. Therefore it is reasonable to assume that the computing power which this hardware provides can be used quite liberally to achieve such a reduction.

7.5.3 Fault-Dependent Transmission

Even with the local-processing scheme, data transmission utilizes $18.23mJ/24.3mJ = 75.02\%$ or approximately three quarters of total energy consumption. The test setup used in this project uses an optimistic transmission protocol, with a small protocol overhead and no handshaking. An actual networking protocol would likely have to send more data than this project does. This suggests that further gains in energy performance cannot be achieved reducing the size of the data transmission, but instead in the number of transmissions made. This may be achieved by using a *fault-dependendent* transmission scheme, as was originally suggested in the introduction chapter. This has, as calculated, a theoretical reduction potential of 75.02 %, but it is unrealistic to expect a solution in which no sensor-to-host communication is performed until an actual blade fault is detected. But if a signal is only sent every third period, overall energy consumption would be halved, demonstrating its potential.

7.5.4 Further Reductions

The data processing algorithm used in this project does not necessarily represent the algorithm may eventually be chosen by the WiVind project, thus it must be taken into account that the processing cost may rise. Further, one can assume the communication overhead to increase, both through protocol size and the possible inclusion of heart-beat signals.

As even the local-processing scheme uses most of its energy on transmission and processing cost may increase, the most likely reduction potential comes from that of a fault-dependent transmission scheme, so that data is transmitted as rarely as possible.

The data period of the sensor module is 10.24 seconds, that is the amount of time it takes to produce a new set of data. If the heart-beat frequency is set to one signal per day, and the sensor module only transmits heart-beats or if a fault is detected, a sensor on a non-faulty blade would only transmit a frame every 0.000185 period. If one assumes that each transmission requires a more complex communication setup which causes a heart-beat signal to require ten times as much as energy as the frames in the local-processing scheme, one can set up an example of energy cost. Such an example would use the no-transmit energy and a fractional amount of energy on heart-beat signals:

$6.0mJ + (18.23mJ * 10 * 0.000185) = 6.034mJ$ spent per period.

This reduces transmission energy cost to a insignificant part of the total energy consumption, leaving only processing and administration cost. This is the scenario which is approximated by the no-transmit scheme.

For a wind-turbine designed to go for months without inspection, such a low heart-beat frequency is not entirely unreasonable. If sending a crew to inspect the blade and the module takes two weeks, and three lost heart-beats indicate a faulty sensor module (three days without contact with the sensor module), the frequency of the heart-beat signal has only a limited impact on the maintenance routines, but a large impact on the sensor's battery lifetime.

Based on this rationale it is reasonable to argue that a sensor module which only transmits when it detects a faulty blade rather than transmitting all of its sensor data can achieve order-of-magnitude energy savings. The argument holds even if the energy cost of sending a heart-beat signal or a fault-event increases ten-fold.

The main weakness of such a solution is the situation where the sensor module begins to transmit false positives. This may either cause an unwarranted maintenance mission to the wind-turbine or drain the sensor module's energy source prematurely. Reducing the probability of false positives is dependent on an overall reliable sensor module and a robust fault-detecting algorithm.

Thus, the fault-dependent scheme is a highly desireable approach to energy reduction. The communication protocol used to manage communication between the sensor module and the host system should be configurable with respect to heart-beat intervals and communication setup routines to minimize the energy loss of such procedures.

Chapter 8

Discussion

The test results presented in chapter 7 indicate that the idea of the thesis was correct: Energy consumed in the prototype is reduced by processing data locally and only transmitting the results, rather than transmitting all the raw data. The energy saved by this process is of such a magnitude that it is likely valid even if the specifications of the sensor module are different from what was used in this report.

There is also good cause to assume that without taking such a step, other techniques to reduce energy will have minimal impact. Only a radical approach to changing what data that is transmitted causes significant changes in energy consumption.

Performing the analysis of the sensor data within the sensor module may allow the sensor module to autonomously detect faults in the wind-turbine blade it is mounted on. This permits the sensor module to further minimize the use of the radio hardware, presenting the next step in energy optimization. The data transmitted by the local-processing scheme was shown to be mainly protocol overhead, which supports this idea. Once such a local processing is in place it is the frequency of transmissions that becomes the issue and not the size of the payload data.

This observation suggest that in cases where multiple sensor modules on the same blade must cooperate to detect faults, the development of the module should aim to make the co-operation algorithm depend on as little inter-module traffic as possible, as this would quickly become the largest energy sink.

The tools used to test the system was able to sample information about both the power consumption (voltage and current) and software state (program counter and interrupt vector) and store this at a desktop computer. These tools' limitation of only a single pair of voltage/current-channels and somewhat unreliable data storing caused some limitation in the tools' value. The ability to only make one power measurement for the entire prototype caused the connection between energy and software to only be a time-domain correlation. A multi-channel tool could possibly have allowed the causality between energy and software to have been easily identified. Further work on the same prototype should be made with these limitations in mind.

This report developed a sensor module prototype to be used as a testbench. The specifications of that prototype can not be guaranteed to match the specifications which the WiVind project eventually may set up for its own sensor module. Therefore the results presented here are not automatically valid for the WiVind project. The prototype developed for this thesis depends on its own data analysis algorithm and transmission protocol which therefore both are in risk of being too optimistic with regards to the energy consumption.

However, because the savings in energy consumption are so significant, this project report still presents a strong argument that the solution suggested here should be investigated also by the WiVind project. As the WiVind project matures, more accurate prototypes should be possible, which may either help to confirm or discard the results in this report.

Chapter 9

Conclusion

The motivation for the work in this thesis was to investigate energy-reducing techniques for the WiVind project. For this purpose, a sensor module prototype was to be developed and used as a testbench for such techniques.

This prototype was successfully built and used to verify the assumption of the thesis: That the most promising approach to reduce energy is to process sensor data locally within the sensor module and only communicate with the host system when the sensor module has detected a fault. It is also shown that any heart-beat signals used to indicate a non-faulty sensor module may only have a limited impact on long-term energy consumption.

If these techniques are not applied, it is reasonable to expect that any other technique will only have a very limited potential. A valid decision on exactly which techniques to prioritize and their impact is subject to a successful identification of the system's energy sinks, their size and what can potentially be done about them.

The specifications of the sensor module prototype used in this project is expected to be only an approximation to what will eventually be valid for the WiVind project. The results in this report are therefore quantitatively valid only for the work done here, but qualitatively relevant for the WiVind project and other projects which wish to perform low-energy wireless sensoring.

A distributed sensor system in which multiple modules must cooperate to detect faults in the blade represents entirely new challenges and work in this field can be expected to revolve around the energy-optimization of wireless sensor networking. Progress of the WiVind project is likely to allow for thesis work similar to what is done here but with far more accurate system specifications, which may lead to test results of increased value.

In general it can be concluded that event-based communication is a highly valuable approach for the WiVind project, provided that the fault-detection algorithm can be run locally and that necessary communication such as heart-beat signals can be set to an optimal frequency.

Appendix A

Hardware Configurations

Register Name	Register Address	Register Value
ADXL345_DATA_FORMAT	0x31	0x0B
ADXL345_POWER_CTL	0x2D	0x08
ADXL345_INT_ENABLE	0x2E	0x02
ADXL345_INT_MAP	0x2F	0x00
ADXL345_BW_RATE	0x2C	0x09
ADXL345_FIFO_CTL	0x38	0x56

Table A.1: *Sensor register configurations*

Register Name	Register Address	Register Value
CC120X_IOCFG2	0x0001	0x14
CC120X_DEVIATION_M	0x000A	0xD1
CC120X_MODCFG_DEV_E	0x000B	0x00
CC120X_DCFILT_CFG	0x000C	0x5D
CC120X_PREAMBLE_CFG0	0x000E	0x8A
CC120X_IQIC	0x000F	0xCB
CC120X_CHAN_BW	0x0010	0xA6
CC120X_MDMCFG1	0x0011	0x40
CC120X_MDMCFG0	0x0012	0x05
CC120X_SYMBOL_RATE2	0x0013	0x3F
CC120X_SYMBOL_RATE1	0x0014	0x75
CC120X_SYMBOL_RATE0	0x0015	0x10
CC120X_AGC_REF	0x0016	0x20
CC120X_AGC_CS_THR	0x0017	0xEC
CC120X_AGC_CFG1	0x001B	0x51
CC120X_AGC_CFG0	0x001C	0xC7
CC120X_FIFO_CFG	0x001D	0x00
CC120X_FS_CFG	0x0020	0x12
CC120X_PKT_CFG0	0x0028	0x20
CC120X_PA_CFG1	0x002B	0x3F
CC120X_PKT_LEN	0x002E	0xFF
CC120X_IF_MIX_CFG	0x2F00	0x1C
CC120X_FREQOFF_CFG	0x2F01	0x22
CC120X_MDMCFG2	0x2F05	0x0C
CC120X_FREQ2	0x2F0C	0x56
CC120X_FREQ1	0x2F0D	0xCC
CC120X_FREQ0	0x2F0E	0xCC
CC120X_FS_DIG1	0x2F12	0x07
CC120X_FS_DIG0	0x2F13	0xAF
CC120X_FS_CAL1	0x2F16	0x40
CC120X_FS_CAL0	0x2F17	0x0E
CC120X_FS_DIVTWO	0x2F19	0x03
CC120X_FS_DSM0	0x2F1B	0x33
CC120X_FS_DVC0	0x2F1D	0x17
CC120X_FS_PFD	0x2F1F	0x00
CC120X_FS_PRE	0x2F20	0x6E
CC120X_FS_REG_DIV_CML	0x2F21	0x14
CC120X_FS_SPARE	0x2F22	0xAC
CC120X_FS_VCO0	0x2F27	0xB5
CC120X_XOSC5	0x2F32	0x0E
CC120X_XOSC1	0x2F36	0x03

Table A.2: *Radio register configurations*

Appendix B

Results Compendium

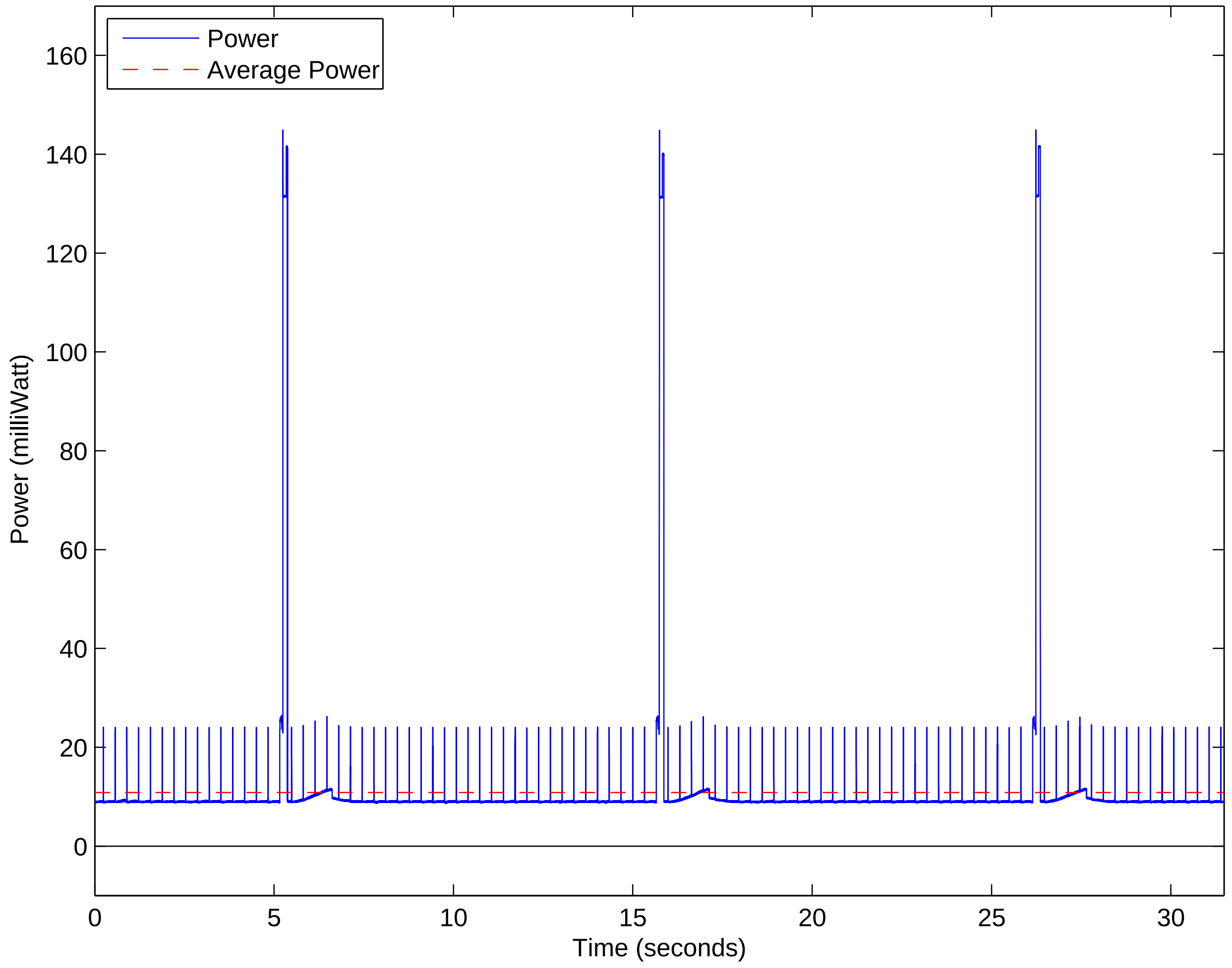


Figure B.1: *Local-processing scheme and power control on the MCU only. Average power: 10.83 milliWatts*

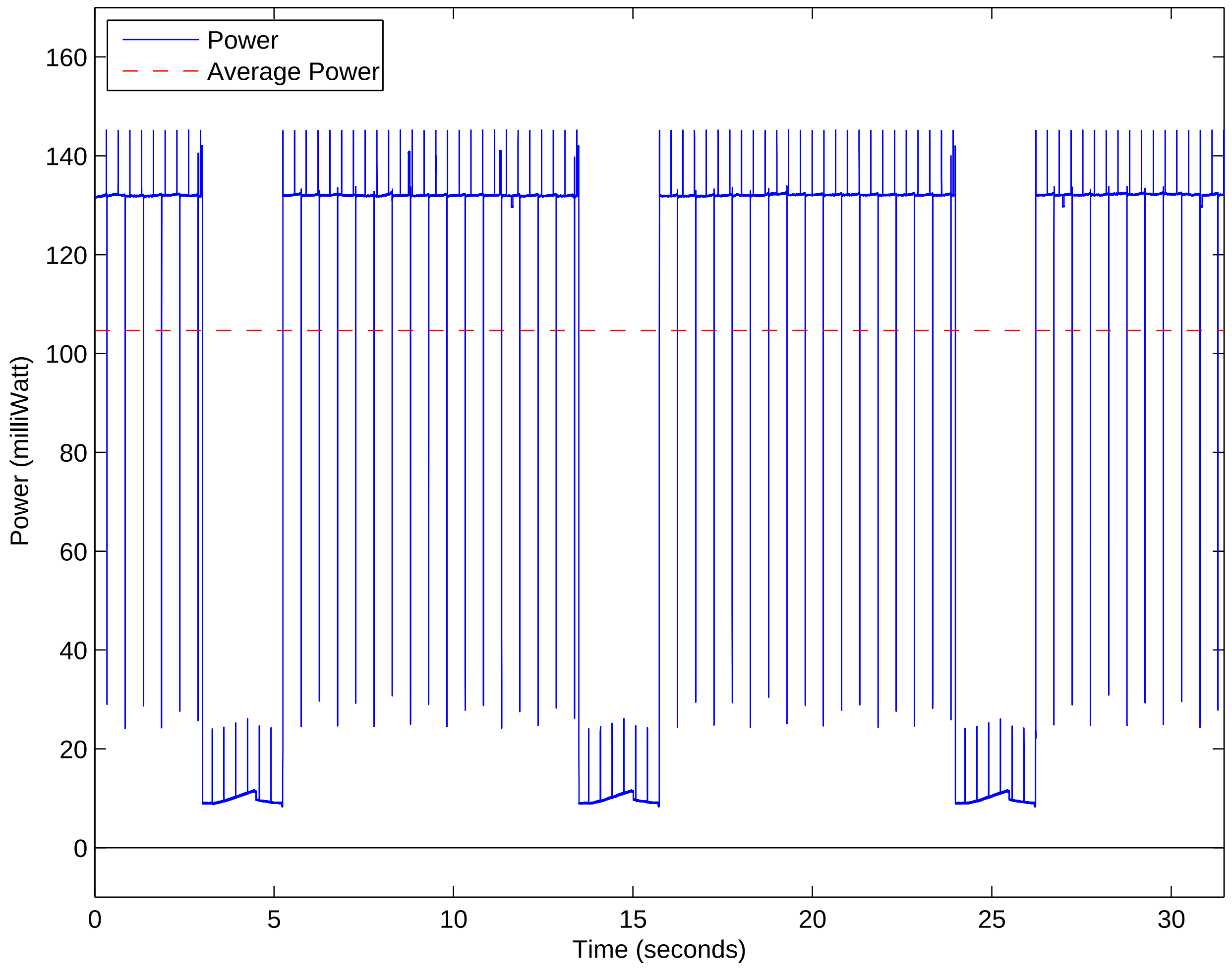


Figure B.2: *Raw-data scheme and power control on the MCU only. Average power: 104.66 milli-Watts*

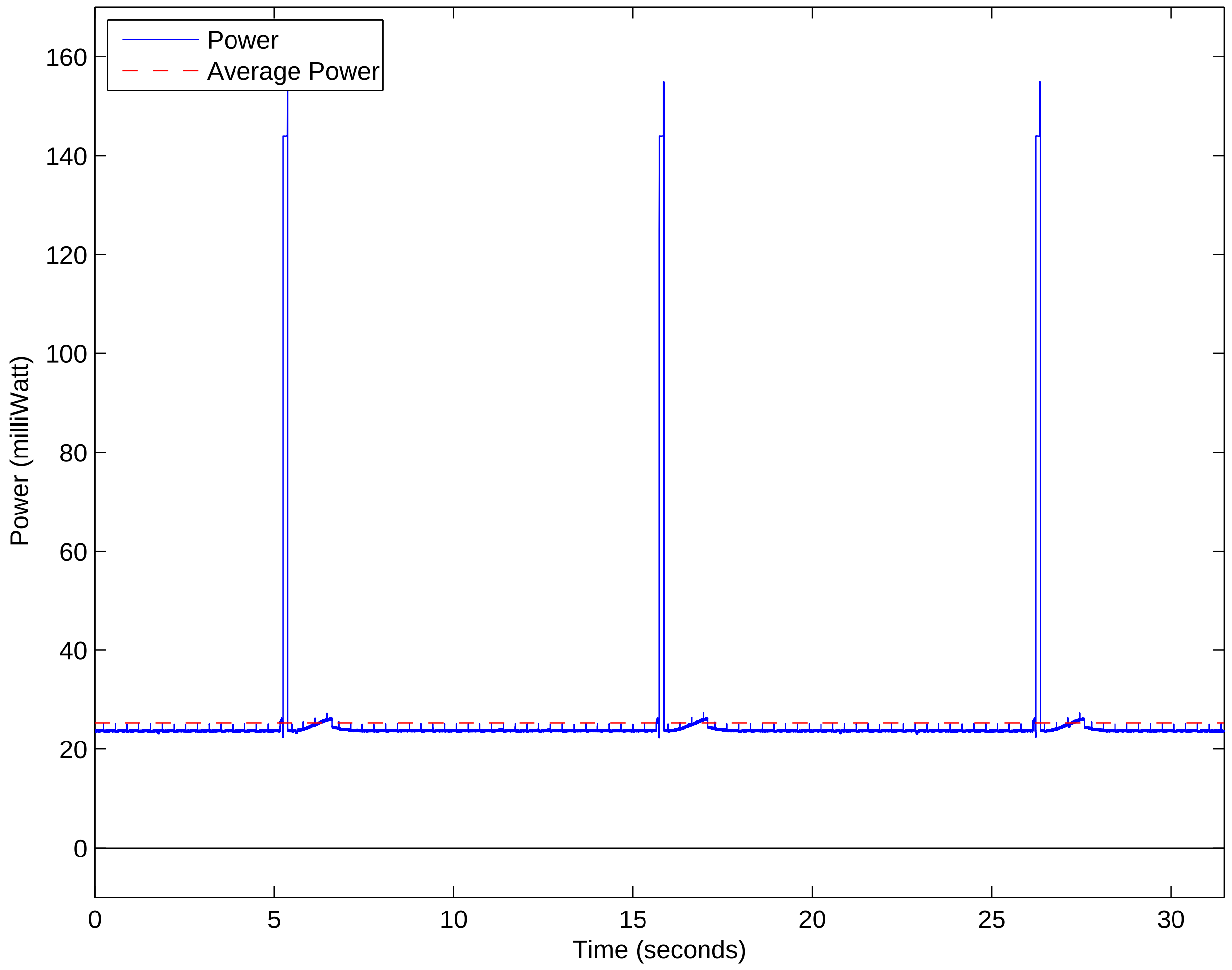


Figure B.3: *Local-processing scheme and no power control. Average power: 25.29 milliWatts*

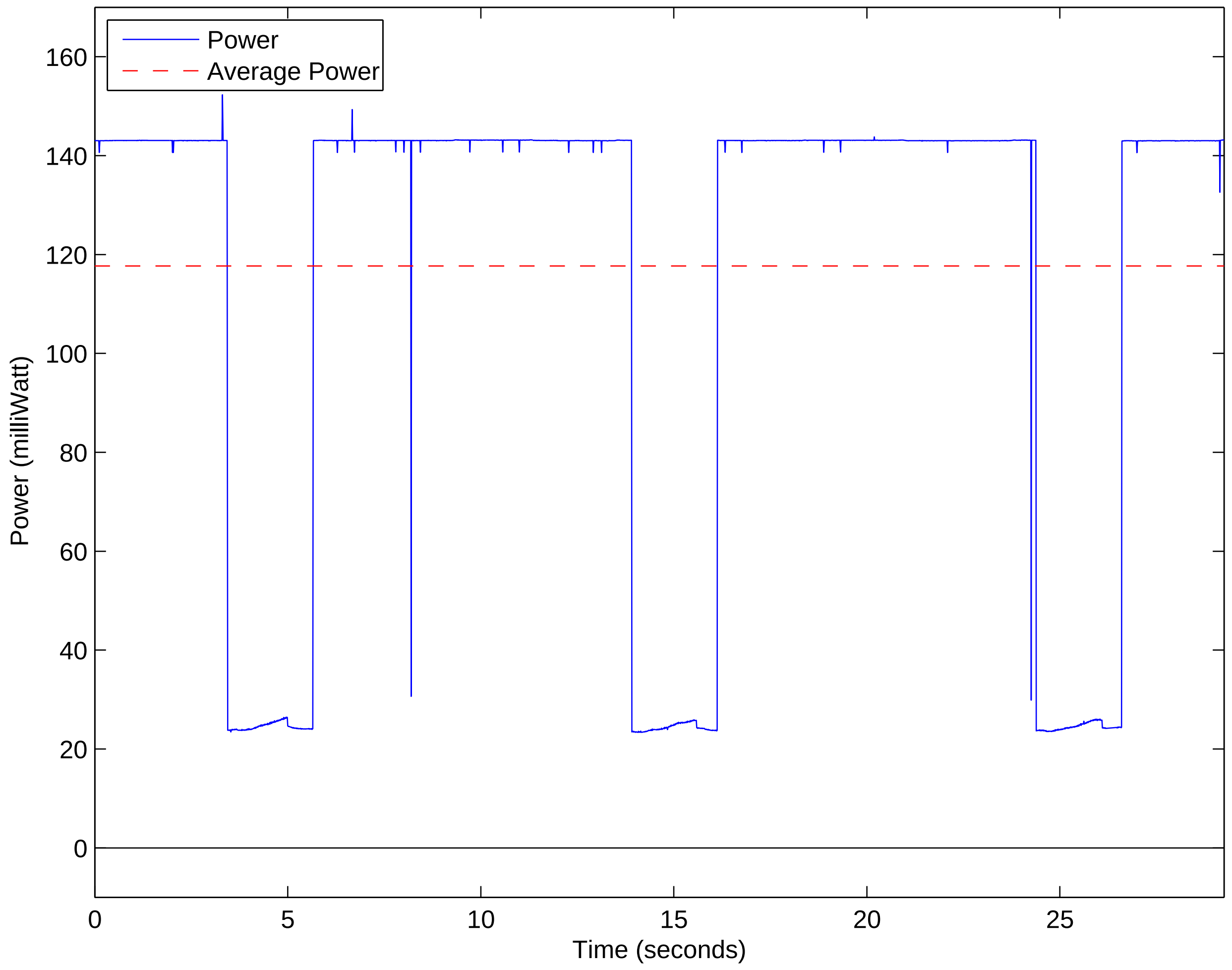


Figure B.4: *Raw-data scheme and no power control. Average power: 117.70 milliWatts*

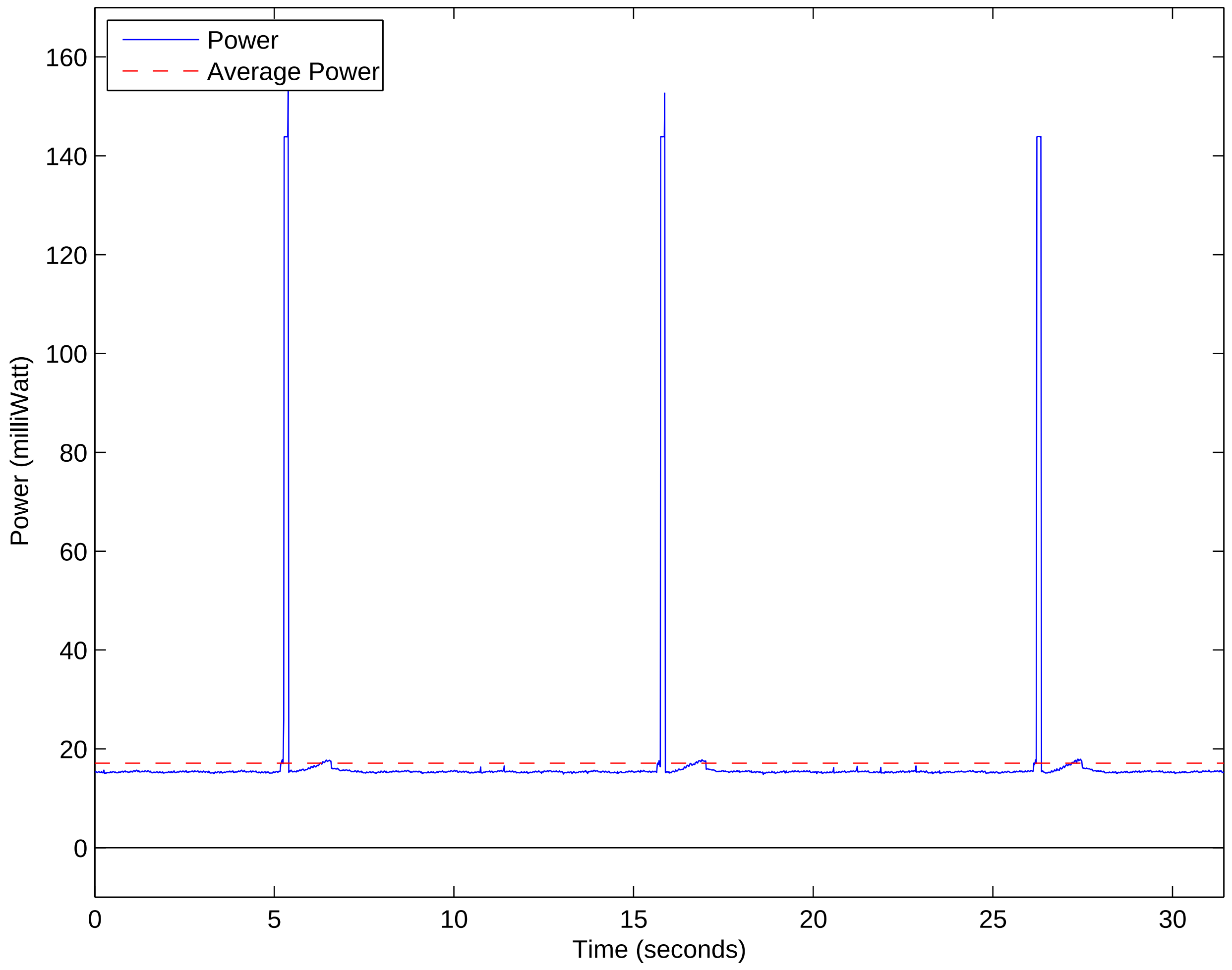


Figure B.5: *Local-processing scheme and power control on the radio only. Average power: 17.10 milliWatts*

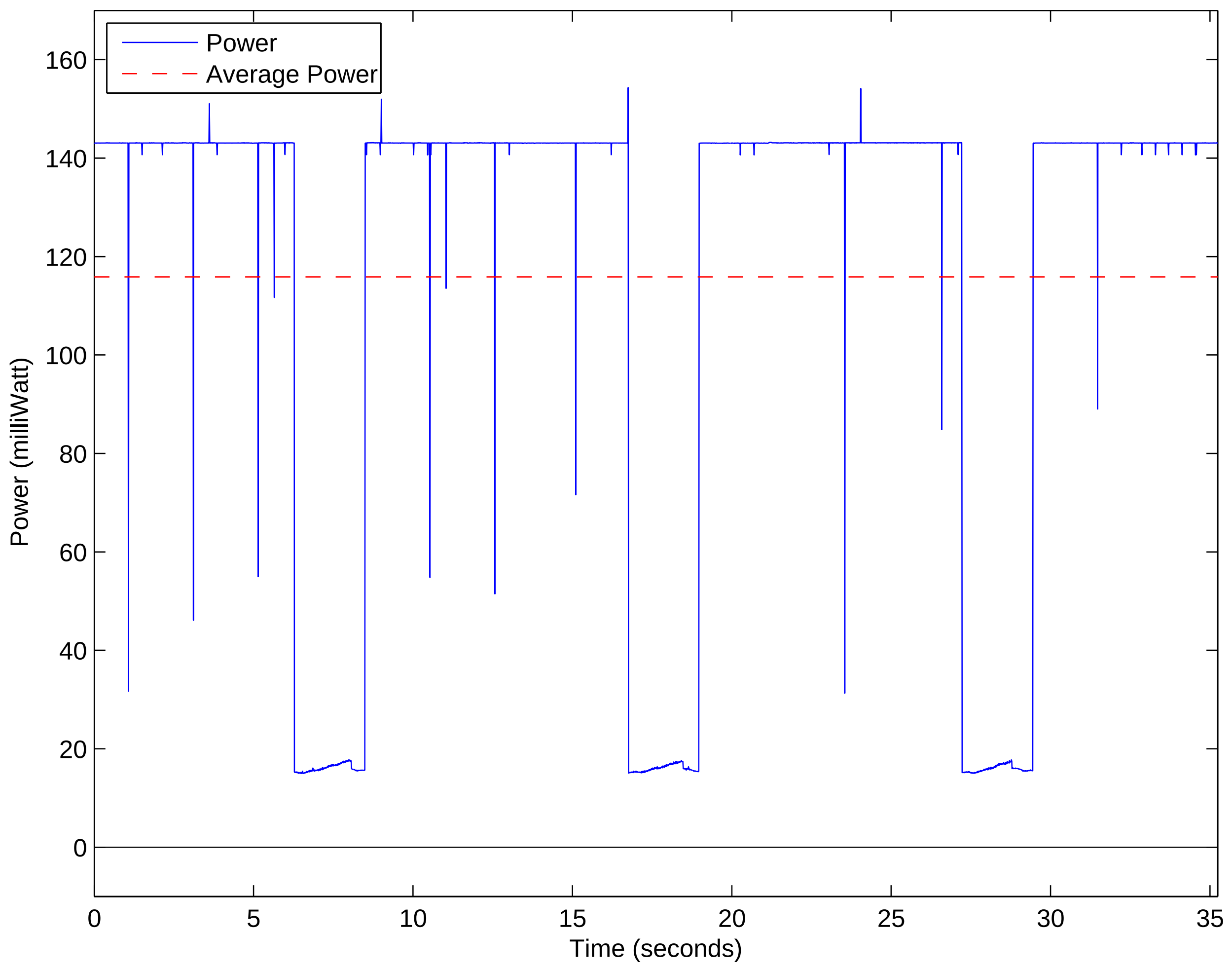


Figure B.6: *Raw-data scheme and power control on the radio only. Average power: 115.84 milli-Watts*

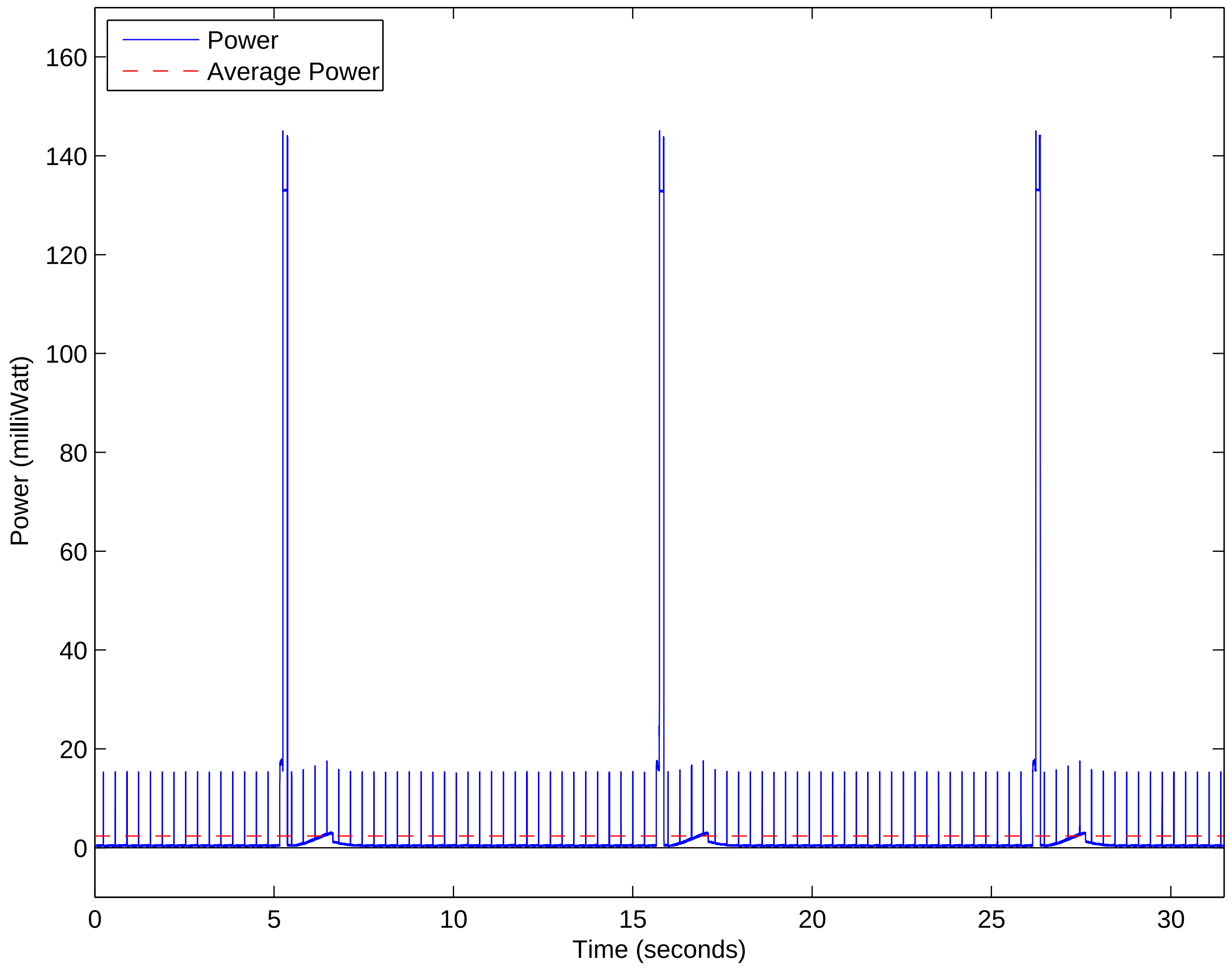


Figure B.7: *Local-processing scheme and power control on both MCU and radio. Average power: 2.37 milliWatts*

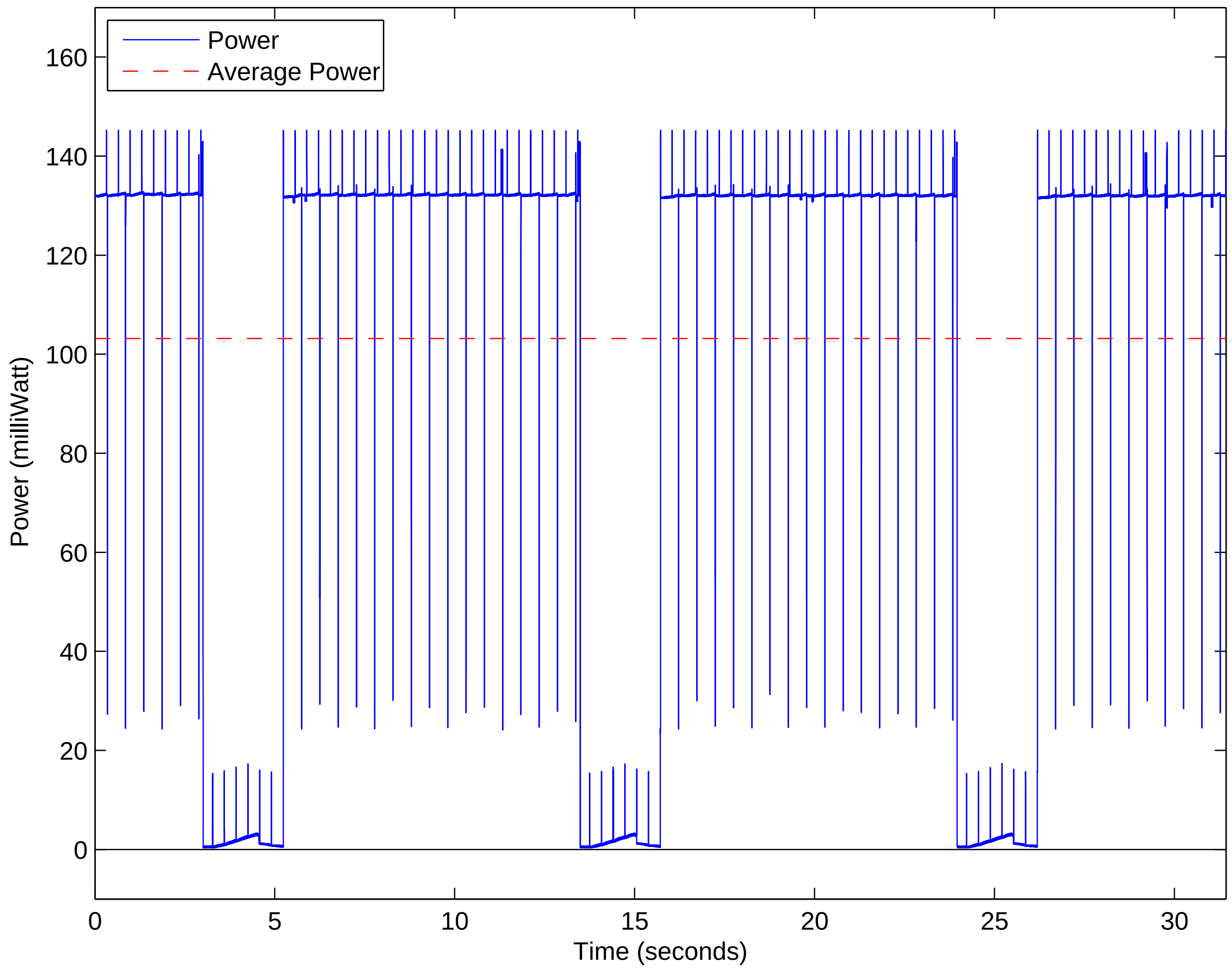


Figure B.8: *Raw-data scheme and power control on both MCU and radio. Average power: 103.17 milliWatts*

Appendix C

Digital Files

In alphabetical order there are five categories of files.

The matlabscripts contains a bundle of Matlab Script files that were written to work with the test data files used in the project work. They import data to Matlab, store data sets in .mat-files and produce plots for the thesis report.

The datasources is the set of test data which was sampled by the energy-aware profiler. They are named as follows:
powermeter_#1_MCU#2_R#3_5min.csv
Where #1 indicate whether it tests a (L)ocal-processing scheme, an (R)aw-data scheme or a (N)o-transmit scheme, #2 indicates either a (M)anaged MCU or an (U)nmanaged MCU and #3 indicates either a power-(C)ontrolled radio or an (U)ncontrolled radio.

The hostsystem zip-file is the source code for the host system, stored as an IAR workbench project. The src zip-file contains the source code developed for the sensor module. The wivind zip-file contains the entire sensor module software as an Simplicity Studio software project.

Bibliography

[1] Janusz Bryzek. Impact of MEMS technology on society. *Sensors and Actuators A: Physical*, 56(1):1–9, 1996.

[2] ECC. ERC recommendation 70-03. Regulation, Electronic Communications Committee, February 2014.

[3] Andrea Goldsmith. *Wireless communications*. Cambridge university press, 2005.

[4] ITU-R. Radio regulations. Regulation, International Telecommunications Union – Radiocommunication Sector, 2012. Volume I, Chapter II, Article 5, paragraph 5.138 and 5.150.

[5] Khurram Shahzad, Peng Cheng, and Bengt Oelmann. Architecture exploration for a high-performance and low-power wireless vibration analyzer. *Sensors Journal, IEEE*, 13(2):670–682, 2013.