Power Efficient Duty-cycling with Ultra Low-power Receivers

M.Sc. Thesis

Marcus Chang
marcus@diku.dk

Department of Computer Science
University of Copenhagen
Denmark

June, 2006

Typesetted with LaTeX 2_ε

Abstract

In this thesis, we investigate the use of ultra low-power receivers to perform power efficient communications by exploring two orthogonal approaches to the problem.

First, by using the DCF77 time-code signal, we are able to synchronize a Wireless Sensor Network (WSN) to within (3.3 ± 1.3) ms accuracy, without the use of the conventional radio. We identify the limiting factor to be the time-code decoding chip and an analysis reveals that with a more efficient decoding chip the platform supports μs precision.

We subsequently use this global synchronization in our scalable, self-configuring, scheduled communication protocol, and achieve a power consumption equivalent to a duty-cycle of the primary radio of 0.4% - 0.001 %, which is at least an order of magnitude better than similar protocols. This combination of out-of-band synchronization together with self-configuration makes deployment and joining a WSN completely transparent.

Secondly, we explore the use of out-of-band signalling to reduce latency and improve duty-cycling by implementing a low-power, low transmission rate secondary radio. We show that we can achieve 73 ms latency with one-hundredth of the amount of energy normally required to have no latency in the network. By using this secondary radio to facilitate deployment and joining a WSN we achieve a similar reduction of power consumption.

When used in power efficient discovery, however, we find that the low-power consumption will eventually become a significant problem, and regular scheduled communication based on timers will be more energy efficient. By combining the secondary radio with scheduled communication into a Scheduled Discovery Protocol we achieve a scalable, self-configuring, protocol that outperforms even our scheduled communication protocol based on DCF77 synchronization.

keywords: Wireless Sensor Networks, TinyOS, DCF77, time synchronization, out-of-band synchronization, low-power listening, out-of-band signalling, Freescale EVB13192

Acknowledgements

A special thanks goes to Jørgen Kragh Jakobsen at Oticon for helping me out with the electronic aspects of this thesis, especially with choosing the correct hardware in the critical start-up phase.

On the same note, a special thanks to Jesper Noes Rasmussen at Power Play Electronics for helping me build the prototype boards, and lending me the necessary measuring instruments to perform detailed hardware analysis.

I cannot thank Jan Flora enough for showing me how to get the upper hand on the Freescale motes. Especially for helping me access the mote's hardware and use the radio as a clock source.

To Esben Zeuthen for setting me up on the Re-Mote testbed and helping out when the testbed started acting up.

My adviser Philippe Bonnet for supervising this project.

To Randi Laursen and Henrik Juul Nielsen goes my thanks for proofreading this thesis.

Finally, I am grateful for the support from my friends and family through all these years. Thank you!

Contents

List of Figures

List of Tables

1 Introduction

We first present the context in which this thesis takes part. Secondly, we discuss the problems we wish to treat together with the approach we wish to seek. Finally, we report on our main contributions and give an outline for the rest of this thesis.

1.1 Context

1.1.1 Wireless Sensor Networks

A Wireless Sensor Network (WSN) is characterized by a network of small autonomous units equipped with sensors and radio. These units form a wireless network and solve a given task in collaboration with each other. The units are often referred to as motes, as the vision of WSN is to have dust sized units, and they are thus typically equipped with limited processing and storage capabilities.

The collected data from the sensors are typically either stored in the network for later retrieval or transmitted to a base station for out-of-network processing and storage. This base station is often referred to as the gateway as it connects the motes on the WSN with external computers. Because the physical layout of a fully deployed WSN often means that not all of the motes are within communication range of a base station, data are routed through the network. This multi-hop routing implies that at least some of the motes must be able to work as routers.

The motes are usually equipped with their own power supply which makes power consumption a key constraint in WSN. As it is often impossible to recharge or replace batteries in a fully deployed WSN, the power supply puts an effective limit to the motes' lifespan. The most effective way to conserve energy is to keep all but the necessary components turned off. The ability to turn on components only when they are needed, and otherwise have them turned off, is known as duty-cycling. The difference in lifespan between a mote with all its components turned on and a duty-cycled mote can be of several orders of magnitude. Duty-cycling is thus an integral part of WSN.

1.1.2 TinyOS

The key constraints when developing a WSN application are the limited resources on each device. With limited power supply, processing power, and storage, the energy consumed during processing, communicating, and sensing must be optimized throughout the system and across the different layers in order to achieve a maximum lifespan for the motes. TinyOS [1] is an open-source operating system built specifically for WSN. It uses an event driven execution model where incoming events post tasks to a single task queue. All tasks in this queue run to completion in a FIFO order and are only interrupted by events.

With a component based hierarchical architecture, the boundary between hardware and software is clearly defined due to each component is either implemented in hardware or software. However, although clearly defined, the boundary itself is highly flexible as software/hardware functionality can easily be moved to hardware/software, an entire component at a time, without affecting the rest of the system. This component abstraction can also be used for cross-layer optimization and efficient duty-cycling by turning entire components off.

Also, this component-based architecture minimizes the overall program size because only essential components are included. The TinyOS core itself only requires 400 bytes of code and data memory combined, and by only including essential components the overall program size can be kept at a minimum. All these features make TinyOS a much preferred operating system in the WSN community.

1.1.3 Hogthrob

The Hogthrob project commenced in February 2004 and has a duration of three years. The project's goal is to build a sensor network infrastructure for online sow monitoring. The parties involved are:

- Informatics and Mathematical Modeling, Technical University of Denmark
- Dept. of Computer Science, University of Copenhagen
- Dept. of Animal Science and Animal Health, The Royal Veterinary and Agricultural University
- National Committee for Pig Production
- IO Technologies

For veterinarians efficient and accurate sow monitoring can facilitate the collection of substantial empirical data for building a precise model of the sow behavior. Current sow monitoring equipment is restricted to RFID tags placed on each sow's ear, identical to earmarking technology used to register domestic cats.

With these tags feeding monitoring can be performed by placing RFID readers at the feeding stations. The time each sow spend at the trough can then be measured and logged in a database. Similarly, the heat period of a sow can be detected by placing a boar in a nearby paddock, and subsequently monitor which sows are particularly interested in the boar. These methods are, however, neither efficient nor effective because there exists a strict hierarchy in a pigsty, and lower ranking sows might be too intimidated to go near the trough and the boar.

If a sow needs special attention, finding it using solely the RFID tag requires the veterinarian to manually scan each sow with a RFID reader. This process is both

cumbersome and time consuming. Clearly, the process of finding a particular sow can be made more efficient by using an active tag capable of distinguishing itself from the others by emitting some sort of signal when given a specific command.

A recent field test, designed to measure the activity of the individual sows, was performed by [3]. In this experiment, acceleration data were collected continuously and stored on each mote with the radio turned off in order to save power. When the storage was almost full, the radio was turned on briefly to allow burst transmissions of the collected data to a nearby gateway. To facilitate ordering and correlation of data, the measurements were timestamped on each mote. In order to further save power, it is planned for future experiments for the motes to be inactive for at least several weeks after a heat period is detected, as the interval between heat periods is of this scale.

1.2 Problems

The key issue we wish to investigate in this thesis, is the duty-cycling of the radio and the problems and trade-offs this causes. Due to the radio being an integral part of any WSN and easily the one component with the highest energy consumption, an efficient duty-cycling of the radio can easily lead to the highest energy savings in the system.

The direct consequence of duty-cycling the radio is its effect on communication. The challenge is to know when someone is transmitting a message, so the radio can be ready to receive, and, conversely only transmit when it is certain that someone is listening, since anything else will be a waste of power.

According to [2], there are two basic approaches to perform duty-cycled communication:

- Scheduled communication (synchronized)
- Power efficient discovery (unsynchronized)

These will be discussed in detail in the following sections.

The indirect consequences are those that are caused by the communication channel not always being available. Since, in principle, there are as many problems as there are applications that uses a duty-cycled communication channel, we will focus on two problems that are related to the Hogthrob project and in particular the experiment mentioned above. We will specifically look at issues regarding:

- Latency

- Deployment

These will also be discussed in the following sections.

1.2.1 Scheduled Communication

In scheduled communication, the transmitting mote and the receiving mote(s) have synchronized their internal clocks, and based on either a predetermined or a negotiated schedule, the transmitting mote knows when a communication window occurs on the receiving mote(s) and can, thus, transmit accordingly. This is similar to TDMA except that the purpose of the time division is power savings and not media fairness. The weaknesses are, however, the same as both have the problems of keeping the clocks synchronized, assigning schedules to each mote and making this schedule available to the other motes either by assignment at deployment time or negotiation at operation time.

Whether the schedules chosen are complex and unique to each mote or simple and highly generic, assigning schedules can be done at deployment time, and although it can be a big scalability problem this is completely free with respect to power consumption. Keeping the clocks synchronized, however, is far from free. Due to clock jitter[1] and clock drift[2] even a synchronized network will eventually lose synchronization and resynchronization must be performed.

Because of all synchronization protocols to some extent are based on exchanging messages, and since radio communication is the most energy consuming operation, the cost of keeping a WSN synchronized can be very expensive if the period between data messages is longer than the necessary synchronization messages. However, if data messages are sent frequently, compared to synchronization messages, synchronization can be performed almost free by exploiting the data messages to piggyback synchronization information. The time between resynchronization can be minimized by using high-precision clocks. These are, however, often expensive to buy and in some cases also more energy consuming than the regular ones.

In essence, the problem of performing scheduled communication in a power efficient way can be reduced to a time synchronization problem. Besides scheduled communication, time synchronization is essential in many applications e.g. in the above Hogthrob experiment the timestamping of measurements makes it possible to correlate measurements with video recordings. Thus, an efficient and precise time synchronization would be of general interest. Hence, we state the problem:

> How can we perform time synchronization in a power efficient way?

[1] The accuracy with which a clock can be read

[2] The relative error in the precision of a clock's time measuring ability.

1.2.2 Power Efficient Discovery

When two unsynchronized motes, both operating with a duty-cycled radio, wish to communicate with each other, a discovery phase must take place first. This usually consists of the would-be receiving mote broadcasting a beacon followed by a short communication window. A mote that wishes to initiate communication must listen for this beacon and make a transmission within the following communication window.

The obvious weaknesses of this approach are the conflicting needs of the receiving and transmitting mote: The receiver wishes to have a long period between each beacon as a beacon transmitted with no motes listening is wasted. Similarly, the transmitter wants a short period between each beacon because this would give a shorter period of wasted listening. Depending on the frequency between data messages, it can be advantageous to have a short beacon period if the frequency is low as this would imply that the transmitter must have many but short listening periods. Correspondingly, if the frequency between data messages is low, a long beacon period would imply that the transmitter would have few but long listening periods.

In essence, the problem with discovery lies in the long periods of wasted listening for a beacon, and the wasted beacon transmissions when no one is listening. With the real issue being that the radio generally is the most energy consuming component on motes. Hence, we state the problem:

> How can we reduce the cost and/or periods of idle listening?

1.2.3 Latency

A common problem with both approaches mentioned above, and to duty-cycled operation in general, is how to communicate with a radio that is turned off. The simple answer to this is to wait until the duty-cycling turns the radio on again. This trade off between low-power operation and high latency is a fundamental side-effect of duty-cycling and must be solved by other means.

In the Hogthrob project, it is essential for the veterinarian to be able to find a particular sow, quickly. If the radio is turned on, this can be as simple as transmitting a message to the mote carried by the sow instructing it to either flash a light or emit a sound. However, if the radio is turned off, the veterinarian would have to wait until the next communication window opens which could easily be in several days or even weeks which would be unacceptably long. What is needed is another way to turn on the radio besides the internal clock. Hence, we state the problem:

How can we reduce the latency from communicating with a duty-cycled mote?

1.2.4 Deployment

Closely related to synchronization is the problem of deployment. Without synchronization a big amount of energy is being consumed at the deployment phase in order to establish a common starting point for the experiment. The reason for this common starting point could be either to minimize power consumption as the first mote deployed has to wait until the last mote is deployed, in order to begin duty-cycling simultaneously, or to synchronize data collection with a specific event e.g. the beginning of a month.

This problem is in many ways similar to the issues of joining an operating WSN where a new mote must listen for a potentially very long period in order to estimate the intervals of the duty-cycle periods. If new motes could join an operating WSN without consuming too much energy, this could potentially reduce the power consumption of deployment as well because the motes deployed first could start without causing the following motes to consume too much power. This only holds if synchronized data collection is not needed.

This deployment issue is a fundamental problem with WSNs, and becomes an even bigger issue when combined with scalability. In the Hogthrob project, the measurements performed by the motes are complemented by video footage of the sows. It is thus of particular interest to be able to start both the measurements on all the motes simultaneously with the video recordings. Hence, we state the problems:

How can we:

- reduce the power consumption at deployment time?
- reduce the power consumption of joining an operating WSN?

1.3 Approach

We wish to investigate how ultra low-power receivers can be used to solve the problems mentioned above. Ultra low-power receivers are particularly interesting because they use significantly less power than the normal radios found on sensor motes. This reduction can easily be of several orders of magnitude with normal radios operating in the 100 mW range and ultra low-power receivers at merely 0.2 mW. This figure is particularly interesting because in the area of energy scavenging a general rule of thumb is that whether it is solar power, thermal energy, wind energy e.t.c., it is always possible to extract 0.1 mW from the environment. Although,

we are not quite there yet the technology is close to make idle listening completely free with regards to energy consumption. The matching ultra low-power transmitters have, however, only a single order of magnitude lower power consumption.

The trade off for this low power consumption comes with the functionality and transmission rate of the transmitters/receivers. With a normal radio being able to transmit in the 250 kbps range, the low-power counterparts operate in the 1 kbps range. Similarly, the supported data format is typically just a single signal being set either high or low, which implies that data encoding/decoding must be done either by separate circuitry or in software. Thus, ultra low-power transmitters/receivers are only suitable to replace the normal radios when the application requires an insignificant amount of bandwidth.

The ultra low-power transmitters/receivers will hence be used primarily as a secondary communication channel, rather than a substitute for the normal radios, and with the purpose of improving duty-cycling of the normal radio. The two orthogonal domains we wish to explore are:

- Persistent transmitter
- Persistent receiver

1.3.1 Persistent Transmitter

In this domain, it is assumed that a signal is always being transmitted, and whenever a mote turns on the low-power receiver a signal will always be received. Assuming that the signal received on all the motes in the WSN is the same, this can be used to synchronize the entire WSN without any of the motes transmitting a single message.

Such signals do exist already. In particular radio controlled household clocks have been available from any local hardware store in Europe for the past decade, thanks to the German DCF77 timecode signal. Because of this technology being quite old, the method is very durable and the components easily available and quite cheap as well.

We wish to combine this technology with a WSN to achieve not only local, but global time synchronization, in a highly scalable and cheap way, both money and power consumption wise. With a precise time synchronization the problem with scheduled communication would indeed be reduced to a problem of exchanging schedules and joining an operating WSN would be completely transparent. Similarly, deployment issues could be reduced to having all the motes turn on their radio on a particular time every day and listen for the ”go” signal. We wish to investigate the domain where the power consumed by the low-power DCF77 receiver outperforms a regular time synchronization protocol and if this time synchronization can be used for efficient deployment of and joining a WSN, specifically:

Can we use the DCF77 signal to:
- perform time synchronization in a power efficient way?
- reduce power consumption at deployment time?
- reduce power consumption when joining an operating WSN?

1.3.2 Persistent Receiver

In the other end of the spectrum lies the always listening receiver. This can be used for out-of-band signalling, specifically it can be used to remote control any given mote within range e.g. to remotely turn on the radio on a mote. If the frequency of data transmission is high enough the cost, of having the receiver turned on always, will be paid in full by the obsolete beacon and listening periods in power efficient discovery. This out-of-band signalling would immediately reduce the latency to the fixed transmission time of the low-power transmitters/receivers. Similarly, deployment and joining issues would also vanish due to the remote control capabilities of the out-of-band signalling. We wish to investigate the potential reduced latency and power savings achieved by out-of-band signalling, specifically:

Can we use out-of-band signalling to:
- reduce power consumption of idle listening when performing discovery?
- reduce latency caused by duty-cycled communication?
- reduce power consumption at deployment time?
- reduce power consumption when joining an operating WSN?

1.4 Contributions

Our contributions are the following:

- We bring DCF77 global time synchronization into WSN and TinyOS, in a cheap and efficient manner.
- We provide a proof of concept with the implementation of our scheduled communication protocol based on the DCF77.
- We argue that this protocol is scalable, uses ultra low-power, has no deployment cost overhead at all and allow motes to join an operating WSN on-the-fly without any power consumption penalties.
- We offer an easy and efficient way to reduce latency with our out-of-band signalling, which is more than just a wake-up channel, but rather a low-power secondary radio, allowing both remote control and low-rate data transmission.

- We provide a proof of concept with the implementation of our scheduled discovery protocol based on out-of-band signalling.
- We argue that this protocol is scalable, ultra low-power, has very little deployment cost overhead, and allow motes to join an operating WSN on-the-fly, with the same reduced power consumption penalty.
- We give a thorough analysis of the Internal Clock Generator module on the Freescale EVB13192 based on actual measurements. This can be used in future applications to better understand timing critical issues.
- We implement a low-power receiver board for the DIKU testbed, allowing future students to explore DCF77 time synchronization and out-of-band signalling.

1.5 Road-map

This thesis consists of three parts:

1. The development of the necessary hardware boards and drivers for the low-power receivers that we are going to evaluate. This work is presented in Section 2.
2. The evaluation of the DCF77 time synchronization and subsequent use in scheduled communication. This work is presented in Section 3.
3. The evaluation of the potential for low-power receivers as a means for out-of-band signalling in power efficient discovery and to reduce latency. This work is presented in Section 4.

In each part, we begin by presenting the necessary background information with regards to technical information and related work done on the subject. We then discuss the design and implementation of our solution, and subsequently perform an evaluation of this. Finally, each part concludes with a discussion of future work that needs to be done and a summary of the entire section.

After these three parts we present our conclusions and contributions.

2 DIKU Low-power Receiver Board

In this section, we report on the design and development of the DIKU Low-power Receiver Board.

In the first part, the target platform is presented in Section 2.1.1 and the choice of radio modules is discussed in Section 2.1.2. In the second part, the physical layout of the receiver board is shown in Section 2.2.1 and the necessary TinyOS components to drive the radio modules are developed in Section 2.2.2. A direct measurement of the power consumption in the actual implementation is then done in Section 2.3. Finally, in Section 2.4 future improvements are discussed and a summary of the development can be found in Section 2.5.

2.1 Background

2.1.1 Freescale EVB13192

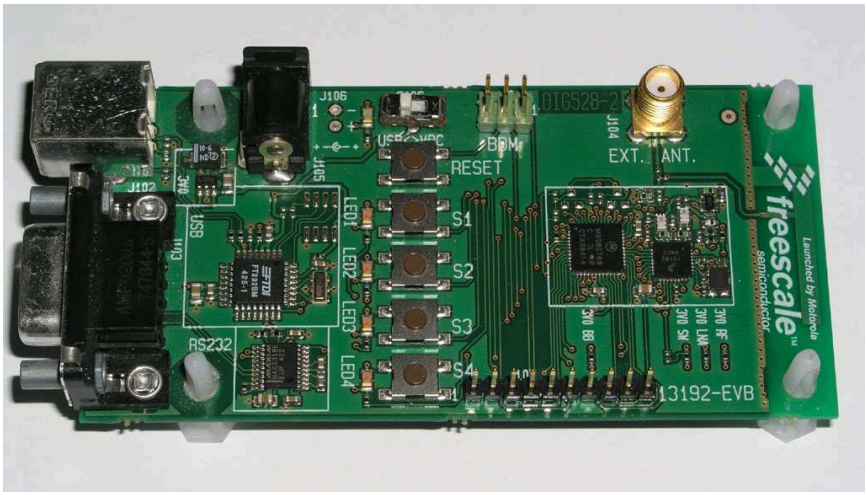


Figure 1: The Freescale EVB13192 mote.

In this thesis we use the Freescale EVB13192 [4] mote as our target platform. The mote consists of:

- Freescale MC9S08GT60 MCU
- Freescale MC13192 radio
- 9-pin RS-232 serial connector
- serial-over-USB connector
- 4 buttons

- 4 LEDs

The mote can be powered either by an external power supply (3.5V-12.0V) or by the USB-port.

MCU

The main modules of the Freescale MC9S08GT60 [5] are:

- 0-40 MHz HCS08 8-bit CPU
- 4K RAM
- 60K FLASH
- Internal Clock Generator
- Keyboard Interrupt
- I^2C (Inter-Integrated Circuit)
- 2x Timer/PWM
- 1x ADC (Analog-to-Digital Converter)[3]
- 2x Serial Communications Interface modules (SCI)
- 1x Serial Peripheral Interface module (SPI)

Internal Clock Generator

The MCU is clocked by an Internal Clock Generator (ICG), which is driven either by the `Internal Reference Generator` or an external clock source. On the EVB13192, the radio can be used to drive the external clock.

The `Internal Reference Generator` consists of two reference clock sources, one designed to be approximately 8 MHz and one being possible to trim by $\pm$ 25% starting at 243 kHz. This trimming process requires an external reference value though e.g. the radio clock. Since the value of this register is stored in non-volatile memory, once trimmed the internal clock should only have a temperature induced drift, which is stated by the manufacturer to be within -0.3% - 0.1%.

The downside of the register being stored in persistent memory is, however, that unless a trimming procedure is explicitly performed it is not possible to determine the precise value of the internal clock, as any prior programs could have altered the register and thus the value of the internal clock.

The ICG can be run in four different clock modes:

[3] Multiplexed over 8 channels.

- Self-Clocked Mode (SCM)
- FLL[4] Engaged Internal (FEI)
- FLL Bypassed External (FBE)
- FLL Engaged External (FEE)

Where SCM mode is based on the 8 MHz internal reference clock, FEI mode is based on the 243 kHz internal reference clock, and the FBE and FEE modes use the external clock as reference. The main differences between the modes are the precision, power consumption and available frequencies.

Assuming that the external clock has a higher precision than the internal clocks, the precision in ascending order would be: SCM, FEI, FEE and FBE. Assuming that the increased precision from the external clock also comes with an increased power consumption, the same ordering holds for power consumption as well.

If precision is more important than power consumption, the external clock can be connected directly to the timers bypassing many of the ICG's modules. In this configuration, the FEE and FBE modes give the same precision of the timers because the modules differentiating the two modes are bypassed anyway. This mode of operation will later be referred to as external clock (XCLK) mode.

Looking at the frequency range the SCM mode can drive the system clock from 3MHz-20MHz, FEI and FEE modes from 4MHz-20MHz and FBE mode less than 8MHz.

Timer/PWM (TPM)

Closely related to the internal clock are the two Timer and Pulse-Width Modulation modules, TPM1 and TPM2. Each module has its own 16-bit counter, which can be driven by either the system clock or the external clock. Furthermore, each counter is associated with 3 and 2 channels for TPM1 and TPM2 respectively, with an optional external pin associated with each channel. The individual channels can be set to operate in either input capture, output compare, or buffered edge-aligned pulse-width modulation.

In input capture, the external pin is monitored for either rising edges, falling edges or both. When an event occurs, the value of the associated counter is copied to a register and an interrupt is raised. This can be used to sample a signal in an processing efficient way i.e. without continuously sampling.

[4] FLL - Frequency-locked loop

In output compare, a 16-bit value can be stored in the same register as before and when the value in the counter is identical to this an interrupt is raised. This can be used as an asynchronous alarm clock instead of busy waiting.

Finally, in buffered edge-aligned pulse-width modulation, the channel can drive a square-wave on the associated external pin, with the register determining the frequency and width of the wave. This can be used to generate an external clock source.

Modes of operation

In order to reduce power consumption, the MCU can be run in five different modes: Run, Wait, Stop1, Stop2, and Stop3.

- Run-mode is the normal execution mode. All modules are turned on and the CPU executes instructions from memory.
- Wait-mode is similar to Run-mode, except the CPU is turned off. Any interrupts will make the MCU enter Run-mode.
- Stop1-mode is the lowest possible power state. All internal circuits are shutdown except the voltage regulator and the ADC which are kept in a low-power state. This mode can only be exited by either setting the external RESET-pin or IRQ-pin.
- Stop2-mode is similar to Stop1-mode except the contents of RAM and the current state of all I/O-pins are maintained. All registers have to be saved to RAM before entering Stop2-mode though and I/O port registers must be restored before pin latches are opened. This mode can be exited by either setting the external RESET-pin, IRQ-pin or by an RTI interrupt (Real-Time Interrupt).
- Stop3-mode is similar to Stop2-mode except the states of all the internal registers are maintained as well, removing the need for saving and restoring these registers. This mode can be exited by either setting the external RESET-pin, IRQ-pin, an RTI interrupt or by one of the Keyboard Interrupt pins.

By turning off these various modules, the power consumption can be reduced significantly e.g. the power consumption when in Run-mode at 8MHz is 6.5mA which can be reduced to 25nA, 550nA, and 675nA for Stop1, Stop2, and Stop3-mode respectively.[5]

[5]For reasons unknown no power consumption data are listed for Wait-mode. However, given that only the CPU is turned off one should expect a power consumption in the range of mA.

Radio

The key features of the Freescale MC13192 [6] are:

- 2.4 GHz ISM band
- IEEE 802.15.4 compliant
- 16 channels (5 MHz intervals)
- 250 kbps transfer rate

Due to the strict timing requirements of the IEEE 802.15.4 standard, the combined error of the radio's internal clock circuit and its external 16MHz crystal is less than ± 40 ppm. This clock can be used as an external clock source for the MCU allowing the latter to take advantage of the high-precision clock.

Like the MCU, the radio can run in different modes in order to reduce power consumption, with the available modes being: Off, Hibernate, Doze, Idle, Transmit, and Receive.

Due to Idle-mode being the normal operation mode, the main functional difference between the low-power states Off, Hibernate, and Doze-mode is the transition time to Idle-mode. A full listing of the different transition times and each modes power consumption can be found in Table 1.

Mode	Transition Time	Power Consumption
Off	25 ms to Idle	0.6 μW
Hibernate	20 ms to Idle	3.0 μW
Doze	300 μs to Idle	105 μW
Idle		1.5 mW
Transmit Mode	144 μs from Idle	90 mW
Receive Mode	144 μs from Idle	111 mW

Table 1: Transition times between the Freescale MC13192 radio's different operation modes and each modes power consumption. Data from [6].

Evaluation Board Layout

The EVB13192 was originally designed as an evaluation board for the MC13192 radio transceiver and as such also equipped with a BDM-interface (Background Debug Module) and 10 connector pins are wired directly to the MCU for easy access of on-chip modules. Combined with the relatively large form factor, this platform is quite suitable for development and attachment of peripherals.

However, since the primary purpose of the board was to evaluate the MC13192 radio, little effort has been put into the actual layout of the board and the choice of pins used on the MCU for wiring, meaning several module specific pins are used as General Purpose I/O (GPIO) pins instead of using dedicated GPIO pins for this purpose. Specifically,

- Of the eight Keyboard Interrupt pins, the four not used by the buttons are used as data control pins for the two serial interfaces.
- Four of the total five pins associated with the TPM module are being used to drive the LEDs.
- The I^2C interface is connected to the radio as normal data pins.

This greatly limits the functionality of any attached peripherals. The 10 connector pins that are connected to the MCU has the following functionality:

- 1x TPM1 Channel 2
- 4x ADC Channel 1, 2, 3, and 7
- 1x Oscillator output
- 3x High-current GPIO pins (max. 10.0 mA)
- 1x Low-current GPIO pin (max. 2.0 mA)

Through the BDM-interface, it is also possible to access: 3.0V, ground, the RESET and BDM-pin on the MCU.

The most striking limitation of this is that the TPM1 pin is the only available pin capable of raising an interrupt. This implies that any peripherals connected to the other pins must be accessed by polling instead of being event driven. Furthermore, the interrupt raised by the TPM1 pin is not of any significant priority and can thus only bring the MCU out of Wait-mode and not out of any of the Stop-modes. As mentioned above, only the RESET, IRQ and Keyboard Interrupt-pins can do this and none of these pins are available without modifying the board i.e. removing one of the buttons.

2.1.2 Low-power Receiver Modules

As stated in Section 1.3, we wish to explore two orthogonal regimes of operation e.g. the persistent transmitter and the persistent receiver. Because these two approaches are not mutual exclusive, we wish to be able to explore both on the same

receiver board. The radio controlled clock signal we wish to receive implies that it is a transmission source we cannot turn off, hence in order to evaluate the two regimes, it is necessary to have two different kinds of low-power receivers, each operating on a different radio band. This is to avoid interference between the two modes of operation.

Given the specifications of the MCU and the evaluation board, it is required of the low-power receiver modules to transfer data using CMOS/TTL logic and be able to operate with 3.0V power supply. Any other specifications would require the need for extra components such as level converters etc. which would increase both overall cost and power consumption.

Conrad DCF77 Receiver

Figure 2: Conrad DCF Receiver. Printed-circuit-board connected to a ferrite rod antenna and mounted with a 4-pin screw connector. Picture taken from [7].

To evaluate the persistent transmitter regime, we choose the DCF77 Receiver Board from Conrad International [7]. This board includes a ferrite rod antenna, and has a 4-pin screw connector for easy attachment. Combined with the circuits being CMOS/TTL compatible and having an operating voltage range between 1.2V to 15.0V, this makes an excellent board for evaluation purposes. The decoding of the transmitted radio signal into a CMOS/TTL signal is done by a Temic U4224B [8] time-code receiver chip.

The module costs 9.95€.

RF Solutions AM-HRR8-433 and AM-RT5-433

For persistent receiver evaluation, a AM-HRR8-433 receiver [10] and a AM-RT5-433 [11] transmitter from RF-Solutions [9] were used. Both operate in the 433 MHz AM radio band and has CMOS/TTL compatible input/output. The receiver requires 3.0V supply voltage, while the transmitter can use 2.0V - 14.0V supply voltage.

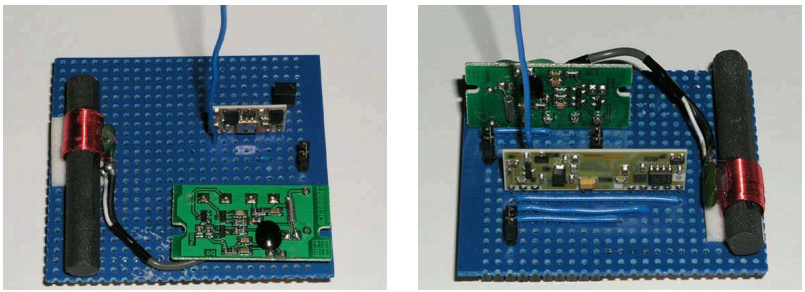
AM-HRR8-433 AM-RT5-433

Figure 3: RF-Solutions 433MHz AM receiver (left) and transmitters (right). Pictures taken from [9].

The receiver is rated to have a reception range up to 50 meters and a maximum data rate of 2000 Hz. For the transmitter, the transmission range is up to 70 meters and the maximum data rate is 9600 Hz.

The receiver module costs 7.70 € and the transmitter 8.33 €, but different from the DCF Receiver Board, neither the receiver module nor the transmitter module come with an antenna included.

2.2 Design and Implementation



DIKU Transmitter Board DIKU Receiver Board

Figure 4: Pictures of the `DIKU Low-power Receiver Boards`.

The actual boards used in this thesis can be seen in Figure 4. On the left, the `DCF Receiver Board` can be seen in the front, connected to the ferrite rod, and the `AM-RT5-433` transmitter can be seen in the back, connected to the blue whip antenna. On the right, the `DCF Receiver Board` is mounted at the back, and the `AM-HRR8-433` receiver can be seen in the front.

2.2.1 Board Layout

Given the available pins discussed in Section 2.1.1, the most efficient way to sample a signal is by using the TPM1-pin with the associated channel set to input capture. Any changes in the signal will thus raise an interrupt, and by the virtue of the timer module, the exact time of the event is known. This is significantly more efficient than using any of the other pins as GPIO-pins since continuously sampling the signal to detect any changes is both a waste of processing time and power.

Having both type of receivers on the same board with only one TPM1-pin available is a problem, though. With our primary concern being power consumption, using one of the GPIO-pins is not an option. Thus, the only viable solution is to connect both type of receivers to the TPM1-pin in a way that allows multiplexing between the two.

Since it should be possible to turn the modules on and off anyway, multiplexing between the two modules could be as simple as only having one module turned on at a time, as long as the output signal from the turned off module is stable. This turns out to be the case and by programming the MCU to use its internal pull-up device the output signals from the two receiver boards can be connected directly to the TPM1-pin simultaneously. Also, using the programmable pull-up device instead of an external resistor helps save a component.

In order to turn the two receivers and the one transmitter on and off independently, three control pins from the MCU are needed. However, because three of the available connector pins are high-current GPIO-pins anyway, using these pins directly as power sources saves the need for extra transistors which again saves power.

As there is no special requirement for the input to the transmitter module, any GPIO-pin will do.

Schematic

For this thesis five boards equipped with both the `DCF Receiver Board` and the `AM-HRR8-433` receiver called the `DIKU Receiver Board` were built. A diagram of the wiring can be seen in Figure 5. Also, a single board equipped with both the `DCF Receiver Board` and the `AM-RT5-433` transmitter called the `DIKU Transmitter Board` was built. A diagram of the wiring can be seen in Figure 6.

Because the choice of pins is not mutually exclusive, it is possible to combine the `DIKU Receiver Board` and the `DIKU Transmitter Board` into the `DIKU Transceiver Board`. This was not done in this thesis, however.

Antenna

As mentioned above, neither the AM-HRR8-433 receiver nor the AM-RT5-433 transmitter came with an antenna. For this thesis a simple whip-antenna was

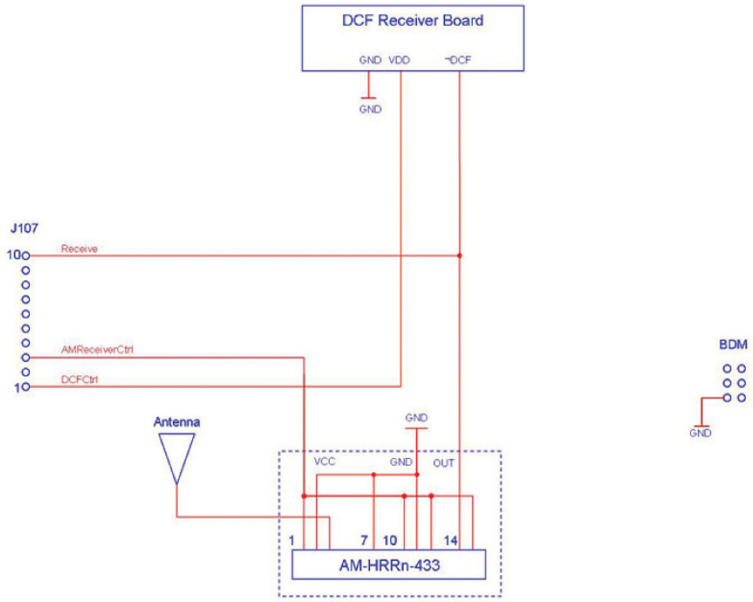


Figure 5: Schematic diagram of the `DIKU Receiver Board`.

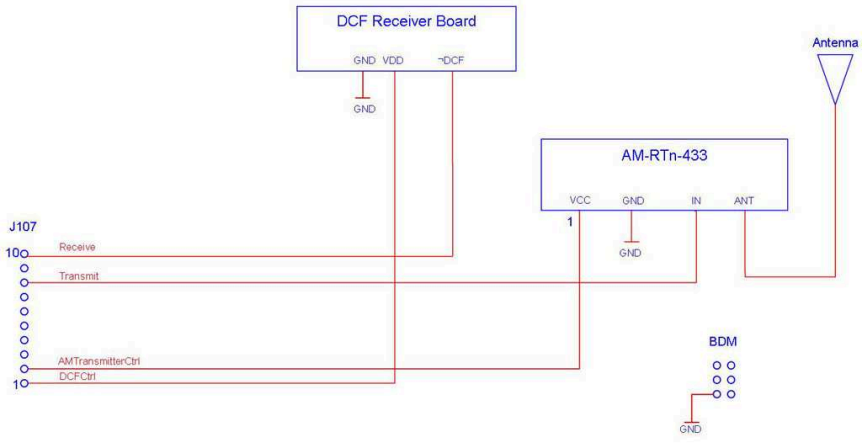


Figure 6: Schematic diagram of the `DIKU Transmitter Board`.

mounted on the boards by attaching a single wire to the antenna pin of the respective modules. With a carrier frequency of 433MHz and estimating the propagation velocity of the radio signal to be $c = 299792458\ m/s$ the wavelength can be calculated to be:

$$\lambda = \frac{c}{\nu} = \frac{299792458\ m/s}{433000000\ s^{-1}} = 0.691m$$

This is not a very practical length for an antenna mounted on a palm-sized mote and therefore a 1/4-wavelength wire (17.3 cm) was used instead. As much as possible of the bottom surface was grounded, to complete the whip-antenna, but as this ground plane could not be distributed evenly due to the board layout, the antenna became slightly directional.

2.2.2 TinyOS

The configuration and usage of the MCU pins on the EVB13192 platform are done through memory-mapped registers. Since these are made available in TinyOS through global variables, there is as such no need for special drivers to access the pins. However, as the TPM1 channel 2 pin would be used extensively by both receiver modules, a dedicated module to control the entire TPM1 module was developed. Besides helping to reuse components, a control component for the TPM1 module helps fill a void in supporting the platform as the module is currently unused by any other TinyOS code developed for the EVB13192 platform.

Also, the currently available timer modules (alarm clock and counters) are based on the TPM2 module. By developing separate timer functionality based on the TPM1 module, alarm clocks and counters used by the receiver and transmitter boards would become transparent to any other components. This would allow all other applications, currently developed to use the standard timer functionality, to continue to work in conjunction with any components developed to utilize the low-power receivers. Because efficient radio communication can be viewed as a tool for all other sensor applications, this is a priority.

With the power control to the three modules being almost identical bare a pin number, a parametrisized component was developed to turn a module on and off. This made reusing the code easier. On the other hand, only one component was expected to access the transmitter, and therefore this was done directly on the global variables controlling the data signal.

TPMM

As mentioned above, the TPM1 module consists of a counter and three channels. Only one of the pins associable with the TPM1 module is available though as the two others are used to drive LEDs. It is thus important not to change the operation of these two pins as this might interfere with any application using the LEDs. The

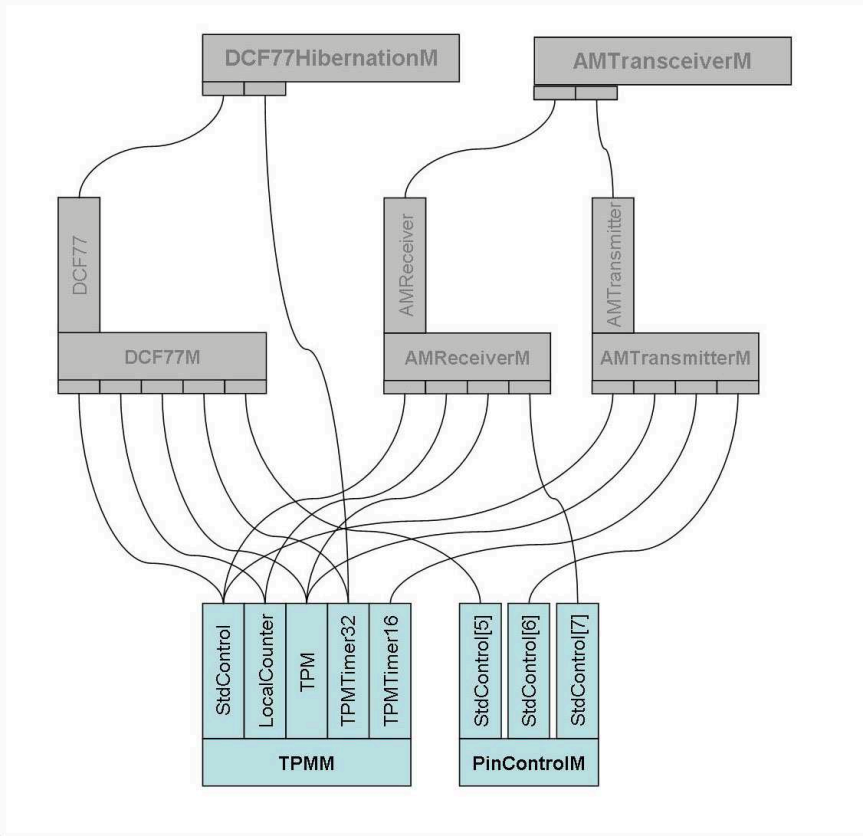


Figure 7: Component overview of the TPMM and PinControlM modules.

last pin, which has both receivers connected, must obviously be used to trigger an event when a change is detected on the pin.

The counter can be driven by either the internal clock or the external radio clock if the radio is on. It stands to reason that the ability to choose between the low-power internal clock and the high-precision external clock is useful. A consistency check is performed though before choosing the external clock as source based on the current operation mode of the ICG.

Due to the counter being only 16-bit wide, it will overrun in a matter of milliseconds. Whenever possible, a high divider is chosen with the condition that there is at least 1 ms between each count. In order to increase the counting range from mere seconds to hours, whenever an overflow of the timer occurs, several software timers are incremented. These timers being a 64-bit counter running from the starting of the module, a 32-bit counter running from the last received event on the channel 2 pin and finally a 32-bit alarm clock counter.[6]

With two unused channels, it is possible to make two separate hardware alarm clocks. Given the problem at hand, it was decided to make two different kinds: A 32-bit alarm clock (with the help of software counters) to allow long term asynchronous wake-up and a 16-bit hardware only alarm clock for small interval interrupts, useful when encoding signal. Both alarm clocks were made repeating in order to assure a stable periodicity. This was to facilitate a smooth signal encoding and a stable second counting useful when doing synchronization.

These modules have been made part of DIKU's contribution to the TinyOS project at SourceForge,[7] and can be used in any TinyOS application by adding the line `SENSORBOARD=lpreceiver` to the application's `Makefile`.

2.3 Evaluation

With minimal necessary functionality available, receiver and transmitter boards were attached to a mote each (as seen in Figure 8) and, by the virtue of three specially mounted jumpers, it was possible to measure the actual energy consumption directly from this real setup. The figures can be seen in Table 2.

The different transmission rates were obtained using the newly built alarm clock by alternating the output on the transmitter signal each time the alarm clock fired. By looking at the 600 Hz signal sent into the transmitter with an oscilloscope, an

[6] The software counters are incremented in such a way that in order to actually read one of them, the 16-bit from the hardware counter must be added as well. This was done in order to be compliant with the standard TinyOS timestamp, which is 32-bit wide and counts the number of clocks since the mote is turned on. The commands reading the counters does automatically add the hardware counter before returning a value.

[7] http://tinyos.sourceforge.net

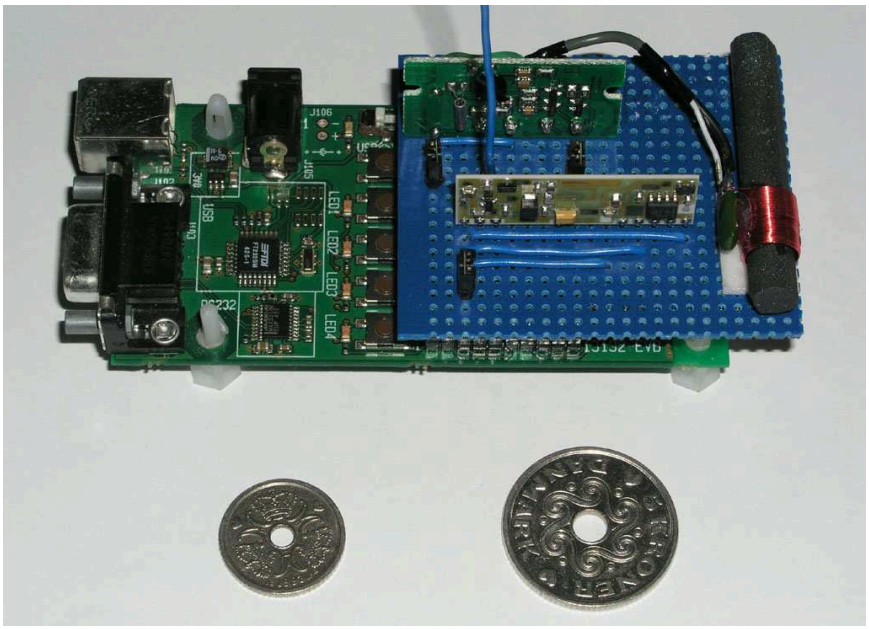


Figure 8: Freescale EVB13192 with DIKU Receiver Board attached. Danish `1 krone` and `5 krone` coins in front for size comparison.

Module	Mode	Power (mW)	Power (mA)
AM-HRR8-433	Idle	1.35 ± 0.03	0.45 ± 0.01
	Receiving	1.32 ± 0.03	0.44 ± 0.01
AM-RT5-433	Idle	0.00 ± 0.03	0.00 ± 0.01
	Transmitting @ 2.0kHz	7.80 ± 0.09	2.60 ± 0.03
	@ 1.2kHz	8.49 ± 0.03	2.83 ± 0.01
	@ 0.6kHz	8.58 ± 0.03	2.86 ± 0.01
DCF Receiver Board	Receiving	0.21 ± 0.03	0.07 ± 0.01

Table 2: Power consumption of the three different radio modules. Measured at 3.0V voltage and when running on the Freescale EVB13192.

even square wave was observed with the width corresponding to the set alarm clock interval. Looking at the received signal through the oscilloscope yielded the same result. At higher speeds, however, the squares became rounded asymmetrically resembling a shark fin. The reason behind this is probably the GPIO pin on the MCU not being fast enough to generate a perfect square wave at high frequencies.

The result of this asymmetry is that even when a supposedly symmetrical signal is transmitted the interval between edges is not constant, and because it is this interval that the TPM1 module counts, any decoding procedures must be able to compensate for this.

This morphing from a square wave to a shark fin wave at higher data rates could help explain the apparently odd result of higher data rates causing lower power consumption. Due to power consumption being directly related to the area beneath the wave, a shark fin wave will consume less power compared to a square wave.

Another interesting observation is the power consumption of the AM-HRR8-433 receiver. First, the receiver consumes less power when receiving than when it is not. Second, according to the specifications given by the manufacturer, the power consumption should be 0.5 mA at 3.0V. The latter is interesting because the same manufacturer also produces a receiver that, according to the same specification, only consumes 0.07 mA at 3.0V. This is the same value measured for the `DCF Receiver Board` and thus an entirely realistic value. Unfortunately, this AM-HRR18-433 receiver was sold out when this thesis was done.

2.4 Future Work

With the current set of boards, only one of them is equipped with a transmitter. This is adequate for the problems we wish to investigate in this thesis e.g. single hop out-of-band signalling. If out-of-band signalling in multi-hop routing were going to be explored, transceiver boards with both transmitter and receiver should be built. The design of the receiver and transmitter board and the corresponding software components do indeed account for this and should work without any modifications. Special care had to be taken with regards to the antenna because two antennas on the same board, with a common ground, might cause undesirable interference. This could be solved by having the two modules share one antenna through a multiplexer.

The antenna is currently just a 17.9 cm long wire soldered directly to the board. As the picture shows, this is not a very practical solution given the form factors. Also, the more sophisticated antenna would undoubtedly yield a better transmission range and quality.

Finally, for this to be a real low-power receiver/transceiver board, the AM-HRR18-433 must be used instead of the AM-HRR8-433. Because of the form factor, func-

tionality and requirements are identical to the receiver currently mounted, achieving an order of magnitude power reduction could be as simple as resoldering the boards.

2.5 Summary

In this section we developed the `DIKU Receiver Board` and `DIKU Transmitter Board` and the necessary TinyOS components in order to use the boards with the Freescale EVB13192 evaluation board.

The functionality was verified with the help of an oscilloscope. It was found that the transmitter/receiver setup was indeed working although the signal suffered from distortion at high data rates.

Power consumption of the different modules was then measured when fully operational. These measurements showed the low-power radios to use one to three orders of magnitude less power than the on-board radio. This comes with the cost of data rate with the low-power radios having three orders of magnitude less data rate.

Finally, several improvements to the board were presented, with the most significant one being a transceiver board with a truly low-power receiver.

3 Time Synchronization

In this thesis, we wish to study the importance of time synchronization in the regime of scheduled communication. However, the need for accurate time synchronization goes beyond this. Generally, any application, which requires two or more motes to agree on any event, requires some sort of synchronization. For instance, in a vehicle tracking system the lack of synchronization could lead to a single vehicle being classified as several vehicles by the WSN.

If the synchronization only involves the motes within a single WSN to be in agreement with each other, the synchronization is called internal. However, if the motes are being synchronized with a device outside the WSN e.g. a gateway connecting several WSNs, the synchronization is called external. The association of a point in time with an event is known as timestamping and the time of the event is usually referred to as the timestamp.

For any synchronization, the motes are being synchronized to a relative time frame meaning that the timestamp of one event only makes sense when compared to the timestamp of another event. Also, timestamps can only be compared to signify a temporal difference if the timestamps originate from motes synchronized to the same time frame. If the time frame being synchronized with is widely used e.g. UTC, the synchronization is called global and the timestamps are often referred to as absolute.

We will in this section show how to use the German DCF77 timecode signal to externally synchronize the motes within a WSN. The advantages of using the DCF77 signal are in the nature of the long-wave signal, being its very long range, and penetrating ability i.e. allowing reception within buildings. Also, due to DCF77 signal reception being relatively cheap both in regards to equipment and power consumption, it is quite feasible to equip every single mote in a WSN with a DCF77 receiver. This reduces any scalability problems related to regular time synchronization protocols to a constant cost overhead when manufacturing the mote.

The use of a secondary radio and a transmission source powerful enough to encompass the entire network is also known as out-of-band synchronization. Furthermore, because the time-code contains an absolute timestamp i.e. Central European Time, each and every mote will be globally synchronized in a very power efficient way and with complete fault-tolerance.

With this global time synchronization we can perform scheduled communication without the need of exchanging synchronization messages over the radio if the the schedule has been distributed prior to the deployment. In this case, the cost of deployment would reduce to either begin at a pre-determined time or to listen for the ”begin” message regularly according to a schedule. Joining an operating

WSN would be transparent in this case as synchronization and schedule are already known to the joining mote.

First, Section 3.1.1 describes the DCF77 signal in detail and Section 3.1.2 presents previous work on achieving global synchronization within a WSN. Secondly, Section 3.2.1 analyzes the different parameters needed to achieve optimal precision and in Section 3.2.2 the TinyOS components needed for DCF77 decoding are developed together with a scheduled communication scheme utilizing the global synchronization as a proof-of-concept. These are subsequently evaluated in Section 3.3. Finally, Section 3.4 discusses the road for future work and Section 3.5 summarizes this section.

3.1 Background

3.1.1 DCF77

This section is based on information from the *Physikalisch-Technische Bundesanstalt* [12].

The *Physikalisch-Technische Bundesanstalt* is the German national metrology institute and has, since 1978, been in charge of realizing and disseminating legal time in Germany. This has been done by telephone service, the Internet and a long-wave radio signal called DCF77.

DCF77 is the international call sign that all radio stations' which transmissions exceed national borders, are required to have according to the International Telecommunication Union. The name was chosen in accordance with the preassigned series of call signs for Germany and is loosely an acronym for German long-wave transmitter located near Frankfurt with a carrier wave of 77.5 kHz.

The signal is transmitted from Mainflingen, 25 km from Frankfurt, by a 50 kW transmitter and an omni-directional antenna. Because of the long wavelength, the signal has a maximum range of 1900 km during daytime and 2100 km at nighttime.[8] This means that the DCF77 signal can be received in most of Europe, as can be seen by the signal coverage map in Figure 9.

Signal reception can be attributed to two parts, a ground and sky wave with the sky wave being a reflection from the atmosphere. Within 600 km from the source the ground wave dominates the received signal, and because of the direct flight, the signal strength is constant.

Between 600 km to 1100 km, the signal strength of the ground wave and the sky wave is of the same order of magnitude, and because of the different flight paths,

[8] This is caused by the cooling of the atmosphere during night, which causes the altitude of the reflective layer of the atmosphere to rise.

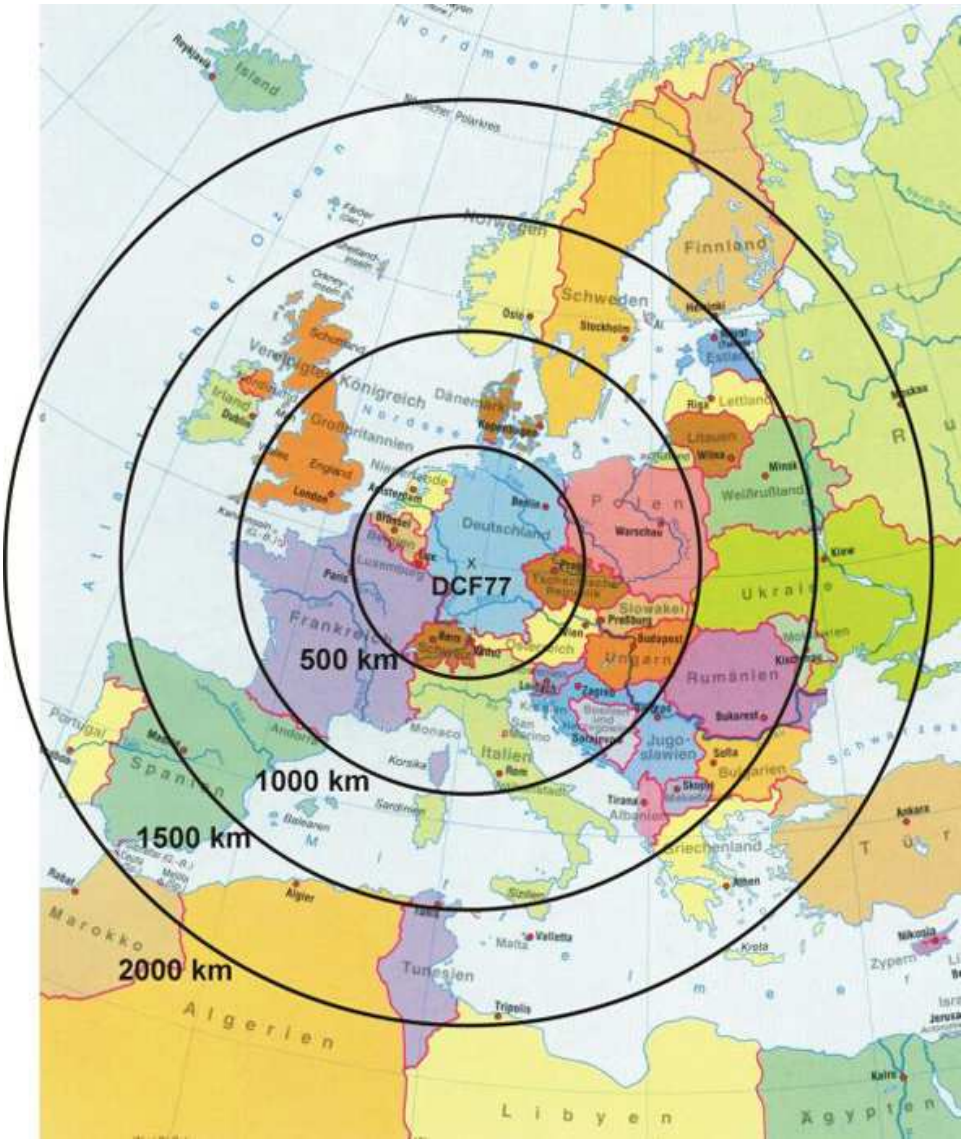


Figure 9: Map of Europe overlaid with distances to Frankfurt, Germany. With an effective range of 1900-2100 km the DCF77 signal can be received in most of Europe. Picture taken from [12].

the two signals might have different phases causing both cancellation and resonance. Due to this phase shift being mainly caused by atmospheric fluctuations, the change of phase is a slow process and complete cancellation should not be observed more than a few times a day. However, this also means that when it occurs it lasts for tens of minutes. By using a directional antenna pointed at either the ground or the sky wave, this cancellation effect can be greatly reduced.

Above 1100 km, only the reflected sky wave can be received. Since the reflective atmospheric layer is temperature dependent, signal strengths may vary over time.

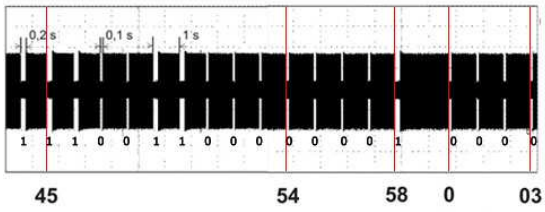


Figure 10: Signal strength of the DCF77 signal between the 44th to the 3rd second. The red lines mark the beginning of the 45th, 54th, 58th, 0th, and 3rd second respectively. Figure taken from [12].

The timecode is encoded onto the radio signal as a 59-bit long binary sequence by amplitude modulation. This is done by reducing the strength of the carrier wave by 25% at the beginning of each second. This binary sequence is transmitted once a minute, starting at the zeroth second, and with no signal reduction at the 59th second which is used to indicate the beginning of a new minute. By keeping the signal reduced for either 100 ms or 200 ms, a `0` or `1` can be encoded onto the signal. A plot of the signal strength between the 44th and 3rd second can be seen in Figure 10.

The radio signal is derived from an atomic clock on-site with a precision of 2×10^{-12}. However, because the propagating long-wave signal has a phase time of $\frac{1}{77.5kHz} = 12.9\ \mu s$, the highest achievable precision must be of the same order of magnitude. The precision of the beginning of each second is controlled to be within $\pm 0.3\ \mu s$.

Depending on the location of the receiver, time-of-flight must be taken into account as well. The distance from Denmark to Mainflingen is approximately 600-800 km, which translates into a delay of 2-3 ms. If the location of the mote/WSN is known, however, this can be compensated for. If the geographic extend of a WSN is limited, the motes within the WSN will have the same time-of-flight offset since the motes will receive the signal almost simultaneously.

With the number of seconds within a minute being countable from the beginning of each timecode sequence, only a timecode with resolution of minutes and lower is

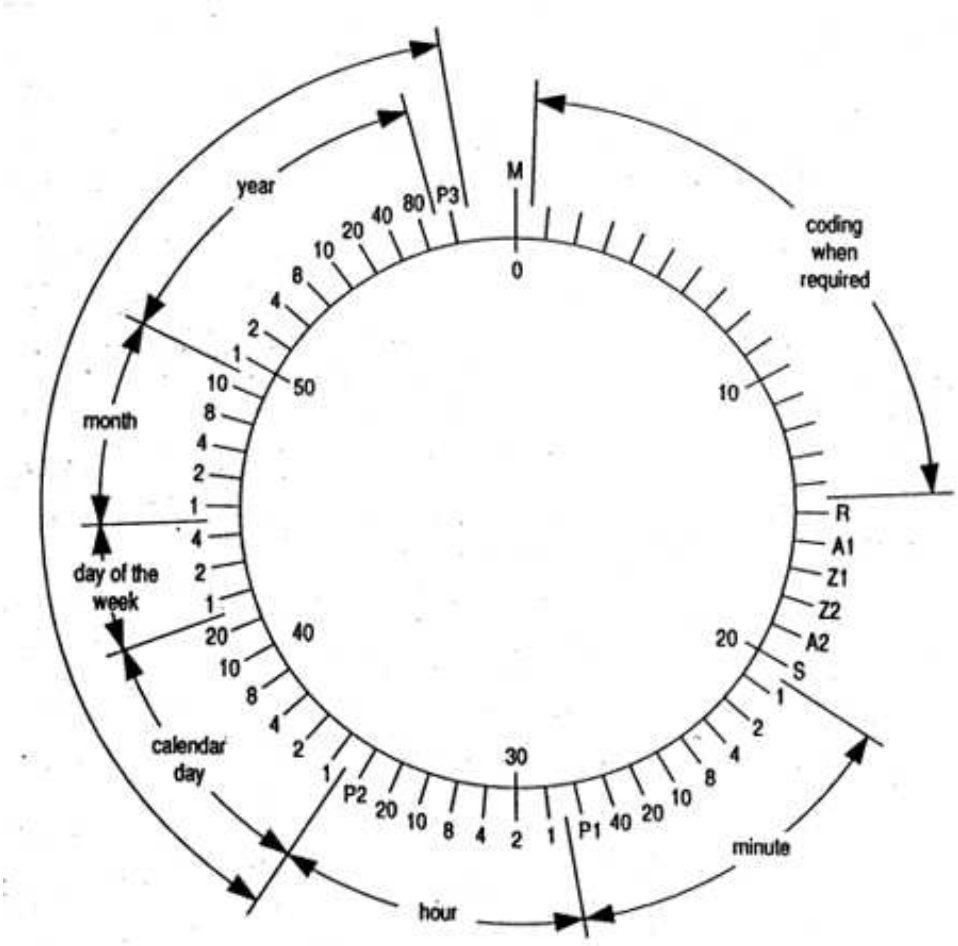


Figure 11: Encoding scheme for the DCF77 timecode. Picture taken from [12].

needed. As can be seen from Figure 11, the timestamp encoded contains the time in hours and minutes, and the date in day of month, day of week, month, and year, with the year being the number of years in the current century.

Table 3 shows the status and control information also encoded in the DCF77 signal.

Name	Status
Bit 0	Control bit: Always 0.
R	A call sign for technical assistance, indicating a problem at the transmission site.
A1	Indicating a change of daylight saving times, transmitted for an hour before each change.
Z1	Timezone being transmitted: Central European Time (CET).
Z2	Timezone being transmitted: Central European Summer Time (CEST).
A2	Announcement of the insertion of a leap second, transmitted an hour in advance, a whole hour.
S	Control bit: Always 1.
P1	Parity bit no. 1, maintains even parity for bit 21-28.
P2	Parity bit no. 2, maintains even parity for bit 29-35.
P3	Parity bit no. 3, maintains even parity for bit 36-58.

Table 3: List of the control and status bits encoded in the DCF77 signal. Description given by [12]

With redundant transmitter and antenna, the DCF77 signal is transmitted continuously with a guaranteed[9] annual availability of 99.7%. In 2005, this figure was as high as 99.98% when not counting interruptions lasting less than 2 minutes.

This high availability of the transmitter is, however, not equal to a high availability at the receiving end. The main reason for this being interference, either natural interference e.g. atmospheric conditions, weather, obstructing mountains etc. or human interference e.g. TVs, computer monitors, industrial machines, steel frame buildings.

3.1.2 Related Work

Although the field of time synchronization in traditional wired networks is well studied, the inherent resource constraints in WSNs impose new challenges and different requirements to the synchronization protocols. Because energy consumption is of paramount importance, a traditional Round-Trip-Time (RTT) calculation approach, known from traditional wired networking [15], would be prohibitively expensive and not very scalable. Instead, we wish to seek a scalable solution that minimizes power consumption.

[9] This guarantee is part of the agreement between Physikalisch-Technische Bundesanstalt, which supplies the signal, and the contractor responsible for the transmission, Deutsche Telekom AG.

Recent studies have moved the field to the WSN regime where many of these new protocols have been investigated and summarized by [14] and [13]. Particularly, a paradigm known as ”Reference Broadcasting” [16] leverages the power of the wireless link as a broadcast medium to achieve a common reference point for all motes within the broadcast radius.

This is used in ”Reference Broadcast Synchronization” (RBS) by [17] to perform an internal synchronization of the network by having a reference mote broadcast message beacons. The reception time of these beacons is recorded by each of the other motes and put in a local timetable. By exchanging timetables with all the other, each mote is capable of calculating an offset that minimizes the overall RMS error of the local timetables. The drawback of this approach is the increased overhead caused by the exchange of timetables among the motes, which has a quadratic increase in the number of participating motes. The authors also propose a multi-hop variant where regular motes themselves can become reference motes, and theorize that the mean error will grow as $O(\sqrt{n})$, where n is the number of hops.

This lack of scalability is improved in ”TSync” by [18], which reduce the quadratic complexity into a constant, by assuming that the transmission delay between the reference mote and each of the other motes is the same. Then, by having the reference mote calculate the transmission time to one particular mote in a regular RTT fashion, and afterward broadcasts this delay to all the other motes, all motes within communication range can be synchronized with a total of three messages transmitted, regardless of the number of motes in range. The authors implemented both RBS and TSync on the MANTIS [19] platform and found that the mean synchronization error was 21.2 μs for TSync and 20.3 μs for RBS in a single-hop network, and 29.5 μs for TSync and 28.9 μs for RBS in a 3-hop network. This was achieved on a system where the clock jitter was measured to be 15.3 μs.

For very large networks, a multi-hop approach is not very practical as the probability for a lost packet increases for each hop. Also, for a partitioned network, an external synchronization is necessary meaning one or more motes must be able to synchronize with an external source, which again means these motes constitute a critical point-of-failure. Even with redundancy, an elaborate election scheme had to be in place in order to ensure the election of a new reference mote.

Due to the strict timing requirements in measuring the Time-Of-Flight of radio signals, the Global Positioning System (GPS) can be used as an extremely precise clock source, and, when mounted on all motes, as an efficient way to perform out-of-band global synchronization for an entire WSN simultaneously.

This has been done by [20] in the ZebraNet project where the movement of zebras was monitored with the help of sensor motes mounted with GPS. Besides using the GPS to record the location of each zebra, the global clock was used to facilitate ra-

dio communication. The problems they encountered, however, were an extremely high power consumption of the GPS receiver and limited reception beneath tree tops.

These have been two of three main reasons for not equipping every WSN mote with a GPS receiver, the third being a very large price tag. However, recent progress has been made in making GPS receivers both energy efficient and usable indoor, and commercial products with this new technology are already available. The `u-blox LEA-4T` [21] has a power consumption of 39 mA at 3.0 V. and need, under good conditions, 3.5-34 seconds to acquire a timestamp with 50 ns accuracy. Depending on the structure, some indoor reception is possible, as shown by the field test [22]. However, with a price tag of 95.0€ only very few applications could justify an expenditure of that magnitude.

As we will show later, the nanosecond accuracy will be wasted on the Freescale platform and for one-tenth the price of a GPS receiver we achieve a global synchronization only slightly worse than the supported precision.

A MAC protocol that is based on scheduled communication is the SCP-MAC presented in [32]. They combine channel polling, known from power efficient discovery, with scheduled communication by scheduling the periods where channels are polled. They claim they can achieve ultra low duty-cycles of 0.01-0.1 %, which is one to two orders of magnitude better than other MAC protocols. Their own TinyOS implementation shows only a modest 2.5 times lower energy consumption. Nevertheless, we will later use their rather optimistic duty-values as comparison.

3.2 Design and Implementation

3.2.1 Precision

As mentioned above, the uncertainty of the beginning of each second in the transmitted DCF77 signal is ± 0.3 μs. In order to achieve the same amount of precision at the receiving end, the equipment and the signal play significant parts. Disregarding the signal for the moment, the analysis of the precision of the motes can be split in two parts:

First, the resolution and precision of the clock on the mote determine how precise events can be timestamped, including the beginning of each second event from the DCF77 signal. On the EVB13192 evaluation board, the clock is instantiated as a free running counter driven by the ICG. The precision of the ICG is determined by the driving clock, which is either the internal crystal or the radio clock, and the operating mode of the ICG.

Second, the precision of the `DCF Receiver Board` determines how precisely the analog amplitude modulation of the radio signal can be converted into a digital signal. This conversion is done by the `Temic U4224B` timecode decoder chip.

ICG

In order to evaluate the precision of the ICG, several measurements were performed. First, both the internal clock and the radio clock were measured. This was done by attaching a `Philips PM 6668 High Resolution Counter 1GHz` directly to the mote. This was easy with the radio clock as it is possible to intercept the clock's signal path from the radio chip to the MCU. However, since the internal clock is built directly into the MCU, the former approach was not possible.

Instead, the TPM module was programmed to emulate an external clock as mentioned in Section 2.1.1, and the output of this pin was measured with the counter. Because the TPM is driven by the ICG, the measured precision and discrepancy are of course affected by both the uncertainties in the TPM and ICG module, and not just the internal clock. However, it does give valuable insight into the system as a whole.

The precision of each measurement is determined by how many significant digits with which it is possible to count the clock. Even though the clock counter is capable of measuring with seven significant digits, jitter will often result in the clock being measured with less than seven digits precision. The discrepancy between the expected clock and the measured one causes the clock drift as this numerical difference between counts results in the actual clock running at a different speed than expected.

It is important to differentiate between jitter and drift as it is possible to compensate for drift, but only within the uncertainty caused by the jitter.

For reference, all measurements were conducted on two motes.

Mode	Mote A			Mote B		
MHz	Frequency <kHz>	σ <μs>	δ <ppm>	Frequency <kHz>	σ <μs>	δ <ppm>
16	15999.97 ± 0.01	0.63	2	15999.85 ± 0.01	0.63	9
8	7999.990 ± 0.001	0.13	1	8000.007 ± 0.001	0.12	1
4	3999.995 ± 0.001	0.25	1	4000.003 ± 0.001	0.25	1
2	1999.997 ± 0.001	0.50	2	2000.001 ± 0.001	0.50	1
1	999.9990 ± 0.0001	0.10	1	1000.000 ± 0.001	0.10	0

kHz	Frequency <Hz>	σ <μs>	δ <ppm>	Frequency <Hz>	σ <μs>	δ <ppm>
62.5	62499.92 ± 0.01	0.16	1	62500.06 ± 0.01	0.16	1
32.786+	32786.85 ± 0.01	0.31	26	32786.91 ± 0.01	0.30	28
16.393+	16393.41 ± 0.01	0.61	25	16393.45 ± 0.01	0.61	27

Table 4: Measurements of the different clock outputs from the radio of two different motes.

The measurements of the radio clock can be seen in Table 4, with the `Mode` column

being the frequency specified in the manual, `Frequency` the measured frequency in kHz, σ the corresponding precision in μs, and δ the drift in ppm (parts-per-million).

First, it can be seen that all measurements are with 7 significant digits and that the uncertainty is on the last digit. As this is also the limit of the measurement device, this uncertainty might be caused by the counter and not the clock. This would explain why the uncertainty is of the same order of magnitude regardless of the selected frequency. Nevertheless, the measured precision is thus at least in the 10^{-1} μs range.

Second, the difference between the rated frequency and the actual measured frequency is on the 6th or 7th digit, which is clearly measurable by the instrument. The relative error is less than 10 ppm for the modes with an exact frequency and less than 30 ppm for the last two modes. This translates into a drift of 10 μs and 30 μs, which means that the main contribution of uncertainty from the radio clock comes from drift and not jitter.

Third, the measurements are quite consistent between Mote A and Mote B, which means that the precision and drift most likely are of the same order of magnitude for all the motes.

Mode	Mote A			Mote B		
	Frequency	σ <μs>	δ <ppm>	Frequency	σ <μs>	δ <ppm>
SCM	1027 $\pm$ 1 kHz	974	27000	988 $\pm$ 1 kHz	1012	12000
FEI	1221.5 $\pm$ 0.2 kHz	164	221500	1219.4 $\pm$ 0.2 kHz	164	219400
XCLK	3906.243 $\pm$ 0.001 Hz	0.26	46851	3906.252 $\pm$ 0.001 Hz	0.26	46849

Table 5: Measurements of the different ICG modes with the TPM module as external clock.

The measurements of the internal clock can be seen in Table 5, with the `Mode` column being the selected operating mode of the ICG, `Frequency` being measured frequency, σ the corresponding precision in μs, and δ the drift in ppm.

It should be noted that the comparison being made is between the different clock sources, with the SCM mode being based on the 8 MHz internal reference clock and the FEI mode on the 243 kHz internal reference clock. During emulation of an external clock by the TPM module a divider of 8 was introduced thus leading to only a fraction of the internal clock being measured. This means that when set the internal clock is set at 8 MHz in SCM and FEI mode the output is 1MHz. In XCLK mode the external radio clock is connected directly to the TPM module, and with the radio clock set at 32.786 kHz the output is 4.09825 kHz.

First, by comparing the measurements in XCLK mode with the ones obtained by measuring the radio directly, it can be seen that the same 7 digit precision remains,

and that the difference between the two motes is of the same order of magnitude as when measuring the radio clock directly. However, there appears to be a discrepancy of 192 counts between the two methods.

This delay is probably caused by the TPM module and the process of generating an external clock since a delay in the ICG module of this magnitude would result in a drift of $\frac{192}{4.09825kHz} = 47ms$. This would make any timing critical applications on the EVB13192 platform impossible, which clearly is not the case as this particular setup can be used for IEEE 802.15.4 MAC layer handling, which requires timing operations in the μs range.

It thus stands to reason that the measurements of the two other clock sources also contain this delay, and that this must be taken into account when using the absolute value to estimate the drift. However, since the uncertainty does not seems to be affected by this delay, the measured jitter can still be used to estimate the precision of the clocks.

Second, the precision of the SCM mode appears to be of the orders of 10^3 μs and the FEI mode of the order of 10^2 μs, which is four and three orders of magnitude worse than the radio clock. That the precision of FEI mode is an order of magnitude better than SCM mode can primarily be contributed the FLL module being engaged in FEI mode.

Third, the drift in SCM mode is of the order of 10^4 ppm, which also happens to be the same order as the aforementioned delay. However, as the measurements of Mote A show a higher value than expected, the delay would cause an even larger drift, hence it is safe to assume that the drift must be of the orders of 10 ms as well. The drift in FEI mode is of the order of 10^5 ppm, which is clearly larger than the delay. But, due to the precision being three orders of magnitude higher than the drift, it must be possible to remove this high discrepancy between the expected value and the measured one by trimming the 243 kHz reference clock. The problem, thus, lies in an untrimmed clock rather than in an imprecise one.

Temic U4224B

In order to measure how precise the analog amplitude modulation of the radio signal can be converted into a digital signal, the performance of the `Temic U4224B` chip was evaluated. It is known that the beginning of each pulse in the DCF77 signal is defined within ± 0.3 μs, thus, regardless of the encoded value, the length between a second is defined within ± 0.6 μs.

By measuring the time between seconds with the timers, the DCF77 signal can be used as a point of reference and negate uncertainties in the ICG caused by drift which were shown in the last section to be a more dominant source of error than the jitter. Because the precision of the ICG can be of the same order of magnitude

as the transmitted signal when used in this way, any uncertainties caused by the `Temic U4224B` should be detectable.

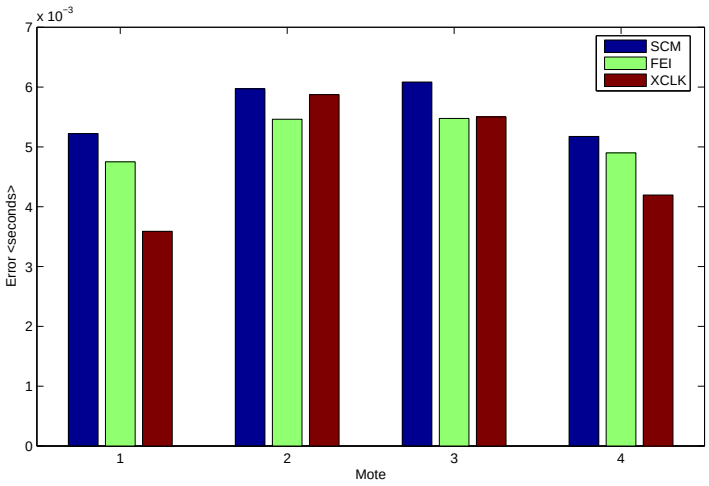


Figure 12: The standard deviation of the length of a second based on the DCF77 signal, measured with the timers.

With the application `TestDCF77Chip`[10] over 1000 seconds were counted individually, and by calculating the mean an estimate of the actual system clock was obtained. By dividing the standard deviation of the counts with this system clock, the standard deviation of the length of a second was found. This was done for four motes and with three different clock sources. The internal clock was kept at 8 MHz while the external clock was at 32.786+ kHz. The result can be seen in Figure 12.

Regardless of the clock source, the uncertainty of each second lies between 3 ms and 7 ms. This is four orders of magnitude worse than the uncertainty at the transmission time. This uncertainty is also worse than the precision of the ICG and thus the measuring precision, meaning the uncertainty must come from the `Temic U4224B`. Looking at the datasheet for the `Temic U4224B` [8] confirms this as the precision of each pulse transition is only defined with ± 35 ms accuracy. Since the precision of the entire synchronization process cannot be more precise than the definition of the individual second, this uncertainty from the `Temic U4224B` has a huge impact.

To verify that the `Temic U4224B` indeed is at fault here, and to investigate if the delay is random or perhaps common among several `Temic U4224B` chips, the length of each individual transition of the DCF77 signal was measured by counting. This was done for four motes, using the radio clock, and the time-code itself to align the measurements. The standard deviations were subsequently calculated for each

[10] Available under the DIKU contribution directory to TinyOS at SourceForge.

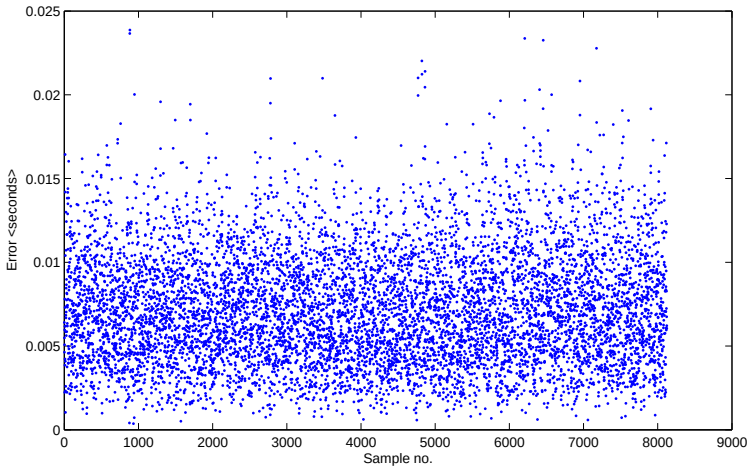


Figure 13: The standard deviation of the agreement between 4 motes on the number of counts between each transition in the DCF77 signal, measured with the timers.

and every transition. With the measurements aligned, each standard deviation is an estimate of the agreement between the four motes. A plot of all the standard deviations can be seen in Figure 13.

If the delay from the `Temic U4224B` was common for the four motes, a pattern should be visible from the standard deviations. This is clearly not the case. As a matter of fact, the distribution of the standard deviations appears to be completely random.

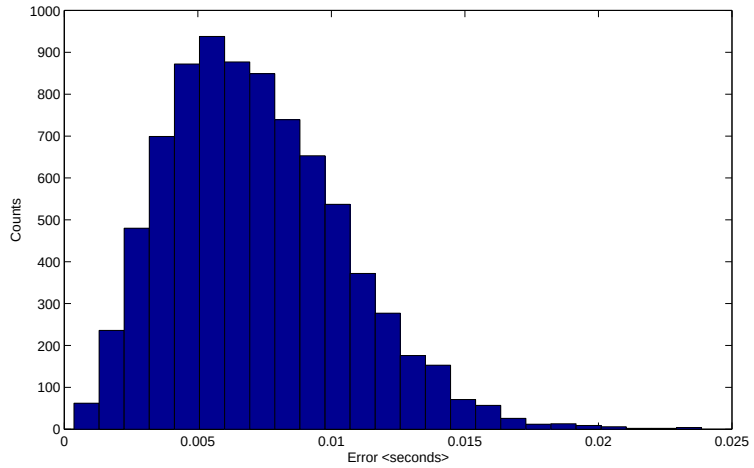


Figure 14: A histogram of the standard deviation of the agreement data.

By plotting the standard deviations in a histogram, one gets Figure 14. Assuming the histogram follows the Normal Distribution, the precision of the `Temic U4224B` is (7 ± 3) ms. This is higher than one might suspect after studying Figure 12. The main difference being one was based on the sum of two transitions and the other on the individual transitions. Apparently, it is possible to even out the randomness. This would be the case if over-counting one transition would mean the under-counting of the subsequent transition. In this case, it would be possible to average out the errors from each transition.

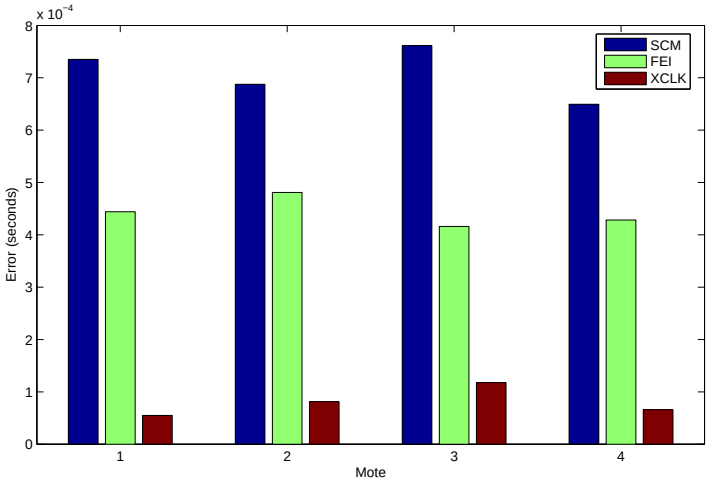


Figure 15: The standard deviation of the length of a minute based on the DCF77 signal, measured with the timers.

To verify this, the first experiment with counting the length of a second was redone, but this time to count the length of an entire minute. Because of the DCF77 timecode being exactly 1 minute long, this interval is well defined by the DCF77 signal. The result of this is shown in Figure 15.

First, it can be seen that the error has fallen with at least an order of magnitude. This was expected due to the number of references has been increased by 60 i.e. a minute.

Second, it is now possible to see a distinct difference between the three clock sources. Interestingly, the limit of the SCM and FEI mode is almost identical to the estimated precision from the previous section. The precision of the XCLK mode is, however, far from reached, and thus shows the precision to be $(8 \pm 3) \times 10$ μs, when one counts with respect to a whole minute and not just a second.

To summarize: By measuring an entire minute, it is possible to reduce the uncertainty of each second by at least one order of magnitude. The usability of this, however, is somewhat limited since the beginning of a second must always be used

as a point of reference. Within the same minute, or any other period based on the same reference point, the difference between two points in time can be measured with μs precision.

The data files containing the measurements shown above can be downloaded from:

http://www.powerplay.dk/marcus/3.2.1-precision-chip.rar
http://www.powerplay.dk/marcus/3.2.1-precision-secmin.rar

3.2.2 TinyOS

We wish to use the global synchronization obtained from the DCF77 signal to improve scheduled communication. However, because medium access is not the only application that would benefit from a global synchronization, we have split the task at hand up in two parts:

- A DCF77M module that handles the general synchronization aspects, such as reception and decoding of the DCF77 signal and methods to easy access this information. Any application in need of ms accurate synchronization can use this module.
- A DCF77HibernationM module that functions as an alarm clock, similar to the standard TinyOS alarm clock modules, but with the main difference being the range. This can be used in scheduled communication to signal the precise moment to start listening or transmitting.

The two components can be seen in context in Figure 16, and both have been made part of DIKU's contribution to the TinyOS project at SourceForge,[11] and can be used in any TinyOS application by adding the line `SENSORBOARD=lpreceiver` to the application's `Makefile`.

DCF77M

In order to use the DCF77 time-code in TinyOS, a full 59-bit signal must be received and successfully decoded. As the signal might be corrupted or completely lost due to interference, we wish to seek an approach which maximizes robustness and minimizes corrupt decoding on the cost of performance. Because CPU time is much less power consuming than idle listening with the radio, a robust synchronization will be more power efficient than a less complex algorithm when used in scheduled communication. We will later show that interference can be an annoying obstacle and that the algorithm we present, although robust, still is light enough to allow other applications to function. A few assumptions must be made, though, and limitations will be presented as well.

[11] http://tinyos.sourceforge.net

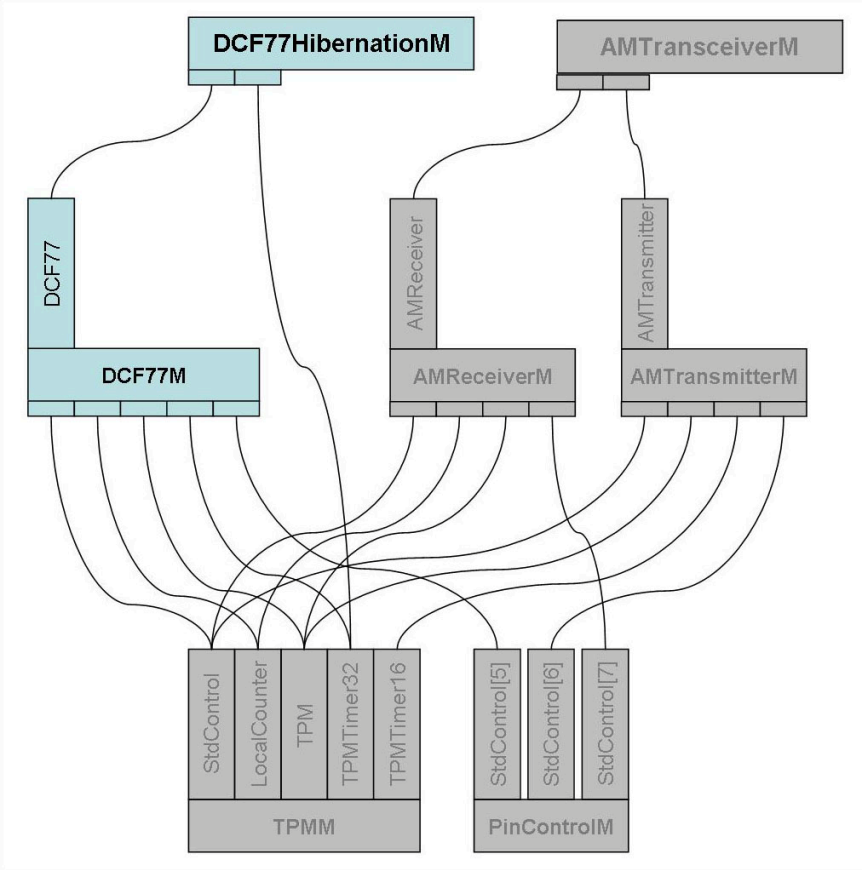


Figure 16: Component overview of the DCF77M and DCF77HibernationM modules.

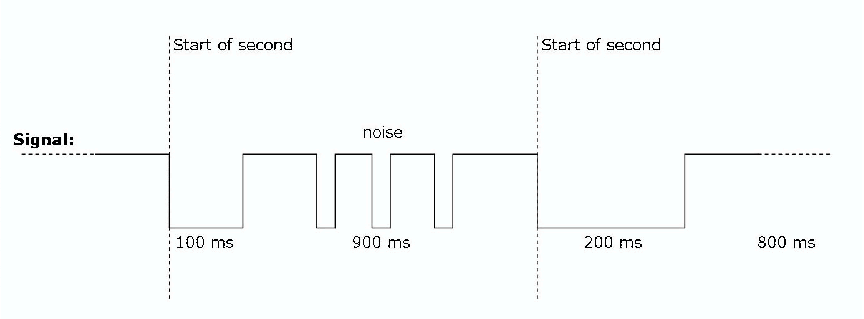


Figure 17: Simplified model of a DCF77 signal with noise.

The first assumption we make is about the noise. Since the signal received by the TPM module is converted to a timestamp of the last detected edge, and the value of the edge itself, we assume that any noise received will be detected as an edge as well, and that the associated timestamp will hold the length of the noise. This is illustrated in Figure 17. By summing-up all the timestamps of the noise, it will still be possible to detect when enough time has passed since the last second. However, this assumption breaks down if the frequency of the noise is so high that the TPM module cannot differentiate amongst them and timestamp them correctly. Under such circumstances, it is doubtful that any successful reception is possible anyway. Also, this summing-up of noise will not be done at the beginning of each second since the duration of the modulation is often of the same magnitude as the noise.

Second, it is assumed that an estimate of the clock driving the TPM module is known. Because this value is used by other TinyOS modules as well e.g. the UART, it is not a limiting assumption. This value is mainly used to help identify the different kind of edges in the DCF77 signal, and also to sum out the noise as mentioned above. Since clock drift might be an issue, the clock value is only used as a guideline in the initializing phase, and an estimate which is off by as much as 50% can still be used.

As shown above, the DCF77 signal contains three parity bits. Because a lost or undeterminable pulse is detectable, it is possible to use the parity bits not only to discard but also to repair a corrupt signal under certain conditions. This feature will, however, not be used due to one bad pulse possibly being the result of a bad signal, and a voluntary loss of synchronization is much more preferred than an faulty synchronization.

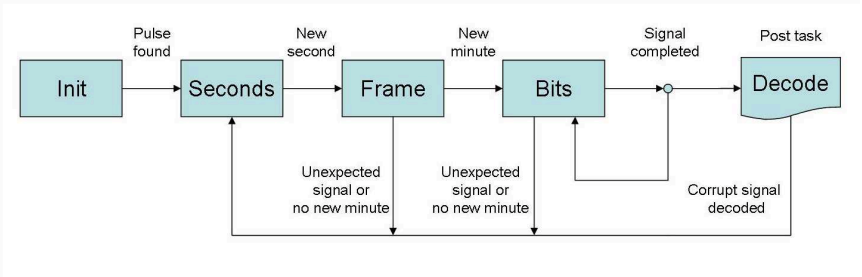


Figure 18: Finite-State-Machine of the reception process of the DCF77 signal.

The approach we seek in receiving the DCF77 signal correctly can be described by the Finite-State-Machine shown in Figure 18.

In the `Init` stage, nothing is known except an estimate of the system clock. This is used to calculate an estimate of the width of a 100 ms pulse, 200 ms pulse and an acceptable deviation value of 50% as well. When a signal matches a low pulse

with the width of 100 ms or 200 ms (plus-minus the deviation), a second has started (100 ms or 200 ms ago) and the state changes to `Seconds`.

In the `Seconds` stage, we still look for a low pulse with the width of 100 ms or 200 ms. When this occurs, a new second has started. By summing-up the passed counts since the last second, an estimate of the system clock is found. This new and more precise value of the system clock (precise within (7 ± 3) ms as shown above) is used to calculate better estimates for a 100 ms pulse, 200 ms pulse and deviation. The state then changes to `Frame`.

In the `Frame` state, we look for the beginning of a new DCF77 signal frame. In essence, we look for two seconds to pass without any low pulses with the width of 100 ms or 200 ms in the middle. If this is detected, we progress to the `Bits` state. However, if a frame has not been detected within a minute or pulses appear where they are not expected, we regress to the previous `Seconds` state. Any regression results in the restoration of the initial clock estimates as the new estimates might be the root to the unexpected signal.

In the `Bits` state, the pulses are decoded based on their width and inserted into a buffer. When a new frame is detected i.e. the buffer is full, a decoding task is posted and the buffer is switched. The counts accumulated during the past minute are used to calculate an even better estimate of the clock and pulse widths. This count can also be used in the subsequent minute together with the new accumulating count to calculate a fraction of a minute. This fraction can be converted to a timestamp with μs accuracy.

As before, if any unexpected signals are detected or a new frame is not detected within a minute, the system regresses to the `Seconds` state. This regression can also be initiated by the decoding task if either the parity does not match or if the newly decoded DCF77 timestamp is not consistent with the known time. The latter can only occur if the mote is already synchronized, though.

If the timestamp is successfully decoded, the values are stored as the present time, a flag is set to indicate the mote is synchronized, and an alarm clock is set to refire every second based on the calculated system clock. This alarm clock can be used by other components as a heartbeat pulse with ms accuracy. When the mote loses synchronization, this alarm clock continues to count the seconds since the last synchronization. It is thus possible to maintain a fairly accurate synchronization even though the DCF77 signal is lost periodically. Because each second is only defined with (7 ± 3) ms accuracy, this will also be the drift-per-second for the out-of-synchronization time. Also, whenever the mote changes state, a signal is sent to the component above since certain applications might need to act differently when synchronized and when unsynchronized.

Finally, the module includes several commands to read the DCF77 timestamp.

These include both reading the time in milliseconds since midnight and formatted in HHMMSSmmm,[12] with the pure millisecond timestamp being suitable for arithmetic operations and the formated one being readable by humans. The date can only be read as YYYYMMDD,[13] however, if any arithmetic operations need to be performed on the scale of years, the module supports standard UNIX timestamping[14] which combined with the millisecond timestamp gives the highest available precision. The Unix timestamp also has the feature of being widely used which means timestamps taken directly from a measurement from a mote can be used without any conversion in almost any application.

DCF77HibernationM

With the ability to do general purpose global synchronization, we wish to apply this feature to scheduled communication. The main feature we need is to be able to signal a component at a predetermined time similar to the regular alarm clocks, but with the precision of the DCF77 signal. We also wish to turn off as much as possible when we wait i.e. hibernation.

As we showed in Section 3.2.1, the main contribution to drift on the EVB13192 platform is caused by the discrepancy between the believed clock rate and the actual measurable clock rate. This is especially the case when the radio clock is not used and the internal clock has not been trimmed properly. By using the measured clock rate from the DCF77M module, we can remove this uncertainty and keep the drift fixed at (7 ± 3) ms at most. This is, of course, several orders of magnitude worse than the drift from the radio clock, but to minimize power we assume that the radio clock is not present when we need to hibernate. Because the enhanced precision from the FEI mode is of no use due to the (7 ± 3) ms uncertainty, the mote is kept in SCM mode to further save power.

Since power consumption is the key issue, we do not wish to simple idly wait with the `DCF Receiver Board` turned on at all time, but rather rely on the internal clock whenever possible and only turn on the `DCF Receiver Board` when necessary. To do this we need to make some assumptions:

- Synchronization takes at least 1 minute, due to the length of the DCF77 signal, and at most 2 minutes under ideal conditions. To compensate for turn on time, jitter causing the `DCF77M` module not to start in the `Init` state, and in general to favor robustness we define synchronization time to be 3 minutes.

[12] HH = hour, MM = minute, SS = second, mmm = millisecond.

[13] YYYY = year, MM = month, DD = date.

[14] Standard Unix time counts the number of seconds since the Unix epoch 00:00:00 UTC on January 1, 1970.

- Until now, the temperature induced clock drift has not been taken into account as this is a relatively slow process compared to the every second pulses from the DCF77 signal. However, if the mote is going to hibernate for days, even months, this drift must be taken into account. According to the datasheet, the drift is between -0.3% - 0.1% in the operating range of the mote. Together with the (7 ± 3) ms uncertainty from the DCF77 decoding chip, we assume total drift to be no more than 1.0%. This is an overestimate and in general, the drift will be lower because the two contributions can negate each other.
- We assume that the decoded DCF77 signal is always correct, meaning that, IF a synchronization is achieved the time and date will be correct.

To set a wakeup time, a millisecond offset and a sleep interval in seconds must be set. By splitting this up in two, the maximum sleep interval is increased three orders of magnitude, and, due to the offset itself not being restricted to fall within a second, this splitting can be used in an efficient way to allocate timeslots to different motes e.g. by having an offset based on the address.

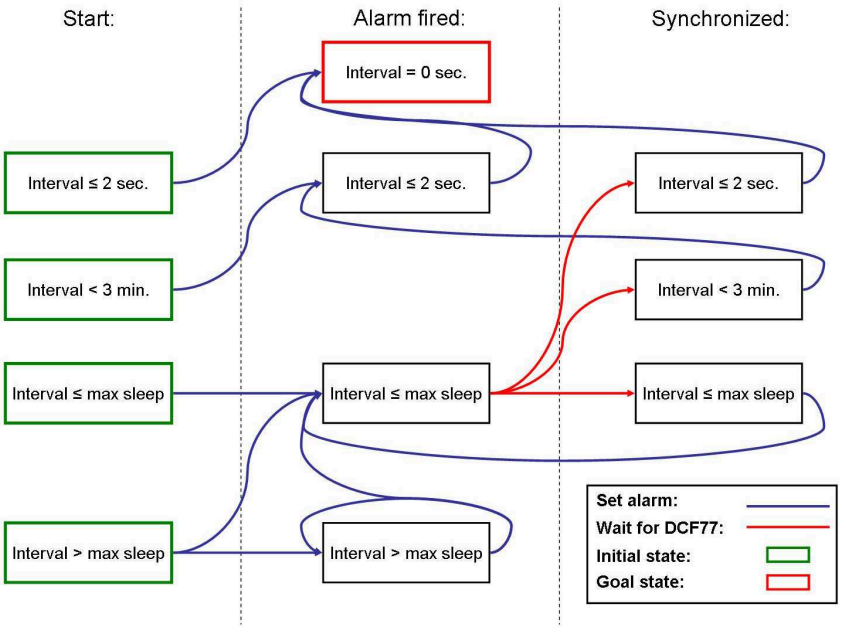


Figure 19: Finite-State-Machine of the hibernation process.

The Finite-State-Machine shown in Figure 19 has been used to control the hibernation process. The FSM can be regarded as 2-dimensional, with one dimension being controlled by the event:

`{Start}`, `{Alarm fired}`, and `{Synchronized}`

And the other dimension being controlled by the remaining sleep interval:

`{Interval = 0}`, `{Interval` $\leq$ `2 sec.}`, `{Interval < 3 min.}`, `{Interval` $\leq$ `max sleep}`, and `{Interval > max sleep}`.

State changes occur when an event is triggered and the remaining sleep interval is changed. This happens in the following way:

- **Interval = 0**
  This is the goal state. When this state is entered, the calling component is signaled and hibernation is over.

- **Interval $\leq$ 2 sec.**
  The sleep interval and the offset are combined to set an alarm clock, and the remaining sleep interval is set to zero in order to indicate a wakeup must be signaled when the alarm is fired. The value 2 was chosen to ensure enough time to set the alarm clock since `less than 1` could result in the alarm being set too late.

- **Interval < 3 min.**
  This is based on the synchronization time discussed above because, with less than 3 minutes to wakeup, there is not enough time to synchronize. The alarm clock is set with the sleep interval minus 2 seconds, and the remaining sleep interval is set to 2 seconds.

- **Interval $\leq$ max sleep**
  Because the timers, controlling the alarm clock, have to count several thousand counts each second, there is an upper bound on how many seconds the alarm clock can be set. If the sleep interval is less than this bound, it is possible to sleep the entire interval with only firing the alarm clock once. However, this is not a very robust approach as this upper bound easily can be days, and thus the clock would have drifted too far by then.

  In our case the timer controlling the alarm clock is 32-bit wide which would give a maximum sleeping time of 0xFFFFFFFF counts. However, since the alarm clock is based on the 16-bit timers from the TPM module, the maximum consistent sleeping time is 0xFFFF0000 counts.

  `Start` and `Synchronization`
  Since we chose robustness over performance, in this case power consumption, we set the alarm clock to the sleep interval, minus the 3 minutes needed for synchronization and minus 1% to account for drift. Also, the remaining sleep interval is set to 3 minutes without the drift compensation. The reason behind this is that if a synchronization is possible afterward, the sleep interval will be set correctly anyway, and, if not, it will still be possible to

wakeup before the actual wakeup time, and thus give the calling component a chance to recover in time.

`Alarm fired`
At this point we have an underestimate of the remaining time, due to the compensation for drift. The `DCF Receiver Board` is turned on and a 3 minute long alarm is set. If synchronization is not possible within this period, the receiver is turned off and the alarm is set again. If enough time remains for another synchronization attempt, the alarm is set to the remaining sleep interval minus 3 minutes. Otherwise, the synchronization attempt was the last one possible and thus the wakeup time is now.

- **Interval > max sleep**
  If the remaining sleep interval is larger than the possible maximal sleep interval, a sleep time as close as possible is chosen, but with at least 3 minutes left afterward to perform a synchronization. Again, the sleep interval set in the alarm clock is compensated for 1% drift but the remaining sleep interval is not.

3.3 Evaluation

With the DCF77 components complete, the performance of the setup was evaluated. This was done qualitatively by evaluating the functionality and working conditions, and quantitatively by measuring the agreement of the synchronization. Finally, a simple scheduled communication protocol was developed based on the hibernation module.

3.3.1 Quality of Service

The DCF77M module was tested with the `TestDCF77Clock`[15] application which essentially works as a clock. After a successful synchronization, the heartbeat-pulse is used to post a task every second which prints the current date and time to the console.

A screenshot of the application running through Re-Mote [23] can be seen in Figure 20.[16] An EVB13192 mote programmed with the `TestDCF77Clock` application and connected to a laptop was subsequently tested different in locations. The main observations we made were:

- Steel frame buildings completely shield the DCF77 signal.

[15] Available under the DIKU contribution directory to TinyOS at SourceForge.

[16] Because the output from the mote runs through several layers before reaching the Re-Mote window, it cannot be used to measure timing. The clock inset is thus mainly for reference and not comparison.

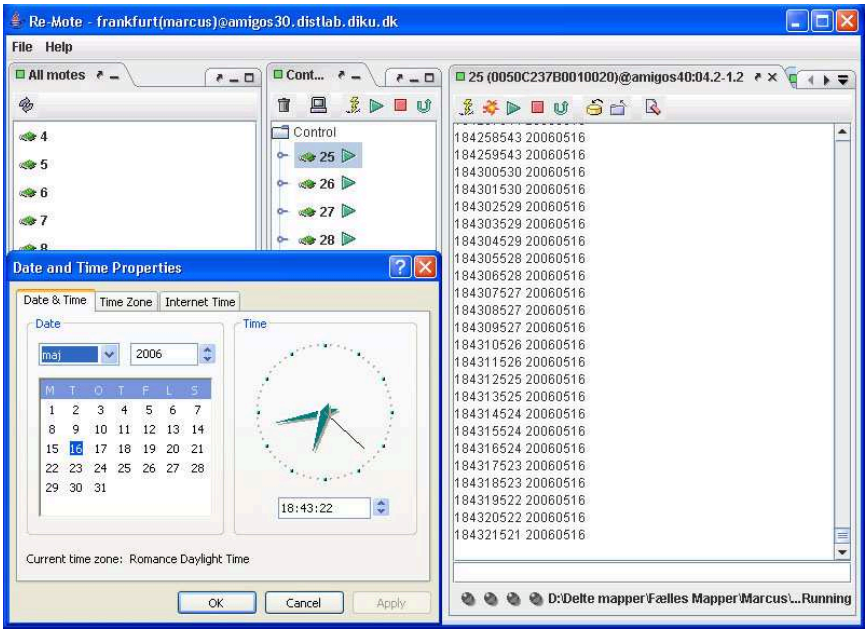


Figure 20: Screen-shot of the output from a mote running the `TestDCF77Clock` application taken through *Re-Mote* [23], with the *Windows XP Date and Time* window inserted.

- Domestic and generally older buildings have excellent reception.
- TVs and computer monitors can interfere if they are placed within 5-10 meters.
- Unshielded computers e.g. laptops can interfere within 1 meter.

The buildings tested were ITU[17] (steel frame), DIKU[18] (older building), and several domestic residents.

To evaluate the downtime at the receiver end, 5 motes were programmed with a modified version of the `TestDCF77Clock` application[19] and set to run continuously for more than two days. This was done during the weekend because of the motes being placed at DIKU, hence, during office hours the reception can be of varying quality, mainly caused by people using their laptops and the nearby arcade machine. The result of this experiment can be seen in Figure 21.

[17] IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S.

[18] Dept. of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen Ø.

[19] This application only prints the date and time of the event of acquiring and loosing synchronization.

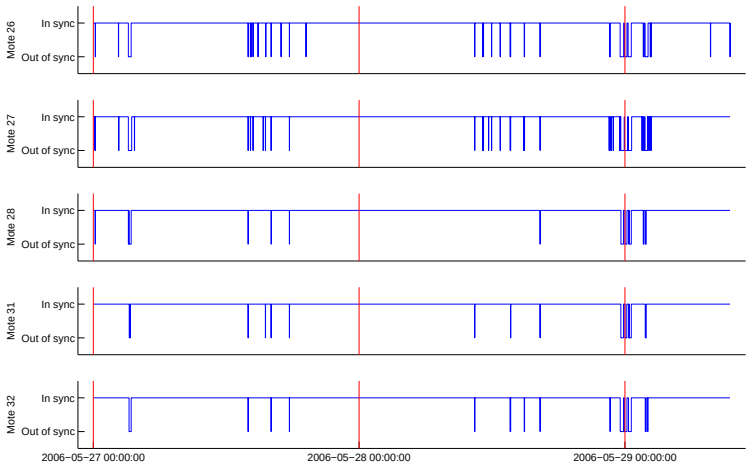


Figure 21: Graphical representation of five motes being in or out of synchronization, with the red lines marking the beginning of a new day.

First, by comparing the periods of downtime, it can clearly be seen that these fall inside large windows, often several hours long, but common to all five motes. The agreement between the five motes suggests that the cause of this downtime must be external interference. This is probably caused by signal cancellation between the aforementioned ground wave and reflection wave due to the shifting altitude of the reflective atmospheric layer caused by temperature changes. This is consistent with DIKU being approximately 700 km from Frankfurt.

	Mote 26	Mote 27	Mote 28	Mote 31	Mote 32
Downtime	3.5 %	4.0 %	1.9 %	1.7 %	2.2 %

Table 6: The ratio between periods without synchronization and the total observed time.

Second, although some peaks are common for all five motes, not all of them are. This means that some receivers are experiencing less interference than others. This becomes evident when taking the ratio between the accumulated time spent without synchronization with the total time observed as shown in Table 6. Since the five motes are placed within 1 meter of each other, they must be experiencing the same atmospheric cancellation. However, local fluctuations caused by reflections, and localized interference caused by nearby radio sources can still occur as shown by the previous test. This emphasizes the importance of an environment with as little human made interference as possible. Whether this interference can be reduced by more expensive receivers and antennas remains to be explored.

By plotting the length of each period without synchronization in a histogram, as

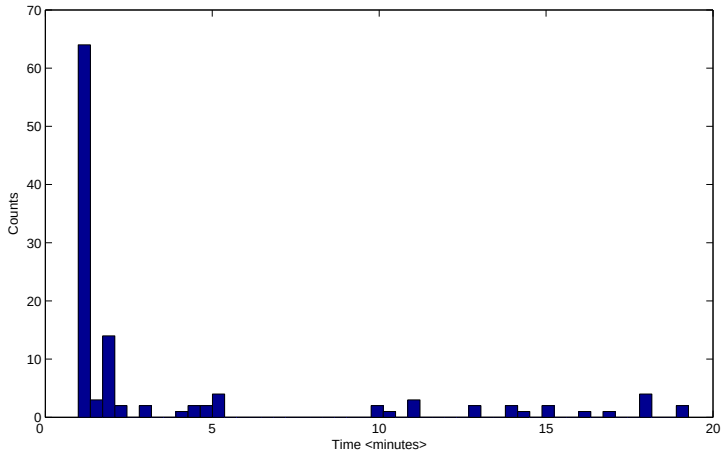


Figure 22: The distribution of the length of each period without synchronization.

shown in Figure 22, it can be seen that although the total downtime is of the order of 2-4 %, the majority of these periods is less than 5 minutes long. As a matter of fact, 75 % of all the periods have a duration of 3 minutes or less.

The data files containing the measurements shown above can be downloaded from:

http://www.powerplay.dk/marcus/3.3.1-quality-of-service.rar

3.3.2 Agreement

The main reason to do synchronization in the first place is for a group of motes to agree on when an event took place. To test the quality of the synchronization, the application `TestDCF77Agreement`[20] was used. The test consists of one transmitter, and four receivers. The transmitter broadcasts three packets every second, and the receivers timestamp the reception time of the packet and print this to the console. Because the motes are placed next to each other, the propagation time of the electromagnetically wave can be neglected, and for all intents and purposes the four motes receive the packets at exactly the same time. Since we wish to measure the quality of the synchronization, the external radio clock was used to achieve optimal precision.

By calculating the standard deviation of the four timestamps associated with each packet, the uncertainty can be estimated. This was done for 10000+ packets and the result can be seen in Figure 23.

First, it can clearly be seen that the distribution is not random, and that the errors are clustered together on plateaus. This is in sharp contrast with the randomness

[20] Available under the DIKU contribution directory to TinyOS at SourceForge.

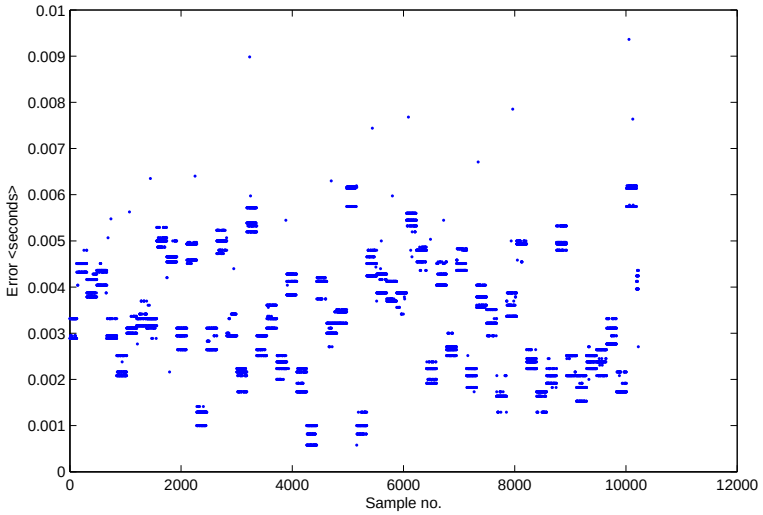


Figure 23: The standard deviation of the agreement of reception timestamps.

encountered when evaluating the `Temic U4224B` chip directly which was shown in Figure 13. The reason behind this is the way each second and millisecond are calculated. By accumulating the seconds over an entire minute, the precision in determining events inside the same minute is increased to μs precision. Because the reference point only changes once a minute, all timestamps within the same minute are thus clustered together. This confirms the use of accumulated seconds to increase precision within a minute.

Second, the deviations are of the orders of ms which means the `Temic U4224B` chip is indeed the weakest link. A more precise decoding chip would instantly increase the precision with at least an order of magnitude.

Ordering the standard deviations in a histogram as shown in Figure 24 the distribution appears to be Normal Distributed. The precision of the synchronization can then be calculated to be (3.3 ± 1.3) ms. Not surprisingly, this is approximately half the precision found for the `Temic U4224B` chip in Section 3.2.1 since when counting the length of a segment, the uncertainty of both the beginning and the end will contribute to the precision. However, when using the beginning of a second as reference point, the same uncertainty is only counted once and not twice, thus increasing the precision to the double.

In other words, the jitter of the DCF77 driven clock is (3.3 ± 1.3) ms and the drift is (7 ± 3) ms.

Since in-band synchronization with reference broadcasting and MAC layer timestamping has not been implemented on the Freescale EVB13192 platform, it is not

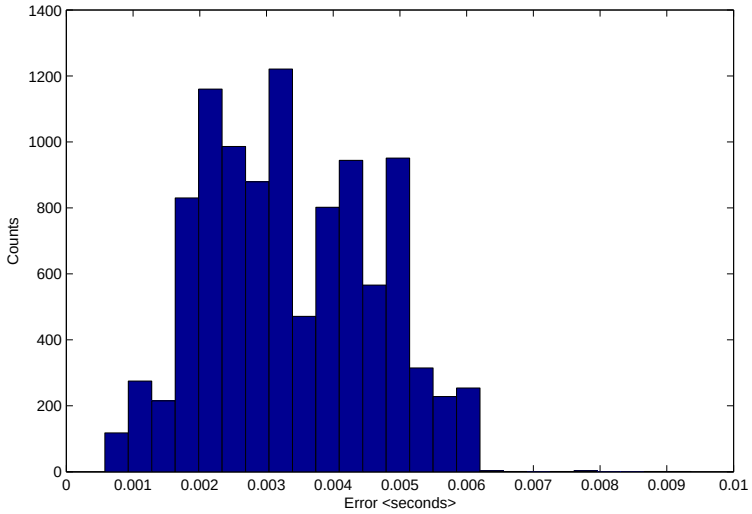


Figure 24: A histogram of the standard deviation of the agreement of reception timestamps.

possible to perform a direct comparison between the two approaches. However, because no synchronization can be more precise than the clocks that are being synchronized, it stands to reason that, when using the Freescale mote in SCM or FEI mode, a precision of 1 ms or 0.1 ms respectively is the highest achievable when performing in-band synchronization. With the mean error in multi-hop reference broadcasting growing with the square-root of the number of hops, it would require 10 or 100 hops in order for the error to reach the same order of magnitude as the DCF77 driven clock.

This holds for the XCLK mode as well, but since the precision of the radio clock is greater than the individual CPU instructions, the clock is not the limiting factor. Rather the uncertainty caused by the processing power is the limiting factor. It is thus not possible to give an estimate on the precision without knowing the overhead caused by implementation.

In a data acquisition application, such as the recently conducted Hogthrob experiment, measurements must be timestamped before further processing. Assuming that a precision of (3.3 ± 1.3) ms is acceptable, the question is then how much impact does the `DCF Receiver Board` has on the energy budget of a mote.

Based on the results from [3], we assume that we have a platform with a power supply of 18.9 kJ[21] By assuming different lifetimes based on this capacity, we

[21] This is the approximately energy stored in three rechargeable NiMH batteries, under the assumptions that each battery has a mean voltage of 1.25 V during discharge, and a 1400 mAh capacity when self-discharge is taken into account.

can calculate the reduced lifetime caused by the power consumption of the `DCF Receiver Board`. This is done by first calculating average energy consumption for the platform:

$$P_{plat} = \frac{18.9\ kJ}{\Delta t}$$

The new lifetime can then be found by solving the equation:

$$\Delta t_{new} = \frac{18.9\ kJ}{P_{plat} + P_{DCF}} = \frac{18.9\ kJ}{P_{total}}$$

These reduced life expectancies for different values have been calculated in Table 7.

		DCF Receiver Board		
Δt <d>	P_{plat} <mW>	P_{tot} <mW>	Δt_{new} <d>	Loss <%>
7	31.25	31.46	6.9	0.67
14	15.63	15.84	13.8	1.33
30	7.29	7.50	29.2	2.80
60	3.65	3.86	56.7	5.45
90	2.43	2.64	82.8	7.95
180	1.22	1.43	153.5	14.73
360	0.61	0.82	267.5	25.68

Table 7: Estimated lifetime reductions when using the `DCF Receiver Board` to timestamp measurements.

In the aforementioned Hogthrob experiment, the motes were equipped with power supplies with a similar capacity as used in our calculations [3]. The experiment was set to last for 20 days, however, the motes lasted for 30-40 days before depleting their power supply. Given this time frame and power supply, timestamping with DCF77 synchronization would indeed be feasible, as the reduced lifetime of the mote would barely be a day, which is well within the safety margins of the motes lifetime.

The data files containing the measurements shown above can be downloaded from:

http://www.powerplay.dk/marcus/3.3.2-agreement.rar

3.3.3 Scheduled Communication Protocol

As a proof-of-concept, a simple scheduled communication application, `TestDCF77Schedule`,[22] was built on top of the `DCF77HibernationM` module. A communication frame of

[22] Available under the DIKU contribution directory to TinyOS at SourceForge.

20 ms was chosen in order to allow for drift, turn on time of the radio, and packet preparation and transmission time.

Each mote was assigned a timeslot based on the `TOS_LOCAL_ADDRESS` variable, and this was used as offset in the `DCF77HibernationM` module. The mote with the lowest address was assigned as gateway, thus being the first to turn on its radio. This mote would return to hibernation when either it had received a message from the last mote or an alarm had fired. The regular motes were set to hibernate immediately after the transmission was done as no messages were sent to these motes in this test. An alarm clock could just as easily have been used.

The hibernation period was set to one whole day starting at midnight i.e. the motes would hibernate until midnight (plus an offset), turn on their radios and broadcast a message to the gateway, and then hibernate until next midnight. Three motes were used in this test, one gateway and two regular motes. The TPM module was driven by a 62.5 kHz clock. After two days the gateway had printed the following:

```
# DCF77 in sync
# Entering hibernation until 1146528000
# 165600111 20060501
# DCF77 out of sync
# 165600125 20060501
# DCF77 in sync
# 235000103 20060501
# DCF77 out of sync
# 235000116 20060501
# DCF77 in sync
# 235900101 20060501
31 418
32 441
# Re-entering hibernation until 1146614400
# DCF77 out of sync
# 461 20060502
# DCF77 in sync
# 184400104 20060502
# DCF77 out of sync
# 184400118 20060502
# DCF77 in sync
# 234300100 20060502
# DCF77 out of sync
# 234300114 20060502
# DCF77 in sync
# 235900109 20060502
31 421
```

```
32 437
# Re-entering hibernation until 1146700800
# DCF77 out of sync
# 457 20060503
```

By looking at the time between the first and second wakeup, it can be seen from the transcript that the gateway synchronized itself three times during that day. The maximum sleeping time of 0xFFFF0000 counts, combined with the clock rate of 62.5 kHz, and subtracted 1% to account for drift results in 18 hours, 53 minutes and 51 seconds or just 68031 seconds. The actual sleeping time of the mote was between 18 hours and 41 minutes and 18 hours and 43 minutes. This gives a total drift of 651-771 seconds or an average drift of 9-11 ms each second which is slightly more than the expected (7 ± 3) ms. This might be the effect of changing temperature which should have an effect over a period this long.

DCF77

Assuming the worst case, where the `DCF Receiver Board` has to be turned on for 3 minutes in order to be synchronized, the energy used by the module during synchronization is:

$$E_{DCF77} = P_{DCF77} \cdot \Delta t_{3\ minutes} = 0.21\ mW \cdot 180\ s = 38\ mJ$$

In comparison, using the `u-blox LEA-4T` GPS receiver once, starting up cold, would consume:

$$E_{GPS} = P_{GPS} \cdot \Delta t_{34\ seconds} = 118\ mW \cdot 34\ s = 3978\ mJ$$

This is two orders of magnitude more than the DCF77 receiver, and due to the internal clock not being more precise than 0.1 ms, the period between resynchronization would be the same. This means that the nanosecond precision would not be useful to reduce overall power consumption.

Using a communication frame of 20 ms, and observing that the power consumption of the Freescale MC13192 radio is 90 mW and 111 mW for transmission and reception respectively, the amount of energy spent during a communication frame is at most:

$$E_{radio} = P_{radio} \cdot \Delta t_{20\ ms} = 111\ mW \cdot 0.020\ s = 2.2\ mJ$$

Thus, the total amount of energy spent in order to be able to communicate once every day is:

$$E_{regular} = 3 \cdot E_{receiver} + E_{radio} = 3 \cdot 38\ mJ + 2.2\ mJ = 116\ mJ$$

$$E_{gateway} = 3 \cdot E_{receiver} + E_{radio} + N_{motes} \cdot E_{radio} = 116\ mJ + N_{motes} \cdot 2.2\ mJ$$

For the regular motes and for the gateway respectively, N_{motes} is the number of regular motes in communicating with the gateway. This amount of energy spent at the regular motes corresponds to idle listening with the Freescale radio for $\frac{116\ mJ}{111\ mW} = 1.045\ s$.

If the scheduled communication requires more than one communication period a day, the number of synchronizations performed at each period decreases as the 1% drift compensating has reduced influence. Since the hibernation algorithm will try to sleep as long as there is still enough time left to perform a synchronization i.e. 3 minutes until wakeup, the required number of synchronizations for different periods are:

- 1 synchronization when period is between 3 minutes and 5 hours
- 2 synchronizations when period is between 5 hours and 19 hours
- 3 synchronizations when period is between 19 hours and 37 hours

If the period is less than 3 minutes, the `DCF Receiver Board` will always be on. The power consumed during a day in this case is:

$$E_{receiver} = P_{receiver} \cdot \Delta t_{1\ day} = 0.21\ mW \cdot 86400\ s = 18144\ mJ$$

The amortized energy consumption as a function of the interval between transmissions, can be found by assuming that each communication period must be at least 20 ms long i.e. consume 2.22 mJ. The result is thus a piece-wise linear function:

$$\begin{array}{lcl} t \in [0.02; 180] & \mapsto & \frac{18144\ mJ}{86400\ s} \cdot t + 2.22\ mJ \\ t \in]180; 18000] & \mapsto & 1 \times 37.8\ mJ + 2.22\ mJ = 40.02\ mJ \\ t \in]18000; 68400] & \mapsto & 2 \times 37.8\ mJ + 2.22\ mJ = 77.82\ mJ \\ t \in]68400; 86400] & \mapsto & 3 \times 37.8\ mJ + 2.22\ mJ = 115.62\ mJ \end{array}$$

By dividing this energy function with the power consumption of the Freescale radio, we get a correspondence between DCF77 power consumption and idle listening. By subsequently dividing this amount of idle listening with the time interval itself, we get a DCF77 power consumption converted to a Freescale radio duty-cycle figure.

E.g. the amount of energy spent in order to be able to communicate once every day corresponds to idle listening with the Freescale radio for 1.045 seconds which translates into a duty-cycle of $\frac{1.045 seconds}{86400 seconds} = 0.001\%$.

The duty-cycle of the DCF77 as a function of time between communication can be seen in Figure 25. According to [32], protocols like S-MAC, B-MAC, and T-MAC can achieve duty-cycles of 1-2 % while their own SCP-MAC can achieve 0.01-0.1 %. As can be seen in the figure, the DCF77 starts with a duty-cycle of 0.4 % which is lower than the regular MAC protocols, and after 3605 seconds it drops below 0.01 % which is lower than SCP-MAC. As a matter of fact, when counting intervals in days, our duty-cycle becomes an order of magnitude lower than SCP-MAC.

Timers

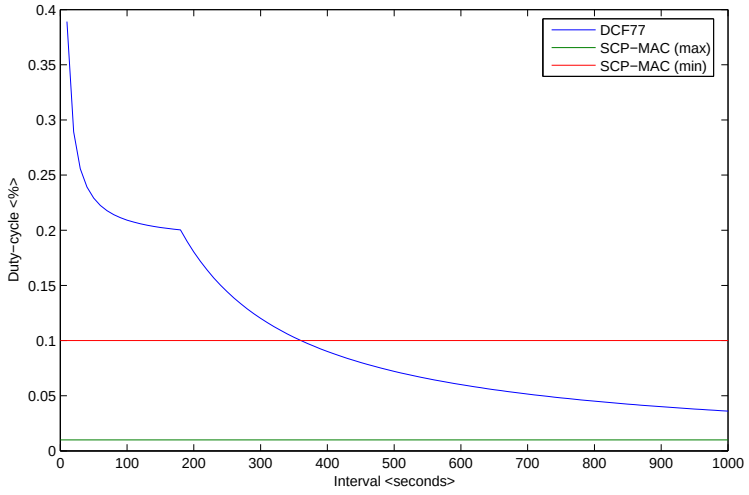
As should be evident by now, drift has an enormous impact on scheduled communication as the time needed for listening is linearly related to the amount of drift between the motes. Two motes with a δ drift each must in the worst case have one of the motes to listen for $2 \times \delta$ in order to be certain that the transmitted packet falls within this period. Thus, the longer the interval between packets is, the longer is the listening period needed to ensure possible reception.

Because synchronization messages can be piggy-backed on regular data messages, there is little difference energy wise between synchronization messages and data messages with piggy-backed synchronization. Hence, in the following analysis of the energy consumption, we do not distinguish between the two, but rather focus on the actual time between transmissions regardless of the type.

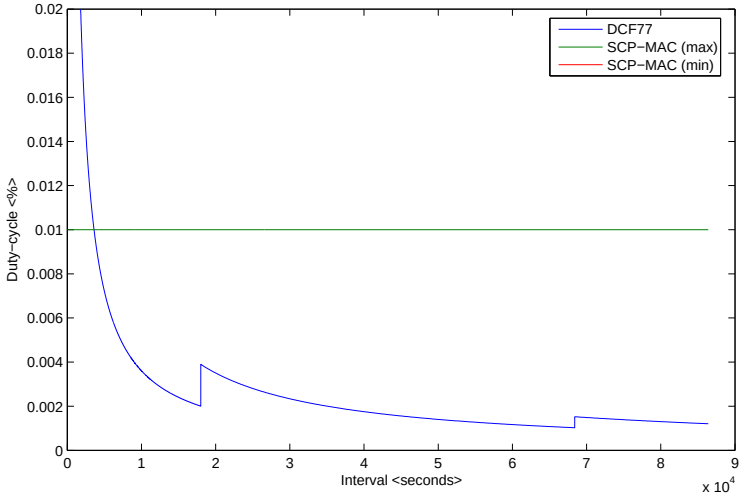
Furthermore, we estimate power consumption used by two motes and with only one transmitted message at each opportunity. Because of this being the absolute minimal setup with any additions automatically raising the power consumption, this is suitable to establish an upper limit of performance.

On the Freescale EVB13192 platform, the drift is circa 10 ms in SCM mode, and at least 0.1 ms in FEI mode, assuming the reference clock has been trimmed. Otherwise the drift could be as bad as 250 ms. However, as it is possible to compensate for drift up to the precision of the jitter, the minimal drift is 1 ms in SCM and 0.1 ms in FEI mode. The amortized energy consumption is thus:

$$\begin{array}{lrcl} \text{SCM:} & t \in [0.001; 86400] & \mapsto & 111\ mW \cdot 0.001\ ppm \cdot t \\ \text{FEI:} & t \in [0.001; 86400] & \mapsto & 111\ mW \cdot 0.0001\ ppm \cdot t \end{array}$$



(a)



(b)

Figure 25: The amount of duty-cycling as function of time between communication.

This is the minimal amount of energy each mote must use to ensure synchronization as a function of time between each packet. The $2 \times \delta$ drift has been averaged over the two motes and because the transmission time is much less than the listening period, the contribution from this has been omitted.

After one day, the drift has accumulated to 86.4 seconds for the SCM mode and 8.64 seconds for the FEI mode. The energy consumed by the radio for listening in these periods is:

$$E_{SCM} = P_{radio} \cdot \delta \cdot \Delta t_{1\ day} = 111\ mW \cdot 0.001 ppm \cdot 86400\ s = 9590\ mJ$$

$$E_{FEI} = P_{radio} \cdot \delta \cdot \Delta t_{1\ day} = 111\ mW \cdot 0.0001 ppm \cdot 86400\ s = 959.0\ mJ$$

When using the external clock, drift falls to circa 1 μs, but at the cost of increased power consumption. Keeping the external clock on for a day would consume more than:[23]

$$E_{doze} = P_{doze} \cdot \Delta t_{1\ day} = 0.105\ mW \cdot 86400\ s = 9072\ mJ$$

And on top of that, the listening period would still have to be at least:

$$t_{listen} = 2 \times \delta_{external\ clock} \cdot \Delta t_{1\ day} = 2 \times 1\mu s \cdot 86400\ s = 172.8\ ms$$

Amortizing over two motes, the minimum energy consumed as a function of time between each transmission is:

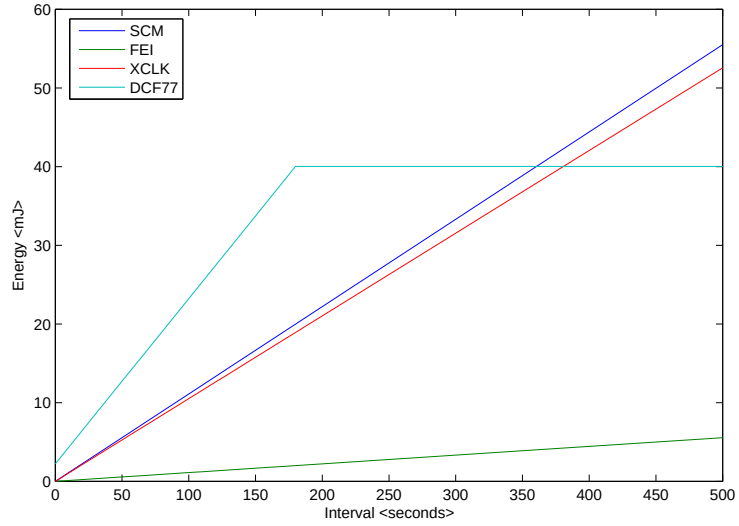
$$\text{XCLK:} \quad t \in [0.001; 86400] \quad \mapsto \quad (111\ mW \cdot 0.000001\ ppm + 0.105\ mW) \cdot t$$

Thus, the power consumed for transmitting one message a day is:

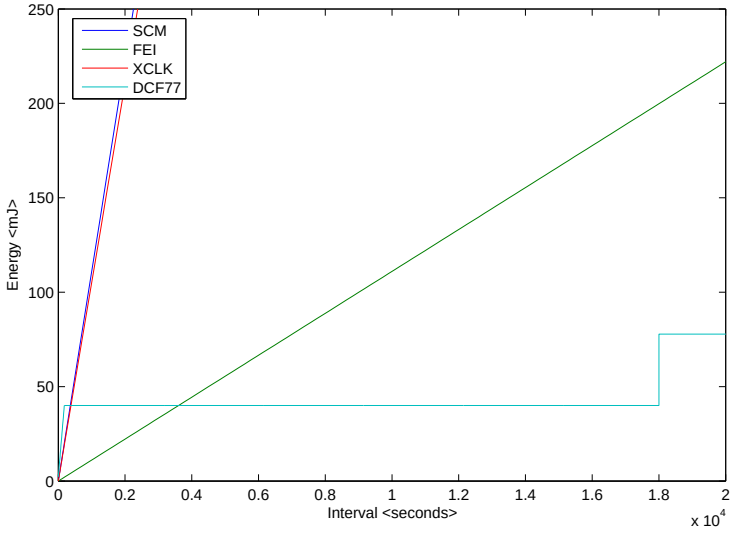
$$\begin{aligned} E_{XCLK} &= (P_{radio} \cdot \delta + P_{doze}) \cdot \Delta t_{1\ day} \\ &= (111\ mW \cdot 0.000001 ppm + 0.105\ mW) \cdot 86400\ s \\ &= 9081\ mJ \end{aligned}$$

The four different power profiles have been summarized in Figure 26 and 27 with the power profile for the regular DCF77 mote being used, and not the gateway mote.

[23] The exact power consumption when in the lowest possible state with the external clock enabled (called Acoma Doze Mode) is not mentioned in the datasheet. The value used is the one without the clock enabled.

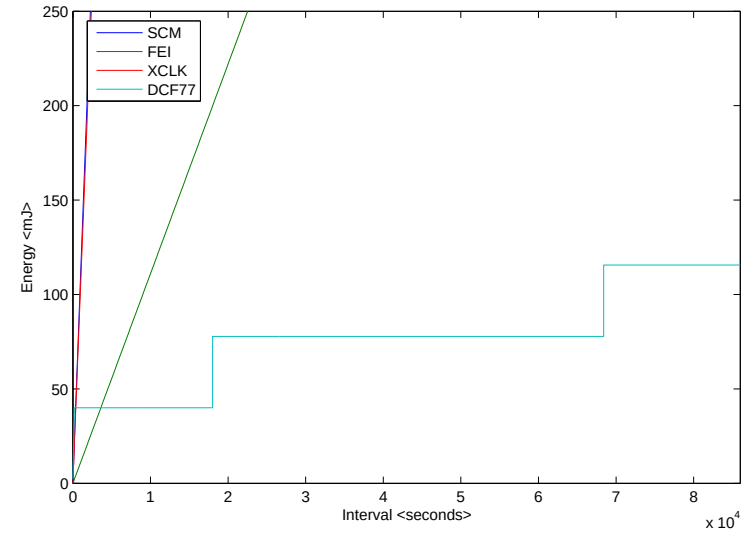


(a)

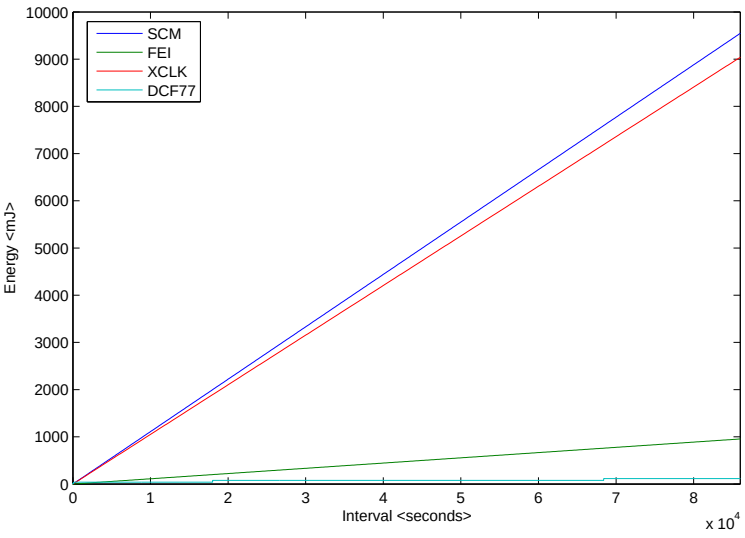


(b)

Figure 26: Part 1 - Energy consumption comparison as a function of time between packet transmissions (continues in Figure 27).



(a)



(b)

Figure 27: Part 2 - Energy consumption comparison as a function of time between packet transmissions (continued from Figure 26).

Figure 26a shows the energy profiles in the interval 0 to 500 seconds. The profile for the DCF77 synchronization does not start in origo due to a fixed window of 20 ms being used. Because of this overhead, the DCF77 synchronization is not very efficient when the data rate is high i.e. the interval between transmissions is low. After 180 seconds, the duty-cycling of the `DCF Receiver Board` begins which causes the energy consumption to remain constant.

With an interval larger than 361 seconds, a break-even occurs between the DCF77 profile and the SCM profile with the high drift from the latter offsetting the overhead from the former. After 381 seconds, the increased power consumption caused by using the external clock overtakes the power needed for the DCF77 receiver.

In Figure 26b, the interval has been increased to 20000 seconds, and another two turning points occur. First, after 3605 seconds, the DCF77 uses less energy than the FEI mode. Second, after 18000 seconds, the DCF77 increases its power consumption with 37.8 mJ which is still less than any of the other profiles.

In Figure 27a, the last increase of the DCF77 occurs at 68400 seconds, and in Figure 27b the four power profiles can be seen in direct comparison.

Mode	Interval <seconds>						
	180	361	381	3605	18000	68400	86400
SCM	20	40	42	400	1998	7592	9590
XCLK	19	38	40	379	1892	7190	9052
FEI	2	4	4	40	200	759	959
DCF77	40	40	40	40	78	116	116

Table 8: Absolute energy values taken at different turning points for comparison. All figures are in mJ.

The absolute values for the different turning points have been put in Table 8. Besides the aforementioned turning points, where the DCF77 outperforms the timer based synchronization, the FEI mode outperforms the SCM mode and XCLK mode with one order of magnitude regardless of the chosen interval. However, this is under the assumption that the internal reference clock has been trimmed. If this is not the case, power consumption can be even worse than the SCM mode. Surprisingly, comparing the SCM and XCLK modes, the energy penalty induced when using the external clock is immediately negated in form of less time needed for listening because of less drift. The power difference is very small though with the external clock marginally better than SCM mode.

Because our scheduled communication protocol is self-configuring, every mote with a clear DCF77 signal can from the moment it powers on obtain the schedule used by the WSN and be synchronized with all the other motes, without exchanging a single message over the radio. This nullifies any power consumption penalties associated with both deployment and joining an operating WSN. Since this is

true regardless of the size of the WSN, any scalability problems are pushed back at manufacturing and programming of the WSN because of the need for unique addresses on each mote.

3.4 Future Work

First, although the `DCF Receiver Board` from Conrad was very suitable for this thesis, given its form factor, price tag, and it could be used out-of-the-box, several shortcomings were found. As our experiments showed, the precision of the `Temic U4224B` is only (7 ± 3) ms which is one to two orders of magnitude worse than the rest of the system. Another decoder chip with a better precision would immediately increase overall precision. Also, since interference is an issue, the antenna supplied with the `DCF Receiver Board` might not be the optimal solution. It should be explored if the amount of interference could be reduced by equipment of a higher quality.

Second, in its current form the scheduled communication protocol is vulnerable to DCF77 reception loss. With the only fall back plan to try again at the next scheduled communication window. Further study is needed to investigate if the interference has a certain periodicity and if it can be avoided by either using better equipment or constructing a schedule that avoids the periods with high interference.

Third, although the developed TinyOS modules are capable of receiving and decoding the DCF77 signal, only the time and date information are actually used, and even then only as-is. This is not a problem as long as the receiver is placed in a country that shares the same timezone and daylight saving time conventions as Germany which it is in the current case. However, a more general module must be able to take the actual timezone into account. This could be done with a simple command to add a signed offset to all time calculations.

On the same note, in its current form the Unix timestamp is calculated with respect to the local timezone, and not as it should be with respect to the UTC.[24] Since UTC is the same as CET+1 and CEST+2, the offset can be implemented by using the summer time information encoded in the DCF77 signal.

Fourth, the decoding algorithm does not account for leap seconds, meaning when a leap second is inserted, the DCF77 signal becomes 60-bit long instead of 59-bit. In its current form this will result in the module losing synchronization. As this information is encoded in the DCF signal an hour before the event, it is possible to compensate for this.

Fifth, although still experimental, the first 0-14 bits are planned to be used as emergency signals, and, depending on the application, this information might be useful for a mote as well e.g. if the mote is part of an alarm system.

[24] UTC - Universal Coordinated Time

Finally, the out of synchronization counting only converts the seconds since the last synchronization into a correct timestamp. Due to the complexity of implementing a Gregorian calendar into a mote, the conversion of hours into days is not performed.

3.5 Summary

In this section we developed the necessary components to decode the DCF77 signal, and we used the achieved global synchronization to develop a hibernation module similar to the standard alarm clocks but with a very high precision, making it suitable for long-term sleeping periods.

A subsequent evaluation showed that the synchronization was valid within (3.3 ± 1.3) ms. The limiting factor was found to be the `Temic U4224B` DCF decoding chip which precision is (7 ± 3) ms. Or in other words, the DCF driven clock has a jitter of (3.3 ± 1.3) ms and a drift of (7 ± 3) ms. Given a better decoding chip, we also showed that μs precision is supported by the Freescale EVB13192 platform, with 100 μs in FEI mode and 1 μs with the external clock. However, the latter comes with increased power consumption.

We argued that if the precision mentioned above is sufficient, the DCF77 synchronization is a feasible way to perform global timestamping of measurements in a data acquisition application. By calculating the energy budgets for different lifetime expectancies, we found that under conditions similar to the recent Hogthrob experiment, the energy overhead from the `DCF Receiver Board` is affordable.

We investigated the amount of reception downtime caused by interference, and found this to be approximately 2-4 % during a day, with 75 % of these periods being less than 3 minutes in duration. In an application where the receiver is often turned on, this interference is of little concern, since the out-of-synchronization counting is adequate enough to keep track of time. However, applications that rarely activate the receiver must contingency plan for the event where a clear DCF77 signal is not present.

A simple proof-of-concept application was developed to utilize the hibernation module for scheduled communication. We showed that the energy needed to power the `DCF Receiver Board` sufficient enough to allow for one transmission window a day, would only suffice for 1 second of listening with the Freescale radio. This corresponds to a duty-cycle of 0.001 % which is an order of magnitude more than the maximum achievable in FEI mode.

Given some basic assumptions, we estimated the energy consumption, as a function of time between each transmission, for both the three regular clock modes and the DCF77 driven clock. We found that if the interval between transmission was more than:

- 361 seconds, the DCF77 method would be more efficient than using the SCM mode.
- 381 seconds, the DCF77 method would be more efficient than using the external clock.
- 3605 seconds, the DCF77 method would be more efficient than using the FEI mode.

with the DCF77 method's weakness being the imprecision of the `Temic U4224B` chip, as this dictates an unnecessarily large transmission window. It should be noted though that the estimates for the DCF77 method are based on an actual implementation, while the three other estimates are based on ideal best case scenarios between two motes, and with the transmission of one message only. Since this rarely holds for any synchronization algorithm, the break-even points between the DCF77 method and the others are upper bounds, and in practice the DCF77 method will out-perform the others much sooner.

Also, we argued that by the merits of self-configuration and a DCF77 receiver on every mote, our scheduled communication protocol, makes deployment and joining an operating WSN completely transparent, and has no problems with scalability.

Although this specific experiment used the DCF77 signal, and thus is region specific to Europe, other timecoded radio signals do exist, with the JG2AS in Japan and the WWVB in USA just to mention a few. If a radio controlled clock is not available e.g. on the African savannah, the principles shown can still be adopted, by setting up a small long-wave radio transmitter to broadcast a timecode. This can be used in areas with bad reception as well. So, although the signal is region specific, the principles and power consumption considerations are general enough to apply outside this region.

4 Out-of-band signalling

In this section, we explore out-of-band signalling as a method to achieve efficient radio duty-cycling but also as a means to reduce latency. The former being of more general interest since power efficient communication applies to virtually any WSN application, and the latter being more application specific since latency is mostly a concern in live applications e.g. intrusion monitoring, fire control, and where human interaction is needed. Our specific application is the sow monitoring project, and the sow localization problem.

Out-of-band signalling requires (at least) two radios: The primary radio and the secondary radio. In this thesis the power expensive Freescale MC13192 is our primary radio, and the low-power transmitter/receiver pair RF-Solutions AM-RT5-433/AM-HRR8-433 is our secondary radio.

By the virtue of the secondary radio always being able to receive, it becomes possible to give any mote within communication range any commands, with the only delay coming from the transmission speed. Because these speeds are of the order of 10^2 bps, the delay becomes very small. Thus, by transmitting predefined commands, it becomes possible to remote control any given mote within range.

Combined with the small delay, this can be used to turn on the primary radio (or any other component) with very little latency. In the sow localization problem, the command could be to tell a particular sow to turn on an actuator e.g. a flashing diode or a sound emitter. In radio duty-cycling, the command could be to turn the primary radio on or off. On a bigger scale, this can also be used both at deployment time to command all the motes to begin simultaneously, and when joining an operating WSN as no idle listening with the primary radio is needed.

In Section 4.1, we report on related work done on out-of-band signalling. In Section 4.2.1, we present our radio stack and in Section 4.2.2 we develop the necessary TinyOS components. We then use these components in Section 4.3 to evaluate the range of the secondary and to see how we can leverage this for power efficient communication. Finally, we discuss future work in Section 4.4 and summarize this work in Section 4.5.

4.1 Background: Related Work

The idea behind out-of-band signalling is as well-documented as wireless communication. In the WSN regime, most of this work has been concentrated on using out-of-band signalling to reduce power consumption by minimizing the idle listening periods of the primary radio.

The STEM [25] technique uses a dual radio architecture where one radio is used to communicate control signals and the other for transmitting data. By using different

duty-cycles, they can emulate a low-power radio on one channel which they in turn can use to control the other. Their primary goal is to achieve efficient radio communication by managing the topology to avoid race conditions to the medium.

The PicoRadio [24] uses a low-power wake-up channel with the overlaying MAC protocol being able to wake-up a mote when data need to be sent. However, the architecture is based on a channel division scheme where motes must know the channel assignments of the nearby motes and use this channel in the wake-up signal. Our approach is more general with the low-power radio functioning as a secondary communication channel giving more flexibility.

In [27] and [28] they propose several wake-up scheme utilizing RFID technology to achieve out-of-band signalling with virtually no power consumption. However, the downside of using passive radio technology is the limited range and high power consumption at the transmitting end.

A variant of this is used in [31], but instead of using RFID readers and RFID-tags, the authors use energy scavenging of the primary radio wave to trigger an event. By using the primary radio to wake-up the receiver, they save the cost of a special transmitter, however, the reception has to be done with special hardware. The drawback of this energy scavenging is the limited range of 3-5 meters and without discriminating between the signals, all motes within range of the transmitter will wake-up.

Closely related to out-of-band signalling, is in-band signalling with low-power listening. This is done in the B-MAC protocol [30] by listening for very short periods of time. The purpose of this is not to be able to receive an entire packet, but rather to assess if the radio channel is active or not. This is similar to what is done in [31], but without the need for special hardware. The radio being used, however, must have the appropriate functionality exposed to the protocol. Also, all motes within listening range must wake-up as well in B-MAC. In our approach, we can with a smaller power consumption only wake-up a particular mote which is clearly more efficient.

4.2 Design and Implementation

4.2.1 Radio Stack

Traditionally, the approach taken when developing radio protocols is the layered model, also used in wired networking, where the Link layer is responsible for handling the communication between two physical points, and the Physical layer is responsible for the actual bit manipulation of the signal.

We wish to adopt the main concepts from this tried and true model, but due to this being a light weight radio stack, which only purpose is to operate as a secondary

communications channel, a stringent layered implementation will probably be neither practical nor optimal. However, the layered model is an excellent work frame to analyze.

Physical layer

The ultra low-power consumption of the RF-Solutions transmitter and receiver pair comes with the price of low transmission speed and virtually no functionality. Basically, the two units function as a wireless extension to an ordinary single wire signal path, meaning sending a high/low signal into the transmitter will result in the receiver driving a high/low signal out as well.

In other words, it is only possible to transmit a single bit at a time, and, without some sort of data indication signal to signify the change of bits, it is not immediately possible to transmit consecutive identical bits unless sender and receiver have agreed upon a fixed transmission rate. Even then, the clock drift at both ends might alter the values of the signal.

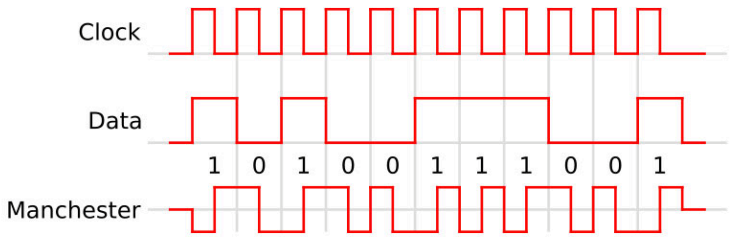


Figure 28: The Manchester Encoding scheme. Picture taken from [33].

To overcome this, we adapt the Manchester Encoding scheme known from its use in Ethernet where bits in a datastream are not encoded by their value but rather by a transition. Meaning a `0` is encoded as the transition from high to low, and a `1` is encoded as low to high. An example of this can be seen in Figure 28.

Preamble	Frame	Trailer
000000	...	11111111

Table 9: Data structure at the physical layer.

By encoding the data onto a clock the sender and receiver can use the transmission itself to agree upon a transmission rate. As in Ethernet we choose to have a preamble to indicate the beginning of a frame and to give the receiver something to lock on to. In order to increase reliability, we also add trailing bits to identify the end. This is illustrated in Table 9.

The downside of Manchester Encoding is the waste of bandwidth as it takes two bits to define one transition. Thus, only half of the bandwidth is used.

Link layer

Bearing in mind that our main application is out-of-band signalling, it would be most appropriate to include some sort of addressing and command in the same frame. Since the standard way of addressing motes in TinyOS is with a 16-bit wide address field, we adopt this for our addressing as well.

Furthermore, to increase correctness, we calculate the parity of the data field and include this value in the header as well. Because our main payload is a single command and not a long message, we choose the less effective but lighter parity check over the more flexible but space consuming CRC check.

As data field, we choose a 16-bit wide field since combined with the address this makes a simple and small packet of 32-bit width which is supported as a basic data type on most platforms. In this way, the address can be shared and is thus both part of the header and the data field. The complete structure can be seen in Table 10.

Preamble	Frame				Trailer
	Header		Header/Data	Data	
	Reserved	Parity	Address		
000000	1	<1-bit>	<16-bit>	<16-bit>	11111111

Table 10: Complete structure of the transmitted data.

Combined with the preamble and trailer, we get a 48-bit wide datastream for each 16-bit command. But since we also Manchester Encode the datastream, the actual effectiveness of this radio stack is 16-bit data per 96-bit transmission, or 17%. This is the price we pay for increased robustness (large preamble and trailer), correctness (parity), and flexibility (full TinyOS address field).

4.2.2 TinyOS

The functionality we wish to implement is out-of-band signalling, both with respect to radio duty-cycling and as a remote controller. The implementation should therefore be flexible enough to accommodate both the asymmetrical case with a pure transmitter and a pure receiver, but also the symmetrical one where a transmitter can act as a receiver as well and vice versa. This was done by developing three distinct modules:

- AMTransmitterM module which only function is to transmit a given value.
- AMReceiverM module which only function is to receive a given value.
- AMTransceiverM module which acts as a wrapper around the AMTransmitterM and AMReceiverM.

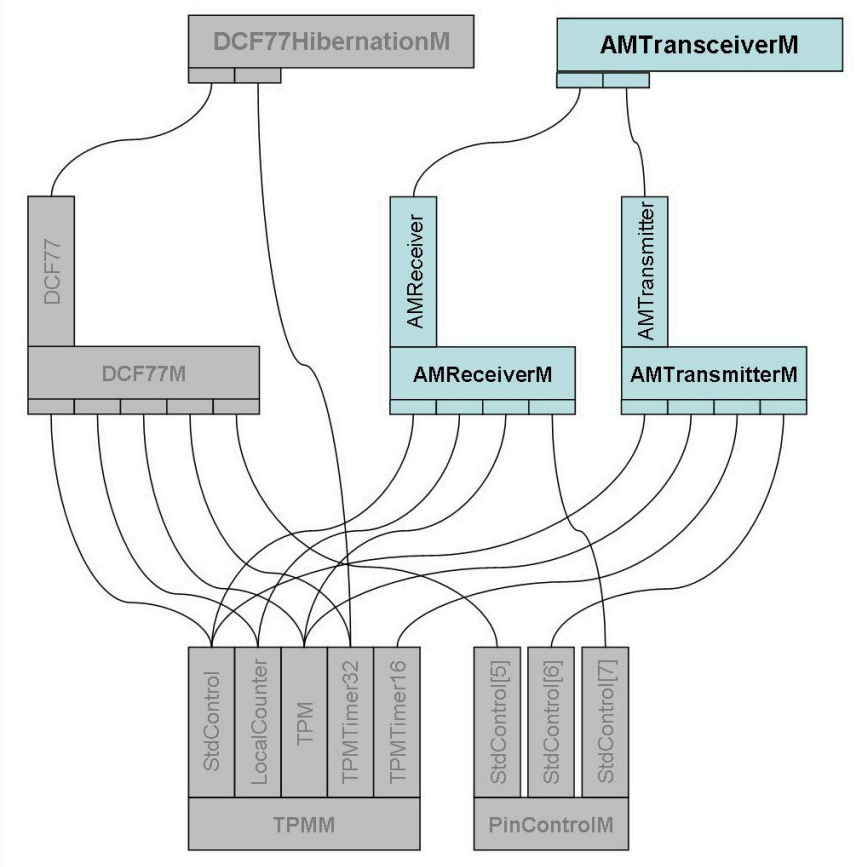


Figure 29: Component overview of the AMTransmitterM, AMReceiverM and AMTransceiverM modules.

Figure 29 shows the three modules in context, which have all been made part of DIKU's contribution to the TinyOS project at SourceForge,[25]. They can be used in any TinyOS application by adding the line `SENSORBOARD=lpreceiver` to the application's `Makefile`.

AMTransmitterM

As the functionality we seek is very similar to that found on the UART, except that we wish to transmit a 32-bit value over the radio instead of an 8-bit value through the serial port, we adopt the main principles from the UART module.

Specifically, the module must be initialized and started before use. An optional baud rate can be specified otherwise the default value will be used. Similar to the UART, a 32-bit value is transmitted asynchronously one at a time with an event signaled when all 32-bits have been sent. Before any transmission begins, though, the parity is calculated and added to the header.

The Manchester Encoding is done by setting the 16-bit alarm clock to the double of the baud rate. Each time, the alarm fires one of two things can happen:

1. A bit is prepared to be transmitted. Due to the encoding scheme, the pin controlling the input signal to the transmitter is set to the inverse value of the bit.
2. A bit is being transmitted. Because the alarm clock is set to the double of the baud rate, we are in fact at a transition which by definition is the inverse of the current signal being sent to the transmitter. Thus, the signal is inverted and a new bit set to be prepared for transmission.

After the last trailer bit is transmitted, the alarm clock is stopped and an event is signaled.

AMReceiverM

Usually, when a Manchester Encoded signal is to be decoded, the signal is continuously sampled at the double clock rate of the expected transmission rate[26], and, since the preamble is known, it can be used to adjust the clock rate until sender and receiver are synchronized. In order for this to be effective, special hardware is needed.

Due to this hardware not being available, continuously sampling a signal which might not even have any data encoded, in software is both a waste of processing

[25] http://tinyos.sourceforge.net
[26] Due to the Nyquist limit.

and electrical power. Instead, we choose the same approach as in decoding the DCF77 signal, namely by using the edge driven timestamped TPM pin. The known preamble can then directly be used to calculate the transmission rate by measuring the time between edges which is exactly what the timestamp reads.

Similar to the DCF77 reception, a state machine is used in order to be able to distinguish between clock estimation through the preamble and decoding of signal. This can also be used to abort reception if the pulses have the wrong width. Also, the state machine is reset if the interval between two pulses is beyond a certain threshold as this indicates that the last reception was a failure, which implies that the state machine has not been reset by itself. When the first edge after a reset is detected, the current timer counter is stored as a reception timestamp.

With the transmission rate known, it is possible to distinguish between single and double pulses with double pulses being a single `0` transmitted after a single `1` and vice versa. In order to increase stability, we define the width of a single and double pulse as an interval and not just a fixed value. This is similar to the deviation value chosen in the DCF77 reception.

By inserting the value of these pulses into a First-In-Last-Out buffer, a transmission can be decoded by matching the first 12-bit of the buffer with a Manchester encoded preamble and the last 16-bit with a Manchester Encoded trailer.

When both of these patterns match, a complete signal has been received, and the header and data field can be obtained by reading every other bit in the buffer. The parity bit can then be used to determine if one bit has been corrupted during transmission. If this check is successful, the received data and the reception timestamp are signaled to the component above by an event. Otherwise two zero values are signaled instead to indicate a corrupt packet has been received.

Strictly speaking, the TOS_LOCAL_ADDRESS should match the address field in the packet i.e. match the first 16-bit of the data field before signalling the component above. However, by doing so, this module would be completely restricted to follow the conventional TinyOS addressing which might (or might not) be the one being used by the application. By delaying the filtering of packets to the component above, the module remains versatile.

AMTransceiverM

Because both the `AMTransmitterM` and the `AMReceiverM` module have simple generic interfaces, it is possible to build functionality on top of both, similar to the Console module being built on top of the UART module. At the moment, we settle for at general purpose transceiver module capable of both transmitting and receiving. The transceiver module is built as a simple wrapper on top of the two former modules allowing commands and signals to pass through directly.

4.3 Evaluation

With a working secondary communication channel, the usefulness of out-of-band signalling can be evaluated. First, the quality and range of the secondary radio are compared with the primary radio. Second, the use of out-of-band signalling for duty-cycling is evaluated, and third, out-of-band signaled duty-cycling is combined with scheduled communication to achieve ultra low-power performance.

4.3.1 Range

By using out-of-band signalling to turn on another mote's radio, a new kind of "hidden terminal"-problem arises, since if the range of the two radios is different, it is possible to wake-up a mote without being able to communicate with it, and conversely, there might be motes one would like to communicate with but with no means to activate.

Hence, we wish to explore the actual range of both the primary and secondary radio. To do this, a test application `TestAMRange`[27] was developed. Because radio transmissions are prone to interference, packet loss and corruption are not uncommon. This is closely related to the maximum range as there is an interval where the packet loss and corruption begin to increase until no packets can be received successfully.

Our approach is thus to transmit batches of 1000 packets and count the number of lost and corrupted packets. By doing this at several distances and, at least for the secondary radio, at different transmission rates, we can estimate the effective range of both radios.

	AM-RT5-433/AM-HRR8-433						MC13192			
	400 bps		600 bps		800 bps		Board		No Board	
Distance	Loss	Error	Loss	Error	Loss	Error	Loss	Error	Loss	Error
5	0.0	0.0	0.6	0.0	0.7	0.0	0.3	0.2	0.4	0.0
10	0.0	0.0	0.7	0.0	0.7	0.0	96.5	0.2	0.0	0.0
15	0.0	0.0	0.6	0.0	0.6	0.0	100.0	-	0.0	0.0
25	1.3	0.2	0.6	0.0	0.7	0.0	100.0	-	3.1	0.2
45	25.6	1.4	65.0	13.8	16.0	0.3	100.0	-	1.6	0.0
Time	120 s		80 s		60 s		< 1 s		< 1 s	

Table 11: Packet loss (%) and packet error (%) at different distances and transmission rates.

Since the 433 MHz radio band is an open commercial band, many wireless devices use this band. To minimize interference and to allow for greater ranges, the experiments were conducted outside in the open in a residential neighbourhood. Also,

[27] Available under the DIKU contribution directory to TinyOS at SourceForge.

all measurements were done with completely clear line-of-sight. The results can be seen in Table 11. The distance is in meters and all measurements are in %. Of course, a packet must be received first in order for it to be corrupt, hence the packet error is measured as a percentage of the received packets and not based on all of them.

Also, it should be noted that although the AM-HRR8-433 receiver is rated at 1000 bps[28] it was not possible to achieve this because any operation between 900-1000 bps was unstable. This is probably caused by the signal between the MCU and AM-RT5-433 transmitter not being a perfect square wave as mentioned in Section 2.3. Since the Manchester decoding requires a signal with consistent pulse widths, a shark fin shaped signal will not be decodable. This was mainly the reason behind choosing to define the pulse widths as intervals instead of fixed values, and because the shark fin effect is present even at 600 bps, it is possible to compensate for this to some extent. Loosening the constraint on the pulse width intervals would compensate even further for this, however, the decreased packet loss would probably cause an increased packet corruption due to misreading pulses.

Whether the right balance between pulse width compensation and signal integrity is found remains to be explored. As a result of this, we take the conservative choice of choosing 800 bps as the highest transmission rate in order to remove this uncertainty.

Two series of experiments were conducted with the primary radio, one with the `DIKU Receiver Board` mounted and one without. This was necessary because the ground plane on the board, which is part of the boards antenna, has an enormous shielding effect on the primary radio. This is clearly seen by the experiments since at 10 meters, there is already almost a complete packet loss. Without the board mounted, however, the range increases significantly with very little packet loss and corruption even at 45 meters. It was unfortunately not possible to test distances further away given the lack of free space. At 25 meters, there is a slight increase in both packet loss and corruption. This is probably interference caused by reflection because the transmitter was close to walls and trees at this distance.

The experiments with the secondary radio showed three interesting results:

First, at short distances (around 15 meters and less), the transmission rate has a small, but measurable, impact on packet loss and corruption with the 400 bps experiments having no loss at all and the 600 and 800 bps both having a loss of 0.6-0.7 %. None of the experiments showed any corruption at this point. At distances beyond 15 meters the packet loss and corruption increase slightly for the lowest transmission rate, and beyond 25 meters loss and corruption become significant.

[28]The actual rating is 2000 Hz, but since we Manchester Encode the signal first, the effective transmission rate is 1000 bps.

Interestingly enough, the highest transmission rate is also the one with the lowest packet loss and corruption. As a matter of fact, loss and corruption are almost constant up to 25 meters, and at 45 meters the increase is significantly less than for the two other transmission rates.

The cause of this is probably the reduced overall transmission time at the higher speeds as the chance of encountering interference from other sources is increased by the amount of time used to transmit. Given the relative low errors at low distances, there is reason to use the highest transmission rate since the benefits from the slower rates at lower distances are too small compared to the significant loss at greater distances.

Second, at 45 meters there does not seem to be a consistent pattern, as the packet loss observed when transmitting at 600 bps is significantly larger than the others. The interference caused by increased transmission times should make the packet loss increase linearly, which apparently is not the case. There must then be something more than the overall transmission time that has an effect on the encountered interference. Because the test application does not transmit continuously but rather posts a task after the completion of each transmission, the packets are transmitted with at certain periodicity which is dependent on the transmission rate. If the interference is caused by transmissions on the same radio band, it is plausible that this source also transmits at a certain periodicity. Hence, certain transmission rates will be more or less effected by interference because of the periodicity of the packets.

Third, with the current implementation of the `DIKU Receiver Board`, the ground plane causes the effective range of the primary radio to be no more than 5 meters. Since the effective range of the secondary radio is at least 15 meters, regardless of transmission rate, it is possible to signal motes with the secondary radio without the motes being able to communicate on the primary radio. Assuming the boards could be connected without shielding for the primary radio, the opposite would occur with a mote being within the range of another mote's primary radio, but with no means of signalling this to each other. Depending on the application and chosen communication protocol, both types of the "hidden terminal"-problem might cause problems. For instance, if the initial discovery is being performed by the primary radio e.g. by building a spanning tree, the shorter range of the secondary radio would cause the tree to collapse.

To avoid this and similar problems, any discovery must be done with the radio that has the shortest range. The real problem is then to know which radio actually has the shortest range at any given time. Since the two radios operate at different frequencies, the interference caused by the actual terrain on-site and perhaps other sources in the vicinity makes it impossible to know beforehand which radio has the farthest range. The only solution is then to perform discovery with both radios and only accept a link if both radios agree.

4.3.2 Remote Control

The main feature out-of-band signalling is offering is a way to remotely control another mote. Whether it is the primary radio that needs to be controlled (as in duty-cycled communication), or an actuator e.g. sound or light emitter (as in low latency sow localization), the framework remains the same:

- Mote A transmits a command to Mote B via the secondary radio.
- Mote B performs the command.

The qualities we wish to evaluate are latency and power consumption. To this end we developed the `TestAMRoundRobin`[29] application which is a data collection infrastructure consisting of one gateway and several regular motes. The regular motes have their primary radio turned off and the secondary radio turned on as default. The gateway has both turned off by default but turn them on periodically to poll the regular motes for data in a round-robin fashion based on the `TOS_LOCAL_ADDRESS` variable. The gateway is also responsible for turning off the regular motes, but if the command is lost, the regular motes will turn off their radio based on a timer as well.

This is in essence `Power efficient discovery` except that instead of a limited listening period our secondary radio has an unlimited listening period. Thus, the gateway is ”discovered” immediately after the transmission of the first ”beacon”.

Six motes were used, one gateway and five regular motes. The interval between polls was set at 5 seconds and each communication window was set at a conservative high value of 500 ms. The timer was set to count at 62.5 kHz and the data packets were 6 bytes long. At typical output from the gateway is shown below:

```
# Gateway wake-up
16584081 16588638 25
16615513 16620070 26
16646945 16651494 28
16678376 16682933 31
16709808 16714365 32
# Gateway sleep
# Gateway wake-up
17053811 17058368 25
17085243 17089800 26
17116675 17121232 28
```

[29] Available under the DIKU contribution directory to TinyOS at SourceForge.

```
17148106 17152663 31
17179538 17184095 32
# Gateway sleep
# Gateway wake-up
17523541 17528098 25
17554973 17559530 26
17586405 17590962 28
17617836 17622393 31
17649268 17653825 32
# Gateway sleep
# Gateway wake-up
17993271 17997828 25
18024703 18029260 26
18056135 18060692 28
18087566 18092123 31
18118998 18123555 32
# Gateway sleep
```

The first column is the counter read at transmission time of the command, the second column is the counter read at reception time of the data from the primary radio and the third column is the address of the source mote. By calculating the difference between the two counts and dividing with the timer frequency, one gets a Round-Trip-Time of 73 ms. Thus, the latency between transmitting a command and receiving a reply over the primary radio is of this order of magnitude when the secondary radio is operating at 800 bps.

Transmission rate <bps>	RTT <ms>
100	496
200	254
300	174
400	133
500	109
600	93
700	81
800	73
900	65

Table 12: Measured Round-Trip-Times at different transmission rates.

In order to isolate which part of the RTT that is dependent on the transmission rate of the secondary receiver and which part is overhead, the experiment was repeated with different transmission rates. The results are shown in Table 12. As the transmission time is inversely proportional to the transmission rate, we expect total latency to be expressed by:

$$\Delta t = \frac{N}{v} + k$$

where N is the number of bits being transfered, v is the transmission rate, and k is the latency not dependent of the transmission rate i.e. overhead.

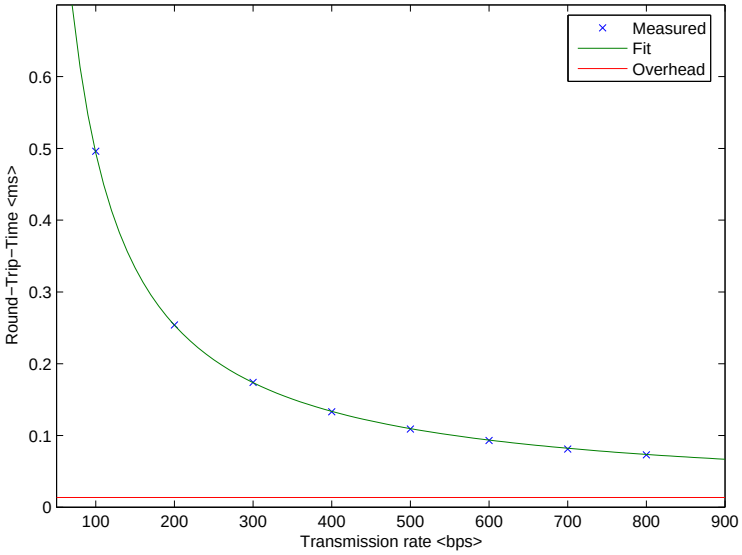


Figure 30: Calculated Round-Trip-Time as a function of transmission rate.

Fitting the values from Table 12 to the equation above yields the transmission rate independent overhead k = 14 ms. This is shown in Figure 30. While k is not dependent on the transmission rate, it is, however, composed of many different factors: Reading counters, packet preparation, packet transmission, packet reception etc. But, by looking at the figure, it becomes clear that even at the highest transmission rate the overhead from k is still four times smaller than the transmission time.

This is important both because of the actual cost of transmitting but also when timing the duty-cycling of components. For instance, turning on the primary radio on the gateway before transmitting the wake-up command would be prohibitively expensive, power consumption wise. Similarly, using the secondary receiver to transmit a goto-sleep command would also be prohibitively expensive. Instead, this should be communicated directly over the primary radio or solely be the responsibility of each individual mote. In this case, the size of the communication window has no impact on power consumption due to the secondary radio always being on, and the primary radio is only turned on when used.

As measured in Section 2.3, power consumption of the AM-HRR8-433 radio is 1.35 mW. There exists also a receiver from the same product line called AM-

HRR18-433 which only has a power consumption of 0.21 mW, but this was not available at the time of this thesis. In the current application, the secondary radio is always on, and thus regardless of the time interval between use, power consumption is linear.

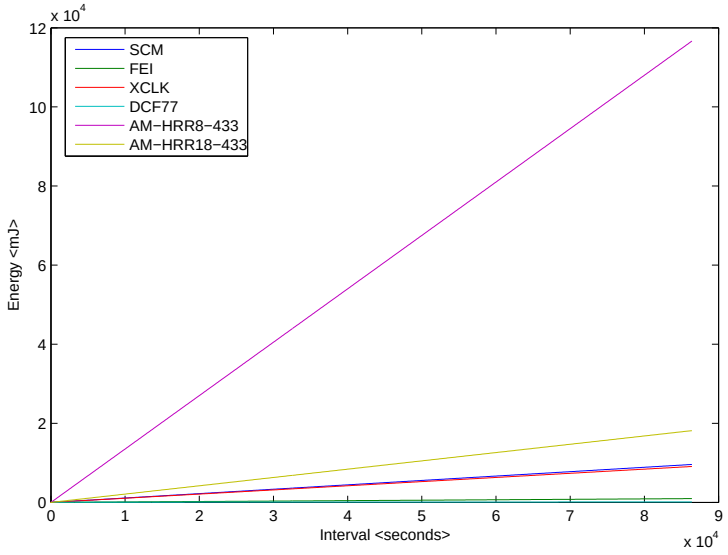


Figure 31: Energy consumption comparison as a function of time between packets.

Revisiting the power consumption estimates of a `Scheduled communication` approach, shown in Figure 26 and 27, our `Power efficient discovery` is not so efficient after all in comparison, as can be seen in Figure 31.

Having the AM-HRR8-433 receiver always turned on is 20 times less effective than using the internal clock in SCM mode to perform scheduled communication. And due to the SCM mode being the most inefficient mode, this is not particularly good performance. If the AM-HRR18-433 receiver had been available, the efficiency would be slightly better with approximately two times more power consumption than in SCM mode.

It should be noted though that the comparison is being made against an idealized single-hop protocol which is working under the assumption that the listening period increases proportionally with the drift. Whether the difference in power consumption is large enough to allow for implementation overhead, and still have the scheduled communication to be more efficient, needs further investigation.

In some applications, this increase in power consumption might be acceptable due to the need of low latency communication e.g. localization in the Hogthrob project. The real question is then how big a lifetime reduction the extra power consumption will result in.

Completely analog to the energy budget analysis performed in Section 3.3.2, we assume that the platform in question is equipped with a power supply with a capacity of 18.9 kJ, which is quite similar to the one being used in the recent Hogthrob experiment [3].

By assuming different lifetimes based on this capacity, we can calculate the reduced lifetime caused by the power consumption of the secondary radio. By using the equations from Section 3.3.2, these reduced life expectancies for different values have been calculated and put in Table 13.

		AM-HRR8-433			AM-HRR18-433		
Δt <days>	P_{plat} <mW>	P_{tot} <mW>	Δt_{new} <days>	Loss <%>	P_{tot} <mW>	Δt_{new} <days>	Loss <%>
7	31.25	32.60	6.7	4.14	31.46	6.9	0.67
14	15.63	16.98	12.9	7.95	15.84	13.8	1.33
30	7.29	8.64	25.4	15.62	7.50	29.2	2.80
60	3.65	5.00	43.8	27.02	3.86	56.7	5.45
90	2.43	3.78	57.9	35.71	2.64	82.8	7.95
180	1.22	2.57	85.3	52.63	1.43	153.5	14.73
360	0.61	1.96	111.7	68.96	0.82	267.5	25.68

Table 13: Estimated lifetime reductions when having the secondary radio always turned on.

As mentioned before, the Hogthrob experiment was set to last for 20 days, but the motes themselves lasted for 30-40 days before depleting their power supply. Given this time frame and power supply, both the AM-HRR8-433 and the AM-HRR18-433 receiver could have been used without comprising the length of the experiment as the lifetime of the motes would only have been reduced by 5 days and 1 day respectively. However, for experiments running for several months, power consumption of the AM-HRR8-433 receiver begins to dominate total energy budget. The AM-HRR18-433 receiver, however, contributes with only a small fraction to the energy budget even when running on the scales of many months.

On the other hand, depending on the deployment method and communication protocol being used, increased power consumption of the secondary radio might be offset by the decreased power consumption of the primary radio at deployment time and when motes need to join operating WSNs.

Usually, the only way to start a new WSN, or add motes to an already operational WSN, is to continuously listen with the primary radio until the communication periods of the WSN are discovered with the risk of new motes depleting their battery. This is a problem inherent to duty-cycling and exists both in scheduled communication and power efficient discovery. Although we cannot remove this, we can greatly reduce the power consumed by using the secondary radio to listen

for communication periods. Because the secondary radio consumes several orders of magnitude less power than the primary radio, the potential savings are equally large.

4.3.3 Scheduled Discovery Protocol

Based on the last section's power consumption prognosis, it becomes clear that although the low-power receivers indeed are low-power, they are not power-free. It is thus important to use them with the same amount of vigilance as with regular radios.

It was concluded by [2] that in a network with periodic data transmissions, scheduled communication would be far superior than power efficient discovery. As discussed in Section 3.3.3, the primary problem with scheduled communication, energy wise, is the long listening periods caused by drift. By combining the low-latency signalling capabilities from the secondary radio with the overall low-power efficiency of scheduled communication, we can achieve the best of both worlds.

The test application `TestAMDuty`[30] was developed to this end. It is similar to the previous round-robin polling application except that the regular motes use the polling signal as a synchronization beacon, and after two consecutive beacons the regular motes can duty-cycle their secondary radio as they would their primary. In other words, instead of turning on the primary radio periodically as in scheduled communication, and instead of having the secondary radio always turned on to control the primary radio, we turn the secondary radio on periodically, and when it is on, we use it to control the primary radio.

Recall from Section 3.3.3 that power consumption for the different clock modes was dependent on the interval between packets, and on the listening period which was caused by drift:

$$\begin{array}{lrcl} \text{SCM:} & t \in [0.001; 86400] & \mapsto & 111\ mW \cdot 0.001\ ppm \cdot t \\ \text{FEI:} & t \in [0.001; 86400] & \mapsto & 111\ mW \cdot 0.0001\ ppm \cdot t \\ \text{XCLK:} & t \in [0.001; 86400] & \mapsto & (111\ mW \cdot 0.000001\ ppm + 0.105\ mW) \cdot t \end{array}$$

In order to use the secondary radio instead of the primary, the listening period must be longer to accommodate for increased transmission time. We showed above that the transmission time at 800 bps was 73 ms. Thus, energy consumed by the secondary radio during this time is:

$$E_{secondary} = P_{secondary} \cdot \Delta t_{73\ ms} = 1.35\ mW \cdot 0.073\ s = 0.099\ mJ$$

Assuming that only one command is necessary i.e. there is no packet loss, power consumption in the different clock modes would reduce to:

[30] Available under the DIKU contribution directory to TinyOS at SourceForge.

$$
\begin{array}{lll}
\text{SCM:} & t \in [0.001; 86400] & \mapsto \quad 1.35\ mW \cdot 0.001\ ppm \cdot t + 0.099\ mJ \\
\text{FEI:} & t \in [0.001; 86400] & \mapsto \quad 1.35\ mW \cdot 0.0001\ ppm \cdot t + 0.099\ mJ \\
\text{XCLK:} & t \in [0.001; 86400] & \mapsto \quad (1.35\ mW \cdot 0.000001\ ppm + 0.105\ mW) \cdot t \\
 & & \quad +0.099\ mJ
\end{array}
$$

This is almost a reduction of two orders of magnitude. Plotting this together with the power consumption in the DCF77 mode, one gets Figure 32.

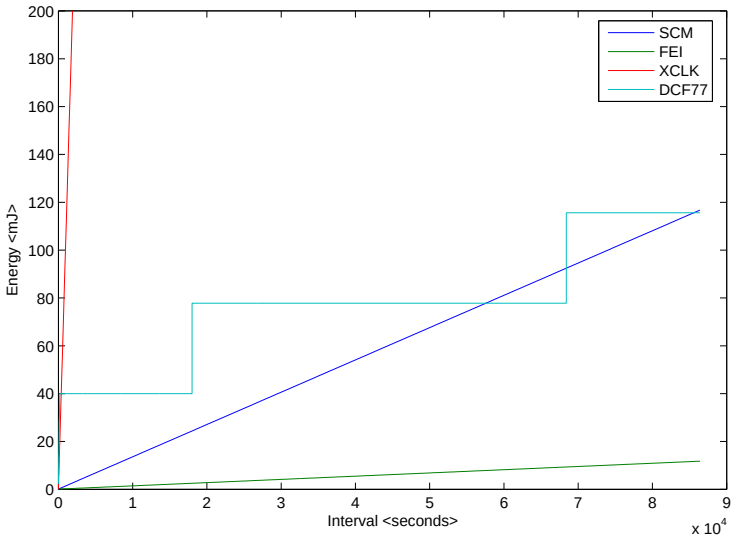


Figure 32: Energy consumption comparison as a function of time between packets.

Compared to Figure 26 and 27, several interesting features can be found:

First, because of the power consumption of the secondary radio being comparable with the power consumption of the external clock, the XCLK mode is no longer more energy efficient than the SCM mode.

Second, there is now a significant period where even the SCM mode is more efficient than the DCF77 mode. With the former being more efficient the first 57644 seconds and between 68400 seconds and 85644 seconds.

Third, the FEI mode is more efficient than any of the other modes at any time interval.

Given the enormous power savings by combining low-power receivers with scheduled communication purely based on timers, it stands to reason that the same fusion can be done with DCF77 scheduled communication. However, actual savings would be minimal since the main power consumption of 37.8 mJ, when in DCF77

mode, is by far from the 3 minutes synchronization time where the DCF77 receiver is on. The savings would then have to come from the actual time where the primary radio is on which is only a small 20 ms window that consumes 2.22 mJ.

In the protocol above, the cost of deployment would be the power consumption of the secondary radio from the moment the mote is turned on, and until the ”begin” command has been received. Similarly, the cost of joining an operating WSN would be the cost of having the secondary receiver turned on for two duty-cycle periods, at most. Since in many protocols this is completely analog to deployment and joining with only the primary radio, reduced power consumption would be equal to the difference between the primary and secondary radio which is two-to-three orders of magnitude. This is a significant reduction.

Also, because our secondary radio is more than just a wake-up signal, as in [27], [28], [31], and [30], we can discriminate between which motes to wake-up, and thus avoiding the power consumption penalty of waking up all motes within communication range. This, combined with each additional mote only needing a constant number of control messages, means that the protocol is quite scalable.

4.4 Future Work

Throughout this section, we have assumed a single-hop architecture and all the power consumption estimates are built around this. Because multi-hop routing is an important feature of WSNs, it would be interesting to explore if low-power receivers could be used with more success in reducing power in this regime. Also, it would be interesting to implement a transceiver solution to investigate how latency can be reduced in multi-hop routing.

Although power consumption from the AM-HRR18-433 receiver still is too high to allow for competitive operation when compared with scheduled communication, it is still an order of magnitude less power consuming than the AM-HRR8-433 receiver. Any future work must be done with this ultra low-power receiver in order to give the most accurate picture of the performance.

With regards to the implemented TinyOS modules, it would be interesting to see if it would be possible to achieve the maximum theoretical transmission rate as stated in the datasheet. This would involve investigating whether the problem truly lies in the MCU not generating a good enough signal, and if this is this case, it should be possible to compensate for this. Either in the encoding by using an asymmetrical clock to drive the signal or in the decoding, by anticipating a skewed clock.

Also, although Manchester Encoding is robust, it is not very efficient. Given the nature of the hardware and the fact that the signal transitions are timestamped very precisely in the TPM module, a more efficient algorithm would use the length of these transitions to encode either an `0` or an `1`, similar to the DCF77 signal. The

transitions themselves would then be used to separate each bit. Assuming a uniform distribution of `0` and `1`, this would give a 50% increase in data transmission.

Related to the encoding, is the actual size of each transmitted packet, as in its current form, there is a 50% overhead when transmitting 32-bits. A more thorough testing might be able to yield a more optimal size of the preamble and trailer.

Finally, if the data transmitted in the network are very light and not very timing critical, it would be beneficial to use the secondary radio to transmit data as well as commands. This could be done quite easily by implementing a wrapper on top of the `AMTransceiverM` module which only purpose is to split and collect smaller packets together into a larger message format.

4.5 Summary

In this section, we investigated the use of out-of-band signalling as a means to reduce power consumption on the primary radio and to reduce the inherent latency in duty-cycled communication.

We first developed the necessary modules in TinyOS to be able to use the low-power transmitter and receiver on the `DIKU Receive Board` as a secondary radio, capable of transmitting 32-bit data at a time. This size was chosen to be small enough to be light and fit into a standard data structure, but also big enough to allow for standard TinyOS addressing and a small payload.

The modules and boards were subsequently stress tested in order to determine their range by investigating the number of lost and corrupted packages at different distances. Together with similar measurements conducted with the primary radio, we were able to quantify the `hidden terminal`-problem. We found that with the receiver board attached, the primary radio was greatly shielded, and the maximum range were no more than 5 meters. Without the board, however, the reception at 45 meters was still acceptable. Beyond 35 meters, the secondary radio began to experience significant packet loss. This was greatly effected by the transmission rate with higher rates being less susceptible to interference.

We then used the secondary radio as an out-of-band signalling device in two different scenarios:

First, we explored the use of the secondary radio as a means to enhance power efficient discovery by using it as a permanent listening radio. We found that energy consumption from being always turned on was prohibitively high when compared to regular scheduled communication.

Specifically, we found that having the AM-HRR8-433 receiver always turned on would be 20 times less effective than using the internal clock in SCM mode,

and even with the ultra low-power version, AM-HRR18-433, power consumption would still be twice as large.

However, the permanent listening also reduces the latency of duty-cycling to the time it takes to transmit a packet with the secondary radio which we found to be around 73 ms. At the same time, we found that the actual transmission time stands for 80% of this overhead which must be taken into account when duty-cycling components.

We argued that the increased power consumption of the secondary radio would be worth its cost if the application required low latency operation. By trading a small amount of a mote's total lifetime, it was possible to reduce latency related problems to the actual transmission time of the secondary radio.

We found that in the recent Hogthrob experiment the mote's lifetime would only be decreased by 5 days and 1 day with the AM-HRR8-433 and AM-HRR18-433 receiver respectively. This would not have comprised the length of the experiment as the motes had a 10 day margin to their lifetime. This low latency could also be used to reduce energy consumption at deployment time and the consumption penalty of joining an operating WSN by the same order of magnitude as the difference between the primary and secondary radio.

Second, we combined the principles from out-of-band signalling with scheduled communication by inserting the secondary radio as a shell around the primary radio. We developed a proof-of-concept application and argued that power reduction was proportional to power consumption difference between the primary and secondary radio. Specifically,

- The higher precision from the external clock could no longer offset the low-power consumption of SCM mode
- SCM mode outperformed DCF77 mode so long as the interval between transmission was less than 57644 seconds or between 68400 seconds and 85644 seconds.
- FEI mode outperformed all the other modes assuming of course that the reference clock had been trimmed prior.

5 Conclusion

The key issue we wanted to investigate in this thesis was how we could utilize low-power receivers to duty-cycle the radio efficiently. We chose two orthogonal approaches to the problem: One where a signal would always be present, the persistent transmitter, and, one where the receiver would always be able to hear a signal, the persistent receiver.

With the persistent transmitter being the DCF77 timecode signal, we showed that we could synchronize a WSN within (3.3 ± 1.3) ms accuracy, and use this global synchronization to perform efficient scheduled communication.

We made our implementation scalable by devicing an algorithm to assign schedules based on the interval between communication periods and each mote's TinyOS address. This self-configuration also removes the high power consumption normally associated with both deployment of a WSN and joining an already operating WSN.

Our evaluation showed that compared to regular timer based scheduled communication there exists a high initial cost of using the DCF77 receiver, but with periods of inactivity larger than 360-3600 seconds, a break-even occurs, and on the scale of day or days, we achieve a duty-cycle of 0.001 % which is at least an order of magnitude better than other protocols.

For the persistent receiver, we developed a low-power, low transmission rate secondary radio and showed that with one-hundredth of the amount of energy normally required to have no latency in the network, we could achieve an approximately 73 ms latency.

In the Hogthrob project, latency caused by duty-cycling is a problem when localizing a particular sow. With a small increase in power consumption, this problem can be solved with the use of our secondary radio. In particular, the recent Hogthrob experiment could have been carried out equally well with our secondary radio fully operational without compromising the lifetime of the motes.

Similarly, with this small increase in power consumption, the high power cost normally associated with both deployment of a WSN and joining an already operating WSN, would be reduced by the same magnitude as the power consumption difference between the primary and secondary radio.

When used in power efficient discovery, we found that low-power consumption would eventually become a significant problem and regular scheduled communication, based on timers, would be more energy efficient.

We then showed that by inserting the secondary radio as a layer around the primary radio, we could reduce the amount of energy spent on idle listening by two orders of

magnitude with the downside being the added latency overhead of approximately 73 ms.

5.1 Contributions

In this thesis, we have contributed the TinyOS community with an efficient and cheap way of performing global time synchronization in a scalable and fault tolerant way. This will benefit both scheduled communication protocols but also data acquisition applications.

Our scheduled communication protocol based on the DCF77 is scalable, uses ultra low-power, has no deployment cost overhead and allows motes to join an operating WSN on-the-fly without any power consumption penalties.

Our out-of-band signalling reduces latency in an easy and efficient way, and is more than just a wake-up channel. It is rather a low-power secondary radio allowing both remote control and low-rate data transmission.

Our scheduled discovery protocol based on the out-of-band signalling is scalable, uses ultra low-power, has 2-3 orders of magnitude less deployment cost overhead and allows joining of an operating WSN with the same reduced amount of power consumption penalty.

Our measurement and analysis of the jitter and drift of the different clock modes on the Freescale EVB13192 can also help others when developing timing critical applications on the platform.

We have also contributed the DIKU testbed with the development and implementation of the `DIKU Receiver Board` on the Freescale EVB13192 allowing both DCF77 time synchronization and out-of-band signalling with a secondary radio to be explored by future students.

5.2 Future Work

We have showed that out-of-band signalling can be used as an effective means to reduce latency from duty-cycled network without inducing too large energy penalty. However, since our signal decoding is done in software, the secondary radio is still limited to control the duty-cycling of components connected to the MCU. What really is needed to achieve ultra low-power and low-latency is a separate decoding solution implemented in hardware, capable of comparing the address from the message with a pre-stored value on the chip. The MCU itself can then be duty-cycled as well on equal footing with the radio, and since dedicated hardware is used for the sole purpose of signal decoding, this can be done very energy efficiently.

References

[1] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler and Kristofer Pister: “System Architecture Directions for Networked Sensors”, ASPLOS 2000, Cambridge, November 2000. http://www.tinyos.net/papers/tos.pdf

[2] Mads Dydensborg: “Connection Oriented Sensor Networks”, Ph.D. Dissertation, Dept. of Computer Science, University of Copenhagen, December 2004. http://www.dydensborg.dk/dissertation.pdf

[3] Klaus Skelbæk Madsen: “Experimental Sensor Network: Lessons from Hogthrob”, Master Thesis, Dept. of Computer Science, University of Copenhagen, Spring 2006. http://hogthrob.42.dk/thesis.pdf

[4] Freescale Semiconductor Inc.: “MC13192 Evaluation Board Reference Manual”, MC13192EVBRM/D, Rev. 0.0, 08/2004. http://www.freescale.com.

[5] Freescale Semiconductor Inc.: “MC9S08GB60/GB32/GT60/GT32GT16 Data Sheet HCS08 Microcontrollers”, http://www.freescale.com.

[6] Freescale Semiconductor Inc.: “MC13192 2.4 GHz Low Power Transceiver for 802.15.4”, MC13192/D, Rev. 2.4, 07/2004. http://www.freescale.com.

[7] Conrad International: “DCF Receiver Board”, http://www.conrad.com.

[8] Temic Semiconductors: “Time-Code Receiver with Digitized Serial Output”, Datasheet, Rev. A4, 26-Apr-99 http://www.temic.com.

[9] RF-Solutions Ltd.: http://www.rfsolutions.co.uk

[10] RF-Solutions Ltd.: “AM Super Regenerative Receivers. AM-HRRN-XXX”, http://www.rfsolutions.co.uk/acatalog/DS016-11.pdf.

[11] RF-Solutions Ltd.: “AM Hybrid Transmitter. AM-RT4-XXX AM-RT5-XXX”, http://www.rfsolutions.co.uk/acatalog/DS013-3.pdf.

[12] Physikalisch-Technische Bundesanstalt: http://www.ptb.de/en/org/4/44/442/dcf77_1_e.htm

[13] Kay Römer, Philipp Blum and Lennart Meier: “Time Synchronization and Calibration in Wireless Sensor Networks”, In: Ivan Stojmenovic (Ed.): Handbook of Sensor Networks: Algorithms and Architectures, John Wiley & Sons, ISBN 0-471-68472-4, pp. 199-237, September 2005 http://www.vs.inf.ethz.ch/publ/papers/wsn-timecali.pdf.

[14] Bharath Sundararaman, Ugo Buy and Ajay D. Kshemkalyani: “Clock Synchronization for Wireless Sensor Networks: A Survey”, Ad-Hoc Networks, 3(3): 281-323, May 2005. http://www.cs.uic.edu/∼ajayk/ext/ClockSyncWSNsurvey.pdf.

[15] F. Christian: “Probabilistic clock synchronization”, Distributed Computing, Vol. 3, pp. 146-58, 1989

[16] Joseph Y. Halpern and Ichiro Suzuki: “Clock Synchronization and the power of Broadcasting”, Distributed Computing, 5(2):73-82, 1991.

[17] Jeremy Elson, Lewis Girod and Deborah Estrin: “Fine-grained Network Time Synchronization using Reference Broadcasts”, In *Fifth Symposium on Operating Systems Design and Implementation* (OSDI 2002), December 2002.

[18] Hui Dai and Richard Han: “Tsync: A Lightweight Bidirectional Time Synchronization Service for Wireless Sensor Networks”, ACM SIGMOBILE Mobile Computing and Communications Review, 8(1):125-139, January 2004.

[19] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, R. Han: “MANTIS: System Support For MultimodAl NeTworks of In-situ Sensors”, 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA) 2003, pp. 50-59, http://mantis.cs.colorado.edu/media/p348-han_public.pdf

[20] P. Zhang, C. Sadler, S. Lyon, and M. Martonosi: “Hardware Design Experiences in ZebraNet”, Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys) 2004, November 2004. http://www.princeton.edu/∼csadler/sensys04.pdf

[21] u-blox AG: “ANTARIS 4 GPS Modules. System Integration Manual (SIM)”, http://www.u-blox.com/customersupport/gps.g4/ ANTARIS4_Mdoules_SIM(GPS.G4-MS4-05007).pdf

[22] u-blox AG: “ANTARIS SuperSense. Field Tests”, http://www.u-blox.com/products/Product_Summaries/ GPS_SuperSense_Tests(GPS.G3-X-04004).pdf

[23] Esben Zeuthen and Jan Flora: “Re-Mote”, http://www.distlab.dk/remote/

[24] Chunlong Guo, Lizhi Charlie Zhong, Jan M. Rabaey: “Low Power Distributed MAC for Ad Hoc Sensor Radio Networks”, IEEE GlobeCom 2001, November 2001.

[25] Curt Schurgers, Vlasios Tsiatsis, Saurabh Ganeriwal, Mani Srivastava: “Topology Management for Sensor Networks: Exploiting Latency and Density”, ACM MobiHoc 2002, June 2002

[26] Eugene Shih, Paramir Bahl, Michael J. Sinclair: “Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices”, MOBICOM ’02 September 23-26, 2002, Atlanta, Georgia, USA

[27] Carla F. Chiasserini, Ramesh R. Rao: “Combining Paging with Dynamic Power Management”, IEEE INFOCOM 2001

[28] Primož Škraba, Hamid Aghajan, Ahmad Bahai: “RFID Wake-up in Event Driven Sensor Networks”, http://www.standford.edu/∼primoz/sigcomm04

[29] Matthew J. Miller, Nitin H. Vaidya: “A MAC Protocol to Reduce Sensor Network Energy Consumption Using a Wakeup Radio”, IEEE Transactions on Mobile Computing May/June 2005, Volume 4, Number 3, pp. 228-242 http://www.crhc.uiuc.edu/∼mjmille2/rsrch/tmc04.ps

[30] Joseph Polastre, Jason Hill, David Culler: “Versatile Low Power Media Access for Wireless Sensor Networks”, SenSys ’04, November 3-5, 2004, Baltimore, Maryland, USA. http://www.polastre.com/papers/sensys04-bmac.pdf

[31] Lin Gu and John A. Stankovic: “Radio-Triggered Wake-Up Capability for Sensor Networks”, Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS’04) http://www.cs.virginia.edu/papers/Radio_Triggered.pdf

[32] Wei Ye, John Heidemann: “Ultra-Low Duty Cycle MAC with Scheduled Channel Polling”, USC/ISI Technical Report ISI-TR-604, July 2005

[33] Wikipedia - The Free Encyclopedia: “Manchester Encoding”, http://en.wikipedia.org/wiki/Manchester_encoding