ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. Dr. L. Thiele
Fall term 2010

SEMESTER THESIS

Interface Development for Next Generation FlockLab

Author:
Dominic Just

Supervisors:
Christoph Walser
Federico Ferrari
Marco Zimmerling

Abstract

Wireless Sensor Networks are often deployed in outdoor settings, where it is difficult to access them. Therefore, it is crucial that they work reliably when deployed. Special testbeds are designed for research and development. Gaining better knowledge about communication or energy consumption is an important topic of current research. Nevertheless, current Wireless Sensor Network testbeds often provide only very limited functions for power profiling.

The new FlockLab architecture of ETH Zurich offers detailed monitoring of sensor nodes including power measurement. An Analogue to Digital Converter samples the current drain of the node with high accuracy and speed. The transfer of those measurements sets several requirements in terms of real time and of computational resources.

In order to read measurement data using an Analogue to Digital Converter, a suitable driver is required. The goal of this semester thesis was to write a driver for the Linux kernel which adds support for this device. The purpose of this driver is to read the values of the Analogue to Digital Converter and to provide an interface for its functionality

Furthermore, the driver was tested to verify its correctness. It is capable of sampling the power in a way that it allows extracting accurately the behaviour of the power consumption of a node.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

A Wireless Sensor Network (WSN) is a network of many sensor nodes which communicate with each other. The sensor nodes are equipped with measurement sensors specific to their application. In a WSN, the measurement data from each node is forwarded to other nodes until it eventually reaches a data sink. The sink further processes and analyses the measurement data. WSNs are an ongoing interest in current research.

Usually, sensor nodes in WSN are battery powered and can communicate wirelessly with neighbouring nodes. A single sensor node basically consists of sensors and a radio. In order to gain more measurement data for research, a testbed is designed to test sensor nodes and to extract data of the sensor nodes. The observer is the most important part of a testbed and it is responsible for controlling all devices in a testbed.

For testing WSN hardware, an important question is whether the testbeds can be simulated or whether tests including real hardware have to be carried out. In the FlockLab project, tests including real hardware are carried out. These testbeds can process large amounts of data over years in order to gain higher quality information out of the real environment then simulations get.

ETH Zurich is currently developing a new version of the FlockLab[1] testbed architecture. This new architecture supports time-accurate power measuring and the facility of state extraction. These two features allow detecting problems which only occur in certain cases and cannot be simulated. Moreover, experiments can be carried out easily in an environment without having to use expensive lab instruments.

One of the most interesting other WSN testbeds is MoteLab at Harvard University. MoteLab is an indoor testbed and consists of 190 permanently deployed nodes. In the MoteLab testbed, there is one dedicated node in the network that supports power profiling with a sampling rate of 250Hz using external devices.

In real applications, the WSN nodes are battery powered and should be capable

of running a long time without needing to be maintained. Therefore, one important task in the development of sensor nodes is to reduce the energy consumption to a minimum. MoteLab and FlockLab are currently the only WSN testbeds, which supports power profiling. However, MoteLab's sampling rate of 250Hz is hardly sufficient to track the power consumption of every possible state in the system. The sampling in MoteLab is also very expensive, because external devices are used. Therefore, only one node's power consumption is observed.

1.2 Contributions

For measuring the power consumption of the WSN testbed, new hardware was introduced. This hardware is capable of sampling the power consumption of a node with high speed and accuracy. To use the functions of the hardware, a driver has been implemented. This driver reads the measurement of the ADC as soon as it is ready and stores it temporarily. Then, the samples can be handed to user space for further analysis. For controlling the ADC by programs running on the sensor node, the driver should provide some adequate interfaces.

Chapter 2

Background and Related Work

A Wireless Sensor Network (WSN) is a network of distributed sensor nodes. Sensor nodes are commonly used for measuring environmental data such as air pressure and temperature. In a WSN, each node is equipped with an antenna to communicate with neighbouring nodes. The measurement data is forwarded along multi-hop paths to the data sink, which is often not directly reachable by a node.

WSNs are an ongoing field of research. Several different WSN testbeds have been developed to support the development of WSN applications and networking protocols. They help to identify bugs and ensure an acceptable system performance prior to deployment. In particular, since nodes are typically powered by batteries, power measurements from a WSN testbed provide valuable guidance in improving the energy-efficiency of the protocols employed.

2.1 Existing WSN Testbeds

One example of a WSN testbed is MoteLab[2] at Harvard University. MoteLab is an indoor testbed and consists of 190 permanently deployed nodes. The nodes communicate via Ethernet to a server, which is responsible for scheduling tasks that run on the nodes. MoteLab provides an advanced web-interface for scheduling tasks. The scheduling policy assigns a time window to each user, during which tasks of that user are executed. However, after a task has been scheduled, the user has no fine-grained control anymore.

The TWIST testbed[3], developed at Technical University of Berlin, is another publicly available WSN testbed. It consists of 204 sensor nodes permanently deployed in an indoor environment. Nodes in the TWIST testbed are optimised within a segmented architecture. In a segmented architecture, there are different types of nodes. Some nodes serve as gateways between different types of nodes that use incompatible radio technologies or protocols. These gateways are called Super Nodes in the TWIST testbed architecture and have fewer constraints in terms of memory and computational resources as the majority of nodes have.

The TWIST testbed is designed to experiment with failures of single nodes. In

order to simulate a failure of a node, the node can be powered off, which has the same effect to the network as a failure of the node. On the other hand, an inactive node can be powered on in order to simulate a new node joining the network.

2.2 Power Profiling in WSN Testbeds

In the MoteLab testbed, there is one dedicated node in the network that supports power profiling. This node is connected to a networked Keithley digital multimeter. This multimeter supports sampling the power consumption continuously with a rate of 250Hz. The power samples are time-stamped and automatically logged on the server to make it available for further analysis. However, a sampling rate of 250Hz might not be sufficient in order to track the power consumption of every possible state in the system. For example, a significant amount of the consumed power is used by the radio, which may have uptimes of only a few milliseconds. Therefore, it is crucial to provide fast sampling during the uptime of the radio. The other problem of MoteLab is that the power measurement is very complex and expensive technology has to be used. The TWIST testbed does not offer functions to track the power consumption. It only provides the possibility to which the USB power supply of all nodes can be turned on and off. This feature allows to evaluate the battery lifetime for a given WSN application or protocol.

Chapter 3

From Hardware to Software

ETH Zurich is currently developing a new version of the FlockLab[1] testbed architecture. FlockLab offers detailed monitoring of sensor nodes, which includes power measurements and chip monitoring. A set of 9 nodes and one server are currently deployed in an indoor test setting. In order to transmit data, each node communicates directly with neighbouring nodes to forward measurement data. Finally, the data reaches the server and is either stored in a data base or further processed. Amongst others, the goal of FlockLab is to be providing a faster, easier and cheaper option for power measurement of each node than its competitors do.

This thesis focuses on measuring the power consumption of sensor nodes. Therefore, it is important to look into the functionality of the used ADC and the transmission of samples from the ADC. Then, it is crucial to understand how data is being processed on the Linux kernel of the observer before it can be further used in software.

3.1 Layout of the FlockLab Observer

The underlying hardware of this project is the FlockLab board, which is depicted in Figure 3.1. There are two devices attached to the FlockLab board: the target node and the observer.

The observer consists of a Gumstix verdex pro XL6P [4], which is an embedded Linux device. It has a CPU, main memory, and flash memory in order to store the operating system and for files containing data (e.g. logs). The observer node runs Openembedded Linux. Among others, the observer is responsible for the following tasks:

- All devices on the FlockLab board are controlled by the observer. Any device on of the FlockLab board can signal the observer, when it needs its attention. Then, the observer is supposed to process data or control the devices.
- The observer collects all measurements of the devices located on FlockLab board and processes them. It brings the data to a form to send it over WLAN.

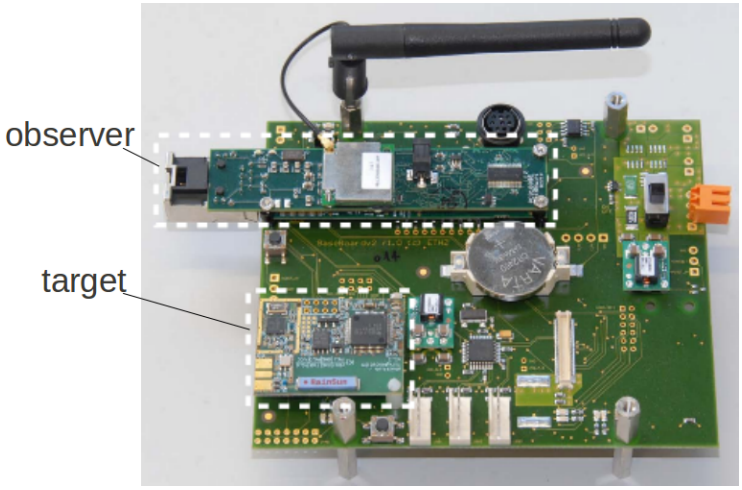

Figure 3.1: The current version of the FlockLab board consists of a base board with the observer and the target node.

There, the data is forwarded though an ad-hoc multi-hop network until it finally reaches the base station. It is also crucial to add additional information to the data (e.g. time stamps) to ensure a correct interpretation of it later on.

The target nodes (c.f. Fig 3.1) are the parts of interest in the FlockLab project. In the current testbed, TinyNodes184 from Shockfish SA [5] are used. One important parameter, which is measured on the current FlockLab board, is the power consumption. Therefore, the current draw of the target node is sampled by an analogue to digital converter (ADC), which is brazed on the FlockLab board. The ADC then passes the information to the observer using the SPI protocol, as described in the next section.

3.2 The Analogue to Digital Converter

A new ADC has been introduced in this design. This new ADC allows to monitor the power consumption using a high sampling rate to provide more accurate measurements. According to the data sheet of the new ADC[6], the Texas Instruments ADS1271 provides tree modes of operation, which are summarized in Table 3.1. In high-resolution mode, the Signal to Noise Ratio (SNR) is the highest, whereas high-speed mode offers sampling with 105 kilo Samples per Second (kSPS). The low-power mode allows to measure with a power consumption of only 35mW. The ADC uses a reference voltage V_{ref} and the measurement voltage $V_{meas} \in (-V_{ref}, V_{ref}]$ in order to compute the ratio:

$$V_{OUT} = C_{res} \cdot \frac{V_{meas}}{V_{ref}}, \qquad C_{res} = \begin{cases} 2^{15} & \text{high-speed, low-power mode} \\ 2^{23} & \text{high-resolution mode} \end{cases} \tag{3.1}$$

Table 3.1: The Modes of Operation of the ADC.

Mode	Sampling Rate	Power Draw	SNR
high-speed mode	105kSPS	92mW	106dB
high-resolution mode	53kSPS	90mW	109dB
low-power mode	53kSPS	35mW	106dB

Equation 3.1 shows that in high resolution mode, a change of the number V_{OUT} amounts to a much smaller difference in the variable V_{meas} as it does in high speed or low power mode. After a sample is ready on the ADC, the ADC outputs V_{OUT} to the observer as a signed 24 bit integer using the SPI protocol.

3.2.1 The SPI Protocol

The ADC's outputs are transmitted serially to the observer using the SPI protocol, which is depicted in Figure 3.2. The SPI format is extended with an nDRDY signal, which goes low if new data is ready. This signals that there is a new sample ready to be read. The nDRDY pin is connected to the processor of the observer. When the nDRDY pin goes from high to low, an interrupt is triggered. This interrupt makes the observer starting the serial transmission of the data using the SCLK clock. The SCLK clock makes the ADC transmitting one bit to the observer in each SCLK clock cycle. The output DOUT is the data output and the input CLK is a clock, which is required for the integrated circuit of the ADC.

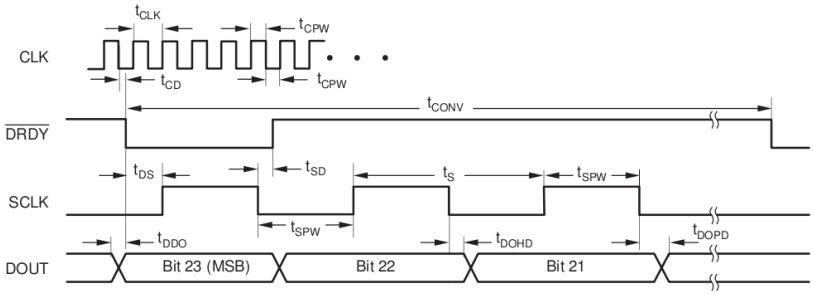

Figure 3.2: This figure depicts the course of events in the SPI protocol. CLK and SCLK are inputs, whereas nDRDY and DOUT are outputs of the ADC. Source: [6].

3.2.2 Transferring Samples to the Observer

The ADC for the new design of the FlockLab board is significantly faster than the one in the old version. Unfortunately, this leads to problems with the speed of the previously used functions. In particular if the handling of interrupts is too slow, it can happen that new data arrives before the current data transfer has been completed. This results in corrupted data, as depicted in Figure 3.3.

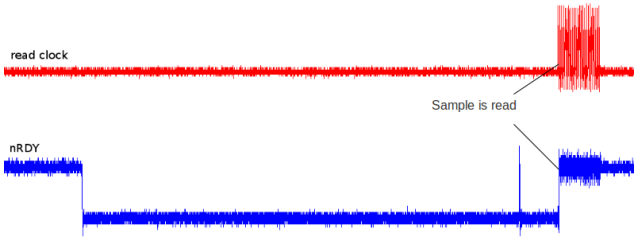

Figure 3.3: After the falling edge of the not ready (nDRDY) signal, the sample is not read until it is overwritten by the next sample (indicated by a pulse of nDRDY).

The way of transferring data has therefore been redesigned with a strong focus on speed. The old version of the FlockLab board used already existing functions in order to transfer the data. These functions were generic for all SPI transfers. They supported many different hardware functions and were designed to handle only sparely occurring SPI transfers. Since there are many frequently occurring transfer events in the new ADC, the new driver only uses the absolutely necessary instructions to transfer a sample and is tailored to the used ADC. This makes transferring samples much faster compared to the old version.

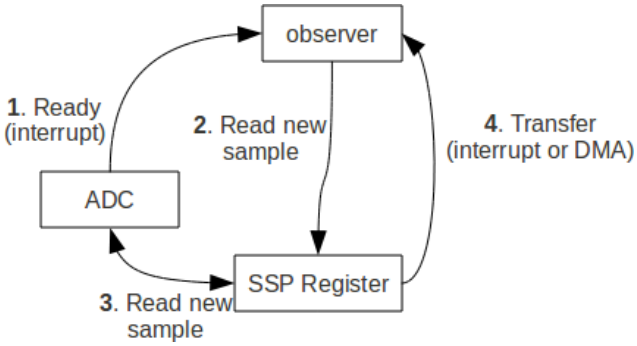

Figure 3.4: The ADC signals the observer that a sample is ready (1). The CPU starts the SSP (2) to let it read the new sample (3). After the sample is in the SSP register, the sample can be transferred to the observer using an interrupt or Direct Memory Access (DMA).

Figure 3.4 depicts the way of transmitting data from the ADC to the observer. When a new sample is ready on the ADC, it has to be transferred to the observer. The observer is in charge of controlling, when which device has to send any samples.

Whenever a new sample is ready, the observer has to be interrupted in order to read the sample. When an interrupt is triggered, the interrupt handler starts the Synchronous Serial Protocol (SSP), which works as a master for serial reading out of the ADC's value. This value is then stored in the CPU internal SSP registers. When this transfer is complete, either another interrupt can be triggered or a DMA transfer can be requested to transfer the temporarily stored value from the SSP registers to the observer.

3.3 The Linux Kernel as Interface between Hardware and Software

On the observer, a Linux kernel is responsible for controlling the hardware. A kernel's task is mainly to provide hardware functions to user space programs. The kernel acts therefore as an interface between hardware and software. For doing this, the processes in the kernel have several privileges. For example, they can access any area in memory or any device. This privileged area is called kernel space, whereas the term user space refers to the non privileged area, where all processes except the kernel are running.

According to [7], the Linux kernel is built of discrete modules. Each module adds specific functions to the kernel (c.f. Figure 3.5). The basic features such as process management, support of file systems or memory management are permanently in the kernel. A feature, which is only used in certain circumstances (e.g. to support printers or music players) is typically loaded into the kernel whenever it is necessary. Once, its functionality is no longer used, the module is removed from the kernel in order to keep the kernel simple.

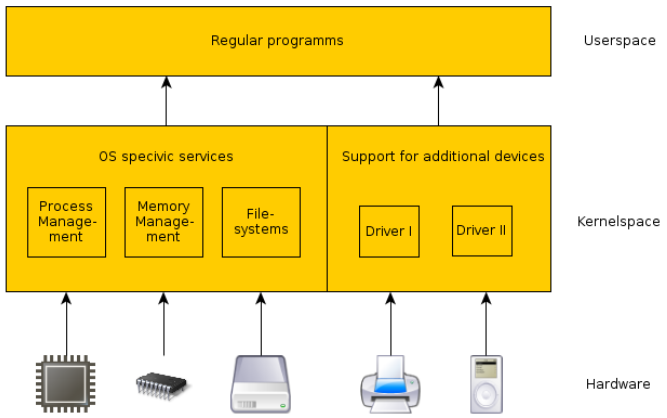

Figure 3.5: The kernel acts as an interface between software (in user space) and the hardware.

The driver for controlling the ADC is implemented as a kernel module. A driver is software, which generally runs in kernel space and provides hardware functions in an abstracted way to other programs. The driver for the ADC controls the hardware, which is responsible for managing the measurement process as described in Section 3.2.2. It also provides functions to user space to control the kernel module. A deeper insight of the functioning of the kernel module is provided in Chapter 4.

The kernel module normally interacts with a process running in user space. For this thesis, the user space process is mainly to verify the correct functioning of the kernel module. However, since the kernel module is generic, any process in user space can use the kernel module's functions.

Chapter 4

The Program Flow

The flow of data described in the last chapter is controlled by the observer. This controlling is implemented as a kernel module, which manages the control sequence according to Figure 3.4. Chapter 4 offers a detailed understanding of the structure of the kernel module. It also describes how each part of the driver is exactly implemented and how it interacts with other parts. The kernel module's main task is to enable the ADC to measure the power consumption. Moreover, the non trivial task of efficiently storing the samples and the alternative approaches are discussed in detail. Additionally, the kernel module offers two functions that can be called from user space. The first one allows to control the sampling of the ADC (I/O control), whereas the second function is used to pass data from kernel space to user space (reading).

4.1 Overview

The driver consists of two parts. One part runs in user space and the other part runs in kernel space. For controlling the kernel space from user space, the common Linux interface *I/O control* is used. In order to pass the samples to user space, the *read* function can be called from user space. Figure 4.1 provides an overview of the whole driver.

The core functionality of the driver is running in kernel space and is split in two functions. A measurement process collects all the measurements from the ADC and a read process converts the values and provides them to user space. In user space, there are two processes. One process controls how many measurements should be sampled. The other process is responsible for processing and storing the measurement data in a database.

When running the driver, the user space process, which is responsible for controlling the sampling, first, has to tell the kernel module how many samples it should record. By doing this, the measurement function, which is further described in Section 4.2.1, begins to record the requested number of samples. After enabling the measurement, the user space process can call the read function, which makes

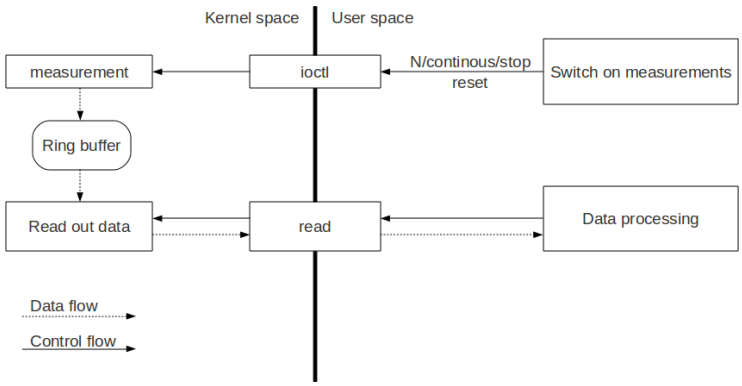

Figure 4.1: On the right of this figure, several processes in user space use the functions of the driver. On the left side, several processes in kernel space run concurrently in order to provide the desired functions to user space.

the read process in the kernel module reading a requested number of samples. The read process is described in more detail in Section 4.2.2. For storing the measurement data temporarily until it is finally transferred to user space, an adequate data structure (the ring buffer) is used and discussed in Section 4.3.

4.2 Set-up of the Kernel Module

The kernel module's main task is to enable the ADC to measure the power consumption. Additionally, the kernel module offers two functions that can be called from user space. The first one allows to control the sampling of the ADC (I/O control), whereas the second function is used to pass data from kernel space to user space (reading).

4.2.1 Program Flow for Measurements

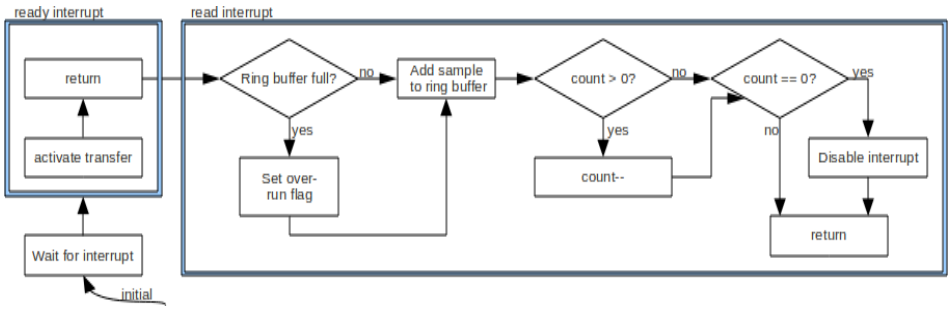

Figure 4.2: The ready interrupt enables the transfer of the new sample to the CPU. The read interrupt handler is responsible for storing the sample in the ring buffer

The measurement function supports two different measuring modes. The dis-

crete measuring mode reads a predefined number of samples and stops sampling after enough samples are collected. In continuous mode, the driver samples until the stop I/O control is called.

The program flow of the measurement process is depicted in Figure 4.2. First, a ready interrupt is triggered to make the hardware read the mew sample from the ADC. In order get the values into the driver, another interrupt is used (read interrupt). After the whole sample is read of the ADC, this interrupt is triggered in order to let the SSP begin to transmit data from the SSP register to the driver. Before storing any value, it has to be checked whether the ring buffer is already full. If so, an overflow occurred, this has to be redirected to user space when the user reads values the next time. An overrun-flag holds the information, about the occurrence of an overflow. In case of an overflow, the measurement continues and the oldest value is overwritten with the newest value (ring buffer). If the ring buffer is not full, the new value is just added to the ring buffer. The variable count (c.f. 4.1), which holds the information, how many samples are still needed to be measured, has always to be updated, too. The measurement process stops further measuring by disabling the ready interrupt, as soon as there is no more data to sample (counter is 0).

Since the ADC used in this project supports a high sampling rate, it is important to read the values immediately after a new sample is ready. Otherwise, there might be the risk of losing samples, if they are read while the next sample is already at the output of the ADC. Therefore, the ready interrupt is triggered, as soon as there is a new sample ready. Figure 4.3 illustrates the point of time, in which the ready interrupt is triggered. The interrupt handler starts the SSP immediately in order to transmit the new measurement sample to temporary auxiliary registers. The interrupt handler of the ready interrupt terminates immediately after enabling the SSP functionality and passes CPU computing resources to other processes.

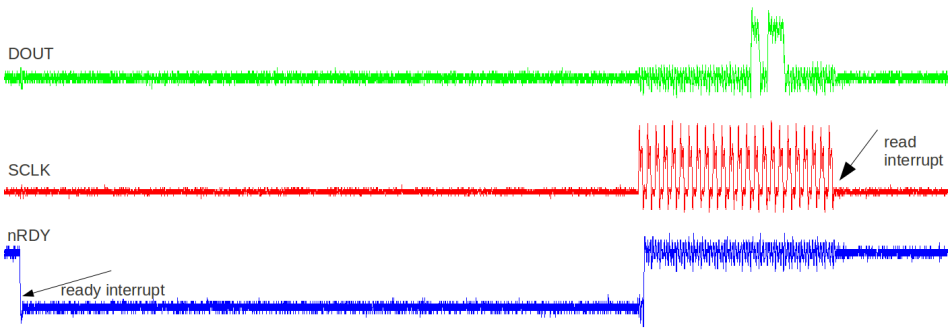

Figure 4.3: Timing for reading a 24 bis sample. At the falling edge of nDRDY (not ready), the ready interrupt is triggered, which starts the process of reading the sample. At the data output, the value 0x0002C0 is sent. SCLK clocks the serial data transfer. After the value is read from the ADC, the read interrupt is triggered in order to process the new sample.

Table 4.1: Meaning of the Variable count and the Argument N of the I/O Control.

count, N	Meaning
>0	*Discrete mode.* The ADC samples a desired number of samples and stops afterwards.
0	*Sampling disabled.* The driver disables the ready-interrupt and therefore stops sampling.
<0	*Continuous mode.* The ADC samples continuously.

The variable *count* holds the measuring mode according to Table 4.1. In continuous mode, count holds a negative number, whereas the discrete mode is selected by setting the counter to a positive value. The value 0 of the count variable stops the measuring.

In discrete mode, the count variable acts as a counter which depicts how many values are still needed to be read. This counter has to be decremented whenever a new value is read. The measurement function then waits for the next ready-interrupt in order to read the next sample. As soon as either enough values are collected or the I/O control command with the value 0 (stop measurement) is called, the interrupt is being disabled. After disabling the interrupt, new values don't result in a program action and therefore the whole measurement is stopped.

For simplicity, the driver is designed that this function and the reset I/O control (c.f. 4.2.3) are the only functions that can disable the read interrupt.

4.2.2 Program Flow for Reading

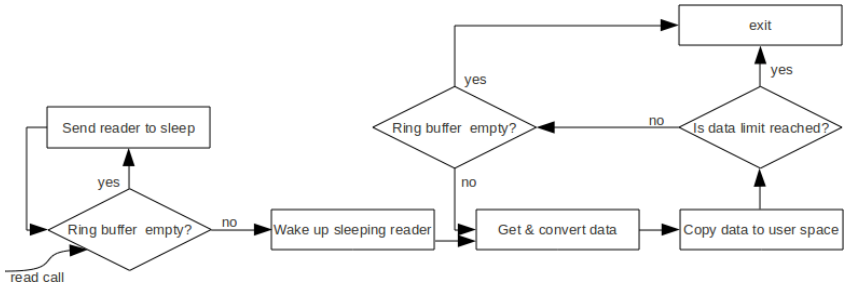

Figure 4.4: When there is a read request and there is data ready, the data is being converted and copied to user space. After the process function exits the read-loop, the number of read samples is returned.

When the read function is called from user space, the ring buffer is checked to be empty. If the ring buffer does not contain any values, the reader is sent to sleep and the read process is blocked until there are values ready. When there are values in the ring buffer to be read, any sleeping reader processes in user spaces are waken

up and the reading function starts. Figure 4.4 gives a comprehensive overview of how the reading process is working.

The read process consists of a loop that executes until the FIFO ring buffer is empty or the requested number of data is read. When exiting the loop, the number of read samples is returned to user space. The following three steps are used to perform the reading process:

1. *Reading from ring buffer.* The first sample of the ring buffer is read.
2. *Converting.* Depending on the mode in which the ADC is running, the samples in kernel space are either 24 bit (high resolution mode) or 16 bits wide (high speed and power saving mode). In user space, however, there is only a 32 bit data type available. Therefore, the samples have to be converted into the right format before the data can be copied to user space.
3. *Copying to user space.* Address space of processes in kernel space and processes in user space are different. Moreover, access to a memory address of the kernel space from user space is not allowed for security reasons. Therefore, the sample has to be copied to user space.

4.2.3 Program Flow for I/O Control

In order to support several functions to control the driver, the I/O control function is used. The I/O control function is a standard function for controlling kernel modules from user space. The I/O control function has a variable number of arguments. The first argument, which is mandatory, is used to specify what has to be executed within the I/O control function. Depending on the value of this argument, one of the two branches out of *ioctrl* in Figure 4.5 is taken. The current driver supports the following two functions, which can also be seen in Figure 4.5:

- *Measure.* This function lets the driver read either a specified number of elements, sample continuously, or stop a currently ongoing measurement.
- *Reset.* This function resets the driver completely. It can be called after an error occurred, in order to ensure that the whole state of the driver is reset to its initial values, or when initializing the driver.

4.2.3.1 The Measure I/O Control

The measure command lets the driver read either a specified number of elements or starts continuous sampling. N is used as argument of this I/O control. N defines what the ADC is going to do according to Table 4.1. By using the measure I/O control, the internal state *count* is updated according to the following rule:

- N>0: If count is greater or equal than 0, count is increased by N. Therefore, a measure-I/O control with a positive value makes the driver measure N more

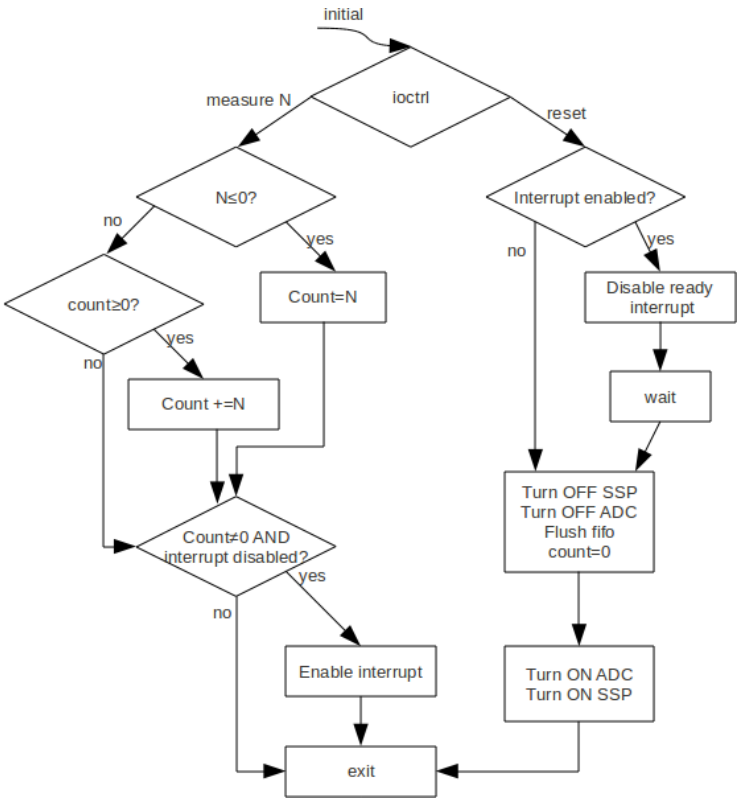

Figure 4.5: The I/O control function, which supports measure (right) and reset (left) of the ADC.

values than already specified in count. If the driver is measuring continuously, a measure-I/O control doesn't have any effects.

- N=0: The count variable is reset to 0 and the measurement is stopped in the next cycle.
- N<0: The count variable is set to N and the driver is going to measure continuously.

If the ready interrupt is not enabled and the driver is supposed to measure, the ready interrupt will be enabled too. However, the measure-I/O control never disables an interrupt. The measurement function is supposed to disable the ready interrupt when count is 0. If this I/O control is called with the argument 0, the interrupt handler of the measurement function is called one more time and disables the interrupt after sampling one more measurement.

It is important to ensure that the check, increment, and setting of count cannot be interrupted. If they are interrupted or another process in kernel space was using them concurrently, it could lead to incorrect behaviour of the driver due to race conditions.

4.2.3.2 The Reset I/O Control

This function resets the driver completely. The reset I/O control can be called after an error or while initializing the driver. The reset I/O control first disables any ready interrupts. Then, the SSP and the ADC are restarted, the ring buffer is cleared, and the counter is set to 0.

When the ready interrupt is disabled, read interrupts can still occur in order to transfer data. If the read interrupt is triggered before the SSP registers are turned off, there is one more sample available in the driver, which is dropped anyway when the ring buffer is cleared. On the other hand, it can happen that the second interrupt handler is not executed yet when the I/O control turns off the SSP. Therefore, the I/O control routine waits a certain amount of time in order to ensure that all pending interrupts are executed. This makes the reset I/O control slow, which is no problem because this I/O control is only used sparsely.

4.3 Data Structure for Storing Measurement Samples

In order to fulfil the very demanding requirements in terms of efficiency, it was crucial to carefully select the data structure to save the samples from the ADC. The data structure should be capable of storing a new value using as few machine instructions as possible. Such efficient data structures would lead to a short interrupt handler and therefore to a good reliability concerning real time constraints. One focus was to choose a data structure easy to apply and ensuring correct behaviour of the driver. Another important requirement is to support flexibility.

An adequate data structure should store the measurements in a way that makes it possible to handle the data in the following ways:

- Read a number of values and return all samples to user space. This approach requires copying with a speed of around 410 Kbytes/s[1] in high speed mode. Although this speed is supported by the hardware, it is very resource intensive.
- Read a given number of samples and average them. If p samples have to be averaged, the copying of the data to user space could be done with much less effort, because it is not necessary to copy every single sample to user space separately. Averaging can also be done in different ways. It might be advantageous to calculate the Root Mean Square (RMS) average according to $I_{RMS} = \sqrt{\frac{1}{p}\sum_{i=1}^{p} i^2}$. Knowing the RMS average allows to compute the power dissipation with a given resistance according to $P = I_{RMS}^2 \cdot R$
- In certain scenarios, the user may be interested in getting the maximal or the minimal value only in order to find out the best or worst case power consumption. If only the maximum or minimum of the values over a certain time have to be copied, copying to user space would be very efficient.

[1] Copying speed $V_{cp} = 105496 \frac{Samples}{s} \cdot 4 \frac{Bytes}{Sample} = 412.09375 \frac{Kbytes}{s}$. Since the programs in user space uses the 32 bit wide integer data type, 4 Bytes have to be copied for each sample.

If the averaging functions are selected, it is not possible to compute the average completely in kernel space. One problem is that there are no floating point instructions available in kernel space. Additionally, there should be a focus on putting as many operations as possible to user space. Processes in kernel space execute with a much higher priority than processes in user space. Therefore, it is best to compute only the necessary operations to copy data to user space in kernel space. After the data is in user space, it can then be further computed whenever resources are available. In order to allow (RMS) averaging in user space, two numbers have to be transmitted to user space: The (squared) sum over all samples and the number of samples. With these numbers, the (RMS) average can then be computed in user space.

4.3.1 Simple Summation

This approach sums up all new values and holds a counter in order to track how many samples are already collected. After enough samples are summed up, the sum and the number of summed values are returned to user space in order to compute the average. This function could easily be modified in order to return the squared sum. It can also be modified easily in order to track the maximal or minimal value only. A completely different approach has to be used if all samples have to be passed to user space. The following pseudo code shows how this approach works:

```
clear & init:
 int avg_sum=0;
 unsigned int count=0;

add new value:
 avg_sum+=get_new_value();
 count++;
```

This approach uses only 6 instructions to work off one sample. Moreover, only two memory words are required to hold all the necessary variables. Therefore, this is a very effective approach if not all samples have to be passed to user space.

However, it has to be ensured, that there are no arithmetic overflows of the avg_sum variable. The most important downside is the lack of flexibility. It is indeed possible to modify the function described above to compute the squared sum of the samples or to evaluate the maximum or the minimum of all values. However, it is not possible to support entire flexibility, because the single samples are not stored.

4.3.2 The FIFO Ring Buffer

The needed flexibility however can be provided by a ring buffer. Because it stores each sample individually, the ring buffer is the approach finally applied for storing the samples. In order to use the ring buffer functionality, the kfifo interface provided by the kernel can be used. The kfifo is a generic kernel First-In-First-Out (FIFO)

implementation, which manages all routines to ensure the proper functioning with concurrency. Since is supports reading and writing at the same time, it is an ideal structure to support communication between the measure and the read function as depicted in Figure 4.1. Additionally, it is already optimized to be as fast as possible, while still providing all necessary functions. The FIFO data structure is more expensive than the simple summation, but still feasible. Because of is reason, the kfifo ring buffer was chosen to hold the data in this project.

Chapter 5

Evaluation of correct Operation

5.1 Influence of real time constraints

It is very important to meet real time constraints. If interrupts cannot be executed in time, measurement values get lost. It is therefore crucial to watch a measurement of a large amount of data closely to give a statement, whether all real time constraints are fulfilled.

5.1.1 Experimental Set-up

A logic analyser was connected to the ADC and monitored three pins: The clock pin (CLOCK), the data output (DATA) and the not ready pin (nREAD). The ADC was also connected to the FlockLab board. The driver was running on the observer and a user space program was especially designed to make the ADC continuously measure the power consumption using the ADC and its driver. The logic analyser was set up to trigger a measurement as soon as the clock signal starts oscillating for the first time. It then records the fist 100ms of the power measurement of the ADC.

The results of this measurement are depicted in Figure 5.1.

5.1.2 Results and Discussion

Figure 5.1 shows that there are indeed some problems concerning real time constraints. It happens that some of the samples get lost. The largest gap in the measurement was around 250μs. During the first 100ms, only 3 of those large gaps were captured. There are of course more gaps with much shorter durations, too. There are also gaps which are not obvious in Figure 5.1.

The results of this measurement however don't jeopardise the correct functioning of the driver. The gaps in Figure 5.1 don't corrupt the measurement. In the worst case, the power consumption of the radio is wanted and the radio is turned on for just a very short time and the measurement stalls exactly at that time. However,

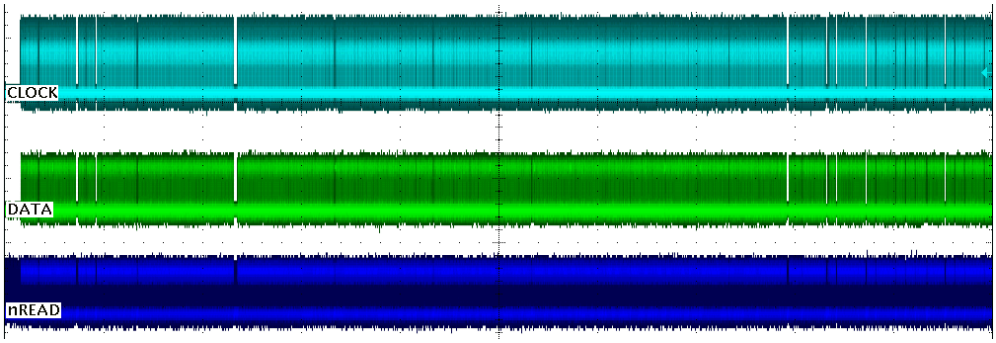

Figure 5.1: This figure shows the results of the experiments regarding real time constraints. It shows the first 100ms of the read process of the ADC. The vertical dashed lines represent 10ms.

the shortest supposable uptime of the radio exceeds 250μs by far! This means that the results of the ADC are good enough to always fully determine the power consumption of the target node.

The shorter gaps are not problematic, either. Even if they occur more frequently than the large gaps, the small gaps in the measurement are just small losses of samples compared to the high sampling rate of the ADC.

Chapter 6

Conclusion and Future Work

The goals of this thesis where the design and implementation of an ADC driver for the next generation FlockLab testbed.

The software implementation of the ADC driver allows measuring the power consumption with a high number of samples, which offers a comprehensive understanding of the sensor node's power consumption. The driver of this semester thesis provides generic functions to user space in order to allow a comfortable use of the ADC.

However, there are some open problems. For example, it can happen that the driver is not fast enough to handle all interrupts within their deadlines. This results in missing some samples. However, only a small number of samples are lost compared to the very high sampling speed of the ADC. Therefore, the number and the distribution of missed interrupts does not jeopardise the correctness of the measurement results.

The driver is documented and programmed in a way that allows easy extension with new functionality. In addition, the following improvements or extensions are worthwhile:

Instead of using the read interrupt to make the observer read a sample from the SSP registers, this could be done with DMA. Using DMA would halve the number of interrupts in the driver, and effectively reduce the CPU load. Less interrupts can also be advantageous in terms of missed deadlines. Since an interrupt is always executed atomically, an ongoing interrupt handler at the time when the next sample becomes ready delays the read of the new sample until the currently ongoing interrupt handler is completely executed.

New functions could be added in order to transfer only averaged values, minimums or maximums to user space, as described in Section 4.3. This would lead to an extended I/O control interface, which has to offer a function to enable this newly added mode of operation. It is likely, that this idea could accelerate the driver.

Bibliography

[1] ETH Zurich. Home page of the flocklab testbed. http://www.flocklab.ethz.ch/wiki/, December 2010.

[2] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. Motelab: a wireless sensor network testbed. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05, Piscataway, NJ, USA, 2005. IEEE Press.

[3] Vlado Handziski, Andreas Köpke, Andreas Willig, and Adam Wolisz. Twist: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, REALMAN '06, pages 63–70, New York, NY, USA, 2006. ACM.

[4] Gumstix inc., 2010. http://www.gumstix.net/Hardware/view/Hardware-Feature-Overview-Sheets/Gumstix-Verdex-Pro-Feature-Overview/112.html.

[5] Shockfish SA. Tinynode 184. http://www.tinynode.com/index.php?id=132, November 2008.

[6] Texas Instruments Inc. 24-bit, wide bandwidth analog-to-digital converter (rev. f). http://www.ti.com/lit/gpn/ads1271, October 2007.

[7] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers, 3rd Edition.* O'Reilly Media, Inc., 2005.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Engineering and Networks Lab (TIK)

Fall Term 2010

SEMESTER THESIS

for
Dominic Just

Supervisor: Christoph Walser
Co-Supervisors: Federico Ferrari, Marco Zimmerling

Start date: 21 September 2010
End date: 4 January 2011

Interface Development for Next Generation FlockLab

Introduction

Wireless sensor networks (WSNs) are composed of small, battery-powered nodes that are typically equipped with a microcontroller, a radio transceiver, and one or more sensors. Such a WSN can, for instance, be deployed on a mountain to monitor the environmental conditions and their impact on permafrost [1]. The sensor nodes are not only constrained by their limited energy supply (battery) but also in terms of processing power, communication capabilities, and available RAM. These limitations make the implementation of a WSN application a challenging task. To support the application development process from the very beginning, several test platforms have been proposed including simulators [2] and testbeds consisting of real sensor nodes [3, 4].

We are currently developing a service-oriented testbed architecture, called FlockLab [5, 6], which allows for detailed monitoring and stimulation of sensor nodes. Time-accurate state extraction and power measurements, taken at the same time at several nodes, enable thorough testing and debugging of distributed WSN applications. As shown in Figure 1, these services are provided to the tester by pairing the *target (sensor) node* with a powerful *observer*. The observer runs openembedded Linux and has full control of the pins of the target node to monitor or influence its behavior. Moreover, the observer can use the output of the FlockBoard's analog-to-digital converter (ADC) to accurately measure the power consumption of the target node over time. The FlockLab hardware architecture is generic in the sense that it is capable of supporting different types of target nodes. This is accomplished through a *target adaptor* (see Figure 1) that provides a hardware interface between the observer and the target node; to support possible future node types only a new adaptor is required while all other FlockLab components remain unchanged. Previous testbeds [3] are tight to a single target node and only allow for detailed measurements on individual nodes.

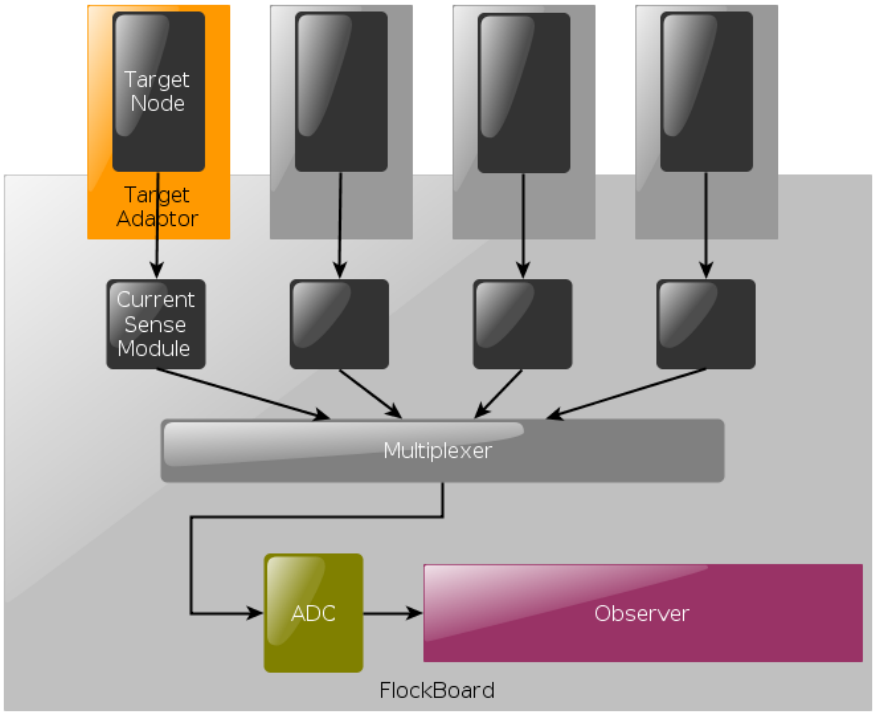

Figure 1: FlockBoard layout and interfaces for power measurements on the target nodes. The colored interfaces indicate the parts considered in this thesis.

Problem Definition

The goal of this semester thesis is to design and implement a Linux kernel driver for an ADC needed for power measurements in FlockLab. A major goal is to evaluate the operation mode of the ADC which fits best the needs of FlockLab. Furthermore, a hardware adaptor board should be designed that allows to attach Tmote Sky [7] sensor nodes to FlockLab.

This work particularly focusses on:

- implementation of a kernel driver for an ADC,
- evaluation of the ADC operating modes regarding speed vs. resolution,
- adaption of the existing power measurement service to the new ADC,
- design of a hardware adaptor board for the Tmote Sky sensor node platform using Altium Designer [8]
- test and validation of the implemented functionality.

In order to reach these goals, the project should proceed according to the following steps:

1. The student should write a project plan and identify its milestones (thematically and concerning time). In particular there should be enough room for the final presentation and the report.
2. The student should find and study related work in the area of Linux kernel driver development, printed circuit board (PCB) design and WSN testbeds. The results of this literature research should be written down as a first chapter of the report.

3. The student should design and implement a Linux kernel driver for the 24-Bit, wide bandwith ADC ADS1271 [9].
4. The student should compare the different operating modes of the ADC, evaluating the tradeoff between speed and resolution of the measurements.
5. The student should design, produce and test a hardware adaptor board to attach Tmote Sky sensor nodes to FlockLab.
6. The student should design and implement test scenarios to demonstrate that all implemented components operate correctly and efficiently.

Organization

- Duration of the Work:
  This Semester Thesis starts 21 September 2010 and has to be finished no later than 4 January 2011.
- Project Plan:
  A project plan with its milestones is held and updated continuously. Unforeseen difficulties that change the project plan have to be documented and should be discussed with the supervisors in a timely manner.
- Weekly Meetings/Reports:
  In regular (weekly) meetings with the supervisors, the current state of the work, potential difficulties as well as future directions are discussed. The day before the weekly meeting a brief status report should be sent to the supervisors commenting on these issues, in order to allow an adequate meeting preparation for the student and the supervisors.
- Research Journal:
  The work's progress is written down in a research journal that is handed in to the supervisor at the end of the project.
- Beginners Presentation:
  Approximately two to three weeks after the start the student presents the objectives of the work as well as some background on the topic. The presentation should not exceed 5 minutes and consist of no more than three slides.
- Final Presentation:
  By the end of the project, the student will present the achieved result. The presentation should not exceed 15 minutes.
- Documentation:
  At the end of the project, no later than *4 January 2011*, the student will have to hand in a written report. Together with the system implementation/software this report is the main outcome of the project. Code has to be commented extensively, allowing a follow-up project.
- Evaluation of the work:
  The criteria for grading the work are described in [10].
- Finishing up:
  The required resources (e.g., laptop, keys) should be cleaned up and handed back in.

References

[1] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, and M. Yuecel, "PermaDAQ: A scientific instrument for precision sensing and data recovery in environmental extremes," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2009)*, p. (to appear), 2009.

[2] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003)*, pp. 126–137, Nov. 2003.

[3] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A wireless sensor network testbed," in *Proc. 4th Int'l Conf. Information Processing Sensor Networks (IPSN '05)*, pp. 483–488, Apr. 2005.

[4] M. Dyer, J. Beutel, L. Thiele, T. Kalt, P. Oehen, K. Martin, and P. Blum, "Deployment support network - a toolkit for the development of WSNs," in *Proc. 4th European Workshop on Sensor Networks (EWSN 2007)*, pp. 195–211, 2007.

[5] J. Beutel, R. Lim, A. Meier, L. Thiele, C. Walser, M. Woehrle, and M. Yuecel, "Poster abstract: The flocklab testbed architecture," in *Proc. 7th ACM Conf. Embedded Networked Sensor Systems (SenSys 2009)*, (Berkeley, CA, USA), pp. 415–416, November 2009.

[6] C. Walser, "Flocklab." http://www.tik.ee.ethz.ch/ flocklab.

[7] Moteiv, "Tmote sky." http://sentilla.com/files/pdf/eol/tmote-sky-datasheet.pdf.

[8] Altium, "Altium designer." http://www.altium.com/products/altium-designer.

[9] T. Instruments, "Texas instruments, ads1271." http://focus.ti.com/docs/prod/folders/print/ads1271.html.

[10] TIK, "Notengebung bei Studien- und Diplomarbeiten." Computer Engineering and Networks Lab, ETH Zürich, Switzerland, May 1998.