A Heterogeneous System Architecture for Low-Power Wireless Sensor Nodes in Compute-Intensive Distributed Applications

Eine heterogene Architektur für energieeffiziente drahtlose Sensorknoten in rechenintensiven verteilten Anwendungen
Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation von Dipl.-Inform. Andreas Engel aus Sonneberg
Tag der Einreichung: 09.11.2015, Tag der Prüfung: 17.12.2015
Darmstadt 2016 — D 17

1. Gutachten: Prof. Dr.-Ing. Andreas Koch
2. Gutachten: Prof. Dr.-Ing. Christian Hochberger

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Eingebettete Systeme
und ihre Anwendungen

A Heterogeneous System Architecture for Low-Power Wireless Sensor Nodes in Compute-Intensive Distributed Applications
Eine heterogene Architektur für energieeffiziente drahtlose Sensorknoten in rechenintensiven verteilten Anwendungen

Genehmigte Dissertation von Dipl.-Inform. Andreas Engel aus Sonneberg

1. Gutachten: Prof. Dr.-Ing. Andreas Koch
2. Gutachten: Prof. Dr.-Ing. Christian Hochberger

Tag der Einreichung: 09.11.2015
Tag der Prüfung: 17.12.2015

Darmstadt 2016 — D 17

Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-57782
URL: http://tuprints.ulb.tu-darmstadt.de/id/eprint/5778

Dieses Dokument wird bereitgestellt von tuprints,
E-Publishing-Service der TU Darmstadt
http://tuprints.ulb.tu-darmstadt.de
tuprints@ulb.tu-darmstadt.de

Erklärung zur Dissertation

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 22.11.2016

_____

(Andreas Engel)

Abstract

Wireless Sensor Networks (WSNs) combine embedded sensing and processing capabilities with a wireless communication infrastructure, thus supporting distributed monitoring applications. WSNs have been investigated for more than three decades, and recent social and industrial developments such as home automation, or the Internet of Things, have increased the commercial relevance of this key technology. The communication bandwidth of the sensor nodes is limited by the transportation media and the restricted energy budget of the nodes. To still keep up with the ever increasing sensor count and sampling rates, the basic data acquisition and collection capabilities of WSNs have been extended with *decentralized* smart feature extraction and data aggregation algorithms. *Energy-efficient processing* elements are thus required to meet the ever-growing compute demands of the WSN motes within the available energy budget.

The Hardware-Accelerated Low Power Mote (HaLoMote) is proposed and evaluated in this thesis to address the requirements of *compute-intensive WSN applications*. It is a *heterogeneous system architecture*, that combines a Field Programmable Gate Array (FPGA) for hardware-accelerated data aggregation with an IEEE 802.15.4 based Radio Frequency System-on-Chip for the network management and the top-level control of the applications. To properly support Dynamic Power Management (DPM) on the HaLoMote, a Microsemi IGLOO FPGA with a non-volatile configuration storage was chosen for a prototype implementation, called Hardware-Accelerated Low Energy Wireless Embedded Sensor Node (HaLOEWEn). As for every multi-processor architecture, the inter-processor communication and coordination strongly influences the efficiency of the HaLoMote. Therefore, a generic communication framework is proposed in this thesis. It is tightly coupled with the DPM strategy of the HaLoMote, that supports fast transitions between active and idle modes. Low-power sleep periods can thus be scheduled within every sampling cycle, even for sampling rates of hundreds of hertz.

In addition to the development of the heterogeneous system architecture, this thesis focuses on the energy consumption trade-off between wireless data transmission and in-sensor data aggregation. The HaLOEWEn is compared with typical software processors in terms of runtime and energy efficiency in the context of three monitoring applications. The building blocks of these applications comprise hardware-accelerated digital signal processing primitives, lossless data compression, a precise wireless time synchronization protocol, and a transceiver scheduling for contention free information flooding from multiple sources to all network nodes. Most of these concepts are applicable to similar distributed monitoring applications with in-sensor data aggregation.

A Structural Health Monitoring (SHM) application is used for the *system level evaluation* of the HaLoMote concept. The Random Decrement Technique (RDT) is a particular SHM data aggregation algorithm, which determines the free-decay response of the monitored structure for subsequent modal identification. The hardware-accelerated RDT executed on a HaLOEWEn mote requires only 43 % of the energy that a recent ARM Cortex-M based microcontroller consumes for this algorithm. The functionality of the overall WSN-based SHM system is shown with a laboratory-scale demonstrator. Compared to reference data acquired by a wire-bound laboratory measurement system, the HaLOEWEn network can capture the structural information relevant for the SHM application with less than 1 % deviation.

Kurzfassung

Drahtlose Sensornetze (*Wireless Sensor Networks*, WSNs) kombinieren eingebettete Sensorik und Rechenleistung mit einer drahtlosen Kommunikationsinfrastruktur, wodurch räumlich verteilte Überwachungsanwendungen unterstützt werden. WSNs werden seit mehr als drei Jahrzehnten erforscht, aktuelle soziale und industrielle Trends wie intelligentes Wohnen oder das Internet der Dinge haben aber auch die kommerzielle Bedeutung dieser Schlüsseltechnologie verstärkt. Die übertragbaren Datenmengen sind durch das Transportmedium und die verfügbare Energie der Sensorknoten beschränkt. Um die wachsende Anzahl an Sensoren und die steigenden Abtastraten dennoch zu bewältigen, wurden die ursprünglich als einfache Datenerfassungssysteme ausgelegten WSNs um Fähigkeiten zur dezentralen Datenaggregation erweitert. Um den dadurch ständig wachsenden Bedarf an verteilter Rechenleistung mit den beschränkten Energieressourcen zu realisieren, werden energieeffiziente Recheneinheiten benötigt.

In der vorliegenden Arbeit wird die *Hardware-Accelerated Low Power Mote* (HaLoMote) Architektur vorgestellt und deren Effizienz für rechenintensive WSN-Anwendungen untersucht. Dabei handelt es sich um eine heterogene Architektur mit einem *Field Programmable Gate Array* (FPGA) für die Hardware-Beschleunigung von Datenaggregationsalgorithmen und einem Funksystem mit integriertem Mikrocontroller für das Netzwerkmanagement und die übergeordnete Steuerung der Anwendungen. Um eine effiziente Energieverwaltung für die HaLoMote zu ermöglichen, verwendet die prototypische Implementierung namens *Hardware-Accelerated Low Energy Wireless Embedded Sensor Node* (HaLOEWEn) ein FPGA mit persistentem Konfigurationsspeicher. Wie bei jeder Multiprozessorarchitektur beeinflusst die Interprozessorkommunikation und -koordination auch die Effizienz der HaLoMote. Daher wurde ein anwendungsunabhängiges Kommunikationsschema entwickelt, welches eng mit den Energiesparmechanismen der Plattform verknüpft und auf schnelle Wechsel zwischen Aktiv- und Ruhemodi ausgelegt ist. Dadurch können Schlafphasen innerhalb jedes Abtastzykluses selbst bei Abtastraten von mehreren Hundert Hertz genutzt werden.

Die vorliegende Arbeit untersucht darüber hinaus das Abwägen zwischen der drahtlosen Übertragung von Sensordaten und deren lokaler Aggregation. Dazu wird die HaLOEWEn Plattform mit herkömmlichen Software-Prozessoren bezüglich ihrer Laufzeit- und Energieeffizienz im Rahmen von drei Überwachungsanwendungen verglichen. Die verwendeten Algorithmen kombinieren Hardware-Beschleuniger für digitale Signalverarbeitungsprimitiven und verlustfreie Datenkompression mit einem präzisen Zeitsynchronisationsmechanismus sowie einem Verfahren zum kollisionsfreien Verteilen von Informationen im Netzwerk. Diese allgemeinen Komponenten können für ähnliche verteilte Überwachungsanwendungen mit aufwändiger dezentraler Datenaggregation wiederverwendet werden.

Eine Anwendung aus dem Bereich der Strukturüberwachung (*Structural Health Monitoring*, SHM) wird für die systemische Evaluation des HaLoMote Konzepts verwendet. Die *Random Decrement Technique* (RDT) ist ein spezieller Aggregationsalgorithmus, welcher das freie Ausschwingverhalten der überwachten Struktur ermittelt, selbst wenn die eigentliche Anregung der Struktur nicht bekannt ist. Dies ermöglicht eine operative Modalanalyse, welche die Voraussetzung für eine autonome Langzeitüberwachung ist. Die Berechnung der RDT auf der HaLOEWEn Plattform benötigt nur 43 % der Energie, welche ein aktueller ARM Cortex-M Mikrocontroller für den glei-

chen Algorithmus verbraucht. Um die Funktionsfähigkeit des gesamten WSN-basierten SHM Systems nachzuweisen, wurde ein Demonstrator im Labormaßstab aufgebaut. Im Vergleich zu einem drahtgebundenen Labormesssystem können die wesentlichen Strukturinformationen vom HaLOEWEn Netzwerk mit weniger als 1 % Abweichung erfasst werden.

Danksagung

Diese Arbeit hätte ohne die Unterstützung vieler Personen nicht entstehen können. Allen voran möchte ich mich bei Prof. Dr.-Ing. Andreas Koch für die Betreuung der Arbeit bedanken. Zahlreiche thematische Diskussionen haben mir mögliche Entwicklungsrichtungen aufgezeigt, ohne mich dabei in meiner Kreativität zu beschränken. Für die Lektorate meiner Manuskripte bin ich ihm ebenfalls äußerst dankbar, insbesondere, da diese häufig sehr kurzfristig erstellt wurden. Darüber hinaus konnte ich Prof. Kochs gute Vernetzung zu wissenschaftlichen und industriellen Kooperationspartnern nutzen, um die Interdisziplinarität meiner Arbeit zu etablieren.

Prof. Dr.-Ing. Christian Hochberger möchte ich für das kurzfristige Erstellen des Zweitgutachtens danken.

Darüber hinaus möchte ich mich bei meinen Kollegen an der TU Darmstadt und dem Fraunhofer-Institut für Betriebsfestigkeit und Systemzuverlässigkeit bedanken, mit denen ich im Rahmen des LOEWE-Zentrums AdRIA zusammenarbeiten konnte. Besonders erwähnt seien dabei meine Koautoren Björn Liebig, Thomas Siebel und Michael Koch, die mich bei der Realisierung verschiedener Versuchsanordnungen unterstützt haben.

Ein ganz besonderer Dank gilt meiner Familie, allen voran meiner Frau Luise. Sie hat mir in den letzten Jahren immer wieder den Rücken frei gehalten und musste dafür selbst viel Geduld, Verständnis und Kraft aufbringen.

Contents

CHAPTER 1
Introduction

In this chapter, the history and current trends in Wireless Sensor Networks (WSNs) are summarized to motivate the relevance of compute-intensive distributed applications. Furthermore, reconfigurable computing for application-specific and hence energy-efficient computing architectures is briefly introduced. When joining both topics, relevant research problems arise that will be addressed in this thesis. Furthermore, the main contributions and the organization of the thesis are summarized in this chapter.

1.1 Importance of Wireless Sensor Networks and Energy-Efficient Computing

Similar to many fundamental technologies, WSNs originated from military applications, such as the networks of acoustic sensors already deployed in the 1950s for submarine tracking [Silverstein1978]. The United States Defense Advanced Research Projects Agency (DARPA) focused their research on distributed sensing, computation and communication within the Distributed Sensor Networks (DSN) project in the 1980s [CMUPDCS1978]. However, the radio transceivers available at that time (e.g., satellite links and early cell phones) could not achieve the required bandwidth, energy efficiency, reliability and integration density, so the first DSNs were based on wire-bound communication.

In the 1990s, advances in wireless communication technologies facilitated the shift from wired DSNs to actual WSNs. Furthermore, military projects such as the DARPA SensIT [Kumar2001] were complemented by university research projects like the Wireless Integrated Network Sensors (WINS) at UC Los Angeles [Asada1998] or the PicoRadio program at UC Berkeley [Rabaey2000]. This opened the WSN technology to a broad spectrum of civilian applications, such as the monitoring of environmental parameters and industrial processes, the tracking of objects, and the detection of events [Chong2003].

At the beginning of the 21st century, the standardization of many wireless communication technologies improved the interoperability and the reusability of the WSN devices, which are also called motes:

- IEEE [802.11] since 1997, i.e., Wireless Local Area Networks (WLANs),
- IEEE [802.15.1] since 2002, i.e., Wireless Personal Area Networks (WPANs),
- IEEE [802.15.4] since 2006, i.e., Low Rate WPANs (LR-WPANs), and
- IEEE [802.15.6] since 2011, i.e., Wireless Body Area Networks (WBANs)

In addition to further improvements in the energy efficiency and density of the sensing, computing, and communication devices, this standardization was the foundation for WSNs to evolve from a subject of scientific research to the base technology for consumer products with an overall market volume of about $1 billion in 2014 [Rawat2014].

(a) Nest [Thermostat] (b) BDI [STS4] (c) BIOTRONIK [BioMonitor]

Figure 1: Examples for wirelessly communicating consumer electronics, industrial products, and medical devices

Nowadays, the research efforts on WSNs are merged with embedded systems and the system of systems design methodologies, yielding Cyber Physical Systems (CPSs) [Mosterman2015]. This class of advanced applications requires the embedded devices to interact with their surroundings more intensively than typical WSNs, e.g., by supporting actuation and Human-Machine Interfaces (HMIs). CPS deployments are no longer isolated from one another to serve only a restricted application. Instead, open communication and interaction standards connect the ever growing number of CPS motes, thus contributing to the development of the Internet of Things (IoT) [Mainetti2011].

In 2014, the global IoT market ($238 billion) was dominated by industrial applications (24 %), automotive applications (22 %), consumer electronics (19 %), and healthcare (13 %) [TMR2015-IoT]. Although the IoT market is not restricted to WSN and CPS products, more and more wirelessly communicating embedded devices are contributing to the IoT domain. Some examples are listed in Figure 1.

Smart Home applications automate the illumination, heating, ventilation, air conditioning, and irrigation of apartments and buildings to minimize their running costs. After its acquisition by Google in 2014, Nest is one of the most popular vendors of Smart Home products. Its Learning [Thermostat], shown in Figure 1a, can be connected to other peripherals like cameras or automated light bulbs via WLAN. As the data rates required by these applications are rather small, IEEE [802.15.4]-based communication is well-suited for smart home applications. Nest thus allied with important home automation players such as Samsung, ARM, Osram, and Somfy to define a proprietary wireless network stack [Thread], which is optimized for Smart Home applications.

Industrial monitoring systems are typically sealed in a robust housing to withstand harsh environmental conditions, and are equipped with large batteries to support an unattended operation of several days. For example, the BDI [STS4] wireless structural testing system, shown in Figure 1b, consists of a Stellaris ARM Cortex-M3 microcontroller (MCU), a four channel Analog-to-Digital Converter (ADC) with 24 bit resolution, a IEEE [802.11] transceiver, and a 67 W h battery. It is used as a temporarily installed data acquisition system for the one-time assessment of the current state of a structure (e.g., bridges). The captured sensor data (up to 96 kbit/s) is transferred to the network operator without further preprocessing, while consuming about 1.5 W on average.

The BIOTRONIK [BioMonitor], shown in Figure 1c, is an implantable medical device used to monitor the heart signals of a patient. It samples three channels at several hundred hertz and automatically detects different types of arrhythmia. Only those critical observations are transferred wirelessly to a base station, where it can be accessed by the attending physician. Due to this in-sensor data preprocessing and feature extraction, the operational live time of the device reaches up to six years.

The [BioMonitor] is a typical example of how the small WSN devices achieve an operational endurance of several months or years on a limited energy storage. As the wireless transceiver typically is the major power consumer of a WSN mote [Jain2015], in-sensor data aggregation must be applied to reduce the required communication rates, and thus the active time of the transceivers. However, data aggregation such as lossless data compression or lossy feature extraction impose additional computational load on the WSN motes. Their compute demand is increased further by the complex encryption algorithms required to secure the communication channels, as privacy is becoming more and more important for all IoT applications.

The importance of wirelessly communicating embedded systems with energy-efficient processing capabilities is also reflected by public research budgets and expected market volumes. The European Commission provides an overall funding of € 1068 million in the Horizon 2020 Information and Communication Technology (ICT) program during 2016 and 2017, out of which € 90 million (8.4 %) are dedicated to Smart CPS, Smart System Integration, Smart Anything Everywhere, and customized and low-energy computing [H2020-ICT]. The overall Horizon 2020 IoT funding in the same period amounts to € 139 million [H2020-FA]. With this budget, the European Commission is strengthening the European industry to participate in a fast growing market. While the worldwide WSN market volume is estimated to exceed $3 billion in 2020 [FrostSullivan2015-WSN], the overall IoT market volume is forecast to $925 billion in 2021 [TMR2015-IoT]. Thus, the demand for wirelessly communicating embedded systems with energy-efficient processing capabilities will keep rising in the next years.

Software-programmable processors like MCUs and Digital Signal Processors (DSPs) are application-independent general purpose compute units. However, they achieve their flexibility at the cost of a reduced energy efficiency [Hameed2010]. In contrast, Application-Specific Integrated Circuits (ASICs) provide the most energy-efficient realization of a particular application. While all building blocks of a WSN processor such as the sensor controllers, digital filters and the radio protocol stacks can be implemented in dedicated hardware, the combination and configuration of these elements is highly application-specific. Therefore, building a complete WSN mote as customized silicon is not economically worthwhile in most scenarios.

As will be detailed in Section 2.2, Reconfigurable Compute Units (RCUs) such as Field Programmable Gate Arrays (FPGAs) provide the flexibility of software-programmable processors, while almost approaching the energy efficiency of ASICs. In the WSN context, FPGAs are well suited to accelerate dataflow-dominated and inherently parallel algorithms, such as multi-channel digital filtering of sensor data, but control flow-dominated algorithms and long-term low-intensity tasks, like time-keeping and the radio protocol, typically do not benefit from hardware acceleration.

1.2 Targeted Applications and Research Problems

In the last section, WSNs have been introduced as a key enabler for emerging technologies such as the IoT or CPS-controlled industrial monitoring, both of which will increase the demand for energy-efficient embedded processing. While the number of WSN installations is expected to grow rapidly, the concrete tasks to be solved by every single mote will vary greatly within the different application domains and deployment scenarios. RCUs have been introduced as processor architectures combining the efficiency of ASICs with the flexibility of software processors. It therefore appears to be promising to integrate reconfigurable computing into a WSN mote to create an energy-efficient platform that can be easily adopted to many different application scenarios.

The heterogeneous WSN mote developed in this thesis targets compute-intensive distributed applications with the potential for aggressive Dynamic Power Management (DPM). The characteristics of these applications can be broken down into the following requirements. First of all, a *sufficient amount of computation* has to be applied to the node-local raw sensor data, either by means of data aggregation, compression, or encryption. In principle, unless sensing random data, every application can benefit at least from lossless data compression. However, the additional complexity and static power drawn by a dedicated hardware accelerator will not pay off, if the in-sensor processing could be handled by a low-power MCU at a low duty cycle. The monitoring of slowly changing environmental parameters (e.g., temperature, moisture, or luminance in the agriculture domain) with sampling periods of several seconds or minutes, is thus not covered by this thesis. Second, a *fixed sampling period* allowing for DPM within each sampling cycle is required. For applications, that have to run as fast as possible (e.g., video processing with maximum frame rates), continuously running hardware accelerators are more suitable and DPM becomes moot. The same holds true, if the required sampling periods approach the transition times between the active and the low-power states of the computational devices. As these transitions typically require tens of microseconds, the targeted sampling frequency should not exceed 10 kHz to achieve duty cycles of 10 % and below.

When combining the often independent research areas of WSNs and RCUs under these application constraints, many new challenges and fundamental questions arise. The following research problems will thus be addressed by this thesis:

1. Which specific RCU is suited best to be integrated into a WSN mote?

2. How to integrate the RCU into the architecture of a WSN mote? To be more specific, how should the sensors, memories, processing and communication modules be arranged, and how should they communicate?

3. How does the RCU affect the DPM strategy of the WSN mote?

4. How does the RCU affect the trade-off between in-sensor preprocessing and wireless data transmission? For which kinds of application do the improved data aggregation capabilities pay off in terms of overall energy consumption?

5. Once the RCU is integrated as application-specific accelerator, which generic WSN services can also benefit from the improved processing capabilities?

Although a specific vibration-based Structural Health Monitoring (SHM) application is used for the system level evaluation, the key findings of this thesis are applicable to many WSN applications of similar complexity.

1.3 Thesis Contribution

This thesis offers solutions for the research problems listed in the previous section. The main contributions cover:

- A heterogeneous system architecture called Hardware-Accelerated Low Power Mote (HaLoMote), which integrates an RCU as a hardware accelerator into a MCU-based WSN mote.
- A HaLoMote implementation called Hardware-Accelerated Low Energy Wireless Embedded Sensor Node (HaLOEWEn), which is used as a proof-of-concept to evaluate the reliability and the efficiency of the heterogeneous system architecture.
- An application-independent communication framework designed to minimize the inter-processor communication overhead of the heterogeneous architecture.
- The hardware and software support for DPM with a special emphasis on fast shut-down and wake-up to benefit from sleep scheduling at the end of each sampling cycle, even at sampling rates of several hundred hertz.
- The hardware acceleration of application independent modules such as DSP primitives (Finite Impulse Response (FIR) and Fast Fourier Transformation (FFT)), a linear regression, as well as a lossless data compression kernel.
- A precise wireless time synchronization protocol, which reduces the computational efforts required to achieve the synchronization accuracy necessary for the targeted sampling rates.
- A contention-free transceiver scheduling for a minimum-effort n-to-all flooding of messages as required by the targeted SHM application.
- A hardware-accelerated processing kernel for the specific SHM application used for the system level evaluation of the HaLoMote architecture.
- A laboratory-scale demonstrator used for the system level evaluation of the HaLoMote architecture. It consists of a truss bridge model equipped with 20 accelerometers controlled by five HaLOEWEn motes, which capture and aggregate sensor signals appropriate to identify relevant modal parameters of the bridge.

Most of these contribution have already been successfully reviewed by the scientific community. The following publications were presented at international conferences in the WSN, RCU, and SHM domain:

[Engel2011] describes the basic HaLoMote architecture as a technology to trade-off node-local processing against date transmission in WSNs. To show its energy efficiency, the power consumption required by the HaLoMote for FIR computations is compared against commercial off-the-shelf (COTS) 8 bit and 16 bit MCUs and DSPs.

[Engel2012a] presents a concrete implementation of the HaLoMote architecture and a generic framework for the communication between its MCU and its RCU. Furthermore, different schemes for driving the RCU clock domain are proposed and their impact on the DPM are evaluated in the context of an SHM application.

[Engel2012b] describes a laboratory-scale demonstrator for a single HaLOEWEn mote and discusses its energy efficiency achieved by a decentralized hardware-accelerated FFT.

[Engel2014a] presents the hardware-accelerated implementation of a lossless data compression module. More specifically, an Adaptive Differential Pulse Code Modulation (ADPCM) encoder was developed and applied to sensor data gathered from a vibrating machinery and the neurons of primates. The benefits of the HaLoMote architecture over a software encoder could be shown.

[Engel2014b] describes a wireless routing protocol optimized for the n-to-all flooding required by the SHM application targeted in [Engel2012a]. Based on reachability- and interference graphs, an optimization problem is formulated to find a contention-free minimum-effort scheduling of sending and receiving messages. The solutions provided by a novel Integer Linear Program (ILP) model and a fast heuristic are compared with state-of-the-art flooding protocols.

[Engel2014c] presents an improved implementation of the hardware-accelerated SHM kernel described in [Engel2012a]. Furthermore, a laboratory-scale demonstrator for a network of HaLOEWEn motes is detailed. This demonstrator is used to evaluate the system level operation of the proposed heterogeneous architecture.

[Engel2015a] describes an improved formulation of the Rolling Linear Regression (RLR) used to optimize clock drift compensation within a high-precision wireless time synchronization protocol. While software-processors already benefit from the proposed algorithm, using a hardware accelerator further reduces the computational overhead of the synchronization protocol.

[Engel2015b] describes a laboratory-scale demonstration of the achievable synchronization accuracy with and without clock drift compensation, as well as the benefits of the hardware-accelerated RLR.

[Engel2015c] compares the measurement accuracy of the HaLoMote-based data acquisition system against a baseline captured by wire-bound laboratory equipment. Furthermore, the energy efficiency of the heterogeneous architecture is compared with the most recent software-processors.

The experiences gained with the HaLoMote architecture and its applications also resulted in additional publications not immediately related to the key topics of this thesis:

[Engel2014d] presents a hardware-accelerated controller for an ultrasonic piezo-electric actuator as the foundation of a haptic feedback system.

[Hochberger2014] presents a Coarse-Grained Reconfigurable Architecture (CGRA) implementation of the ADPCM algorithm described in [Engel2014c]. This paper focuses on aspects of high-level synthesis exploiting instruction-level parallelism.

1.4 Thesis Outline

This thesis is structured as follows.

Chapter 1 motivates the relevance of compute-intensive distributed applications in the context of WSNs. It formulates the research problems that arise when integrating RCUs as energy-efficient hardware accelerator into a WSN mote. Furthermore, it summarizes the main contributions of this research.

Chapter 2 summarizes the main concepts of WSNs and RCUs, as well as the terminology that will be used throughout this thesis.

Chapter 3 discusses related work in the fields of WSN hardware architectures and software services, as well as the class of distributed compute-intensive applications addressed by this thesis.

Chapter 4 presents the proposed heterogeneous WSN architecture HaLoMote and discusses the evolution of its implementation steered by technological developments and application requirements. Furthermore, the communication infrastructure between the compute units, as well as the integration of power management techniques is described.

Chapter 5 details the hardware acceleration of application-independent algorithms such as DSP primitives, lossless data compression, and a linear regression kernel. These modules are used as building blocks for different WSN applications described in Chapter 7.

Chapter 6 presents application-independent wireless communication concepts developed for the HaLoMote, such as a precise time synchronization protocol and a multi-source flooding scheme.

Chapter 7 details the HaLoMote-implementation of three different monitoring applications. The energy efficiency of the hardware accelerators previously described in Chapter 5 is evaluated in the context of these applications.

Chapter 8 gives the system level evaluation results for the targeted SHM application. After describing the laboratory-scale demonstration setup, the measurement accuracy and energy efficiency of the HaLoMote-based measurement system is compared with state-of-the-art wireless and wire-bound measurement systems.

Chapter 9 summarizes the major contributions of this thesis and proposes future research topics that could be addressed to further improve the heterogeneous hardware architecture.

CHAPTER 2
Technical Fundamentals

In this chapter, the main concepts of WSNs and RCUs are introduced to summarize the terminology used throughout this thesis.

2.1 Wireless Sensor Networks and Wireless Sensor Node Architectures

A WSN consists of a number of small embedded systems (termed *sensor nodes* or *motes*), which communicate wirelessly to solve a collaborative task. They are typically used as easily deployable *data acquisition systems* to *monitor* the temporal and spatial characteristics of ambient physical phenomena such as temperature, humidity, luminance, structural vibration, or acoustic pressure. As shown in Figure 2, the data gathered by the WSN has to be collected at a dedicated *gateway* node, where it can be stored persistently or forwarded to a remote *base station* over a Wide Area Network (WAN) such as a cellular mobile radio link. In addition to the monitoring of signals, the *tracking* of objects that can be detected by the motes or which the motes are attached to is a class of typical WSN applications [Rawat2014].

Also the requirements and capabilities of the motes vary strongly between different applications and even within a certain network, the architecture of a typical WSN mote consists of the five generic units shown in Figure 3. Analog or digital *sensors* (Figure 3a) capture the relevant signals at *sampling periods* ranging from several seconds in environmental monitoring [Lazarescu2013] down to several milliseconds in vibration-based structural health monitoring [Battista2013] or even below in acoustic localization applications [Astapov2014].

The sampled data is buffered in the *memory* module (Figure 3b), before it is further processed or transmitted. The processor-internal memory is typically limited to a few kilobytes, which is not sufficient to capture long-term measurements. Therefore,

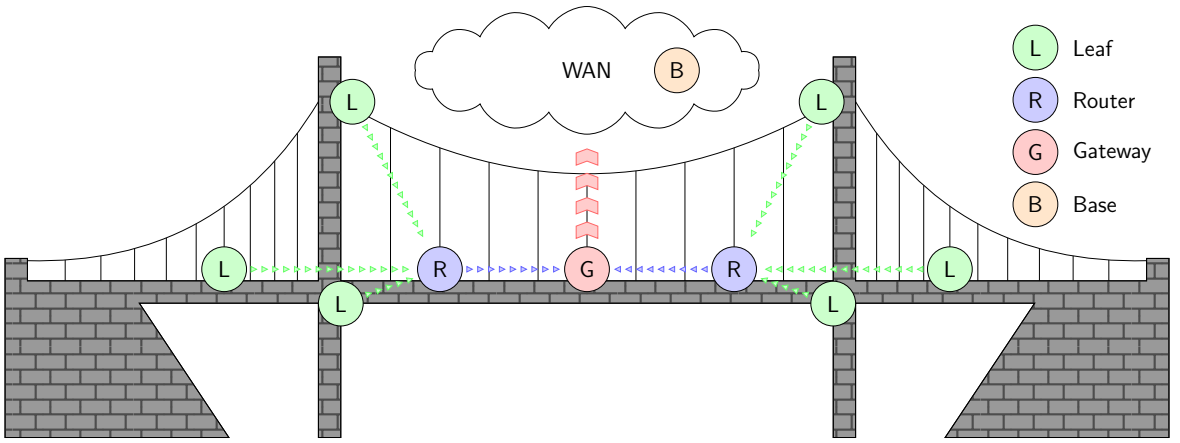


Figure 2: Typical monitoring application realized by a WSN, which forwards the captured data from the leaf nodes over intermediates routers and a dedicated gateway to a remote base station

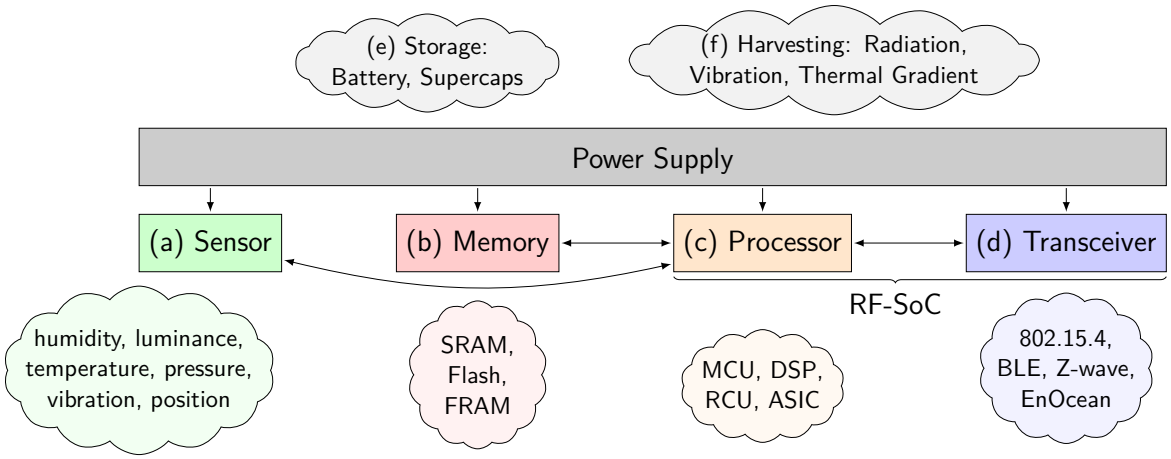


Figure 3: Generic architecture of a WSN mote

additional Static RAM (SRAM) modules or even non-volatile memory (NVM) such as Flash or the more recent Ferroelectric RAM (FRAM) are often integrated into the WSN motes [Yick2008; Zhao2015].

The *processor* (Figure 3c) is the heart of the mote that coordinates the sensors, the memory and the transceiver. The processor realizes the upper layers of the network communication protocol as shown in Figure 4 and performs other administrative tasks such as the time management. Most WSN motes are based on small MCUs such as the 8 bit Atmel [ATmega128] used by the [Mica2] or the 16 bit TI [MSP430] used by the [TelosB]. For more demanding computations like the online compression or encryption of the sampled data, 32 bit ARM MCUs like the Intel [PXA270] integrated in the [Imote2] or powerful DSPs such as the AD [BlackFin] are used as WSN processors [Ravinagarajan2010]. Beyond these typical software processors, the integration of RCUs and ASICs has been proposed by academic WSN projects such as [Shahzad2014; Walravens2014].

The wireless *transceiver* (Figure 3d) realizes the lower layers of the communication protocol stack, as shown in Figure 4. It determines the maximum length (i.e., the *communication range*) as well as the gross data *throughput* rate of each *hop*, i.e., the direct link between a sender and a receiver. Each hop can be either a *unicast* (from one transmitter to one explicitly addressed receiver) or a *broadcast* (from one transmitter to all reachable receivers). The carrier frequencies of the WSN transceivers are typically located in the freely available Industrial, Scientific and Medical (ISM) bands

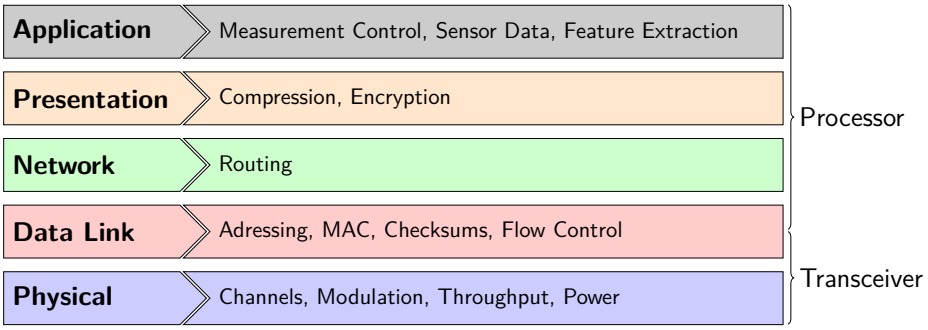


Figure 4: OSI layers used in a WSN communication protocol stack

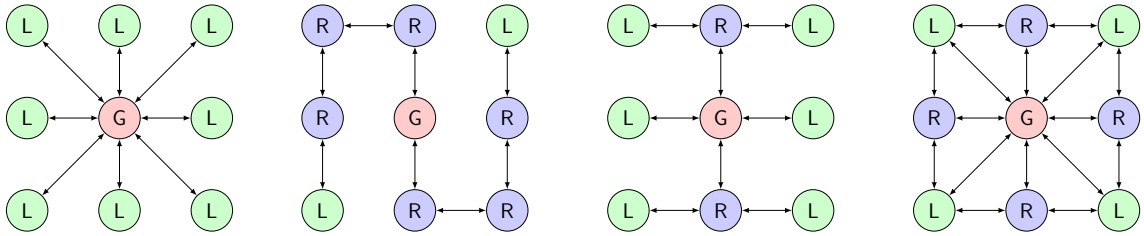
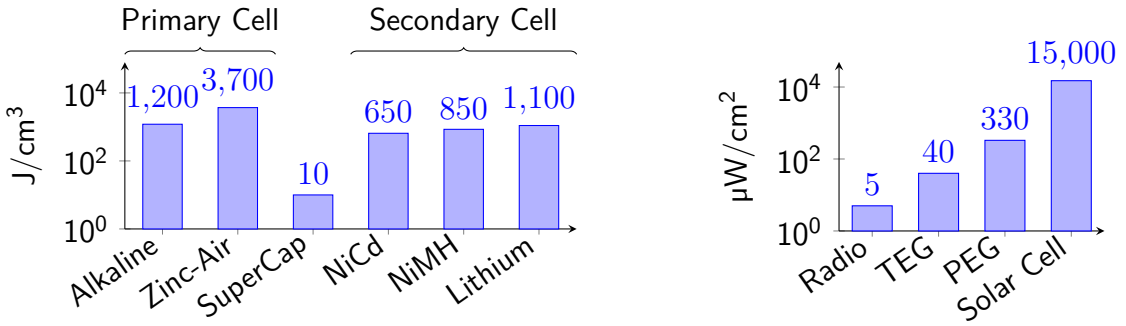


Figure 5: Classification of WSN topologies consisting of leaf nodes, routers, and gateways

at 2450 MHz (worldwide) and 915 MHz (America) or the Short Range Device (SRD) band at 868 MHz (Europe). The IEEE [802.15.4] standard is the foundation of the most popular WSN communication stacks such as [ZigBee], [WirelessHART] or the International Society of Automation (ISA) [100.11a]. It provides a throughput of 250 kbit/s at an outdoor communication range up to 100 m. Alternative transceiver standards include Bluetooth Low Energy (BLE) (1 Mbit/s at 200 m), Z-wave (40 kbit/s at 30 m), or EnOcean (125 kbit/s at 300 m) [Rawat2014]. If the WSN transceiver and processor are combined into one Integrated Circuit (IC), it is called a Radio Frequency System-on-Chip (RF-SoC).

As the communication range of the wireless transceivers is often smaller than the area covered by the WSN, the data from the *leaf nodes* most distant from the gateway has to be transferred over intermediate *routers*. The single-hop interconnectivity between the network nodes can be modeled as a directed graph, which is referred to as the WSN *topology*. If the covered area is smaller than twice the communication range, the *star* topology (Figure 5a) is most commonly used as it does not require intermediate routers. The *chain* (Figure 5b) and the *tree* (Figure 5c) topology can cover larger areas with multi-hop paths without complex routing logic, as the path between each pair of nodes is unique. In contrast, the routing within the *mesh* topology (Figure 5d) is rather complex, as the optimal path between two nodes has to be determined at runtime based on varying information such as the maximum end-to-end delay or the workload and the residual energy of each router. Only a mesh network can handle the failure of intermediate routers without loosing the connectivity between the remaining nodes. The mesh topology is thus preferable for large, dense, and dynamically changing networks, such as those consisting of mobile motes.

All modules of a WSN mote (i.e., sensor, memory, processor, and transceiver) are powered by a limited *energy storage* such as primary or secondary battery cells (Figure 3e). Their capacity is limited by physical constraints, as the motes are typically small to ensure tight integration into the target environment. Two Mignon cells (AA size) occupy about 16 cm^3 and may store up to 16 W h when realized as primary cells with Zinc-Air chemistry (see Figure 6a). A WSN mote with 1 mW average power consumption can be driven for more than two years from this buffer. To extend the maintenance-free system lifetime, ambient energy sources such as electro-magnetic radiation, mechanical motion or temperature gradients have to be tapped (Figure 3f). As shown in Figure 6b, the efficiency of different *energy harvesters* varies over multiple decades. While only 5 µW/cm^2 can be extracted inductively from the radio signals emitted by a 4 km distant 1 MW television broadcast tower [Parks2014], 40 µW/cm^2



(a) Energy storage density

(b) Energy harvesting productivity

Figure 6: Characterization of energy storage and harvesting technologies reported by [Watral2013; Stojcev2009; Parks2014]

can be harvested from a 10 K gradient using a Thermoelectric Generator (TEG). A Piezoelectric Generator (PEG) integrated in a shoe delivers about 330 µW/cm^2 while solar cells may generate up to 15 mW/cm^2 in a sunny environment [Stojcev2009].

As the power supply module determines the physical size of the whole WSN mote, the mote's overall energy consumption must be minimized to enable a compact design, while ensuring the operational life time required by the application. Many well known general *power management* techniques could be applied, but often turn out to be unsuitable in the WSN context. Dynamic Voltage and Frequency Scaling (DVFS) trades of the execution speed of the computational unit against its power consumption and requires appropriate hardware support. While most MCUs typically used on WSNs are able to slow down their clock frequencies by programmable pre-dividers, lowering the core supply voltage is rarely supported by MCUs (e.g., the TI [CC430]) that already aim to be low-power in the first place. Due to these limitations, a more common approach in the WSN field instead uses DPM, i.e., completely shutting down unused components such as the sensors, the transceiver, or (parts of) the processor. This is supported by MCUs providing multiple sleep modes, which trade-off wake-up time against power savings in the deeper states.

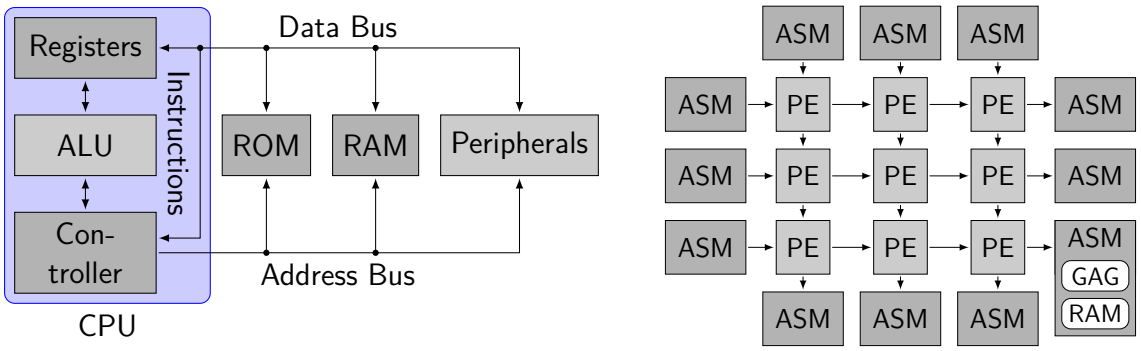
DPM must focus on minimizing the active time of the radio transceiver, as the transceiver draws tens of milliwatts [Akbari2014] and is thus usually the major power consumer of the sensor node [Jain2015]. To reduce the time spent idly listening, i.e., waiting for a message to be received, the in-network *time synchronization* must be sufficiently accurate to tightly synchronize the activities of senders and receivers. To reduce the time spent actually sending (or receiving) messages, the raw data volume generated by the sensors must be reduced before transmission. In many cases, the WSN operator often is not interested in the raw sensor data itself, but in certain features that can be derived from that data. This may be the long term average, the crossing of a threshold, the position of a tracked object, or the visual recognition of certain patterns. Even if the *feature extraction* relies on combining information from all motes, and can thus not be decentralized completely, more general approaches of *data aggregation* and prefiltering such as noise elimination or (lossless) data compression may be applied to reduce the overall communication demands.

For low data rate applications such as environmental monitoring, the powered-on times required for data aggregation (if any), and entering as well as leaving low-power modes, is negligible compared to the energy required for the actual radio transmissions. A COTS mote based on a low-power, but relatively slow 8 bit or 16 bit MCU is well suited to these use cases. For applications requiring higher sampling rates, such as vibration-based SHM, these weak MCUs generally are not able to perform the required preprocessing within the available sampling interval. On the other hand, more powerful 32 bit processors tend to consume more power than the radio transceiver (e.g., 570 mW for the [Imote2] processor [PXA270]) thus limiting the benefits of in-sensor preprocessing. Furthermore, entering and exiting a sub-milliwatt sleep mode takes too long for these more powerful devices (e.g., more than 135 ms for the [PXA270]) to perform DPM in every sampling interval. Active power management on these nodes thus can only occur when measurements are completely suspended.

2.2 Reconfigurable Computing

Computer architects have to trade-off flexibility and usability against performance when engineering a new compute unit. General purpose *software* processors implementing the von Neumann architecture [Neumann1945] consist of a Central Processing Unit (CPU) with attached memories and peripherals, as shown in Figure 7a. The CPU sequentially retrieves values stored in registers and applies them to an Arithmetic Logic Unit (ALU), thus executing basic mathematical operation (e.g, addition, multiplication, or comparison), or exchanges data between the registers and a component attached to the data bus. This can be a Random Access Memory (RAM), a Read-Only Memory (ROM), or another peripheral such as a timer, a coprocessor, or a communication module. The CPU controller determines the operations to be executed based on *instructions* loaded from the memory. In this way, the processor can be configured to solve any problem by storing an appropriate sequence of instructions into the program memory. The software processor thus represents the most flexible and easiest-to-use computer architecture.

This flexibility, however, comes at the expense of a significantly reduced efficiency in terms of energy consumption, execution speed and silicon area. [Hameed2010] reports a speedup of over 500× and a reduction in energy consumption by more than 700× when implementing (parts of) a video codec by an ASIC instead of a general



(a) Software (instruction stream) driven (b) Flowware (data stream) driven

Figure 7: Two basic principles of programmable processor architectures

purpose software processor. In this concrete case, only 5 % of the energy consumed by the software processor was actually spent for the arithmetic operations performing the computation. One reason for the lack of efficiency is the inherently *sequential* nature of the software processor, as only a single arithmetic operation can be performed at once. Most signal processing problems, however, can be partitioned into independent problems that can be solved in parallel. Therefore, processors *specialized* for certain application classes tend to include multiple functional units inside the ALU (e.g., DSPs), provide multi-word buses, registers, and ALUs (e.g., Single Instruction Multiple Data (SIMD) vector processors), allow for multiple parallel operations per instruction (e.g., Very Long Instruction Word (VLIW) processors), or even duplicate the entire CPU resulting in multi-core processors. While these architectures can exploit parallelism at instruction, data, or task level, the degree of parallelism is fixed at fabrication time and often does not match the requirements of a *specific* problem.

The main drawback of software processors is referred to as *von Neumann syndrome* [Hartenstein2007] and is caused by the instruction stream hitting the memory wall. Every operation, arithmetic as well as data transfer, is accompanied by an instruction fetch from a CPU-external memory. The loading of the address and data bus by instruction fetches can be avoided by adding a dedicated instruction bus (i.e., the Harvard architecture [Aiken1946]). However, the performance gap between processing and memory access speed amounts to about three decades [Patterson2012, Figure 2.2], and complex cache hierarchies are required to provide the instructions fast enough. On the other side, these caches increase the energy consumption and the required silicon area of the processor.

Instead of treating the effects of the von Neumann syndrome (i.e., providing instructions faster), its cause (i.e., required instruction fetches at runtime) should be eliminated. This can be achieved by switching from an instruction stream-driven software processor to a data stream-driven *flowware* processor architecture, as shown in Figure 7b [Hartenstein2007]. The latter consists of a (systolic) array of hardwired Processing Elements (PEs), each implementing a (arithmetic) function without (explicit) instruction fetches. Each PE is fed with the output of an Auto-Sequencing Memory (ASM) or the results of its neighboring PEs. The results generated by the systolic array are finally written back to a number of ASMs, which combine a distributed memory with a Generic Address Generator (GAG). The latter contains a data counter and a mapping to the appropriate memory addresses. For a given array of PEs, this mapping determines the overall computed program. Thus, the combined address sequence of all GAGs in the flowware concept corresponds to the instruction sequence of the software concept.

While these data stream processors do not suffer from the von Neumann syndrome, and provide inherently parallel arithmetic, they cannot be regarded as general purpose processors. They are best suited for certain compute-bound regular problems such as convolution, correlation, or matrix multiplications [Kung1982]. To allow for more flexibility, a second level of programmability must be introduced below the flowware level. Instead of using hardwired PEs and inter-PE connections, CGRAs provide mechanisms to reconfigure the behavior of each PE as well as the PE interconnection. This *reconfigurability* can be achieved by means of programmable switches, multiplexers, Lookup Tables (LUTs), or microcode memories (if the PEs themselves are implemented as tiny software processors). In any case, a number of bits has to be written to a dedicated

configuration memory inside the Programmable Logic Device (PLD). This *bitstream* is sometimes referred to as *configware* [Hartenstein2007].

CGRAs are only one possible realization of an RCU. One of the first conceptual descriptions of reconfigurable computing was proposed by [Estrin1960] as an

> *[...] inventory of high speed substructures and rules for interconnecting them such that the entire system may be temporarily distorted into a problem oriented special purpose computer.*

Depending on the granularity and the realization of these substructures, their density as well as the interconnection and routing complexity, different types of PLDs can be distinguished [Kesel2009, Section 3.7]. As shown in Figure 8, the Simple Programmable Logic Devices (SPLDs) realize (multiple) boolean functions in a two stage approach derived from their disjunctive formulation (i.e., sum of products). The Programmable Read-Only Memory (PROM) realization shown in Figure 8a provides reconfigurable switches only in the second stage (i.e., the OR matrix). Each output function is thus realized by its minterm canonical form, which potentially requires many closed switches in the OR matrix resulting in a high leakage current. To allow for logic minimization within each and between multiple output functions, a Programmable Logic Array (PLA) provides configurable switches in both stages, as shown in Figure 8b. The larger number of switches and inputs per logic gate increases the IO-delay and the required silicon of the PLA. As a countermeasure, the Programmable Array Logic (PAL) eliminates the programmability from the OR matrix as shown in Figure 8c, thus still allowing for some logic minimization without the full overhead of a PLA.

Recent PAL devices like the [TIBPAL22V10] support up to 22 inputs and 10 outputs, which are sufficient to realize small decoders or (parts of) state machines. Increasing the number of inputs to support more complex functions is not practical due to the rapidly growing complexity and delay of the switch matrix. Instead, multiple SPLDs are combined with some storage and feedback elements as well as advanced Input/Output (IO) drivers over a central interconnect to form a Complex Programmable Logic Device (CPLD). The largest device of the Xilinx [CoolRunner-II] family (i.e., XC2C512) provides 32 PLAs, each supporting 40 inputs and 16 outputs. CPLDs are thus well

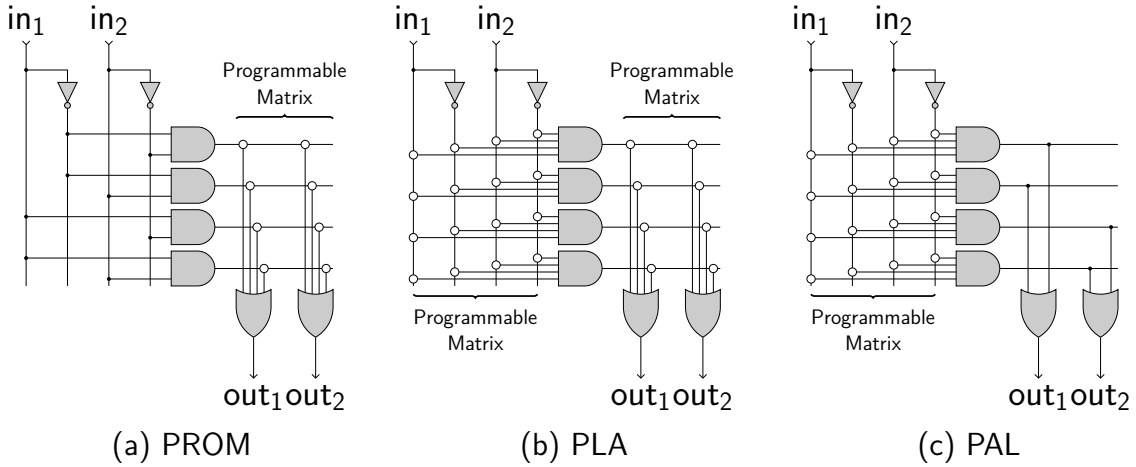


Figure 8: Simple Programmable Logic Devices realizing boolean functions

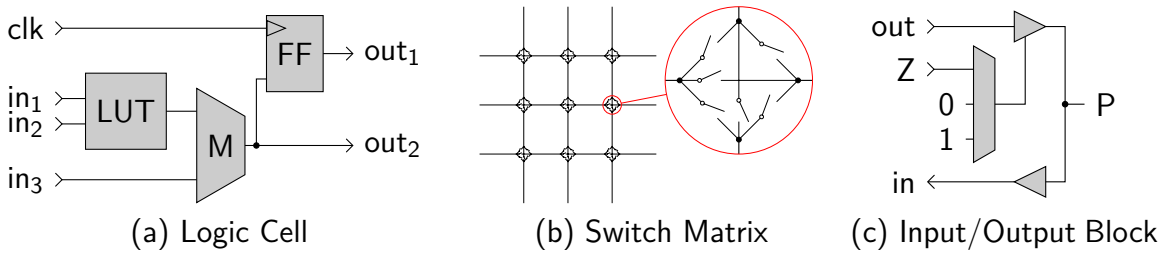


Figure 9: Configurable building blocks of an FPGA

suited to implement small cost-effective controllers for memories and other peripheral components [CPLDAPP].

While boolean functions with tens of inputs are the basic reconfigurable units of CPLDs, Field Programmable Gate Arrays (FPGAs) are *fine-grain* reconfigurable. Their smallest functional units are *Logic Cells (LCs)* including at least a small LUT, a clocked Flip-Flop (FF) and a programmable multiplexer, as shown in Figure 9a. A LC can thus either realize a simple boolean function or a one bit register. Some vendors add additional logic like full-adders and dedicated carry chains into their LCs to enable more efficient implementations of complex arithmetic operations. Furthermore, the LUT sometimes can be reused as shift-register or distributed RAM, if it is not used to realize a boolean function. The LCs are arranged in a regular array and interconnected by some *routing* resources, thus constituting the FPGA *fabric*, as shown in Figure 10. Those interconnects may span the whole FPGA (e.g., for distributing global clock and reset signals), or only a few LCs. Programmable Switch Matrices (SMs) at the junctions of these routing resources consist of six transistors (per junction), as shown in Figure 9b. Each lane is thus connectable to each subset of the three other lanes of the junction.

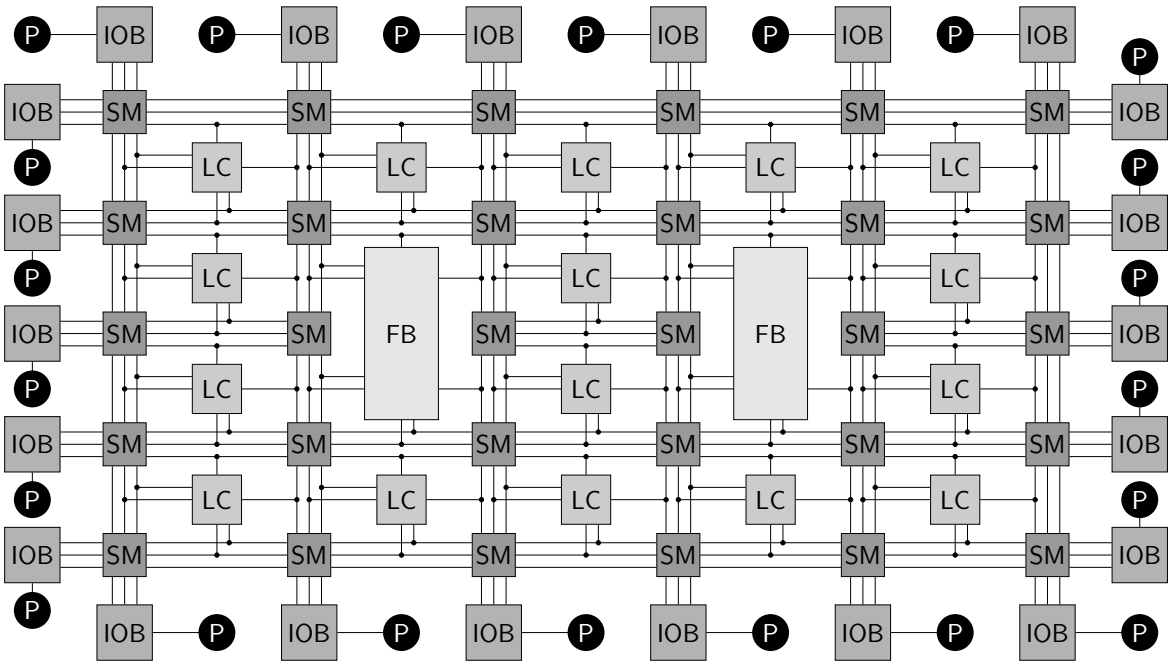


Figure 10: Generic architecture of an FPGA

Besides the LCs, a number of Input/Output Blocks (IOBs) is attached to the routing infrastructure to interface off-chip peripheral components. As shown in Figure 9c, each IO-Pad (P) can be used as input or output, or be driven to high-impedance (Z) to support the implementation of bus protocols. In practice, most IOBs can be configured for different IO standards (i.e., voltage levels, slew rates, differential signaling). Finally, hardwired Functional Blocks (FBs) are connected to the FPGA fabric to reduce the number of logic resources required for some common functionality. This includes Block RAMs (BRAMs), DSP operators, clock conditioning modules like Phase Locked Loops (PLLs), fast serial transceivers, or even complete software processors.

Just as no software engineer manually generates a binary instruction stream to implement a certain algorithm, the bitstream required for an FPGA design is also generated by a *toolchain* from a high-level representation of the algorithm. Figure 11 shows the tools required to transform the design through four different *levels of abstraction*. At the system level (Figure 11a), the design entry and *verification* processes do not differ significantly from a software engineering tool flow. The resulting C, SystemC or even [MATLAB] models are then converted to the Register Transfer Level (RTL) by High-Level Synthesis (HLS) tools such as the proprietary Xilinx [Vivado-HLS] or the open source [ROCCC] compiler. These tools can trade-off execution speed against the number of required logic resources for the resulting hardware by *constraints*, e.g., for loop unrolling, pipelining, operator folding, and the required operator-specific numeric precision at a granularity of 1 bit.

At the RTL shown in Figure 11b, the design is described by Hardware Description Languages (HDLs) such as Verilog or the Very High Speed Integrated Circuit Hardware Description Language (VHDL) in terms of concurrent and sequential operations on signal paths between synchronized registers. The major HDL concepts comprise the *structural* description of a module hierarchy as well as *behavioral* descriptions required to model a synchronized data flow. Besides generating HDL modules manually or by the HLS approach, Intellectual Property (IP) catalogs and core generators provide complete modules for many applications domains. To verify the design, a *testbench* is generated at the RTL. According to the Universal Verification Methodology [UVM], the testbench applies stimuli to the module under test and observes the resulting behavior.

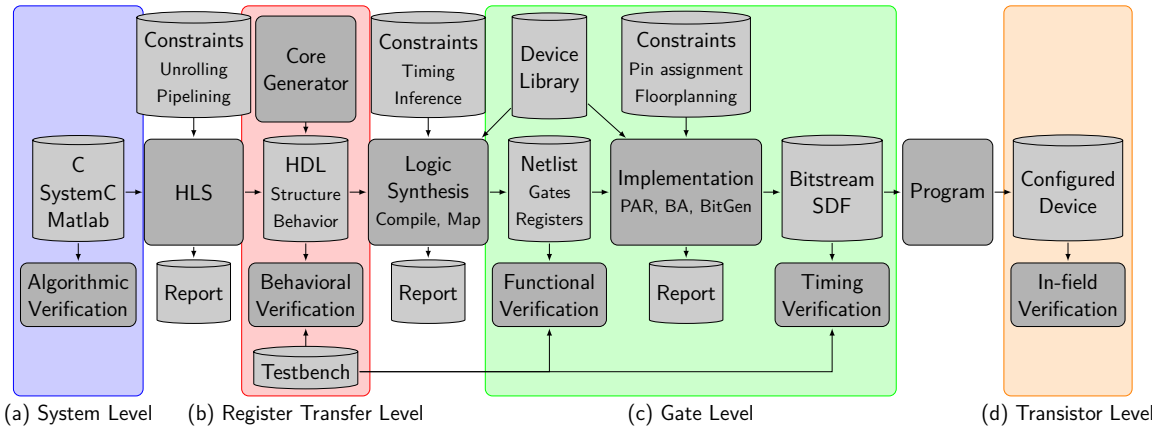


Figure 11: FPGA tool flow transforming resources between different abstraction levels

After the behavioral verification succeeded, the RTL model is transferred to the gate level (Figure 11c) by synthesizing a list of interconnected gates and registers called the *netlist*. Therefore, the synthesis tool first compiles the hardware description into a set of boolean functions. To implement these functions, the subsequent technology mapping selects the appropriate gate primitives directly realizable by the LCs or FBs of the targeted FPGA architecture. Constraints on the achievable clock frequency or the automatic inference of memories and DSP macros can steer this process. The functional verification of the gate level netlist can be driven by the same testbench used at the RTL.

Finally, the gates, registers, and macros of the netlist are assigned to dedicated LCs and FBs inside the target device and interconnected with dedicated routing resources. This Place and Route (PAR) process is controlled by user-defined pin assignment and floorplanning constraints to determine which signal to attach to which physical pin, or to limit the maximum delay between certain gates. After the PAR step, all *gate and wire delays* of the design can be estimated appropriately and are summarized in the Standard Delay Format (SDF), thus allowing for a timing verification after the Back Annotation (BA) of the netlist. Afterwards, the bitstream can be generated. It may also be compressed and encrypted. This bitstream is programmed into the target device for the final in-field verification at the transistor level (Figure 11d).

With decreasing abstraction level, the transformation tools report increasingly precise estimations of the required logic resources and achievable execution speed of the design. As the low-level tools require detailed knowledge about the targeted FPGA architecture, the synthesis, implementation, and programming steps are typically integrated into a vendor specific design environment such as the Xilinx [Vivado], Altera [Quartus], Microsemi [Libero-SoC], or Lattice [iCEcube]. In addition, those Integrated Development Environments (IDEs) typically provide the appropriate simulators, graphical constraint editors, and toolchains for the software-processors integrated into the devices.

Table 1 lists some implementation details of different FPGAs. Their performance can not be judged solely by the amount of LCs, as the number and size of LUTs and FFs per LC varies across different architectures. Furthermore, no common terminology has been established, so each vendor uses another designation for the basic LC and

Vendor	Xilinx		Altera	Lattice	Microsemi	
Market Share	52 %		34 %	6 %	3 %	
Architecture	[UltraScale]	[7-Series]	[HyperFlex]	[iCE40]	[IGLOO]	[Axcelerator]
CMem Type	SRAM	SRAM	SRAM	SRAM+NVM	Flash	Antifuse
Name for LC	Slice	Slice	ALM	LC	VersaTile	Cluster
LUTs per LC (in×out)	8 (6×1 or 5×2)	4 (6×1 or 5×2)	1 (7×1) or 2 (4×2)	1 (4×1)	1 (3×1)	2 (5×1)
FFs per LC	16	8	4	1	1	1
Largest Device	VU440	XC7V2000T	GX5500	HX8K	AGL1000	AX2000
LCs	316 620	305 400	1 867 680	7680	24 576	10 752
IOs	1456	1200	1640	206	300	684
BRAM	90 720 kbit	46 512 kbit	170 356 kbit	128 kbit	144 kbit	288 kbit
DSP Blocks	2880	2160	1980	-	-	-
PLLs	60	24	58	2	1	8

Table 1: Market share [TMR2015-FPGA] of leading FPGAs vendors and logic resources provided by some of their devices

super structures, such as the Xilinx Configurable Logic Block (CLB) clustering one to four slices with dedicated interconnects, or a Xilinx tile which contains a CLB and its surrounding routing logic. Although most datasheets specify some architecture independent measure to rate the available logic resources (e.g., equivalent LCs, LEs, or system gates), the usability of these metrics is quite limited as each vendor uses its own reference value.

Another important difference between the various FPGA architectures shown in Table 1 is the type of the *Configuration Memory (CMem)* used to store the configuration bitstream. SRAM is typically used for this purpose (76 % market share [TMR2015-FPGA]), as it can be written quickly and at the granularity of a single bit, thus allowing for (partial) dynamic reconfiguration at runtime. However, SRAM is volatile and the bitstream thus has to be provided by an FPGA-external entity to be programmed into the FPGA each time the FPGA is powered up. Due to dedicated memory controllers inside the FPGA, an additional off-chip NVM often is sufficient to provide the bitstream. Some architectures like the Lattice [iCE40] actually include this NVM into the FPGA die next to the CMem. The time required for the configuration process varies due to bitstream compression and optional encryption, but several 100 ms are realistic for the larger devices (e.g., of the Xilinx [7-Series]). Instead of using an additional NVM from which the bitstream is copied, the CMem itself can be realized as non-volatile flash (21 % market share) or antifuse (3 % market share) memory. The non-volatile NVM significantly increases the time and power required for the configuration process (up to several minutes). Once programmed before the deployment, the devices can start operating immediately after the power-up sequence. This feature is essential for mobile applications relying on DPM. In contrast to the flash memory, antifuses can only be programmed once by physically melting down isolation between the endpoints of connections. They are thus not appropriate for prototyping applications.

As shown in Table 1, the logic density of the available devices ranges from a few thousand to a few million LUTs and FFs. In combination with the different CMem technologies, a wide variety of applications can thus be targeted. In general, FPGAs are preferable, if the performance or efficiency (speed or energy consumption) of a software processor is not sufficient, and either the product volume is too small to justify the non-recurring engineering costs (NRE) of an ASIC design, or the implemented functionality is expected to significantly change throughout the product lifetime (e.g., for use cases such as prototyping boards, general purpose High-Performance Computing (HPC) engines, or communication/encryption/encoding modules updatable to revised standards). The most relevant market segments for FPGAs are telecommunication (33 %), industrial (21 %) as well as automotive (17 %) applications, and consumer electronics (12 %) [TMR2015-FPGA]. After Intel bought Altera in June 2015, FPGA technology can be expected to offer an alternative to CPUs and Graphics Processing Units (GPUs) in data center applications, with the aim to significantly reduce the energy consumed by server farms for the fast growing big data analytics and cloud computing markets [Putnam2015].

CHAPTER 3
Related Work

Before detailing the contributions of the thesis, this chapter summarizes relevant related research. The review covers the evolution of WSN mote hardware architectures, as well as specific WSN applications and services. The emphasis is placed on compute-intensive tasks and their hardware-accelerated implementation.

3.1 Wireless Sensor Network Motes

Driven by technological advances and application-specific optimization, a large variety of WSN motes have been developed as research prototypes in recent years. Some of these architectures have also become available as COTS devices. Table 2 details the processing and communication elements of some sample WSN motes. Focusing on hardware-accelerated node architectures, this review summarizes software-programmable motes only briefly. A more comprehensive overview can be found in [Lynch2006; Piedra2012].

This discussion is related to the architectural design considerations detailed in Chapter 4. It already has been partially published in [Engel2015c].

3.1.1 MCU-Based Motes

The probably most widely adopted WSN motes were developed at Berkeley. While the Mica mote [Hill2002] is based on an 8 bit AVR MCU, the Telos mote [Polastre2005] is driven by a 16 bit MSP430 processor. Both were redesigned to meet the latest LR-WPAN standard and are now available as COTS devices from MEMSIC as [MicaZ] and [TelosB].

The Libelium [Waspmote] provides the same processing capabilities as the Mica family, but its modularized radio subsystem can be adapted to Near Field Communication (NFC), LR-WPAN, WPAN, WLAN, or even cellular networking. The same flexibility is provided for the sensor subsystem, as more than ten sensor modules for different applications are available.

For more compute-intensive applications like digital image processing, the MEMSIC [Imote2] provides a 32 bit ARM processor. As the platform draws about 1 mW when sleeping and 100 mW when active, its application requires a sufficiently strong power supply.

From the huge variety of WSN prototypes proposed by different research groups, the following three examples stand out in the SHM domain. The SHiMmer mote [Dondi2010] utilizes a Blackfin DSP and is thus well suited for high performance digital filtering. It is actually used for ultrasonic SHM with 25 MHz sampling. The power drawn by this mote can be expected to be even larger than for the [Imote2], but details are not provided by [Dondi2010].

Another WSN mote specialized for SHM applications is proposed by [Araujo2012]. It employs two MCUs to separate the controlling of the sensor and the radio interface. In addition to this heterogeneous processing, the mote also uses heterogeneous

Table 2: Examples of WSN motes with focus on FPGA-based architectures

Reference (Name)	Origin	MCU / DSP	RCU	Transceiver
		based on software processors only		
[Waspmote]	Libelium	8 bit AVR		exchangeable modules
[MicaZ]	Berkeley	8 bit AVR		2.4 GHz IEEE [802.15.4]
[TelosB; TMoteSky]	Berkeley	16 bit MSP430		2.4 GHz IEEE [802.15.4]
[Imote2]	MEMSIC	32 bit ARM5		2.4 GHz IEEE [802.15.4]
[Dondi2010] (SHiMmer)	San Diego	32 bit Blackfin		915 MHz IEEE [802.15.4]
[Araujo2012]	Madrid	32 bit PIC + ARM9		2.4 GHz IEEE [802.15.4] + [802.11]
[Zhao2015] (NFC-WISP)	Washington	16 bit MSP430		13.56 MHz RFID
		based on ASICs		
[Asada1998] (WINS)	Los Angeles	Spectrum Analyzer		915 MHz custom design
[Walravens2014]	Leuven	parallel prefix computations		none
		based on SRAM-FPGAs		
[Kohvakka2006]	Tampere	32 bit Nios (Softcore)	Cyclone	2.4 GHz
[Raval2010]	Dublin	8 bit AVR	Spartan	2.4 GHz IEEE [802.15.4]
[Voyles2010] (RecoNode)	Denver		Virtex 4	2.4 GHz IEEE [802.15.4] + [802.15.1]
[Mplemenos2012]	Crete	8 bit AVR	Virtex 5	2.4 GHz IEEE [802.15.4]
[Lombardo2012] (HiReCookie)	Madrid	8 bit AVR	Spartan 6	2.4 GHz IEEE [802.15.4]
[Goh2012]	Singapore	16 bit MSP430	Spartan 3	2.4 GHz IEEE [802.15.4]
[Yi2013]	Chicago		Zynq 7020	2.4 GHz IEEE [802.15.1]
[Shahzad2014]	Sundsvall	32 bit AVR	Spartan 6	2.4 GHz IEEE [802.15.4]
		based on NVM-FPGAs		
[Vera-Salas2010]	Querétaro		IGLOOnano	2.4 GHz IEEE [802.15.4]
[Berder2010] (PowWow)	Rennes	16 bit MSP430	IGLOO	2.4 GHz IEEE [802.15.4]
[Rosello2011] (Cookie)	Madrid	16 bit MSP430	IGLOO	2.4 GHz IEEE [802.15.4] + Wakeup
[Grassi2012]	Milan	8 bit 8051 (Softcore)	IGLOO	2.4 GHz Xbee Digimesh
[Nylaenden2014]	Oulo		IGLOO	2.4 GHz IEEE [802.15.4]

communication. A secondary LR-WPAN transceiver is dedicated to wireless time synchronization, while raw sensor data is transferred by the faster (but mote power hungry) WLAN transceiver. Again, no power consumption details are reported by [Araujo2012].

The NFC-Wireless Identification and Sensing Platform (WISP), proposed by [Zhao2015] for cold chain monitoring, is located on the other side of the power and performance spectrum. It is powered by and communicating via an Radio-frequency Identification (RFID) circuit and requires 369 nJ per temperature sample.

3.1.2 ASIC-Based Motes

At the beginning of the WSN research, no discrete sensing, processing and communication ICs were available. Early devices like the WINS architecture [Asada1998] were thus completely designed as ASICs.

Nowadays, ASICs are integrated as specialized hardware accelerators to speed-up a small, yet critical part of the DSP chain in a WSN processor. [Walravens2014] proposed an ASIC for parallel prefix computations, that can be used, e.g., for searching elements, detecting peaks, or evaluating polynomials.

Due to their lack of flexibility and high NRE costs, ASICs are not a viable option for a research prototype, and are thus not reviewed here in greater detail.

3.1.3 RCU-Based Motes

As stated in Section 2.2, RCUs can perform complex computations more efficiently than MCUs and DSPs. In real-time applications, the use of RCUs often enables computations that cannot be performed by MCUs or DSPs at all under the given constraints. Table 2 distinguishes between FPGAs based on volatile and non-volatile configuration storage, as these device classes seriously differ in terms of computing power and energy consumption.

SRAM-based FPGAs are attractive for use in sensor nodes performing compute-intensive applications, such as video and image compression and analysis. While the RecoNode [Voyles2010] utilizes the visual information in search and rescue applications, [Shahzad2014] describes a visual particle detection mechanism for a condition monitoring system. Another hardware accelerator used for chemical process monitoring is proposed by [Goh2012]. Other researchers employ the FPGA to accelerate application-independent WSN tasks like the sensor controllers [Yi2013], the routing protocol [Mplemenos2012], or the entire radio interface including fast data forwarding in a multi-transceiver architecture [Kohvakka2006].

However, the energy efficiency of the proposed architectures is rarely addressed by these authors. [Raval2010] reports an 48 % energy reduction when accelerating an 8 bit AVR softcore processor by instruction fusing based on application-specific instruction tracing. A direct comparison against a discrete (i.e., hardwired) processor is still missing. The main problem of the SRAM-based FPGA hardware accelerators, i.e., the time required to load the configuration again after each low-power cycle, is quantified by [Shahzad2014]. The authors state that shutting down and reconfiguring a Spartan 6 design instead of just entering the suspend mode is worthwhile only after a sleep period of at least 235 ms. DPM within each sampling cycle is thus not reasonable for this device in compute-intensive applications with sampling frequencies of several hundred hertz. To actually reduce the energy overhead of the reconfiguration process of SRAM-based

FPGAs, [Lombardo2012] proposed to manually relocate logic resources such that the bitstream is more compressible. For a specific application, a 2.4× reduction of the reconfiguration overhead was reported.

Nevertheless, when energy actually becomes a first-class design goal, non-volatile FPGA accelerators are far more suitable. Sensor nodes using a combination of a Flash-based Microsemi IGLOO FPGA and a wireless transceiver have been proposed [Vera-Salas2010; Stelte2010; Nylaenden2014]. However, despite the power advantages of Flash configuration storage, these implementations have turned out to be sub-optimal: All processing is performed on the RCU (even long-term low-intensity tasks), and when powered down, the radio transceiver is required to wake up the FPGA again. Thus, at least one of the two power-hungry devices, either the FPGA or the transceiver, has to be enabled all the time. Even refinements which use very simple timekeeping on the RCU, such as employing inverter ring-based oscillators [Grassi2012] to power up the receiver periodically at pre-agreed times for data reception, are still sub-optimal: Due to the large timing inaccuracy (drift) of these oscillators, the power-down phases have to be conservatively shortened, leading to the system drawing higher power for longer intervals.

A better choice is a heterogeneous architecture *combining* an RCU and a low-power MCU. The Cookie WSN [Rosello2011] and the PowWow [Berder2010] have joined a small Microsemi IGLOO FPGA with a TI MSP430 MCU and an additional radio transceiver. However, both systems utilize the FPGA only for low-level handling of radio messages, instead of preprocessing the sensor data stream. Furthermore, the use of discrete MCU and radio components carries the burden of slower communication between processor and transceiver, as well as a more complex power management. In Section 4.1, the usage of an RF-SoC next to a Flash-based FPGA is examined in greater detail.

3.2 Common Wireless Sensor Network Services

In this section, existing research related to the modules and services presented in Chapter 5 and 6 are summarized. Parts of this review have already been published in [Engel2014a; Engel2014b; Engel2015a].

3.2.1 Lossless Data Compression

While generic data compression is applicable to almost all kinds of sensor data, more specialized methods such as compressive sensing [Donoho2006] assume highly specific properties in the input signals and are not addressed in this work.

Data compression in WSNs was investigated frequently in the last decade [Kimura2005]. For example, an LZW-compressor was implemented on an [MSP430] MCU to analyze the effect of reduced data rates on the end-to-end packet delay in a multi-hop network [Deng2012]. The energy savings achievable by a nonlinear ADPCM running on the ARM processor of a Beagle Board were investigated in [Kasirajan2012]. However, these authors erroneously considered the achieved compression ratios *directly* as energy savings, completely ignoring the energy required for the *encoding* itself. This gross simplification was not used in [Reinhardt2009], where run-length and adaptive Huffman encoding were implemented on the AVR MCU of a [Mica2] mote. The energy for encoding was determined solely by simulations and datasheet-specifications, using just synthetic data streams with guaranteed statistical properties as inputs.

Hardware-accelerated data compression in context of WSNs focused mainly on lossy image compression in visual surveillance networks, such as the JPEG compression on an Altera EP2C35 FPGA [Zhiyong2009], or the identification of relevant image sections using a Xilinx Virtex II FPGA [Ngau2012]. In addition to not compressing losslessly, these investigations aimed at reducing the required data rate to the throughput limits of the wireless transmission channel, instead of minimizing the overall system energy consumption. The acceleration of a second order ADPCM compressor on a Xilinx XC4000 device was proposed in [Boonyakitmaitree2004], but did not report any energy requirements.

3.2.2 Network Flooding

Congestion-induced packet loss leads to throughput degradation in WSNs and often requires energetically expensive retransmissions. Though collisions can be avoided by Carrier Sense Multiple Access (CSMA) mechanisms, the required carrier-sensing also increases the energy consumption of the sensor nodes. Therefore, routing protocols have a significant impact on distributed applications relying on multi-hop wireless communication. The number of nodes involved to deliver information (a data item) from a source to a sink influences the overall energy consumption of the network, as well as the data throughput. Often, shortest routes cannot be easily discovered, or are undesirable (e.g., when balancing communication load equally over all network nodes, or to provide robust communication over redundant paths). In general, application-tailored routing protocols can exploit special characteristics to improve throughput and energy consumption.

In Section 6.1, a multi-source flooding mechanism is proposed to distribute information from at least two sources to all network nodes at the same time. Flooding information into a WSN has been widely adopted for general tasks such as time synchronization [Maroti2004; Gheorghe2010; Xu2009] or route discovery [Perkins1999]. However, basic blind flooding is quite inefficient, especially in dense networks, as each receiver rebroadcasts the message to be distributed [Ni1999]. Intensive research has been carried out to reduce the number of necessary rebroadcasts. Prior work commonly utilizes knowledge about the nodes locations [Arango2004; Jeong2010] or their local neighborhood [Sheng2005; Lim2001; Lou2002; Agathos2011] to select the forwarding nodes covering the most yet uncovered network nodes. However, all of these flooding protocols assume a 1-to-all data propagation. Unfortunately, unlike physical water or acoustic waves, multiple data waves originating from different source nodes interfere with each other due to channel congestion and signal interferences, such as the hidden terminal problem [Wang2012].

Furthermore, the information from multiple sources should be aggregated as soon as possible to avoid repeated data transfers between the same nodes. Many data aggregation and clustering protocols have been reported [Alnuaimi2013]. By selecting specific nodes as cluster-heads, data is collected locally before it is forwarded to a single sink node. The basic idea of clustering is picked up in the design of the greedy heuristic in Section 6.1.3.

As described in Section 6.1, network routing can be considered as finding a proper transceiver schedule, such that information travels along the links between simultaneously scheduled transmitters and receivers. Using ILPs to solve scheduling problems is a widely adopted approach. In the WSN context, ILPs were used, e.g., for fre-

quency channel assignment [Ahmed2010], task scheduling [DePauw2010], and routing [Hoa2012]. The later one is closely related to the mechanism proposed in Section 6.1, as it generates the ILP formulation based on a directed graph representing the network topology. The ILP solution proposed in [Hoa2012] describes a multi-cycle schedule for transmissions and receptions with a minimal amount of overall required energy. However, as the schedule is used to transport information from multiple source nodes just to a *single* sink, it does not perform flooding. Furthermore, it does not provide information aggregation, and does not consider the transmission ranges and interference between the radio transceivers. All of these aspects are considered in Section 6.1.

3.2.3 Wireless Time Synchronization

Wireless time synchronization aims to compensate the clock deviation between all sensor nodes, which are caused by the different start-up times and the slightly different frequencies of the sensor node's oscillators. Energy-efficient synchronization protocols reduce the communication overhead for synchronization-related timestamp exchanges by estimating the clock drift between different nodes at runtime.

A large variety of wireless time synchronization protocols has been proposed in the last decade [Djenouri2014], with only a few of the employed clock drift estimators not being based on a least squares linear regression. The R^4Syn protocol [Djenouri2012] uses a Maximum-Likelihood-Estimator, which imposes nearly the same computational complexity as the least squares method. In contrast, the Kalman filter used in [Aoun2008] can be executed 10 % faster than a linear regression with a table size of two. However, for synchronization periods of up to 10 s, the average synchronization error of the Kalman filter proved to be larger than for the linear regression-based clock drift estimator.

The Flooding Time Synchronization Protocol (FTSP) [Maroti2004] has become one of the most popular reference implementations for high precision synchronization protocols. It performs the clock drift compensation with a linear regression over the last eight synchronization points, but without justifying the selected regression table size. Furthermore, no details about the regression implementation or its resource requirements are provided. At a 30 s synchronization period, FTSP achieves an average clock jitter of 1.5 µs.

Several improvements of the FTSP have been suggested. In [Castillo-Secilla2013], a temperature-dependent correction factor is introduced to respond faster to temperature-induced clock drift variations. The authors report an average accuracy of 1.1 µs after a 20 K temperature variation at a single node. The Recursive Time Synchronization Protocol (RTSP) [Akhlaq2013] is also based on FTSP, but it uses automatic Medium Access Control (MAC) timestamping, i.e., the timestamps corresponding to the transmission or reception of the IEEE [802.15.4] start of frame delimiter (SFD) are captured by the radio transceiver without software intervention. Furthermore, RTSP dynamically adjusts the synchronization period depending on the current clock drift and accuracy at each node. The average accuracy was improved to 0.3 µs, even when restricting the regression table to two entries to simplify the computational effort for the clock drift estimation.

3.3 Compute-Intensive Wireless Sensor Network Applications

This section provides an overview of WSN applications targeted by this thesis. It is thus related to the use cases described in Chapter 7. Parts of this review have already been published in [Engel2015c]. A more comprehensive and general overview of WSN applications can be found in [Yick2008; Rawat2014].

As outlined in Section 1.2, the applications targeted by this thesis have to provide sufficient potential for node-local computation while running at fixed sampling rates of a few hundred hertz. The demand for in-sensor computation can arise from application-independent services such as encryption, which are not addressed by this review. Instead, the focus here lies on application-specific feature extraction from different application domains.

3.3.1 Condition Monitoring

Condition monitoring systems observe relevant runtime parameters of industrial processes and machinery for early fault detection and predictive maintenance. Most applications observing acoustic signals or structural vibrations require sampling rates of several thousand hertz, such as the 50 kHz bearing fault monitoring described in [Nylaenden2014]. The authors propose to either calculate a Root Mean Square (RMS) value in the time domain, or to detect and classify peaks in the frequency spectrum. For the latter, a hardware-accelerated 256 point radix-2 FFT is applied for the time-to-frequency domain transformation. A similar approach is reported by [Shahzad2014] for the 100 kHz monitoring of a rotating machinery.

Both systems far exceed the range of sampling rates targeted in this work. In contrast, the monitoring of induction motors based on current flow measurements are less demanding, as the relevant physical phenomena to be observed are located close to the power line frequency (e.g., 50 Hz). [Philipp2012] used a Zoom-FFT to detect peaks slightly off from the power line frequency to detect broken bars inside an induction motor. To capture other faults such as dynamic eccentricity [Philipp2012] or inter-turn short circuits [Martinez2013], sampling rates of up to 10 kHz are still required.

3.3.2 Structural Health Monitoring

SHM systems observe global dynamic properties (e.g., eigenfrequencies) of large infrastructure objects, such as bridges, poles, or buildings. Compared to the condition monitoring applications, the observed structures are larger and the relevant (sampling) frequencies thus considerably smaller (below 1 kHz). SHM applications thus better fit the DPM requirements of the heterogeneous WSN mote proposed in this thesis.

WSNs have become popular for SHM in the last decades [Lynch2006]. Several research groups have equipped a number of bridge structures with wireless sensor networks, ranging from small models with artificial excitation [Bocca2011], over medium-size pedestrian bridges [Battista2013], up to large automotive traffic bridges [Chae2012; Hu2013; Kim2007; Cho2010; Pakzad2008].

[Kim2007; Pakzad2008] installed 64 [MicaZ] motes (see Table 2) on the 1280 m main span of the Golden Gate bridge in the San Francisco bay. Each mote was equipped with two high precision and two wide range micro-electro-mechanical (MEMS) accelerometers to capture different excitation scenarios (e.g., traffic, wind, and earthquakes). The

nodes were synchronized with the FTSP [Maroti2004] resulting in a maximum jitter of 10 µs in the 46-hop network. The sensor data captured at 1 kHz are down-sampled to 50 Hz by averaging, and logged to an external flash memory afterwards. The authors report an average power consumption of 358 mW for this sensor sampling. After about 22 min measurement, the raw sensor data is collected at a central gateway, which takes more than 12 h for all 64 network nodes. Finally, an offline system identification is carried out and the resulting eigenfrequencies and mode shapes are compared to theoretical models of the bridge. The first vertical bending and torsional modes were detected with a relative error of 10 % and 19 % respectively.

[Bocca2011] instrumented a 4.2 m long wooden bridge model with 8 [MSP430]-based WSN motes, each equipped with a 3-axis MEMS accelerometer with a resolution of 16 bit and a dynamic range of ±2 g. A clock skew-compensating synchronization protocol was used to keep the temporal jitter between the nodes below 5 µs. While exciting the bridge model with random noise generated by an electro-dynamic shaker, the acceleration sensors captured the movement of the structure at 200 Hz sampling rate. During each 60 s measurement period, the acquired samples are buffered in an external flash memory for later readout without any preprocessing or data aggregation. The authors report problems with lost samples (due to the narrow write throughput of the external memory) and an overall energy consumption of 3.2 J for data sampling, buffering and final transmission. This equates to an average power consumption of 53 mW during the 60 s measurement periods. Finally, the gathered data is used for an offline system identification. Compared to a wire-bound reference system, the first 14 structural modes were identified with a maximum error of 1.035 %.

[Chae2012] installed 45 motes based on 8 bit AVR MCUs on a 550 m suspension bridge. Each mote was equipped with two different accelerometers, i.e., a high precision force-balance sensor for capturing the truss vibration and a low-power MEMS sensor for capturing cable tension. Both sensors were sampled at 60 Hz and the resulting data was directly transmitted to a base station without any data aggregation. The authors reported packet loss problems caused by high data throughput required for transmitting the unprocessed data. Finally, an offline system identification was carried out, but the comparison with a wire-bound reference system consisting of 360 sensors failed due to broken links caused by cable fatigue.

[Hu2013] instrumented a 296 m highway bridge with 6 [MSP430]-based WSN motes. Each mote was equipped with a high resolution (2 mg) MEMS accelerometer sampled at 100 Hz. The network uses an energy-balanced time synchronization, which claims to achieve a maximum jitter of 10 µs. Again, the sampled data was collected in an external memory for each 250 s measurement period, and than transmitted to a central base station without preprocessing. The authors report an average power draw of 290 mW during sampling and data transmission. After an offline-analysis of the gathered data based on multivariate autoregressive models, the detected eigenfrequencies were compared to theoretical models of the bridge. Between 4 % and 18 % difference between the detected and the expected eigenfrequencies was reported.

Except some simple down-sampling, none of the aforementioned monitoring systems is based on compute-intensive in-sensor data aggregation. In contrast, the Illinois SHM project [Cho2010; Battista2013] is based on the powerful [Imote2] system (see Table 2) for complex in-sensor computations. [Cho2010] monitored a 484 m cable-stayed span bridge with 70 nodes, each equipped with MEMS acceleration sensors.

A complete decentralized Operational Modal Analysis (OMA) is performed based on Frequency Domain Decomposition and Stochastic Subspace Identification. However, also measuring only 5000 samples per day with a sampling rate of 10 Hz, the power-hungry ARM processor depleted a 21 A h battery in less than two months.

Based on the same WSN platform, [Battista2013] monitored a 120 m pedestrian bridge with 8 nodes, each sampling a MEMS accelerometer at 100 Hz. The sampled data is analyzed immediately based on a Filtered Hilbert-Huang Transformation, which itself is a combination of modal separation, bandpass filtering, and the Random Decrement Technique (RDT) described in Section 7.3.1. The authors report an overall data reduction of 96 %. At an operational duty cycle of 33 % (i.e., 10 min measurement every 30 min, a 15.6 A h-Li-Ion battery depleted after 14 days. This equates to an average power draw of 172 mW from the 3.7 V supply.

These research projects show that SHM applications typically require sampling rates of several hundred hertz, as targeted by this thesis. Furthermore, algorithms for effective in-sensor data aggregation exist, but they have been rarely used so far - probably due to the required computational power of the WSN mote.

CHAPTER 4

HaLoMote: Hardware-Accelerated Low Power Mote

This chapter details the architecture and implementation of the developed heterogeneous sensor node, the communication infrastructure connecting the computational units, as well as the power management capabilities of the platform. Parts of this chapter have already been published in [Engel2012a].

4.1 Architecture Overview

In Section 2.2, the trade-off between the flexibility of general purpose software processors and the energy- and runtime-efficiency of application-specific processing architectures was described. Providing application-specific hardware accelerators for the compute-intensive distributed data aggregation algorithms described in Section 3.3 should thus improve the computation vs. communication trade-off, and reduce the energy consumption of the overall WSN mote. Apart from a few high volume WSN applications such as smart homes, ASIC-based hardware accelerators are not viable for COTS motes that have to be adaptable to a specific application at the time of deployment or even at runtime. Therefore, this work examines the use of RCUs as hardware accelerators for WSNs.

To integrate an RCU into a WSN mote, the design options for the computation and communication architecture shown in Figure 12 have to be traded-off against each other. As an RCU typically does not provide wireless communication capabilities, at least a radio transceiver must be attached (Figure 12a). These transceivers implement the physical layer (e.g., carrier frequency, symbol modulation) and part of the MAC layer (e.g., channel sensing, address filtering, error checking), and provide a limited message buffer that can be accessed by a digital interface. All other tasks of the radio protocols (e.g., multi-hop routing, transport control) must thus be handled by the RCU. As these are typically rather simple and sequential control flow-dominated algorithms, waking up the RCU just to handle the radio protocol would not be energy efficient. For the same reason, handling the radio stack on a softcore MCU inside the RCU (Figure 12b) should be avoided.

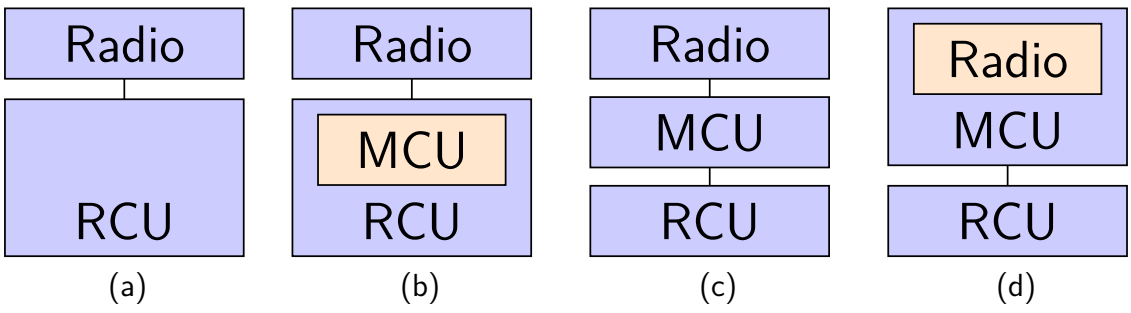


Figure 12: Design options for the computation and communication architecture

When adding a dedicated MCU (Figure 12c) for all low priority tasks such as the radio protocol, basic timekeeping, and the top-level control flow of the application (e.g., periodic sensor sampling), the platform can exploit its heterogeneity by selectively shutting down temporarily unused computation and communication units. This basic DPM principle is also applicable when the radio transceiver and the low-power MCU are integrated into a single device (Figure 12d), as these RF-SoCs allow to suspend either of both units separately. Compared to Figure 12c, the combination of an RCU and an RF-SoC improves the data throughput between the hardware accelerator and the wireless transceiver, as more of the limited number of MCU GPIO pins can be dedicated for the inter-processor communication instead of being required for the communication between the MCU and the transceiver. Therefore, the heterogeneous sensor node developed in this thesis is based on the architecture shown in Figure 12d.

A second fundamental architectural design choice deals with the attachment of digital sensors (or analog sensors with a downstream ADC) and additional memory, often required to temporarily buffer raw or aggregated sensor data. Attaching both peripherals to the RF-SoC (Figure 13a) allows to collect the number of samples required for the data aggregation algorithm without ever waking up the hardware accelerator. However, the entire stream of sensor data has to be transferred over the critical link from the RF-SoC to the RCU to be aggregated. As the throughput between the processing units is further limited by the RF-SoC requiring General Purpose Input/Output (GPIO) pins to interface the sensors and the memory, the transfer of the raw data stream becomes the bottleneck of the architecture.

This bottleneck can be mitigated by a *shared memory* used for the inter-processor communication (Figure 13b and 13c), either by utilizing a dual-port memory, or by synchronizing the memory access of both processors via the remaining direct connections between RCU and RF-SoC. While dual-port memory is typically more expensive than a single-port memory (in terms of chip cost and energy consumption), it enables more parallel operations on the heterogeneous platform such as transmitting one block of the aggregated data while generating the next block. The shared memory approach is most valuable if the sensors are attached to the RF-SoC (Figure 13b) and the data aggregation can be performed on larger blocks of the raw sample data, so the RCU can be kept sleeping until the next block is collected.

However, the sensors may have to be attached to the RCU (Figure 13c and 13d) if the RF-SoC does not have enough GPIO pins to control all peripherals, e.g., if the sensors can not be daisy-chained, or connected to a common bus. Furthermore, many event

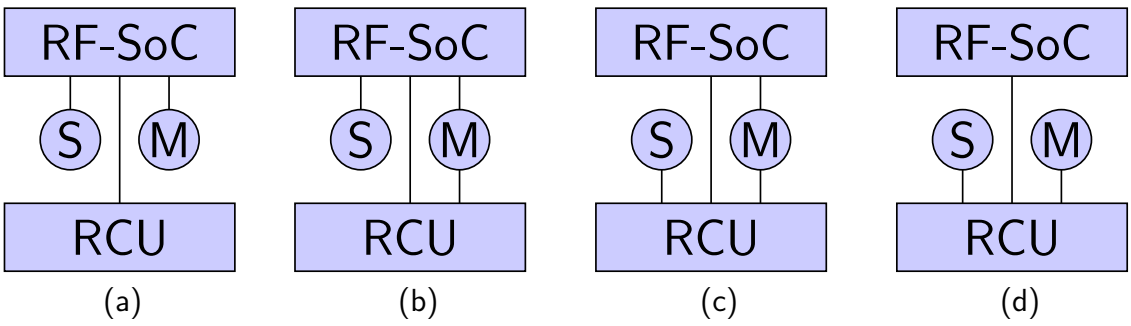


Figure 13: Design options for attaching sensors (S) and memories (M) to the compute units

detection applications require an immediate processing of the sensor data to minimize the detection delay. In these cases, only the aggregated data stream has to pass the bottleneck from the RCU to the RF-SoC. A dedicated shared memory between both computational units is thus not required and the architecture of Figure 13c is not useful.

From the remaining two reasonable architectures (Figure 13b and 13d), the latter was chosen as the foundation of the HaLoMote: With all peripherals being interfaced by the RCU, the HaLoMote supports a broad range of applications with different sensor and memory requirements. This flexibility is essential for the design space exploration targeted by this work.

4.2 HaLOEWEn Implementation

After evaluating two proof-of-concept HaLoMote implementations based on composing discrete evaluation boards for RCU and RF-SoC, a third revision was designed in cooperation with the Microelectronic Systems Research Group at the Technical University Darmstadt [Philipp2011]. Figure 14 shows the main components of this board called Hardware-Accelerated Low Energy Wireless Embedded Sensor Node (HaLOEWEn). As motivated in Section 4.1, it heterogeneously combines a software-programmable RF-SoC and an RCU.

The TI CC2531 (i.e., a [CC2530] with USB controller) was chosen as RF-SoC, as it includes a 2.4 GHz IEEE [802.15.4] transceiver and is thus potentially interoperable with many [ZigBee], [6LoWPAN], [WirelessHART] and ISA [100.11a] devices. Furthermore, the small 8 bit MCU core was chosen deliberately to increase the heterogeneity of the platform, as the CC2531 just handles less complex computations, such as the radio protocol and basic (yet precise) time-keeping. It also has access to the HMI peripherals (e.g., LEDs) on the mainboard.

The RCU used as hardware accelerator is realized as a discrete FPGA. As stated in Section 2.2, FPGAs based on non-volatile configuration storage are best suited for energy constrained applications, as they provide deep sleep modes with fast shutdown

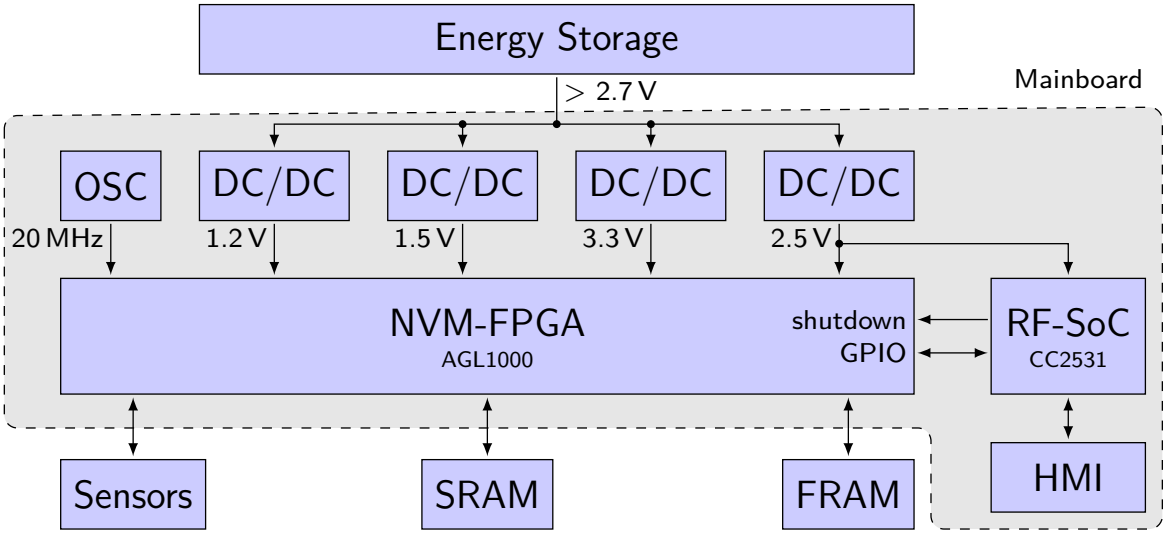


Figure 14: HaLOEWEn version 3 (2010): Basic components

Vendor	Family	since	CMem	largest Device	LCs	V_{CC}	P_{idle}
Actel / Microsemi	[IGLOOplus]	2008	Flash	AGLP125	1,024	1.2 V	17 µW
	[IGLOOnano]	2008	Flash	AGLN250	2,048	1.2 V	24 µW
	[IGLOO]	2008	Flash	AGL1000	24,576	1.2 V	53 µW
	[IGLOO2]	2014	Flash	M2GL150	146,124	1.2 V	10 680 µW
	[ProASIC3nano]	2008	Flash	A3PN250	2,048	1.5 V	4500 µW
	[ProASIC3]	2005	Flash	A3P1000	24,576	1.5 V	12 000 µW
	[ProASIC3L]	2008	Flash	A3PE3000L	75,264	1.2 V	3300 µW
Lattice / SiliconBlue	[iCE65]	2008	hybrid	iCE65L08	7,680	1.0 V	54 µW
	[iCE40]	2011	hybrid	LP8K	7,680	1.0 V	300 µW
	[iCE40ultra]	2014	hybrid	iCE5LP4K	3,520	1.2 V	85 µW
	[MachXO3]	2014	hybrid	XO3LF-6900	6,900	1.2 V	4872 µW
	[XP2]	2007	hybrid	XP2-40	40,000	1.2 V	54 000 µW

Table 3: FPGA devices with non-volatile configuration storage (P_{idle} specifies lowest possible power consumption with RAM retention)

and wake-up times as well as a very low static power draw. At the design time of the HaLOEWEn mote in 2010, only two FPGA vendors actually targeted ultra-low power applications, as shown in Table 3. Actel, which was acquired by Microsemi in 2010, offered the [IGLOO] and [ProASIC3] device families. By storing the device configuration in on-chip Flash memory, the fabric of the FPGAs can be power-cycled within a few microseconds. This process, named *Flash*Freeze*, also preserves the content of the registers and BRAMs holding the runtime information. These Flash-based devices are thus well suited for fast DPM. The [ProASIC3] devices, however, were designed for more compute-intensive applications. Their large idle power consumption is not acceptable for a WSN scenario.

SiliconBlue, which was acquired by Lattice Semiconducters in 2011, offered the [iCE65] device family with a non-volatile bitstream storage, that is copied into the SRAM-based configuration storage at startup. Although the (one time programmable) NVM is integrated into the FPGA die (hybrid CMem), the configuration takes at least 50 ms, which renders fast DPM schemes infeasible. Furthermore, the largest [iCE65] provides less than 7000 logic cells, which would restrict the HaLoMote target applications to very simple aggregation mechanisms. In addition to the FPGAs listed in Table 3, some older CPLD devices like the Xilinx [CoolRunner-II] or the Altera [Max5] also use a hybrid CMem and are thus also not suitable for the HaLoMote.

In the end, only the Microsemi [IGLOO] family was a viable option for the HaLOEWEn implementation in 2010. After Microsemi shifted towards more power-consuming applications with the follow-up [IGLOO2] family and its SoC version (i.e., SmartFusion2), the old [IGLOO] family remains the best choice for ultra-low power, yet demanding reconfigurable computing even today.

The HaLOEWEn mote is built with the large AGL1000 device, but pin compatibility down to the AGL400 is assured. Target applications that can be implemented with a smaller amount of logic resources can thus use the cheaper devices without the need for redesigning the HaLOEWEn PCB. As shown in Figure 14, the FPGA is driven by a 20 MHz external oscillator IC and supplied by four voltage rails generated from a common energy source (e.g., a 3 V battery pack). The 1.2 V rail supplies the

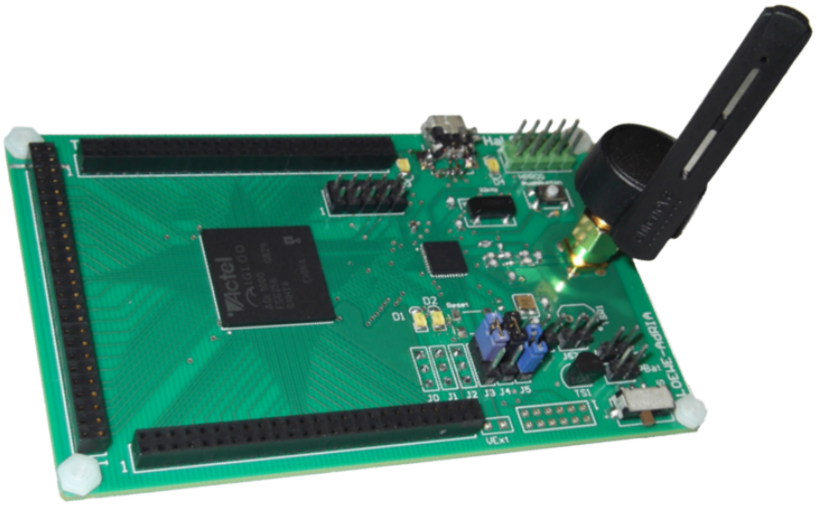
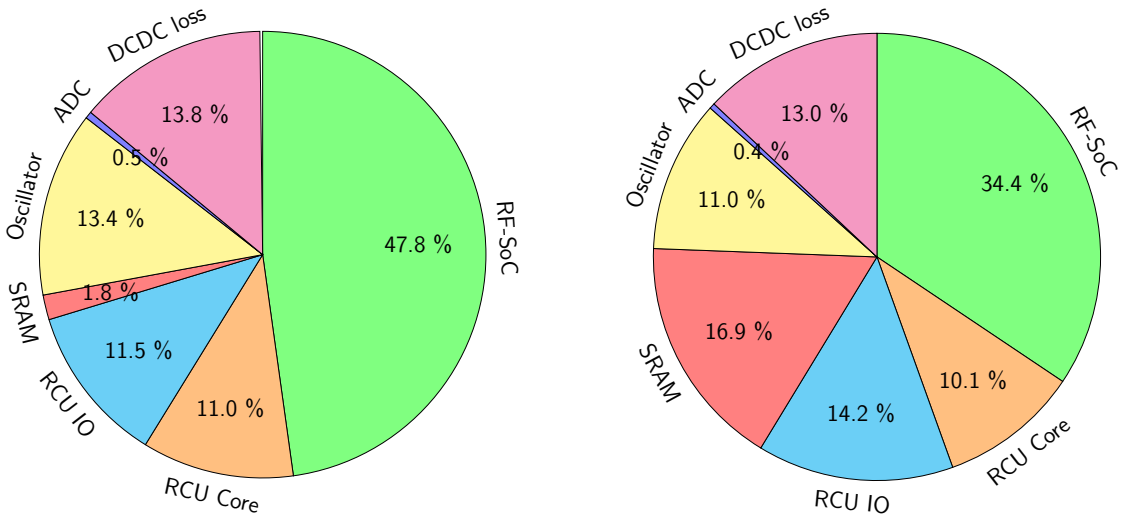


Figure 15: HaLOEWEn version 3 (2010): 100 mm × 62 mm PCB

FPGA core during normal operation, while the 1.5 V supply is required only during Flash programming. The corresponding Voltage Level Converter (DC/DC) can thus be disabled by a jumper, as soon as the device is programmed. The 2.5 V and 3.3 V rails drive the peripherals, the RF-SoC, and the IO banks of the FPGA.

As described in Section 4.1, application-specific sensors and memories are directly attached to the FPGA. By exposing the corresponding GPIOs pins to expansion headers, HaLOEWEn3 can be easily adopted to different applications requiring specific peripherals. The resulting HaLOEWEn PCB is shown in Figure 15.

While the HaLOEWEn mote already exceeded the performance and power efficiency of homogeneous systems for real applications [Engel2012a], detailed power profiling (see Figure 16) revealed that the 8051-based MCU of the HaLOEWEn3 RF-SoC required significant energy, even with the radio transceiver completely shut-down: With the control software on the MCU just initiating RCU operations (i.e., sensor sampling



(a) Idle (392 µW average consumption) (b) Loaded (551 µW average consumption)

Figure 16: Per-component breakdown of the HaLOEWEn3 power consumption

and data accumulation) at 128 Hz, the RF-SoC consumed between 34 % and 48 % of the overall system energy (depending on the actual compute load on the RCU) [Engel2012a]. More than half of the active time of the MCU (31 µs out of 58 µs) was spent waiting for the clock source of the RCU to become stable after waking it up.

These practical experiences led to design improvements for a fourth refined implementation of the architecture, shown in Figure 17. A improved clocking scheme allows the MCU to provide an *auxiliary* clock for the RCU until the main oscillator has completely started up (see Section 4.4.3). Furthermore, the new oscillator runs only at 8 MHz and can be scaled further down by a programmable pre-divider. In most cases, the energy consumption of the RCU-internal clock conditioning component, such as a PLL, can thus be eliminated.

For the HaLOEWEn4, the TI RF-SoC itself was replaced by a more recent Atmel [ATmega256RFR2] device, which not only has higher radio throughput (in a special non IEEE [802.15.4] compliant mode), but also more GPIO pins for communicating with the RCU. Furthermore, it can be operated with just a 1.8 V supply, (instead of the 2.5 V supply used for the CC2531), thus allowing for more efficient switching regulators.

More precisely, a single step-down regulator (1.8 V) is used by HaLOEWEn4 for the peripheral supply instead of two buck-boost converters (2.5 V and 3.3 V) required by HaLOEWEn3. Furthermore, a single configurable step-down regulator is used by HaLOEWEn4 to either generate the 1.2 V or the 1.5 V core supply for the FPGA. The third DC/DC converter used by the HaLOEWEn4 is only required when the FPGA is programmed and can thus be kept down most of the time. By exposing the FPGA Joint Test Action Group (JTAG) interface to the MCU, over-the-air reconfiguration of the whole platform is supported by the HaLOEWEn4.

Furthermore, the power and area-consuming HMI-peripherals (e.g., LEDs, switches) were excluded from the mainboard. While they can still be attached for debugging pur-

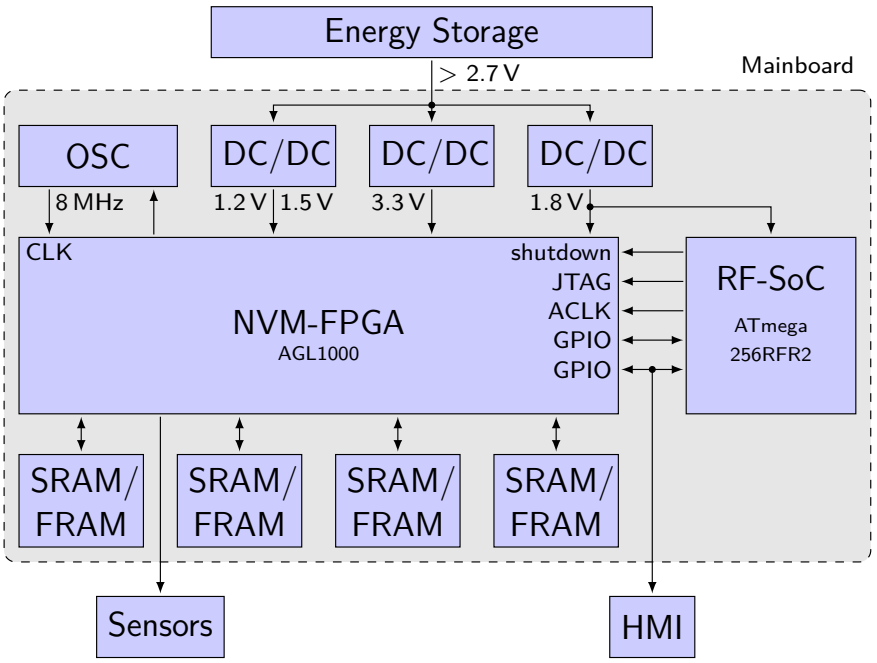


Figure 17: HaLOEWEn version 4 (2014): Basic components

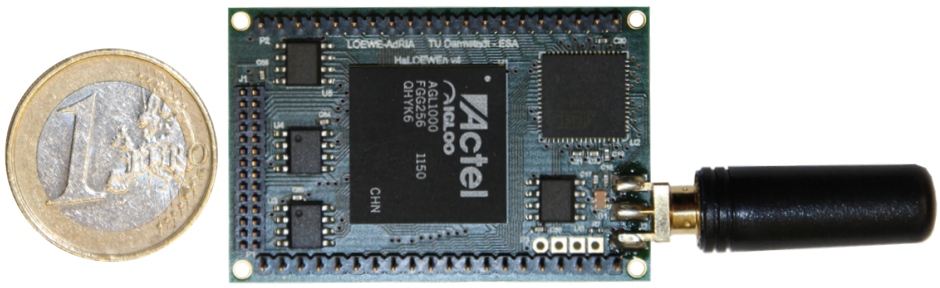


Figure 18: HaLOEWEn version 4 (2014): 46 mm × 30 mm PCB

poses, unused peripheral pins are now used to increase the communication bandwidth between MCU and RCU.

Most monitoring applications require a significant amount of external memory. By directly integrating four 1 Mbit *serial* SRAM devices on the mainboard, less demand was imposed on the expansion headers (now 40 pins, down from 148). This significantly shrunk the overall system size down to 46 mm × 30 mm (see Figure 18). Choosing multiple serial instead of a single parallel memory enables parallel *independently addressed* memory accesses by the RCU. Furthermore, one or more SRAMs can be selectively replaced with pin-compatible non-volatile FRAMs for even more aggressive power management, as described in Section 4.4.2. FRAM was chosen as persistent data storage as it clearly outperforms Flash-based memory in write performance (i.e., access time, granularity, and energy consumption) [Lueders2014]. In particular, the [MB85RS1MT] FRAM modules can be written at 25 Mbit/s while just consuming 5 nJ per byte from the 1.8 V supply rail. As detailed in Table 5, an equally sized Flash device can be written at 260 kbit/s while consuming 332 nJ per byte.

4.3 Inter-Processor Communication

As in every heterogeneous architecture, the communication between the different computing units has a large impact on the system level performance. Commonly, a *middleware* layer is employed to wrap the hardware/software interface.

In order to achieve this for a very constrained system such as HaLOEWEn, a number of issues must be addressed that do not come up for larger desktop or server-class machines. In such systems, discrete RCUs and CPUs usually communicate using memory-mapped IO (MMIO). However, due to pin-count restrictions, the MCU does not externally expose its memory bus, thus necessitating the use of a different interface.

While the Universal Synchronous and Asynchronous Serial Receiver/Transmitter (USART) of the MCU could be used to establish a low-pin count, but rather slow *serial* link to the RCU, a sufficient number of MCU GPIO pins is available to realize a still narrow, but much faster *parallel* bus. It consists of a bidirectional 9 bit to 20 bit data, a 3 bit MCU→RCU command, and a 1 bit clock signal, as shown in Figure 19. The actual data width depends on whether additional (HMI) peripherals also require GPIO pins. An additional sleep signal is used for DPM as described in Section 4.4.

Based on this physical communication layer, a message-based application-independent Application Programming Interface (API) was implemented permitting the MCU to control the execution of multiple hardware (HW) kernels on the RCU. HW kernels are typically used to interface sensors, access the external memory (which is only available

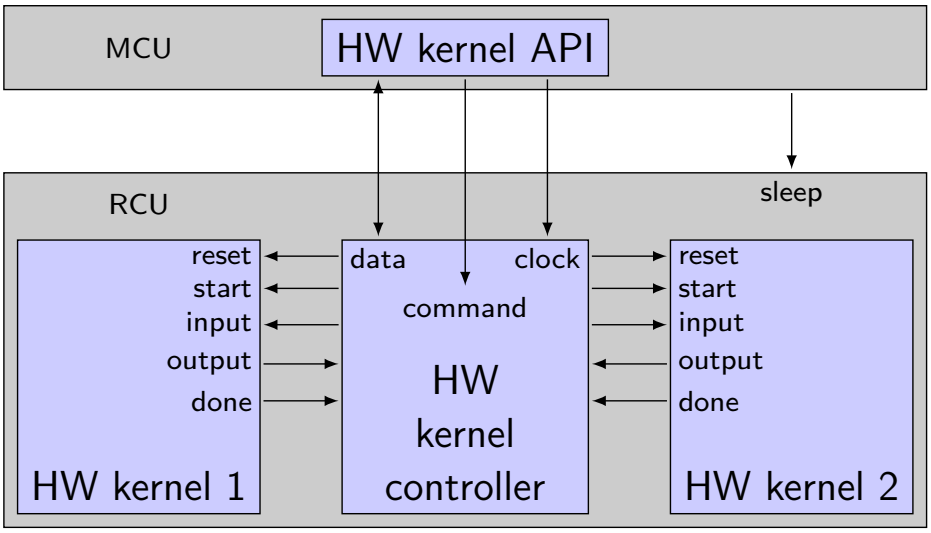


Figure 19: Hardware kernels

to the RCU), implement digital filters, data compression, or other application-specific data aggregation algorithms.

Each HW kernel has input and output data, an internal state, and start/done signals controlling its execution. HW kernels can thus be daisy-chained very easily. To conserve bandwidth on the narrow parallel bus, the HW kernel selection is separated from the data transfers (similar to virtual circuit switching). For example, sensor kernels must be started in every sampling cycle, but a data compression channel is only triggered after a sufficient number of samples has been collected. Thus, the virtual channel only needs to be switched to the appropriate kernel just before and after the data compression. Cyclically executing kernels can be set to auto-restart without MCU intervention, beginning computing again once their previous results have been retrieved.

Table 4 shows the commands implemented by the middleware. On the RCU side, the entire protocol (including marshaling/demarshaling of parameters) is handled in a dedicated HW block shared between the kernels.

The performance of this interface is primarily limited by the MCU-internal memory throughput and GPIO toggle rates. A latency of about 1.5 µs per atomic operation is

Command	data driven by	Semantics and Parameters/Results
OBSERVE	RCU	retrieve execution states of all kernels
READ	RCU	read current output data from virtual channel
WRITE	MCU	write current input data to virtual channel
START	MCU	start kernel with given ID
RESET	MCU	reset kernel with given ID
SWITCH	MCU	switch virtual channel to given kernel ID
INC	MCU	value to be added to the selected kernel index
CONFIG	MCU	reconfigure kernel with selected ID

Table 4: Atomic operations for MCU ↔ RCU communication

achieved for the 8 bit data signals of the HaLOEWEn3, resulting in a throughput of 5.3 Mbit/s. This can be increased by allocating additional GPIO pins not required for peripherals to widen the data bus. An alternative USART-based communication link could achieve up to 4 Mbit/s when utilizing both USART modules of the CC2531 RF-SoC in parallel. The proposed communication framework thus improves the conventional approach by at least 33 %.

4.4 Power Management

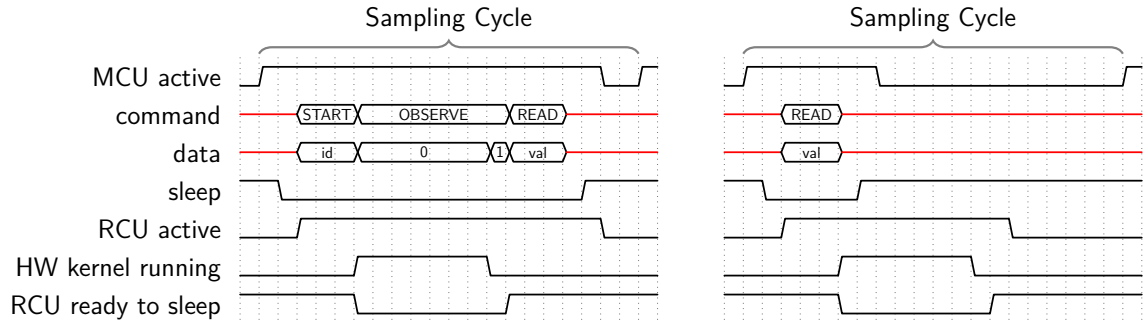
Since both the RCU and MCU devices support DPM, a proper power management scheme should be straightforward: The MCU is woken up periodically at the beginning of each sampling interval to handle system management tasks such as the wireless communication protocol or the high-precision time synchronization (Section 6.2). It also retrieves previously computed data from the RCU (for possible transmission), offloads new computations (if any) to the RCU, and then goes to sleep itself. The RCU is powered up only if it actually has work to do.

The Microsemi [IGLOO] Flash*Freeze mode preserves the hardware-configuration as well as the content of the on-chip state, while reducing its power draw to just 53 µW. This sleep mode can be quickly entered and exited within a few microseconds transition time, which is negligible even for sampling frequencies of several 100 Hz. The MCU signals the RCU to enter and exit the Flash*Freeze mode by (de-)asserting the sleep signal shown in Figure 19.

However, the interdependency between both compute units of the heterogeneous architecture introduces difficulties for the overall DPM strategy, which are addressed in the following sections.

4.4.1 Flash*Freeze Control

Figure 20a shows a sample schedule for a single output-only HW kernel (e.g., a sensor controller). While the first line represents the active time of the MCU, the command, data, and sleep lines correspond to the inter-processor communication signals shown in Figure 19. The fifth line describes the active time of the RCU (i.e., not in Flash*Freeze mode), followed by the time the RCU is actually executing a HW kernel. The last line defines the time periods during which the RCU knows that it is ready to fall asleep, as



(a) Stalling the MCU (b) Delayed Flash*Freeze

Figure 20: Scheduling of DPM and inter-processor communication for a single output-only HW kernel

all HW kernels are completed. This signal is derived by a simple state machine in the HW kernel controller observing the start and done flags of all kernels.

At the beginning of each sampling cycle, the MCU is woken up by its internal timer. Afterwards, the MCU deasserts the sleep signal, so the RCU is woken up. Now, the execution of the HW kernel is initiated by the MCU issuing the START command of the HW kernel API with the id of the targeted kernel passed in the data signal. Once the kernel has started, the RCU is not ready to enter the Flash*Freeze mode. At this point in time, the MCU performs its management tasks (e.g., updating the timer and handling the radio protocol), which is not shown in Figure 20. Complex HW kernels such as the compression of larger datasets (see Section 5.2), will take longer to complete than the MCU requires for its management operations. By using the OBSERVE command, the MCU determines the execution status of the HW kernel and stalls, until the kernel completes execution. Afterwards, the result of the kernel can be read back to the MCU using the READ command. Finally, the MCU asserts the sleep signal again to shutdown the RCU, before the MCU itself falls asleep.

Stalling the MCU just to wait until the RCU can be shutdown, is not energy efficient. Although the MCU could fall asleep earlier and use a GPIO interrupt to wake up as soon as the HW kernel is completed, the additional state transitions of the MCU increase the mote's energy consumption. To decouple the shutdown of the RCU from the shutdown of the MCU, the HW kernel controller inside the RCU delays the actual shutdown request until the RCU is ready to fall asleep (i.e., all kernels finished their execution). This is achieved by a dedicated control bit inside the [IGLOO] fabric. The RCU only enters the Flash*Freeze mode, if both the external and the internal shutdown requests are asserted.

Figure 20b again shows a sample schedule for a single output-only HW kernel, now exploiting the improved Flash*Freeze control to get rid of the lengthy OBSERVE command. Due to the auto-start on read feature, even the START command can be avoided thus significantly reducing the active time of the power-hungry MCU. Note that the value read by the MCU corresponds to the output of the HW kernel from the previous sampling cycle, as the kernel is started *after* its result is fetched. This might be critical for delay-sensitive applications.

4.4.2 Swapping Live Variables to External Memory

Although the [IGLOO] FPGA can enter and exit the Flash*Freeze mode very quickly, the 53 µW static power consumption within this mode is at least an order of magnitude larger than the power drawn by typical sleeping WSN software processors (see Table 17). To further reduce the power consumption of the FPGA, its voltage supply would have to be shut down completely, either by disabling the appropriate DC/DC converter, or by opening a power Field-Effect Transistor (FET). Note that these mechanisms are not yet integrated into the HaLoMote implementations.

Due to the non-volatile configuration storage, the hardware accelerator will be operable again immediately after ramping up its power supply. However, all runtime information hold in the volatile (SRAM-based) registers and BRAMs is lost after each power cycle. Before shutting down the FPGA, any information that will be required again after the power cycle (i.e., the live variables) thus have to be swapped to an external memory, from where it can be restored after the restart of the FPGA.

The data swapping requires additional energy for the memory read and write operations and the additional time the FPGA has to be kept awake. The sleep period with reduced power consumption must be long enough to recoup the swapping overhead. To estimate the minimum sleep time per swapped bit, let P_r, P_w, and P_s denote the power drawn by the memory during reading, writing and when shut down. Furthermore, let T_r and T_w be the achievable throughput for streaming reads and writes, as well as P_a and P_f be the power drawn by the FPGA in active and Flash*Freeze mode. The swapping of n bit live variables for a sleep period t pays off, if

$$t \cdot P_f > \frac{n}{T_w}(P_a + P_w) + (t - \frac{n}{T_w} - \frac{n}{T_r})P_s + \frac{n}{T_r}(P_a + P_r) \tag{1}$$

$$\Leftrightarrow \quad \frac{t}{n} > \frac{1}{P_f - P_s}\left(\frac{P_a + P_w}{T_w} - (\frac{1}{T_w} + \frac{1}{T_r})P_s + \frac{P_a + P_r}{T_r}\right) \tag{2}$$

Table 5 lists the power draw and throughput specification of three different types of memory. The SRAM and the FRAM modules are actually integrated into the HaLOEWEn4, while the third memory is a Flash device with the same capacity and interface. The relative break-even time $\frac{t}{n}$ depends on the power drawn by the FPGA and is thus application-specific. However, assuming $P_a = 30\,\mathrm{mW}$ for a worst case analysis is reasonable (see Table 17).

The resulting break-even points for the different memory types are also listed in Table 5. Due to its low write throughput, Flash memory is not appropriate for the live variable swapping. The SRAM swapping pays off after 62 µs/bit, while the FRAM requires 71 µs/bit to justify the swapping. When directly comparing the overhead of SRAM and FRAM swapping against each other, the latter pays off if

$$\frac{2n}{T_w^{\mathrm{SRAM}}}(P_a + P_w^{\mathrm{SRAM}}) + (t - \frac{2n}{T_w^{\mathrm{SRAM}}})P_s^{\mathrm{SRAM}} > \frac{2n}{T_w^{\mathrm{FRAM}}}(P_a + P_w^{\mathrm{FRAM}}) \tag{3}$$

$$\Leftrightarrow \quad \frac{t}{n} > \frac{2}{P_{s,\mathrm{SRAM}}}\left(\frac{P_a + P_w^{\mathrm{FRAM}}}{T_w^{\mathrm{FRAM}}} - \frac{P_a + P_w^{\mathrm{SRAM}} - P_s^{\mathrm{SRAM}}}{T_w^{\mathrm{SRAM}}}\right) = 327\,\mathrm{\mu s/bit} \tag{4}$$

Thus, for a sample live variable set of 100 B, swapping to SRAM is worthwhile for sleep periods of at least 50 ms, and using FRAM instead of SRAM is reasonable only for sleep periods of at least 262 ms. For a running measurement, the set of the live variables between subsequent sampling periods is rather large, as it typically contains buffered samples (e.g., taps of digital filters) and intermediate results (e.g., accumulated

Device	Type	T_w	T_r	P_w	P_r	P_s	$\frac{t}{n}$
[23A1024]	SRAM	20 Mbit/s	20 Mbit/s	1.8 mW	1.8 mW	1.8 µW	62 $\frac{\mu s}{bit}$
[SST25WF010]	Flash	0.26 Mbit/s	20 Mbit/s	10.8 mW	3.6 mW	-	2993 $\frac{\mu s}{bit}$
[MB85RS1MT]	FRAM	25 Mbit/s	25 Mbit/s	17.1 mW	17.1 mW	-	71 $\frac{\mu s}{bit}$

Table 5: Power consumption and streaming throughput of different 1 Mbit memories operated at 1.8 V

sensor channel statistics). The swapping is therefore not viable for the DPM between subsequent sampling cycles for most of the applications addressed by the HaLoMote with sampling rates of 100 Hz and above. Instead, such swapping can only be used for supervisory power management when the application is not relying on continuous monitoring.

The estimation of the break-even points assume that the number of bits swapped out equals the number of bits swapped in. A dirty bit mechanism known from conventional cache systems may reduce the number of bits to swap out. Classification algorithms such as rainflow-counting [OConnor2010] are typical examples that benefit from this mechanism, as a larger dataset is continuously updated but only a small subset of its entries might have changed between subsequent DPM cycles.

4.4.3 Flexible Clock Generation

Since both the MCU and RCU rely on synchronous digital logic, they require a periodic clock signal. MCUs typically provide internal oscillators based on resonating crystals or simple capacitive circuits, which are automatically turned-off when the MCU is put to sleep. In contrast, the [IGLOO] FPGA requires an additional clock source. Commonly, an external oscillator IC is used for this purpose. The [LTC6930] oscillator clocking the HaLOEWEn4 RCU was primarily selected for its energy efficiency. Nevertheless, it still draws about 580 µW at 8 MHz, which is more than 10× the combined power required by the MCU and RCU in sleep mode. Clearly, the oscillator should also be powered down when the RCU is put to sleep. However, the start-up of the oscillator takes about 45 µs (see Figure 21), which renders the fast Flash*Freeze capability of the RCU (about 1 µs state transition time) a moot point. At higher sampling frequencies, more sophisticated clocking schemes are thus required to support fast and energy-efficient shutdown and wake-up of the hardware accelerator.

In this Section, five different clocking schemes are described and their impact on the DPM scheduling of the HaLoMote is analyzed. The resulting energy efficiency is reported in Section 4.4.4.

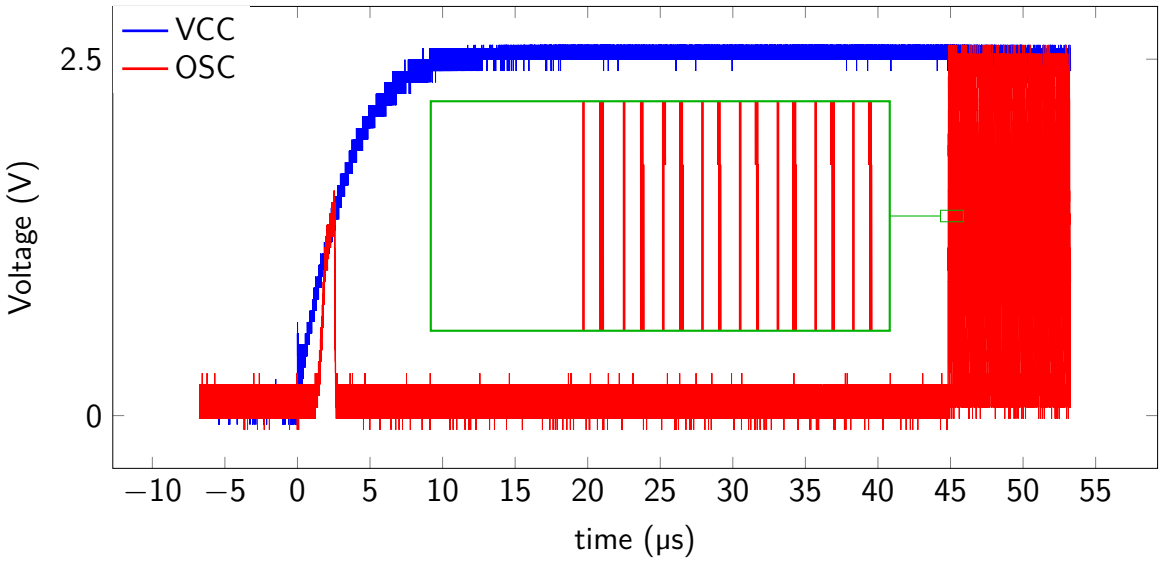
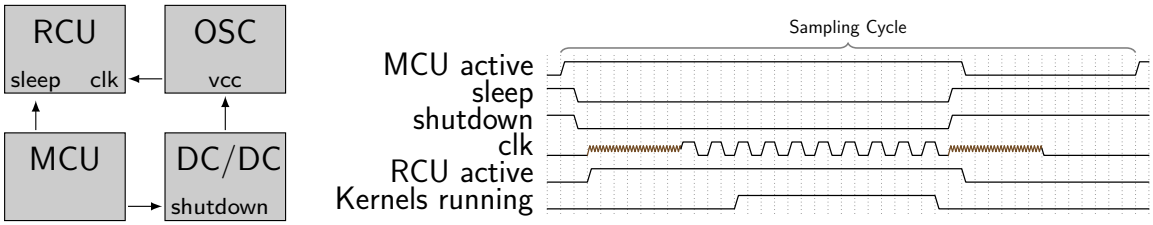


Figure 21: Startup timing of the [LTC6930] external oscillator

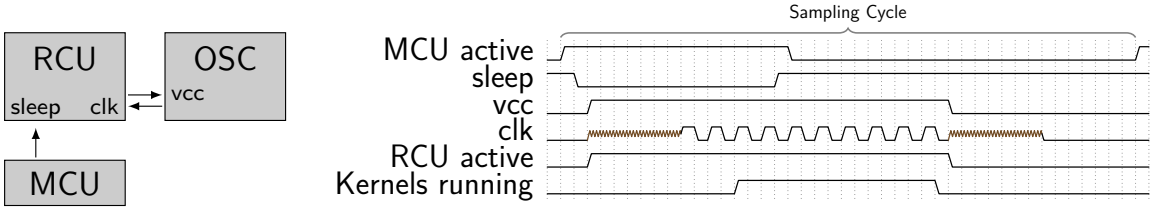


(a) Architecture (b) Power management control sequence

Figure 22: Clocking the RCU by an external oscillator controlled by the MCU

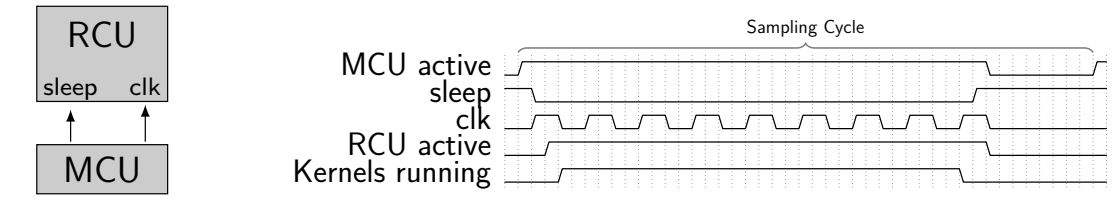
The most obvious clocking scheme is shown in Figure 22a. An external oscillator IC directly drives the RCU while the MCU is controlling the DPM by shutting down both the RCU and the oscillator via dedicated GPIOs pins. Most external oscillators do not provide low-power modes, but require their supply voltage to be detached by a dedicated transistor, or by shutting down the corresponding DC/DC converter. In the latter case, the oscillator keeps toggling even after the converter is shut down, until the voltage of the converter's charge pump capacitor drops below the supply threshold of the oscillator. For example, a typical 10 µF DC/DC output capacitor keeps supplying the [LTC6930] oscillator with 321 µA for about 3 ms, before the supply voltage drops from 1.8 V to 1.7 V. A fast shutdown thus has to be realized by a transistor switch. Figure 22b details the power management timing resulting from the external clock supply. After the MCU woke up and activated the power supply of the oscillator, the hardware kernels cannot be started before the oscillator provides a stable clock signal. As stated above, this long oscillator start-up delay is a major drawback. Even worse, the MCU cannot put itself to sleep as long as the hardware kernels are running, as it has to shutdown the oscillator power supply immediately *after* the hardware accelerator has finished its tasks.

The late shutdown of the MCU can be avoided by directly powering the oscillator with an RCU IO pin (see Figure 23a). The [IGLOO] FPGA can pull an IO pin to ground when entering the Flash*Freeze mode, and back up to the supply voltage level of the IO banks when waking up, even without a clock signal being present. Combined with the sophisticated Flash*Freeze controller described in Section 4.4.1, this clocking scheme unburdens the MCU from monitoring whether the RCU is currently active. Furthermore, as the FPGA output pins are not buffered by a noticeable capacitance, the oscillator stops toggling *immediately* after the FPGA has entered the Flash*Freeze. The lengthy start-up time of the external oscillator, however, remains a weak point of this clocking scheme.



(a) Architecture (b) Power management control sequence

Figure 23: Clocking the RCU by an external oscillator controlled by the RCU



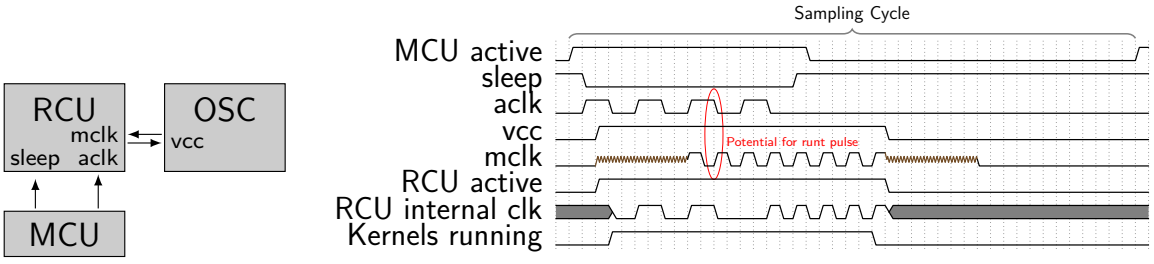
(a) Architecture (b) Power management control sequence

Figure 24: Clocking the RCU by the MCU

Instead of using an external oscillator IC, a clock signal generated by the MCU can drive the RCU as shown in Figure 24a. Many MCUs like the Atmel [ATmega256RFR2] can provide clock signals at dedicated output pins. For all other devices, like the TI [CC2530], an unused Serial Peripheral Interface (SPI) module can be repurposed by repeatedly filling the SPI output register to keep the serial clock running (even though no SPI transfers actually take place). This can be achieved by Direct Memory Access (DMA) transfers without actually invoking the processor. As shown in Figure 24b, the major disadvantage of this scheme is the necessity to keep the MCU active until the RCU has finished its own computations. As before, the MCU must be stalled unless useful computations can actually be issued while waiting for the RCU. Thus, this approach of using a MCU clock can only do better than the external oscillator IC if the accelerator execution time is *shorter* than the oscillator IC startup time.

Figure 25a shows the combination of the two previously described clocking schemes. This *dual clocking* approach uses the fast-starting auxiliary clock (ACLK) provided by the MCU to drive the RCU while the external oscillator is slowly starting up to generate the main clock (MCLK). The [ATmega256RFR2] can generate an ACLK of up to 16 MHz, so about 720 RCU cycles can already be executed before the MCLK becomes stable. As shown in Figure 25b, this avoids stalling the MCU and the RCU at the beginning of a sampling cycle and allows for a fast shutdown of the MCU, as the ACLK is not required to complete the hardware kernel execution (since the MCLK has taken over). In practice, however, several issues have to be addressed when implementing such a dual-source clocking scheme.

First, the RCU itself has to determine whether the MCLK is sufficiently stable after the start-up. A simple counter on the RCU delays the switch-over from the ACLK to the MCLK by an experimentally-determined number of MCLK cycles. To improve the robustness of the MCLK stability detection under voltage and temperature



(a) Architecture (b) Power management control sequence

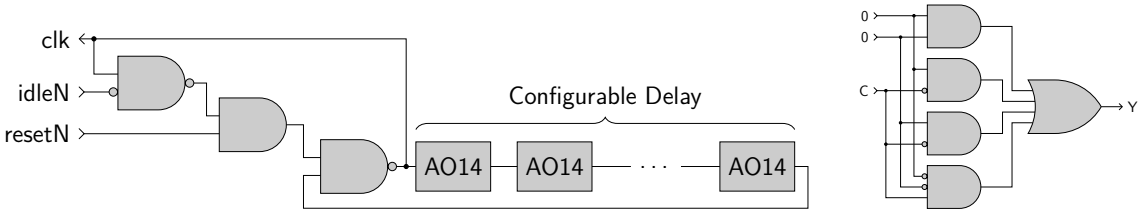
Figure 25: Clocking the RCU by two sources

variations, the number of MCLK cycles within each (or multiple) ACLK cycle can be observed. As soon as the expected number of MCLK cycles was counted in two (or more) subsequent ACLK cycles, MCLK-stability can be assumed. This feature is, however, not yet implemented for the HaLoMote architecture.

Second, a *valid* switching from the ACLK to the MCLK has to be supported. Runt pulses (leading to invalid clock periods shorter than the specified cycle duration) must not reach the hardware kernels, as overclocked hardware would generate unpredictable results. As shown in Figure 25b, switching to the MCLK *immediately* after it is deemed stable may cause such runt pulses and a faulty clock signal. To avoid this, the RCU-internal system clock must be forced low for at least one MCLK cycle after the MCLK became stable and the auxiliary clock went low (ensuring a valid clock signal after the switch-over).

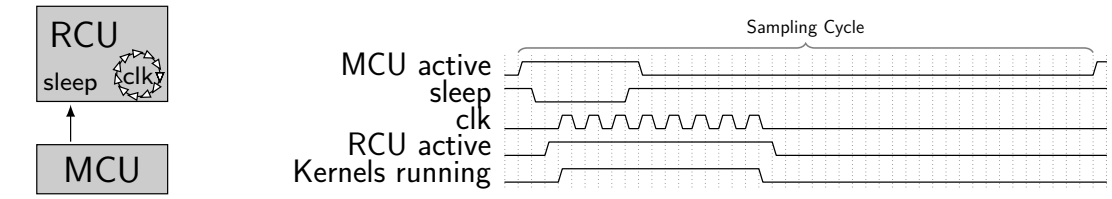
Finally, if all hardware kernels can be completed on the ACLK before the MCU falls back to sleep, the external oscillator should not even be started. To achieve the best energy savings, the decision about whether the MCLK is necessary should be made by the RCU at the start of each sampling cycle, before actually starting the hardware kernels. This is possible for many applications with fixed or predictable accelerator runtimes, as long as the minimum execution time of the MCU (e.g., required for timekeeping) is known.

The last clocking scheme considered generates the clock on the RCU itself, as described in [AC332], requiring neither an external oscillator IC nor the MCU clock. As shown in Figure 26a, this oscillator is realized as a combination of an inverting NAND gate and a configurable number of delay gates to control the clock frequency. The AO14 cell is used as delay gate as it provides the largest IO delay ($\approx 6.3\,\mathrm{ns}$) within the Microsemi cell library for the IGLOO device. By pulling two out of its three inputs to ground (see Figure 26b), the remaining input (C) is not inverted. The delay chain without the first NAND gate is thus not limited to an even length. The oscillator can be controlled by low active signals to immediately force the clock output high (resetN), or only after the next rising clock edge (idleN). The actual clock frequency is sensitive to the number of delay cells as well as the RCU core voltage and operating temperature, since no stable frequency reference (such as a quartz crystal) is involved (see Section 4.4.4). If the associated frequency drift is acceptable for an application, this scheme completely eliminates the flaws of the previous ones, as the MCU is no longer involved in the RCU clock management at all, as shown in Figure 27b.



(a) Ring structure (b) AO14 cell

Figure 26: Implementation of an RCU-internal oscillator, consisting of a configurable number of AO14 cells used as delay elements as described in [AC332]



(a) Architecture (b) Power management control sequence

Figure 27: Clocking the RCU by an internal oscillator

4.4.4 Evaluation

To analyze the effects of the different clocking schemes on the energy efficiency of the heterogeneous platform, a synthetic benchmark related to the target application described in Chapter 7.3 was executed on the HaLOEWEn mote. Essentially, a configurable number of computations and accompanied accesses to an RCU-external SRAM were handled by a HW kernel in every sampling cycle. In parallel, the MCU was performing timekeeping and queried status information from the RCU. The same algorithm was implemented on a TI [MSP430] reference system, to compare the energy efficiency of the heterogeneous architecture against a homogeneous software solution. Detailed information about this benchmark running at 128 Hz sampling frequency can be found in [Engel2012a].

The clocking schemes shown in Figures 23, 24, 25, and 27 were arranged by appropriately connecting prototyping boards for the [IGLOO] AGL1000 FPGA, the [CC2530] MCU and the 8 MHz [LTC6930] oscillator. The current flowing into a common 3.7 V supply rail was measured by an Agilent [34411A] multimeter configured for 100 mA range and an integration time of 2 s to average the current spikes into the switching regulators. Thus, all losses caused by voltage regulators and the oscillator are included in the power measurements.

Figure 28 details the measurement results when sweeping the number of accumulations to be performed per sampling cycle from 0 to 100. First of all, the [MSP430]

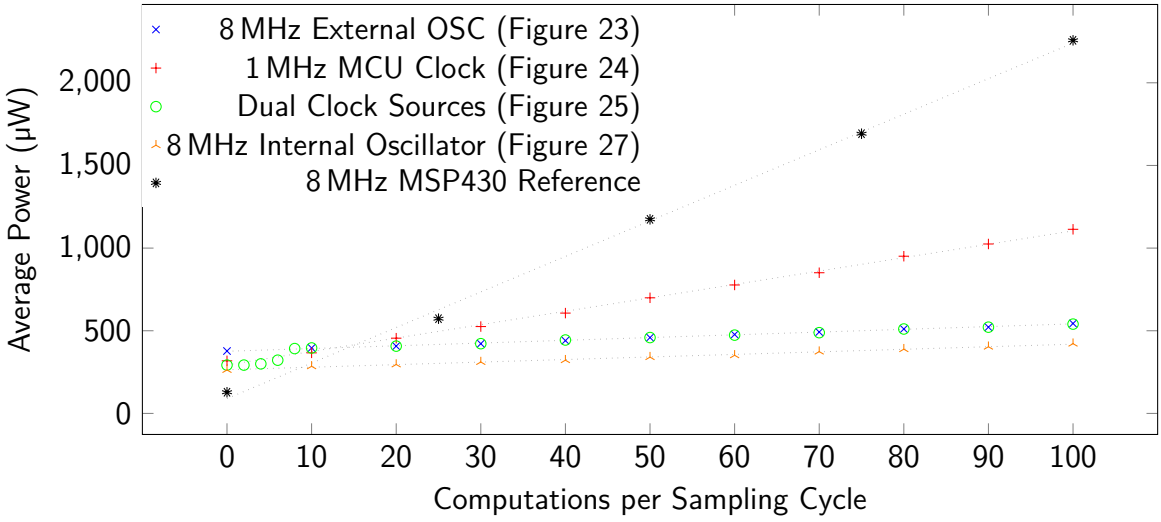


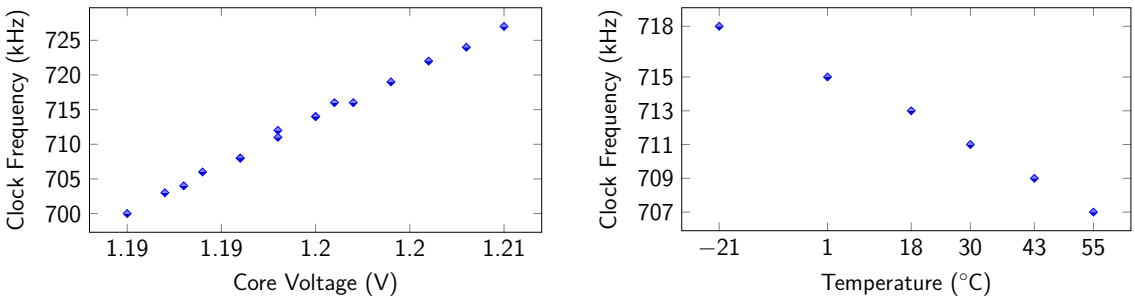
Figure 28: Load-Dependent power draw for the considered clocking schemes

reference system is clearly outperformed by the four HaLOEWEn configurations, if more than 15 accumulations per sampling cycle have to be performed. Obviously, a WSN mote can only benefit from reconfigurable computing, if there is a sufficient amount of computation to be offloaded to the hardware accelerator.

For less than 14 accumulations per sampling cycle, the HaLOEWEn driven by the 1 MHz serial MCU clock draws less power than when driven by the 8 MHz external oscillator. In these cases, the runtime of the HW kernel clocked at 1 MHz is 49 µs, as each accumulation takes 3.5 cycles on average. This closely matches the start-up delay of the external oscillator and experimentally confirms the trade-off discussion in Section 4.4.3.

The experiments have also determined that the MCU requires at least least 26 µs to start the hardware kernel and prepare its timer for the next sampling cycle. As each accumulation may require up to four cycles (i.e., 4 µs at 1 MHz SPI clock) on the RCU, a maximum of six accumulations can be performed running just on the ACLK without starting the external oscillator. This is reflected by the power consumption characteristics of the dual oscillator clocking scheme in Figure 28. For up to six accumulations, this scheme draws even less power than the clocking scheme from Figure 22: It does not require the MCU (which supplies the ACLK) to remain awake and check whether the RCU has completed the computation before the MCU can fall asleep again. Instead, the RCU can continue autonomously as soon as the main clock becomes stable, and the MCU can go back to sleep immediately. If more than six accumulations have to be executed, the main clock is started up at the beginning of each sampling cycle, thus raising the power draw to the level of the external oscillator scheme. With increasing computational load, the dual oscillator scheme then benefits by executing at the rate of the faster main clock.

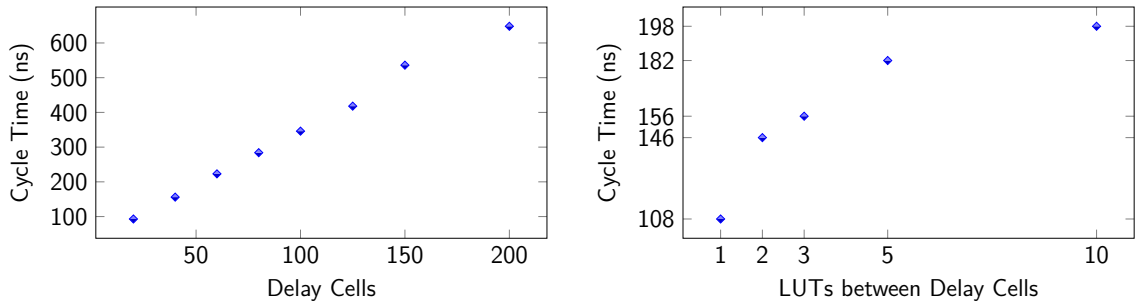
To reduce the system level power draw below that of the dual clocking scheme, all external oscillators and the need to stall any of the computing units before shutdown must be eliminated. As shown in Figure 28, the internal oscillator achieves this goal and outperforms all other clocking schemes. But the usage of such an internal oscillator is quite unconventional, as its frequency is dependent on the operating temperature and the RCU core voltage. As shown in Figure 29a, a ±10 mV core voltage variation translates into a ±13 kHz frequency variation for 400 delay cells. Since the [LTC3388] voltage regulator used to generate the RCU core supply voltage is specified to be stable



(a) Temperature fixed at 20 °C

(b) Core voltage fixed at 1.2 V

Figure 29: Frequency stability of on-chip oscillator with 400 delay cells for variable supply voltage and environmental temperature



(a) Automatically placed delay cells

(b) 40 manually placed delay cells

Figure 30: Effect of delay cell number and distance on the oscillator cycle time

within ±60 mV on its 1.2 V output voltage, the voltage-dependent frequency variation will be limited to ±78 kHz or ±11 % of the 714 kHz center frequency. This far exceeds the temperature-dependent frequency variation, which was observed to be just ±6 kHz or ±0.8 % over a 76 °C temperature range (Figure 29b). Thus, the internal oscillator is a feasible power-saving design choice if its target frequency is conservatively configured to be about 15 % to 20 % less than the maximum clock frequency supported by the specific hardware design (ensuring correct operation even if voltage variations lead to a faster clock).

To selectively control the on-chip oscillator cycle time, two options have been investigated. Increasing the number of delay cells shows a linear effect on the cycle time (Figure 30a). Modifying the distance between the delay cells increases the overall routing delay. To achieve this, each delay cell must be placed manually. Figure 30b shows the effect of longer distances on the oscillator frequency. Due to the segmented nature of the FPGA's configurable interconnect, the relation between distance and delay is non-linear. However, using longer interconnect delays allows for fewer delay cells without changing the target frequency.

To compare the power efficiency of both design options (i.e., longer interconnects or more delay cells), two 5 MHz oscillators have been implemented using 40 and 72 delay cells placed at a distance of 10 and 1 LUTs respectively. The power draw was measured in the RCU Flash*Freeze mode with the oscillator running. Afterwards, the static power consumption in Flash*Freeze mode (with disabled oscillator) was subtracted to determine the power drawn only by the oscillator. The oscillator using 72 delay cells shows a 35 % smaller power consumption (see Table 6). For lower power draw, it is thus advisable to commit more active LUT area for delay cells, than to rely on interconnect segments for controlled delays.

Delay Cells	Distance	Frequency	Power Consumption
40	10	5.02 MHz	272 µW
72	1	5.04 MHz	176 µW

Table 6: Power-Efficiency of on-chip oscillator architectures

CHAPTER 5

Library of Reusable Hardware Accelerator Blocks

The hardware accelerator of the HaLoMote architecture is primarily intended to speed up application-specific data aggregation and feature extraction algorithms. Nevertheless, there are many application-independent algorithms commonly required in WSN scenarios, that might also benefit from hardware acceleration. Potential candidates comprise encryption and authentication, generic data compression, routing, time synchronization, Real-Time Operating Systems (RTOSs), and digital filters. For the following reasons, not all of these tasks are addressed in this work.

Symmetric and asymmetric encryption on RCUs have already been extensively researched [Rodriguez-Henriquez2007]. The corresponding results can be directly applied to the HaLoMote architecture.

Routing algorithms deal with the discovery and maintenance of routes and their overhead is primarily caused by message exchanges (e.g., for neighbor detection) instead of complex computations. Hardware acceleration may be helpful for complex graph problems [Ahmed2009], but the required topology knowledge often prevents a distributed implementation, as targeted by the HaLoMote architecture. Although the hardware acceleration of generic routing protocols is not addressed in this work, a flooding scheme tailored to SHM applications is presented in Section 6.1.

In addition to resource synchronization, event handling, and message passing mechanisms, the central element of an RTOS is the task scheduler. Several attempts were made to realize some [Tang2014] or all [Yan2010] of these elements in hardware. A hardware-accelerated task scheduler, however, must be able to preempt software threads upon occurrence of asynchronous events (i.e., the reception of a radio packet). The RCU thus would remain active most of the time, which is not affordable in energy-constrained WSN scenarios.

Thus, this work is focused on accelerating other functions using dedicated hardware.

5.1 Digital Filters

Digital filters are often required in WSN applications for frequency-dependent signal conditioning (e.g., by FIR filters) or for signal transformations between the time and the frequency domain (e.g., by FFT filters). These digital filters can be implemented as dataflow structures and are thus well suited to be implemented on RCUs. The regular filter structures can also be efficiently generated by HLS tools. In this section, the energy required for hardware-accelerated FIR and FFT filters are compared against appropriate MCU implementations.

5.1.1 Finite Impulse Response Filter

A n-tap FIR filter with the filter coefficients $(a_i)_{i=0}^{n}$ transforms a discrete input sequence $(x_i)_{i\in\mathbb{N}}$ into the output sequence

$$y_m \coloneqq \sum_{k=0}^{n} a_k \cdot x_{m-k} \qquad \forall m \geq n. \tag{5}$$

By selecting appropriate coefficients, the frequency response of the filter can be adopted to realize high-pass or low-pass filters, or to model the behavior of another specific structure. With increasing filter order, the desired frequency response can be modeled more accurately. In WSN applications, FIR bandpass filters are often used for input signal conditioning, e.g., to eliminate low- and high-frequency components such as static offsets or sensor noise.

In [Engel2011], a 384-tap FIR filter modeling the behavior of a bearing structure was generated by [SynphonyHLS] (i.e., the predecessor of [SynphonyModelCompiler]) for the Microsemi [IGLOO] M1AGL1000V2 device. Fixed-point arithmetic with a 16 bit data path, 14 bit coefficients and 7 bit multipliers was used resulting from a bit width optimization process. With a 20× folding of the hardware resources, the filter occupies 71 % of the available logic cells of the FPGA. When running the FPGA at 9 MHz, the FIR execution takes 2.2 µs. At a sampling period of 400 Hz as required by the bearing structure application, the 0.1 % duty cycle results in an average power consumption of 71 µW (FPGA core supply without IO banks).

In comparison, a sequential [MSP430] implementation of the same FIR filter also running at 9 MHz draws 5.6 mW on average. The hardware accelerator thus improves the energy efficiency by 79×.

5.1.2 Fast Fourier Transformation

A n-tap Discrete Fourier Transformation (DFT) filter transforms a discrete time domain sequence $(x_i)_{i=0}^{n-1}$ captured with the sampling frequency f_s into the frequency domain sequence

$$y_m \coloneqq \sum_{k=0}^{n-1} x_k e^{-2\pi i \frac{km}{n}} \qquad \forall m \leq \frac{n}{2} \tag{6}$$

with y_m representing the complex amplitude of the spectrum at the frequency bin $\frac{m}{n} \cdot f_s$.

As stated in Section 3.3.1, many monitoring systems operate in the frequency domain. The FFT rearranges the structure of the DFT into a self-recursive formulation for a more efficient $\mathcal{O}(n \log n)$ implementation. It requires n to be a power of two.

In [Engel2012a], a 256-tap FFT was generated by [SynphonyHLS] for the Microsemi [IGLOO] M1AGL1000V2 device. It uses integer arithmetic with 18 bit accuracy at the input stage. When running the FPGA at 8 MHz, the FFT execution takes 6.3 ms and consumes 46 µJ (FPGA core supply without IO banks).

In comparison, a sequential [MSP430] implementation of the same FFT filter also running at 8 MHz takes 124 ms and consumes 259 µJ. The hardware accelerator thus

improves the runtime-efficiency of the software-processor by 20× and the energy efficiency by 5.6×.

5.2 Lossless Data Compression

In this section, a lossless data compression scheme is described, which is based on minimal assumptions about the nature of the data stream to be compressed. Furthermore, a HaLOEWEn hardware accelerator is detailed, whose energy efficiency is evaluated for two concrete examples in Section 7.1 and 7.2. The implementation and evaluation of the compression kernel have already been published in [Engel2014a].

Lossless data compression can be considered as the most generic data aggregation scheme. It should be applied, if all sensor data has to be forwarded to the base station, either because the feature extraction requires information from all network nodes, or because the network operator requires full-spectrum data for a comprehensive failure analysis.

While reducing the communication demands and the related energy consumption of the sensor nodes, data compression comes at the cost of encoder and decoder complexity. As the decoding is typically performed at the (often mains-powered) base station, the decoder complexity is generally not a major concern. This section thus focuses on the encoder complexity.

To improve the compression quality, many encoders first collect a block of data to analyze its statistical nature before compressing the block with the appropriate settings. This two-pass strategy increases the memory capacity required on the node, as well as the latency between data acquisition and transmission. The block size, and thus the gains in compression quality, is therefore limited by the amount of available memory, or tight real-time requirements in latency-sensitive applications. In addition, both encoder passes require computation time and energy, both of which must be amortized by the reduced communication effort.

5.2.1 Adaptive Differential Pulse Code Modulation and Rice-Encoding

Without detailed knowledge about the data characteristics of a specific application, only the temporal correlation between the samples generated by each sensor can be taken into account. Unless observing random data, the absolute difference between successive samples can be expected to be smaller than the dynamic range of the sensor signal with a high probability.

As described in [Sayood2005, Chapter 11], the Differential Pulse Code Modulation (DPCM) exploits these temporal correlations by predicting the next sensor sample as a linear combination of the previous M observed samples.

$$p_i \coloneqq \sum_{k=1}^{M} a_k \cdot x_{i-k} \qquad \forall i > M \tag{7}$$

$a_1, \ldots, a_M \in \mathbb{R}$ are the prediction coefficients for a predictor of order M. The prediction order can be used to trade-off the compression quality against the computational complexity of the encoder.

Instead of transmitting the samples, the prediction error

$$d_i \coloneqq p_i - x_i \qquad \forall i > M \tag{8}$$

is forwarded to the downstream symbol encoder. If the samples can be predicted accurately, the prediction error sequence has a zero mean and a small variance. To exploit these features, a Rice symbol encoder is used that translate the prediction error sequence into a compressed bitstream [Yeh1991].

As an initial step for Rice encoding, the sequence of signed prediction errors d_i is converted to a sequence of unsigned values ν_i by a simple transformation:

$$\nu_i \coloneqq \begin{cases} 2d_i, & \text{for } d_i \geq 0 \\ -2d_i - 1, & \text{for } d_i < 0 \end{cases} \qquad \forall i > M \tag{9}$$

The relevant property of this transformation is that small absolute values are mapped to small unsigned values. The resulting values ν_i are then encoded into Rice-form bit sequences

$$R_W(\nu_i) \coloneqq \texttt{concat}\left(U(\frac{\nu_i}{2^W}), B_W(\nu_i \bmod 2^W)\right) \qquad \forall i > M \tag{10}$$

with W being the Rice parameter that balances the widths of a zero-terminated unary code U and the binary block code B_W in a bit-wise concatenation. Precise predictions (i.e., $\nu_i < 2^W$) can thus be represented by $W + 1$ bits. Infrequently occurring larger prediction errors, caused by unforeseen spikes, can still be expressed losslessly by exploiting the variable length unary code.

A small and comprehensive example for a third order DPCM and a Rice symbol coder is presented in Table 7. The first three input samples are transmitted in uncompressed form, i.e., as 8 bit block code in this example. The fourth sample is predicted as 17, so the prediction error -7 must be transmitted. After mapping the signed error to the corresponding unsigned value 13, the Rice symbol coder is applied in two steps. First, the run length of the unary code is $13/2^4 = 0$, so only the terminating 0 symbol is generated. Second, the remaining $13 \mod 2^4 = 13$ is represented as 4 bit block code. For the fifth sample, the prediction error is larger. The unary code thus yields a run length of $50/2^4 = 3$, and the remaining $50 \mod 2^4 = 2$ is represented as 4 bit block

i	x_i	p_i	d_i	ν_i	$\nu_i/2^W$	$\nu_i \mod 2^W$	$B_8(x_i)$	$U(\nu_i/2^W)$	$B_W(\nu_i/2^W)$
1	15						00001111		
2	10						00001010		
3	22						00010110		
4	24	17	-7	13	0	13		0	1101
5	8	33	25	50	3	2		1110	0010

Table 7: Example for DPCM and Rice coder: $M = 3, a_1 = 0.5, a_2 = 0.3, a_3 = 0.2, W = 4$

code. Compared to encoding all 5 samples as 8 bit block code, the DPCM achieves a compression ratio of $37/40 = 92.5\,\%$.

As the compression quality of the DPCM scheme depends on the prediction accuracy, the prediction parameters should be adapted to the characteristics of the input data stream. In the ADPCM scheme, this adaption is performed on every block of N samples $x_1, \ldots, x_N$ by calculating the auto-correlation values for the block:

$$r_k \coloneqq \frac{s_k}{N-k} \qquad 0 \leq k \leq M \tag{11}$$

$$\text{with} \qquad s_k \coloneqq \sum_{i=k+1}^{N} x_i \cdot x_{i-k} \qquad 0 \leq k \leq M \tag{12}$$

These correlation values are then used to build a system of linear equations

$$r_k = \sum_{i=1}^{M} a_i \cdot r_{|i-k|} \qquad 1 \leq k \leq M, \tag{13}$$

whose solution results in prediction coefficients that minimize the variance of the prediction error sequence [Sayood2005, Section 11.5.2]. The resulting coefficients must precede the generated bitstream of each block, as the decoder can not perform the coefficient adaption by itself. The coefficients, as well as the first M samples of each block, are encoded by a fixed-length block code.

Resuming the example from Table 7, the correlation values for the $N = 5$ sample block are $r_0 = 290$, $r_1 = 273$, $r_2 = 249$, and $r_3 = 220$. The resulting system of linear equations

$$\begin{aligned} 290 \cdot a_1 + 273 \cdot a_2 + 249 \cdot a_3 &= 273 \\ 273 \cdot a_1 + 290 \cdot a_2 + 273 \cdot a_3 &= 249 \\ 249 \cdot a_1 + 273 \cdot a_2 + 290 \cdot a_3 &= 220 \end{aligned}$$

is solved to get the optimized prediction coefficient $a_1 = 1.13$, $a_2 = -0.05$, and $a_3 = -0.16$. With these coefficients, the compression ratio for this specific example is improved to $90\,\%$.

5.2.2 Hardware-Accelerated ADPCM

The computational demand of DPCM is small compared to other predictive compression schemes such as the Free Lossless Audio Codec (FLAC) or Apple Lossless Audio Codec (ALAC) discussed in Section 7.1.1. However, the online-adaption of the prediction coefficients may overburden low-power MCUs, especially if multiple sensor channels have to be processed in parallel (see Section 7.2.2). This section thus describes the implementation of a ADPCM hardware kernel for the HaLoMote.

Figure 31 shows the structure of the ADPCM hardware kernel for compression of one sensor channel. The compression module is generic and can be configured for static or adaptive prediction. The gray modules of Figure 31 are used only for the adaptive prediction. The block buffer is realized by on-chip BRAM and is initially filled with a block of N samples. For the static DPCM compression, the block buffer is read

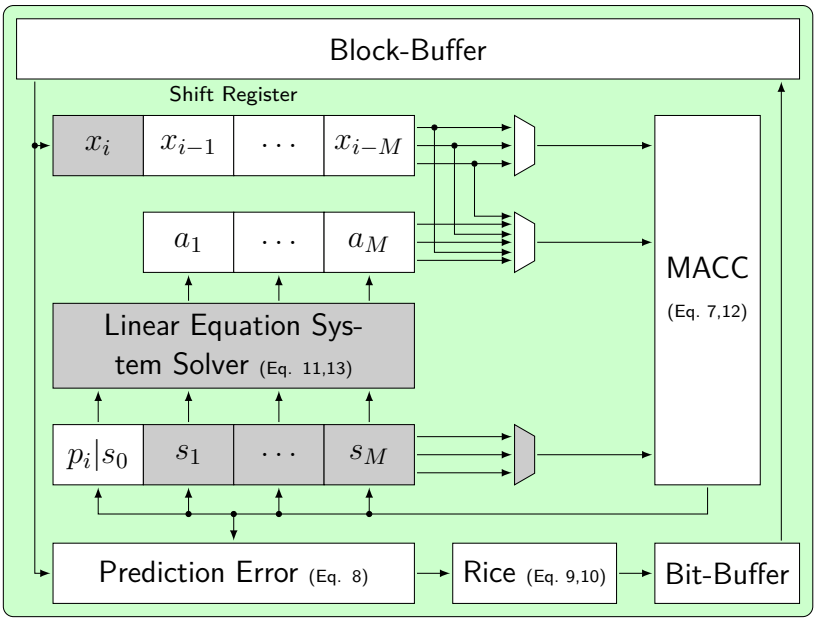


Figure 31: Structure of the generic ADPCM hardware kernel. The gray modules are dedicated to the online coefficient adaption

once to pass the uncompressed data through a shift register. After a new sample was inserted into this shift register, the prediction of the next sample is calculated according to Equation 7. By sequentially multiplexing the appropriate samples and coefficients to the Multiply-Accumulate (MACC) module, p_i is accumulated in M clock cycles. Afterwards, the prediction error is calculated and passed to the Rice coder to produce the bit sequence described by Equations 9 and 10. This bit sequence is sliced into bytes and written back to the block-buffer (now in compressed form) by a bit-buffer module. The Rice coder and the bit-buffer are running in parallel to the sequential predictor.

By using the same block-buffer for the uncompressed and the compressed data, an in-place compression approach is achieved. This allows for more ADPCM modules to be instantiated in parallel for multi-channel compression, as the BRAM of the AGL1000 FPGA used is limited to just 144 kbit. However, this approach presumes a compression ratio *smaller* than 100 % (i.e., compression actually *reduces* the data size) for each prefix of each sample block in the sample stream. If this constraint can not be guaranteed, separated input and output buffers have to be used. As the spikiness of many data streams (and thus the volume of post-compression data) is limited by the inertia of the underlying physical or chemical systems, the possibility to perform in-place compression is often useful.

For the adaptive *ADPCM* compression scheme, an additional coefficient optimization pass precedes the compression pass. During this pass over the block-buffer, the auto-correlation sums s_k of Equation 12 are accumulated. Again, the time-multiplexed MACC module is supplied with the appropriate operands from the sample shift register $(x_i, \ldots, x_{i-M})$ and the accumulator set $(s_0, \ldots, s_M)$. At the end of the pass, the accumulated auto-correlation sums s_k are used to generate the linear equation system that has to be solved to retrieve the prediction coefficients $(a_1, \ldots, a_M)$. As a trade-off between prediction accuracy and the time and energy spent to calculate the coefficients and the prediction values, fixed point arithmetic was chosen. The resulting coefficients

are stored in Q4.12 format. The solver supports first-order predictors ($M = 1$), which simplifies Equation 13 to

$$a_1 = \frac{r_1}{r_0} = \frac{s_1 \cdot N}{s_0 \cdot N - s_0}. \tag{14}$$

By restricting the block size N to a power of two, the single remaining integer division can be performed sequentially in 16 clock cycles. Higher predictor orders could be realized, but were not necessary for the use cases analyzed in Chapter 7.

5.2.3 Test Setup for Energy Efficiency Evaluation

The energy efficiency of the ADPCM kernel is evaluated for two concrete applications in Section 7.1 and 7.2. The test setup used to evaluate the energy efficiency of the in-sensor data compression is, however, application independent in principle, and is thus described in this section.

Figure 32 gives an overview of the HaLoMote test setup for multi-channel ADPCM compression. It is based on the inter-processor communication API described in Section 4.3. The raw data stream to be compressed is hold inside the ROM of the MCU. The data stream can thus be replayed to simplify the comparison of different ADPCM configurations. The data stream may consist of interleaved samples from multiple sensor channels, that are then distributed to multiple ADPCM kernels by a small *WRITE* kernel. The compressed data stream generated by the compression modules is sequentialized by the *READ* kernel and passed back to the MCU, which transmits it wirelessly. The HaLOEWEn radio stack allows parallelizing the radio transmission with the data transfer from the RCU to the MCU. Thus, the MCU duty cycle is not stretched by the inter-processor communication. This is important for the energy efficiency of the system (shorter duty cycles allow longer ultra-low-power sleep phases). Finally, the *COMPRESS* kernel controls the execution of the compression modules and tracks the number of generated output bytes.

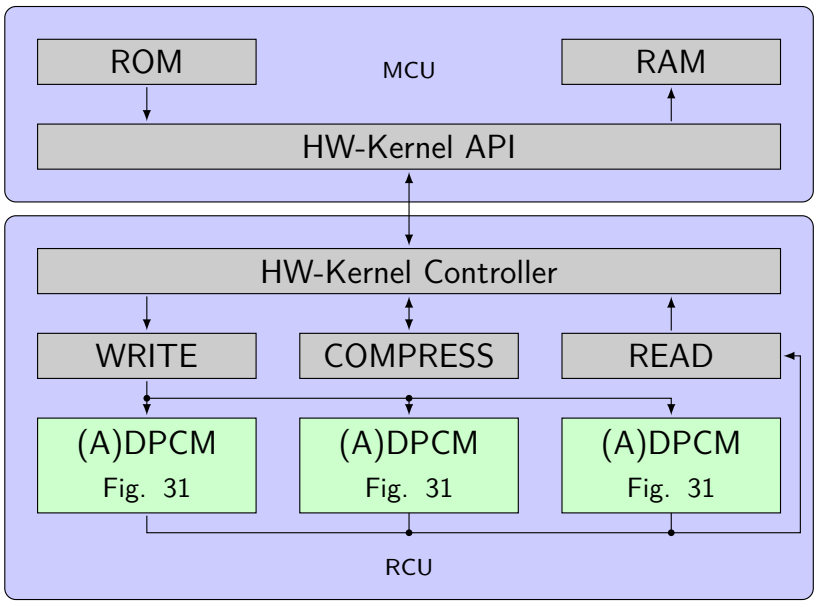


Figure 32: Test setup with multiple ADPCM hardware kernels

Channels	Scheme	BRAM	Core Cells	max Frequency
1	DPCM	8 (25 %)	1681 (7 %)	19.2 MHz
1	ADPCM	8 (25 %)	6728 (27 %)	10.5 MHz
3	DPCM	24 (75 %)	4419 (18 %)	22.4 MHz
3	ADPCM	24 (75 %)	14088 (57 %)	10.7 MHz

Table 8: Synthesis results for the Microsemi IGLOO AGL1000V2 device

Each hardware-accelerated data compression design was synthesized for the Microsemi [IGLOO] AGL1000V2 FPGA using Synplify Pro H-2013.03M-1 with retiming. The compression modules were configured for 16 bit samples, 2048 samples per block buffer and first order prediction, as required by both use cases analyzed in Chapter 7. As shown in Table 8, the design is primarily limited by the available memory and restricted to four channels at the given block size. Even so, some area remains for implementing higher order predictors.

To demonstrate the energy efficiency of the hardware-accelerated data compression, the measurement setup shown in Figure 33 is used. The HaLOEWEn3 mote is supplied by an external 3 V voltage source to power its internal components. The MCU drives the RCU into its low-power Flash*Freeze mode, as long as no hardware-accelerated computations are required. For precise time measurements, an external trigger is asserted by the MCU and recorded by an oscilloscope, as long as a compute task is executed. The average current drawn by the system during the task execution is measured by an Agilent [34411A] multimeter, which provides a resolution of 3 µA at a sampling frequency of 50 kHz.

For comparison with the hardware-accelerated encoders described in Section 5.2.2, the *DPCM* and *ADPCM* compression schemes were also implemented on the TI CC2531 MCU of the HaLOEWEn3 mote. The energy required for transmitting the uncompressed data stream is used as the baseline measurement. Each transmitted packet carries a maximum of 116 payload bytes, i.e., the maximum payload defined by IEEE [802.15.4].

As the runtime of the compression modules and hence the energy efficiency of the compression scheme is is affected by the nature of the data stream to be compressed, concrete results are presented in Chapter 7.

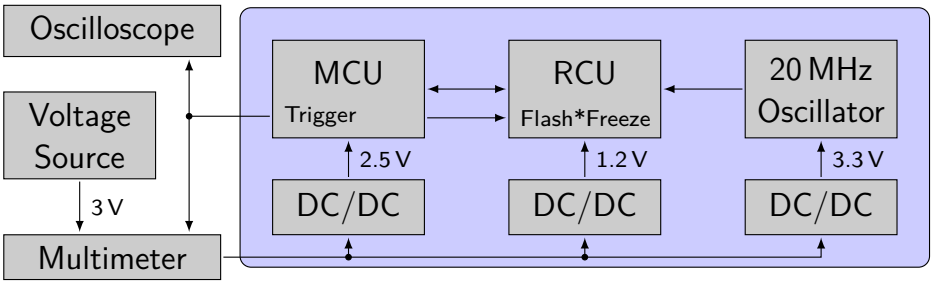


Figure 33: Measurement setup

5.3 Linear Regression

In this section, a resource-efficient formulation of the least-squares Linear Regression (LR) algorithm, as well as its software and hardware implementation is proposed. This approach has already been published in [Engel2015a].

LR can be applied in different WSN scenarios. As shown in Section 6.2, a wireless time synchronization protocol can use a regression-based clock drift estimation to compensate the slightly different oscillator frequencies of all network nodes. In addition, other WSN applications like node Received Signal Strength Indication (RSSI)-based node localization [Vanheel2011], predictive compression [Carvalho2011], or task scheduling [Ravinagarajan2010] have utilized the LR algorithm.

For a list of n data points $(x_k, y_k)_{k=1}^{n}$, the slope of the regression line fitting the points according to the least-squares method is given by

$$m := \frac{\sum_{k=1}^{n} (n \cdot x_k - sx) \cdot (n \cdot y_k - sy)}{\sum_{k=1}^{n} (n \cdot x_k - sx)^2} \tag{15}$$

$$\text{with} \qquad sx := \sum_{k=1}^{n} x_k \tag{16}$$

$$\text{and} \qquad sy := \sum_{k=1}^{n} y_k. \tag{17}$$

Directly implementing this formulation leads to a *linear* runtime complexity with narrow operators, as the dominating multiplications are performed just on differences between sums of data points.

If the LR is applied as a sliding window over a set of data points, i.e., the oldest data point is discarded as soon as a new data point is inserted, the Rolling Linear Regression (RLR) [Neely1966]

$$m := \frac{n \cdot sxy - sx \cdot sy}{n \cdot sxx - sx \cdot sx} \tag{18}$$

$$\text{with} \qquad sxy := \sum_{k=1}^{n} x_k \cdot y_k \tag{19}$$

$$\text{and} \qquad sxx := \sum_{k=1}^{n} x_k^2 \tag{20}$$

can be executed in *constant* time (see Algorithm 2). However, the dominating arithmetic operator of Equation 18 is the multiplication of two sums of data points, which requires a larger operator width than the operations required for Equation 15. Furthermore, additional memory is required to store the rolling sums sx, sy, sxy and sxx for the RLR approach.

To combine the constant runtime complexity of the RLR (Equation 18) with the reduced operator and memory complexity of the simple LR (Equation 15), a *novel* formulation of the *sliding window* LR called *Rolling Linear Regression with Coordinate Transformation (RLRCT)* is proposed in the next section. Although being application independent in principle, the benefits of the RLRCT compared to the RLR are based on the bit width optimization of the underlying operators. Therefore, applica-

tions employing the RLRCT have to specify an upper *limit for the step width* between subsequent data points:

$$L \coloneqq (L_x, L_y) \coloneqq \left(\max_{k \in \mathbb{N}} |x_{k+1} - x_k|, \max_{k \in \mathbb{N}} |y_{k+1} - y_k| \right) \tag{21}$$

5.3.1 Rolling Linear Regression with Coordinate Transformation

The main idea of the RLRCT is to move the origin of the considered coordinate system to the most recent data point in the sliding window, as shown in Figure 34. Let $(x_k, y_k)_{k=0}^{p}$ be the list of data points at a certain point in time. The coordinate transformation translates these data points into

$$(\bar{x}_{p,k}, \bar{y}_{p,k}) \coloneqq (x_p - x_k, y_p - y_k) \qquad \forall k \leq p. \tag{22}$$

This translation does not effect the slope calculated by the LR, but it limits the size of the numerical data to be processed to $n \cdot L$, where n still is the size of the sliding window. The individual arithmetic operations for the RLRCT can thus be implemented more efficiently than without applying the coordinate transformation.

The following formulas are stated only for the x values, but the same statements hold for the y values. To apply the proposed transformation to the rolling sums of the RLR with minimal effort, the RLRCT caches the translation steps

$$dx_k \coloneqq x_k - x_{k-1} \qquad \forall p - n < k \leq p \tag{23}$$

between the last n data points. The absolute coordinates $\bar{x}_{p,k}$ can be recovered for $p - n \leq k \leq p$ by accumulating the translation steps:

$$\begin{aligned} \sum_{i=k+1}^{p} dx_i &\overset{(Eq.\ 23)}{=} \sum_{i=k+1}^{p} x_i - \sum_{i=k+1}^{p} x_{i-1} \\ &= x_p + \sum_{i=k+1}^{p-1} x_i - \sum_{i=k}^{p-1} x_i = x_p - x_k \overset{(Eq.\ 22)}{=} \bar{x}_{p,k} \end{aligned} \tag{24}$$

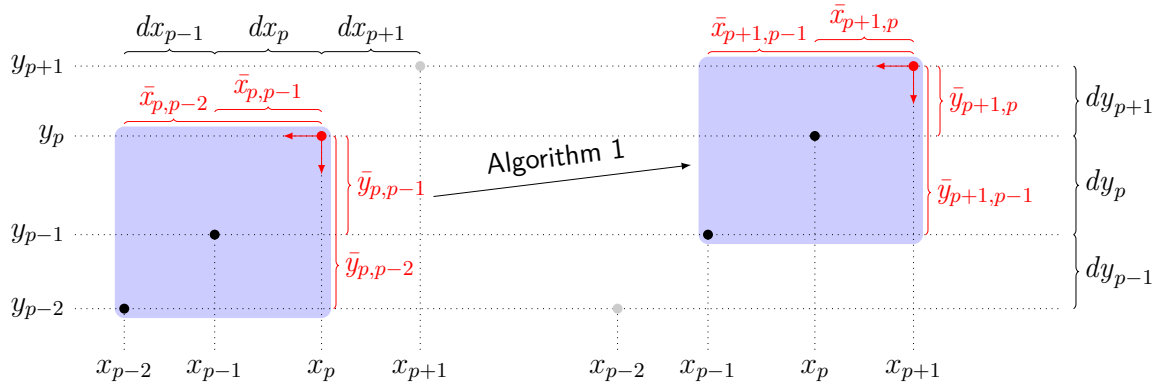


Figure 34: Movement of sliding window and coordinate system by RLRCT update

When moving the sliding window to the next data point x_{p+1}, the coordinate transformation (i.e., translation by dx_{p+1}) must be applied to all coordinates, as

$$\bar{x}_{p+1,k} \overset{(Eq.\ 24)}{=} \sum_{i=k+1}^{p+1} dx_i = dx_{p+1} + \sum_{i=k+1}^{p} dx_i \overset{(Eq.\ 24)}{=} dx_{p+1} + \bar{x}_{p,k} \tag{25}$$

As RLR does in Equation 18, RLRCT also relies on updating the rolling sum

$$\begin{aligned} sx_p &:= \sum_{k=p-n+1}^{p} \bar{x}_{p,k} && (26)\\ &= \underbrace{\bar{x}_{p,p}}_{0} - \bar{x}_{p,p-n} + \sum_{k=p-1-n+1}^{p-1} \bar{x}_{p,k} \\ &\overset{(Eq.\ 25)}{=} -\bar{x}_{p,p-n} + \sum_{k=p-1-n+1}^{p-1} dx_p + \bar{x}_{p-1,k} \\ &\overset{(Eq.\ 26)}{=} n \cdot dx_p - \bar{x}_{p,p-n} + sx_{p-1} && (27) \end{aligned}$$

Instead of explicitly calculating $\bar{x}_{p,p-n}$, another rolling sum is introduced as

$$\begin{aligned} sdx_p &:= \bar{x}_{p,p-n} \overset{(Eq.\ 24)}{=} \sum_{i=p-n+1}^{p} dx_i \\ &= dx_p - dx_{p-n} + sdx_{p-1} && (28) \end{aligned}$$

The sum of coordinate products

$$\begin{aligned} sxy_p &:= \sum_{k=p-n+1}^{p} \bar{x}_{p,k} \cdot \bar{y}_{p,k} \\ &= \underbrace{\bar{x}_{p,p} \cdot \bar{y}_{p,p}}_{0} - \bar{x}_{p,p-n} \cdot \bar{y}_{p,p-n} + \sum_{k=p-n}^{p-1} \bar{x}_{p,k} \cdot \bar{y}_{p,k} \\ &\overset{(Eq.\ 25,28)}{=} -sdx_p \cdot sdy_p + \sum_{k=p-n}^{p-1} (dx_p + \bar{x}_{p-1,k})(dy_p + \bar{y}_{p-1,k}) \\ &\overset{(Eq.\ 26)}{=} sxy_{p-1} + dx_p \cdot sy_{p-1} + dy_p \cdot sx_{p-1} + n \cdot dx_p \cdot dy_p - sdx_p \cdot sdy_p && (29) \end{aligned}$$

is not actually handled as rolling sum in RLRCT. Instead, the entire numerator of Equation 18 is accumulated as

$$
\begin{aligned}
num_p &:= n \cdot sxy_p - sx_p \cdot sy_p \\
&\overset{(Eq.\ 27)}{=} n \cdot sxy_p - (n \cdot dx_p - sdx_p + sx_{p-1}) \cdot (n \cdot dy_p - sdy_p + sy_{p-1}) \\
&\overset{(Eq.\ 29)}{=} n \cdot sxy_{p-1} - sx_{p-1} \cdot sy_{p-1} - (n+1) \cdot sdx_p \cdot sdy_p \\
&\quad + sdx_p(n \cdot dy_p + sy_{p-1}) + sdy_p(n \cdot dx_p + sx_{p-1}) \\
&\overset{(Eq.\ 27)}{=} num_{p-1} - (n+1) \cdot sdx_p \cdot sdy_p + sdx_p(sdy_p + sy_p) + sdy_p(sdx_p + sx_p) \\
&= num_{p-1} - (n-1) \cdot sdx_p \cdot sdy_p + sdx_p \cdot sy_p + sdy_p \cdot sx_p
\end{aligned} \tag{30}
$$

The denominator of Equation 18 is accumulated accordingly:

$$
den_p = den_{p-1} - (n-1) \cdot sdx_p^2 + 2 \cdot sdx_p \cdot sx_p \tag{31}
$$

Algorithm 1 summarizes the RLRCT update procedure. In Line 1 and 2, the translation vector between the old and the new origin of the coordinate system is calculated, before the origin is actually translated in Line 3 and 4. In Line 5, the sliding window is moved by enqueuing the new data point and dequeuing the oldest entry. As no random access to the intermediate data points is required by the RLRCT, the sliding window is implemented as a First In, First Out (FIFO) structure F. The remaining Lines 6 to 12 realize the equations derived above to properly apply the translation vector to all rolling sums.

Algorithm 1: Update procedure for Rolling Linear Regression with Coordinate Transformation

Input: New data point (x, y)
Input: Previously received data point $(\hat{x}, \hat{y})$
Input: Accumulators sdx, sdy, sx, sy, num, den
Input: Sliding Window F
Output: Slope represented as fraction of num and den

1 $dx := x - \hat{x}$;
2 $dy := y - \hat{y}$;
3 $\hat{x} := x$;
4 $\hat{y} := y$;
5 $(\hat{dx}, \hat{dy}) \leftarrow F \leftarrow (dx, dy)$; `// move window`
6 $sdx \mathrel{+}= dx - \hat{dx}$; `// Equation 28`
7 $sdy \mathrel{+}= dy - \hat{dy}$;
8 $sx \mathrel{+}= n \cdot dx - sdx$; `// Equation 27`
9 $sy \mathrel{+}= n \cdot dy - sdy$;
10 $t := (n-1) \cdot sdx$;
11 $num \mathrel{+}= sdx \cdot sy + sdy \cdot sx - t \cdot sdy$; `// Equation 30`
12 $den \mathrel{+}= sdx \cdot sx \cdot 2 \quad - t \cdot sdx$; `// Equation 31`

Algorithm 2: Update procedure for Rolling Linear Regression (without Coordinate Transformation)

Input: New data point point (x, y)
Input: Accumulators sx, sy, sxy, sxx
Input: Sliding Window FT
Output: Slope represented as fraction of num and den

1 $xy \coloneqq x \cdot y$;
2 $xx \coloneqq x \cdot x$;
3 $(\hat{x}, \hat{y}, \hat{xy}, \hat{xx}) \leftarrow F \leftarrow (x, y, xy, xx)$; `// move window`
4 $sx \mathrel{+}= x - \hat{x}$;
5 $sy \mathrel{+}= y - \hat{y}$;
6 $sxy \mathrel{+}= xy - \hat{xy}$;
7 $sxx \mathrel{+}= xx - \hat{xx}$;
8 $num \mathrel{+}= n \cdot sxy - sx \cdot sy$; `// Equation 18`
9 $den \mathrel{+}= n \cdot sxx - sx \cdot sx$;

Algorithm 1 consists of 15 additions and nine multiplications (four of them with a small constant). As shown in Algorithm 2, it is assumed that the RLR buffers the additional values

$$(xy_k, xx_k)_{k=p-n+1}^{p} \coloneqq (x_k \cdot y_k, x_k \cdot x_k)_{k=p-n+1}^{p} \tag{32}$$

for the sliding window to avoid the recomputation of the products required in Line 6 and 7. The RLR requires fewer operations (10 additions, six multiplications) than the RLRCT, but most of the RLRCT arithmetic operates on differences between subsequent data points, instead of absolute data points. If the step width limitation (Equation 21) is significantly smaller than the absolute data points, the bit width of the RLRCT operations can be reduced. As will be shown in Section 6.2.3, this reduction in operator size can amortize the cost of the larger number of operations, thus leading to a more efficient linear regression implementation. Furthermore, the RLRCT requires less memory than the RLR to buffer the sliding window, as only two translation vectors have to be stored per entry instead of two full data points and two data point products.

After execution of the update procedure, the slope of the linear regression line is represented by num and den. The actual division is not executed by the update procedure, to avoid losing accuracy at that point. Instead of representing m as a fixed-point variable that can be multiplied with an value $\boldsymbol{\nu}$ later on, the slope division is scheduled after this multiplication, i.e., $\boldsymbol{\nu} \cdot m = (\boldsymbol{\nu} \cdot num)/den$.

5.3.2 Software Implementation

To analyze the benefits of the proposed RLRCT, Algorithm 1 and 2 as well as Equation 15, and an optimized version of Equation 15 for $n = 2$ was implemented on the CC2531 MCU of the HaLOEWEn3.

Listing 1: 8051 assembler for a sample 24 bit $\times$ 16 bit multiplication $x \cdot y = z$ in the directly addressable memory

```
1  MOV A,_x+0   MOV B,_y+0   MUL AB                          MOV _z+0,A ; z[0]=x[0]*y[0]
2                                                           MOV  R1,B  ; R1=carry
3
4  MOV A,_x+1   MOV B,_y+0   MUL AB    ADD  A,R1             MOV _z+1,A ; z[1]=x[1]*y[0]+carry
5                            CLR A     ADDC A,B              MOV  R1,A  ; R1=carry
6
7  MOV A,_x+2   MOV B,_y+0   MUL AB    ADD  A,R1             MOV _z+2,A ; z[2]=x[2]*y[0]+carry
8                            CLR A     ADDC A,B              MOV _z+3,A ; z[3]=carry
9                            CLR A                           MOV _z+4,A ; z[4]=0
10
11 MOV A,_x+0   MOV B,_y+1   MUL AB    ADD  A,_z+1           MOV _z+1,A ; z[1]+=x[0]*y[1]
12                           MOV A,B   ADDC A,_z+2           MOV _z+2,A ; z[2]+=carry
13                           CLR A     ADDC A,_z+3           MOV _z+3,A ; z[3]+=carry
14                           CLR A     ADDC A,_z+4           MOV _z+4,A ; z[4]+=carry
15
16 MOV A,_x+1   MOV B,_y+1   MUL AB    ADD  A,_z+2           MOV _z+2,A ; z[2]+=x[1]*y[1]
17                           MOV A,B   ADDC A,_z+3           MOV _z+3,A ; z[3]+=carry
18                           CLR A     ADDC A,_z+4           MOV _z+4,A ; z[4]+=carry
19
20 MOV A,_x+2   MOV B,_y+1   MUL AB    ADD  A,_z+3           MOV _z+3,A ; z[3]+=x[2]*y[1]
21                           MOV A,B   ADDC A,_z+4           MOV _z+4,A ; z[4]+=carry
```

The 8051-ISA compliant compilers (e.g., [SDCC], [IAR] or [Keil]) support up to 64 bit integer arithmetic operations with an operator granularity of 8 bit, 16 bit, 32 bit and 64 bit. As the main benefits of the RLRCT algorithm arise from optimized narrowed operator sizes, these four levels of operator sizes are not sufficiently fine-grained. Thus, the required operators (i.e., shift, inversion, accumulation, subtraction, multiplication, division, and movement between memory regions) where implemented in 8051 assembler with up to 96 bit wide representations at a granularity of 8 bit. Operands and results may be constants or arrays allocated in the directly (*data*) or indirectly (*xdata*) addressable memory region of the 8051 MCU. All intermediate variables and accumulators are located in the directly addressable memory and the assembler routines operate on that memory without accessing the stack. The implementation is optimized for execution speed at the expense of a larger code memory footprint. An example for a 24 bit $\times$ 16 bit multiplication is shown in Listing 1.

5.3.3 Hardware Implementation

As stated above, the benefits of the RLRCT arise from fine-grained operator size optimization. Furthermore, the data-dependencies between the subsequent RLRCT-operations allow for parallel executions of Lines 1 and 2, 3 and 4, 6 and 7, 8 and 9, as well as 11 and 12 of Algorithm 1. The RLRCT is thus well-suited for hardware acceleration.

Three different architectures were implemented to trade-off execution speed against resource requirements. All architectures buffer the data points (i.e., the sliding window F) in BRAM and perform the slope division as sequential shift-subtract steps. All additions and multiplications are executed as single-cycle combinatorial logic. In the *Fully Parallel* architecture, every arithmetic operation of Algorithm 1 is implemented with dedicated logic. The data dependencies result in a seven cycle data path for the LR. In the *Single MAC* architecture, a single multiply-accumulate operator is

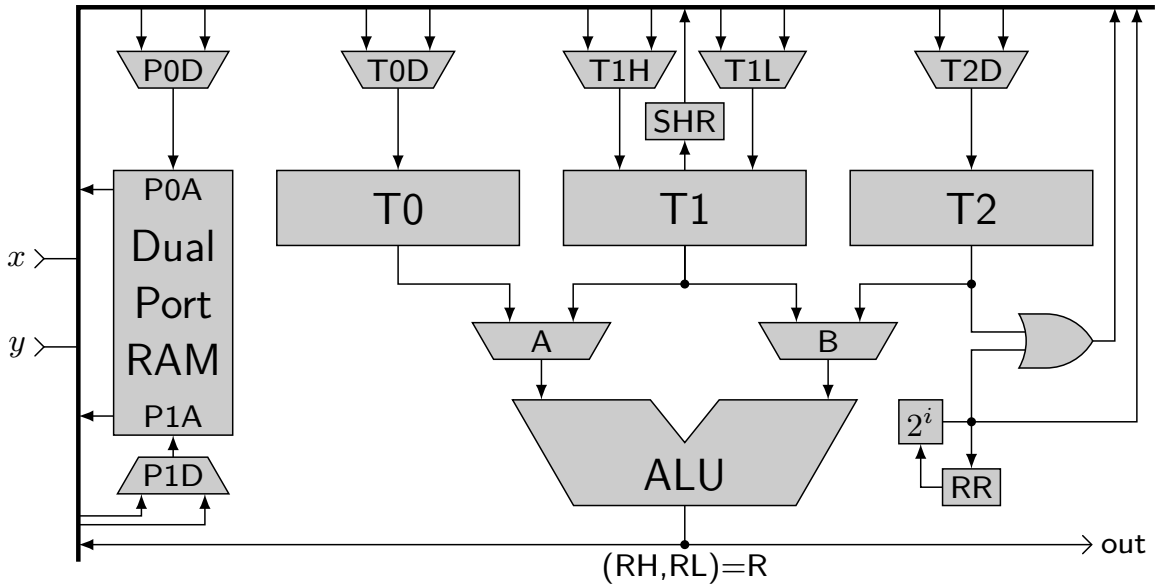


Figure 35: HW Kernel (µArchitecture) for RLRCT

implemented and the registers buffering the accumulators and intermediate results are sequentially multiplexed to this operator, resulting in a 21 cycle computation for the RLRCT.

Finally, the micro-coded *µArchitecture* shown in Figure 35 buffers all accumulators next to the regression table in a dual-port BRAM. To efficiently utilize the BRAM, the larger accumulators (i.e., *num* and *den*) occupy two memory locations (i.e., high and low word). Three double word registers (T0, T1, T2) can be fed with data from the BRAM, inputs of the HW kernels (i.e., x, y), results from previous ALU operations, specific constants, or some derived values required for the slope division. Two registers are selected as operands (A, B) for the ALU, which only supports the required basic operations. Besides computing the sum, difference, product, and absolute difference of A and B, a fraction $\frac{A}{B}$ can be reduced from double to single word precision by right-shifting the numerator and denominator by an appropriate (data-dependent) width.

As shown in Table 9, the 26 bit wide *µInstructions* used to control the µArchitecture are grouped into fields configuring the BRAM addressing (i.e., P0A, P1A), multiplexers (i.e., P0D, P1D, T0D, T1H, T1L, T2D, A, B) as well as the ALU operation. Various efforts have been made to keep the µInstructions compact. For example, not all BRAM addresses are available at both ports, not all registers can be mapped to both ALU operands, and the port / register write enable information is integrated into the related multiplexer configurations. To simplify the control logic of the µArchitecture, the latencies resulting from BRAM and register access have to be considered when coding the µInstructions. Furthermore, predication logic (i.e., T0D = T2D = 101) is used to implement sequential (non-performing restoring) divisions without control flow branches. Each write access to the sliding window F inside the BRAM increments an internal circular counter thus combining the push and pop operation for dx and dy as required by Algorithm 1. Each µInstruction is executed in one RCU clock cycle.

Table 10 lists the 28 µInstructions required to implement Algorithm 1. After an initiation interval of two instructions, the ALU is utilized in every instruction. Ten of

Bits	Field	Description	Interpretation	
25 - 22	P0A	RAM P0 address	0000 : $\hat{x}$ 0001 : $\hat{y}$ 0010 : sdx 0011 : sdy 0100 : sx 0101 : sy 0110 : num low 0111 : den low	1000 : $rnum = num/2^K$ 1001 : $rden = den/2^K$ 1010 : $diff = \lvert rnum - rden \rvert$ 1011 : temp 1100 : FIFO F for dx and dy 1101 : unused 1110 : unused 1111 : unused
21 - 20	P0D	RAM P0 source	00 : no write to P0 01 : RL	10 : x 11 : y
19 - 17	P1A	RAM P1 address	000 : num high 001 : den high 010 : $rden$ 011 : temporary	100 : $neg = sgn(rnum - rden)$ 101 : unused 110 : unused 111 : unused
16 - 15	P1D	RAM P1 source	00 : no write to P1 01 : RH	10 : RL 11 : unused
14 - 12	T0D	REG T0 source	000 : no write to T0 001 : x 010 : y 011 : P0	100 : R 101 : R if R ≥ 0 110 : RL 111 : unused
11	T1H	REG T1 high source	0 : RH	1 : P1
10 - 8	T1L	REG T1 low source	000 : no write to T1 001 : RL 010 : RH 011 : P0	100 : $n - 1$ 101 : SHR(T1) 110 : unused 111 : unused
7 - 5	T2D	REG T2 source	000 : no write to T2 001 : R 010 : P0 011 : n	100 : 0 101 : T2 | 2^i if R ≥ 0 110 : 2^i 111 : unused
4	A	ALU operand A	0 : T0	1 : T1
3	B	ALU operand B	0 : T1	1 : T2
2 - 0	OP	ALU opcode	000 : $A + B$ 001 : $A - B$ 010 : $A \cdot B$ 011 : $(A/2^K, B/2^K)$	100 : $(A < B, \lvert A - B \rvert)$ 101 : T1 < 0 ? $A + B : A - B$ 110 : T1 < 0 ? $A - B : A + B$ 111 : unused

Table 9: 26 bit μInstruction controlling the RLRCT μArchitecture

the 28 instructions actually utilize both BRAM ports. The numerator *num* and denominator *den* representing the slope of the regression line are produced by the ALU in Instruction 21 and 26. Two additional instructions are used to prepare the multiplication of the slope with a certain value ν as required for the time synchronization application described in Section 6.2. Instruction 27 reduces the size of the numerator (to *rnum*) and denominator (to *rden*) by a common factor 2^K chosen large enough, such that *rnum* and *rden* fit into single words inside the BRAM. Instruction 28 derives

#	P0A	P0D	P1A	P1D	T0D	T1H	T1L	T2D	A	B	OP	R	T0	T1	T2	P0	P1	RAM
1	0000	10	000	00	000	0	000	000	0	0	000					$\hat{x}$		$\hat{x}$=x
2	0010	00	000	00	001	0	011	000	0	0	000		x	P0		sdx		
3	1100	01	000	00	011	0	001	000	0	0	001	T0 − T1	P0	R		F		F++=RL
4	0100	00	000	00	100	0	000	010	0	0	000	T0 + T1	R		P0	sx		
5	0010	01	000	00	011	0	000	001	0	1	001	T0 − T2	P0		R			sdx=RL
6	0001	11	000	00	100	0	000	011	0	1	001	T0 − T2	R		n	$\hat{y}$		$\hat{y}$=y
7	0011	00	000	00	000	0	011	001	1	1	010	T1 · T2		P0	R	sdy		
8	0100	01	000	00	010	0	000	010	0	1	000	T0 + T2	y		P0			sx=RL
9	1100	01	000	00	000	0	001	000	0	0	001	T0 − T1		R		F		F++=RL
10	0101	00	000	00	100	0	000	010	1	1	000	T1 + T2	R		P0	sy		
11	0011	01	000	00	011	0	000	001	0	1	001	T0 − T2	P0		R			sdy=RL
12	0110	00	000	00	100	0	000	011	0	1	001	T0 − T2	R		n	num low	num high	
13	0010	00	000	00	000	1	011	001	1	1	010	T1 · T2		(P1,P0)	R	sdx		
14	0101	01	000	00	100	0	000	010	0	1	000	T0 + T2	R		P0			sy=RL
15	0000	00	000	00	100	0	000	000	0	1	010	T0 · T2	R					
16	0011	00	000	00	100	0	100	000	0	0	000	T0 + T1	R	$n-1$		sdy		
17	1011	01	000	00	000	0	001	010	1	1	010	T1 · T2		R	P0			temp=RL
18	0100	00	000	00	000	0	001	000	1	1	010	T1 · T2		R		sx		
19	0000	00	000	00	100	0	011	000	0	0	001	T0 − T1	R	P0				
20	0010	00	000	00	000	0	000	001	1	1	010	T1 · T2			R	sdx		
21	0110	01	000	01	000	0	000	010	0	1	000	T0 + T2			P0			num=R
22	0111	00	001	00	000	0	001	000	1	1	010	T1 · T2		R		den low	den high	
23	1011	00	000	00	100	1	011	000	1	0	000	T1 + T1	R	(P1,P0)		temp		
24	0000	00	000	00	100	0	011	000	0	0	000	T0 + T1	R	P0				
25	0110	00	000	00	000	0	001	000	1	1	010	T1 · T2		R		num low	num high	
26	0111	01	001	01	100	1	011	000	0	0	001	T0 − T1	R	(P1,P0)				den=R
27	1000	01	010	01	110	0	010	000	0	0	011	$(\frac{T0}{2^K}, \frac{T1}{2^K})$	RL	RH				$rden$=RH, $rnum$=RL
28	1010	01	100	01	000	0	000	000	0	0	100	(T0 < T1, |T0 − T1|)						neg=RH, $diff$=RL

Table 10: μInstruction sequence implementing Algorithm 1 on the μArchitecture

the signum $neg = sgn(rnum - rden)$ and the absolute differenz $diff = |rnum - rden|$, which are used later on to calculate

$$\boldsymbol{\nu} \cdot \frac{num}{den} = \boldsymbol{\nu} \cdot \frac{rnum}{rden} = \boldsymbol{\nu} + (-1)^{neg} \cdot \frac{\boldsymbol{\nu} \cdot diff}{rden}.$$

For $\boldsymbol{\nu} \geq 0$, the final sequential division can be executed without explicit sign analysis, as both $\boldsymbol{\nu} \cdot diff$ and $rden$ are not negative. Furthermore, for slopes near one, i.e., $rnum \approx rden$ as expected in Section 6.2, $\boldsymbol{\nu} \cdot diff$ is much smaller than $\boldsymbol{\nu} \cdot rnum$, thus further simplifying the sequential division.

For the evaluation of the resource and energy efficiency of the software and hardware implementations of the RLRCT, a concrete implementation defining the actual operator sizes is required. These results can be found in Section 6.2.3.

CHAPTER 6
Network Communication Primitives

In this chapter, primitives related to the wireless network communication are presented. Although motivated by the target application described in Section 7.3, the routing and synchronization mechanisms are reusable for similar applications. Parts of this chapter have already been published in [Engel2014b; Engel2015a].

6.1 Transceiver Scheduling for Multi-Source Flooding

Routing protocols have a significant impact on distributed applications relying on multi-hop wireless communication. The number of nodes involved to deliver an information (a data item) from a source to a sink determines the networks overall energy consumption as well as the data throughput. Often, shortest routes can not be easily discovered, or are undesirable (e.g., when balancing communication load equally over all network nodes or to provide robust communication over redundant paths). In general, application-tailored routing protocols can exploit special characteristics to improve throughput and energy consumption.

Flooding information into a network by multi-hop rebroadcasting is simple and robust, but also very inefficient [Ni1999]. As stated in Section 3.2.2, many application-independent improvements of the basic blind flooding have been proposed during the last decade to reduce the number of required rebroadcasts. Even better results can be achieved by exploiting some (limited) knowledge about the communication needs of the concrete applications. Even so, the approach presented here is applicable for a wide spectrum of use cases.

The following assumptions are derived from the SHM application described in Section 7.3, but they hold true for many monitoring scenarios. First of all, a monitored object is either excited or not at any point in time. If relevant information is detected at a certain position on the object, it is quite likely that nearby sensors are also detecting events. Thus, instead of assuming communication originating from a single source node, parallel multi-source information flooding should be supported. Detected events can often be characterized just by a few bits specifying the time and place where a specific activity threshold was exceeded. It can thus be assumed that information from multiple source nodes can be *merged* into a single message. Furthermore, if the sensor nodes are attached to a structure, the network topology can be assumed to be static (apart from node failures). And finally, as most distributed monitoring application already rely on a highly accurate time synchronization mechanism for correct data gathering (see Section 6.2), an accurate scheduling of data reception and transmission can also be achieved without additional effort.

As the network topology is assumed to be static and the nodes are time-synchronized to each other, a schedule can be derived that selects appropriate nodes as transmitters and receivers in subsequent time slots (cycles). This static schedule must be known to all nodes and is executed periodically, i.e., the source nodes buffer collected information until the restart of the schedule. The main advantage of this scheme is twofold. First, all nodes not scheduled for reception in a certain cycle can keep their radios turned off

as they do not have to listen for potential incoming data. Second, knowledge of the global topology can be used to deliberately aggregate information from specific source nodes.

Due to the spatial and temporal correlation of the sensor nodes, at each restart of the schedule it is most likely that either *all* source nodes have buffered some information, or that *no* source node has buffered any information. In the first case, the schedule can be executed as intended. In the second case, the source nodes actually scheduled for transmission will not actually start sending, which must be recognized at the receivers scheduled for listening by imposing a short timeout. These receivers, in turn, cancel their scheduled forwarding transmission, thus propagating the transmission avoidance until the end of the schedule. The energy overhead for an unused schedule is thus restricted to a small number of clear channel assessments at the scheduled receivers. The same mechanism of transmission avoidance must be applied if only a *subset* of source nodes provides new information at the start of a new schedule. In this case, the resulting flooding routes may not be optimal for the specific subset of source nodes. However, the probability of those cases is small due to the spatial and temporal correlations between the data sources, as stated above. The overall energy efficiency of the flooding scheme will thus not be significantly affected.

The reminder of this section focuses on calculating an optimal s-to-all flooding schedule for a given network topology.

6.1.1 Network Model and Cost Function

An application independent network model M is defined as

$$M = (N, N_0, E_I, E_C, h) \tag{33}$$
$$N \subseteq \mathbb{N} \tag{34}$$
$$N_0 \subseteq N \tag{35}$$
$$E_I \subseteq N^2 \tag{36}$$
$$E_C \subset E_I \tag{37}$$
$$h : N^2 \to \mathbb{N}. \tag{38}$$

This model combines two directed graphs $M_I = (N, E_I)$ and $M_C = (N, E_C)$ with a common set of network nodes N. M_C represents the networks single-hop connectivity, i.e., node $b \in N$ may receive information directly from node $a \in N$ iff $(a, b) \in E_C$. M_I represents the interference relationship within the wireless network, i.e., the transmission of node a prevents the successful reception of any other information at node b iff $(a, b) \in E_I$. M_C is thus a subgraph of M_I. The information-generating source nodes are denoted as N_0. The function h determines the minimum number of hops between two nodes. These hop counts h are precomputed by the all-pairs-shortest-path algorithm [Floyd1962]. For clarity, a will be used for nodes acting as transmitters, b for nodes acting as receivers, s for source nodes and ν for arbitrary network nodes throughout this section.

The network model M can be generated from the known locations and the transmission and reception characteristics of all network nodes, as shown in Section 6.1.4. A more accurate model accounting for obstacles can be obtained from well known online

topology discovery algorithms (e.g., recursive neighborhood exchange [Dheap2003]), executed once after network deployment. However, detailed topology detection is outside the scope of this thesis and M is thus assumed to be known and static.

A schedule S and an information assignment function I are defined as

$$S : N \times \mathbb{N} \rightarrow \{RX, TX, IDLE\} \tag{39}$$

$$I : N \times \mathbb{N} \rightarrow 2^{N_0} \tag{40}$$

Thus, a schedule S assigns a radio transceiver status $S(v, t)$ to each network node $v \in N$ in every scheduling cycle $t \in \mathbb{N}$. The information assignment function $I(v, t)$ denotes the subset of source nodes that have already forwarded their information to node v before cycle t such that the information is known to v in cycle t. Initially, only the source nodes know about their information:

$$I(v, 0) = \begin{cases} \{v\}, & \text{if } v \in N_0 \\ \emptyset, & \text{otherwise} \end{cases} \tag{41}$$

The information is then propagated within each cycle t from any node a scheduled for transmission to any node b within the transmission range of a if b is scheduled for reception and not disturbed by another transmitter c:

$$I(b, t+1) = I(b, t) \tag{42}$$
$$\cup \begin{cases} I(a, t), & \text{if } (a, b) \in E_C \ \wedge \\ & S(a, t) = TX \ \wedge \\ & S(b, t) = RX \ \wedge \\ & \forall (c, b) \in E_I \setminus \{(a, b)\} : S(c, t) \neq TX \\ \emptyset, & \text{otherwise} \end{cases}$$

The length $L(S)$ of a schedule S is the minimum number of cycles required to flood all source node information to all nodes in the network:

$$L(S) = \min_{t \in \mathbb{N}} \forall v \in N : I(v, t) = N_0 \tag{43}$$

The sum of scheduled receptions and transmissions are defined as the primary optimization goal to minimize the networks overall energy consumption $C(S)$ for a schedule S:

$$C(S) = \sum_{t=0}^{L(S)-1} |\{v \in N : S(v, t) \neq IDLE\}| \tag{44}$$

Radio transceivers draw about the same power when receiving and transmitting data. Thus, receiving and transmitting the same amount of data consumes about the same amount of energy. The cost function therefore does not distinguish between transmission and reception costs. However, weighting factors could be integrated as needed.

For two schedules of equal cost, the shorter schedule yields smaller latency and increased data throughput. Therefore, the schedule length is the secondary optimization

goal. Definition 1 summarizes the optimization problem to be solved in Section 6.1.2 and 6.1.3.

Definition 1 (*Optimal Flooding Schedule*)

For any network model M find a schedule S of finite length, such that

$$C(S) < C(\hat{S}) \vee \left(C(S) = C(\hat{S}) \wedge L(S) \leq L(\hat{S})\right) \tag{45}$$

for any valid schedule $\hat{S}$.

6.1.2 Integer Linear Program for Optimal Scheduling

Finding an optimal solution for the problem stated in Definition 1 requires solving multiple set-cover problems, e.g., finding the smallest number of forwarding nodes to cover the remaining, not yet reached nodes. The optimal scheduling problem is thus NP-complete. To find an optimal solution for non-trivial network sizes in a reasonable time, parallel branch-and-bound algorithms have to be utilized. The network model (Equation 33) and the information propagation rules (Equations 41 to 44) are thus translated into an Integer Linear Program (ILP). A commercial ILP-solver is then utilized to compute the optimal schedule (see Section 6.1.4).

The ILP-solver determines integer values for a set of variables. The solution space is restricted by a set of constraints, which are relations between linear combinations of the variables and constant values. Within this solution space, a single objective is minimized. This objective is also a linear combination of the variables.

When working with binary variables, basic operations like disjunctions and conjunctions have to be replaced by the following constraints:

$$y = \bigvee_{i=1}^{n} x_i \quad \Leftrightarrow \quad y \geq x_1 \wedge \ldots \wedge y \geq x_n \wedge y \leq \sum_{i=1}^{n} x_i \tag{46}$$

$$y = \bigwedge_{i=1}^{n} x_i \quad \Leftrightarrow \quad y \geq \sum_{i=1}^{n} x_i - (n-1) \wedge n \cdot y \leq \sum_{i=1}^{n} x_i \tag{47}$$

To formulate the ILP for the optimal scheduling, the following binary variables are used:

$$\alpha_{a,t} \quad \Leftrightarrow \quad S(a,t) = TX \tag{48}$$

$$\beta_{b,t} \quad \Leftrightarrow \quad S(b,t) = RX \tag{49}$$

$$\gamma_t \quad \Leftrightarrow \quad \exists \boldsymbol{\nu} \in N : S(\boldsymbol{\nu},t) \neq IDLE \tag{50}$$

$$\delta_{s,\nu,t} \quad \Leftrightarrow \quad s \in I(\boldsymbol{\nu},t) \tag{51}$$

$$\epsilon_{s,a,b,t} \quad \Leftrightarrow \quad \delta_{s,a,t} \wedge \alpha_{a,t} \wedge \beta_{b,t} \tag{52}$$

To define the necessary constraints for each schedule cycle t, an upper bound $L_{\max}$ for the length of the resulting schedule is required. This upper bound can be derived from the network model, as a naive sequential blind flooding of all information would not require more then $|N| \cdot |N_0|$ cycles. Alternatively, the outcome of the heuristic

scheduling in Section 6.1.3 can be used to define tighter bounds for the schedule length. This is important, as the ILP formulation requires a total number of $L_{\max} \cdot (2|N| + 1 + |N_0|(|N| + |E_C|))$ variables.

Let $T = \{0, \ldots, L_{\max} - 1\}$ be the set of potential schedule cycles. The objective to be minimized is described as

$$\textbf{Minimize} \quad L_{\max} \cdot \sum_{t \in T} \sum_{v \in N} (\alpha_{v,t} + \beta_{v,t}) + \sum_{t \in T} \gamma_t \tag{53}$$

to satisfy Equation 45. The right sum calculates the number of actually required cycles $L(S)$. The complete distribution of information tested in Equation 43 is ensured by constraints (Equation 59). The left sum calculates $C(S)$ according to Equation 44. It is weighted by $L_{\max}$ to make $C(S)$ the primary objective.

The following constraints are required to distinguish between active and idle cycles:

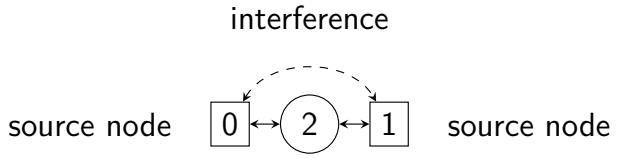
$$\forall t \in T, v \in N : \gamma_t \geq \alpha_{v,t} \tag{54}$$
$$\forall t \in T, v \in N : \gamma_t \geq \beta_{v,t} \tag{55}$$
$$\forall t \in T : \gamma_t \leq \sum_{v \in N} (\alpha_{v,t} + \beta_{v,t}) \tag{56}$$
$$\forall t > 0 : \gamma_t \leq \gamma_{t-1} \tag{57}$$

According to Equation 46, $\gamma_t = \bigvee_{v \in N} \alpha_{v,t} \vee \beta_{v,t}$ is realized by Equations 54 to 56. Thus, a cycle is active, iff any node is scheduled for transmission or reception in this cycle. Equation 57 moves all idle cycles to the end of T.

The following constraints ensure correct information propagation:

$$\forall s \in N_0, t \in T : \delta_{s,v,t} = 1 \tag{58}$$
$$\forall s \in N_0, v \in N \setminus \{s\} : \delta_{s,v,L_{\max}-1} = 1 \tag{59}$$
$$\forall s \in N_0, v \in N \setminus \{s\}, t < h(s, v) : \delta_{s,v,t} = 0 \tag{60}$$
$$\forall s \in N_0, b \in N \setminus \{s\}, t \geq h(s, b), (a, b) \in E_C :$$
$$\epsilon_{s,a,b,t-1} \geq \delta_{s,a,t-1} + \alpha_{a,t-1} + \beta_{b,t-1} - 2 \tag{61}$$
$$3 \cdot \epsilon_{s,a,b,t-1} \leq \delta_{a,s,t-1} + \alpha_{a,t-1} + \beta_{b,t-1} \tag{62}$$
$$\delta_{s,b,t} \geq \epsilon_{s,a,b,t-1} \tag{63}$$
$$\forall s \in N_0, b \in N \setminus \{s\}, t \geq h(s, b) :$$
$$\delta_{s,b,t} \geq \delta_{s,b,t-1} \tag{64}$$
$$\delta_{s,b,t} \leq \delta_{s,b,t-1} + \sum_{(a,b) \in E_C} \epsilon_{s,a,b,t-1} \tag{65}$$

All source nodes know about their own information at any time (Equation 58). In the last cycle, all nodes must know about all information (Equation 59). The information s can not reach node v before cycle $h(s, v)$ (Equation 60). According to Equation 47, the Equations 61 and 62 realize $\epsilon_{s,a,b,t-1} = \delta_{s,a,t-1} \wedge \alpha_{a,t-1} \wedge \beta_{b,t-1}$. Therefore, an information s is transmitted from node a to node b iff a knows about s and is scheduled



$\alpha_{0,0}$	$\alpha_{1,1}$	$\alpha_{2,2}$	
$\beta_{2,0}$	$\beta_{2,1}$	$\beta_{0,2}, \beta_{1,2}$	
γ_0	γ_1	γ_2	
$\delta_{0,0,0}$	$\delta_{0,0,1}, \delta_{0,2,1}$	$\delta_{0,0,2}, \delta_{0,2,2}$	$\delta_{0,0,3}, \delta_{0,2,3}, \delta_{0,1,3}$
$\delta_{1,1,0}$	$\delta_{1,1,1}$	$\delta_{1,1,2}, \delta_{1,2,2}$	$\delta_{1,1,3}, \delta_{1,2,3}, \delta_{1,0,3}$
$\epsilon_{0,0,2,0}$	$\epsilon_{1,1,2,1}$	$\epsilon_{0,2,1,2}, \epsilon_{1,2,0,2}$	

Figure 36: Example network with $N_0 = \{0,1\}$, $N = N_0 \cup \{2\}$, $E_C = \{(0,2),(2,0),(1,2),(2,1)\}$, $E_I = E_C \cup \{(0,1),(1,0)\}$ and its ILP solution (variables set to 1)

for transmission and b is scheduled for reception (Equation 52). According to Equation 46, the Equations 63 to 65 realize $\delta_{s,b,t} = \delta_{s,b,t-1} \vee \bigvee_{(a,b)\in E_C} \epsilon_{s,a,b,t-1}$. Thus, node b knows about information s in cycle t iff it knew about s in cycle $t-1$, or received the information from any node a in cycle $t-1$.

Finally, the following constraints prevent interference and invalid transceiver usage:

$$\forall t \in T, v \in N : \alpha_{v,t} + \beta_{v,t} \leq 1 \tag{66}$$

$$\forall t \in T, b \in N : |N| \cdot \beta_{b,t} + \sum_{(a,b)\in E_I} \alpha_{a,t} \leq |N| + 1 \tag{67}$$

Equation 66 ensures that no node is scheduled for transmission and reception in the same cycle. Due to Equation 67, a receiver b must not hear signals from multiple transmitters. If b is not scheduled for reception, Equation 67 does not constrain the solution space as the number of potential transmitters can not be larger than $|N|-1$.

After passing the Equations 53 to 67 to the ILP solver, the resulting schedule is constructed from the α and β variables.

Figure 36 provides a small example network. For $L_{\max} = 4$, the generated ILP formulation comprises 68 variables and 141 constraints. In the ILP solution, 29 variables are set to 1 representing an optimal schedule ($0 \to \{2\}; 1 \to \{2\}; 2 \to \{0,1\}$).

6.1.3 Heuristic Scheduling

As discussed in Section 6.1.2, finding an optimal schedule is NP-complete and the ILP-solvers will thus not be able to find solutions for larger networks in an acceptable time. To this end, an heuristic algorithm is proposed to find good (see Section 6.1.4 for an evaluation), but not necessarily optimal solutions for larger networks.

The main idea of Algorithm 3 is to select the set of transmitting nodes in each scheduling cycle that transfers the maximum amount of new information to their neighboring nodes. If called with an enabled *collect* flag, the algorithm tries to aggregate the information of all source nodes at a central collector node $v_{collect}$ before

Algorithm 3: Heuristic scheduling for multi-source flooding

Input: Graph model $M = (N, N_0, E_I, E_C, h)$
Input: collect flag
Output: Schedule S

$t \coloneqq 0$;
$S(v,t) \coloneqq IDLE \quad \forall v \in N$;
$I(v,t) \coloneqq \emptyset \quad \forall v \in N$;
$I(s,t) \coloneqq \{s\} \quad \forall s \in N_0$;
if *collect* **then**
    $d \coloneqq \min_{v \in N} \sum_{s \in N_0} h(s,v)$;
    $D \coloneqq \{v \in N : \sum_{s \in N_0} h(s,v) = d\}$;
    $d \coloneqq \max_{a \in D} |\{(a,b) \in E_C\}|$;
    $v_{collect} \coloneqq a \in D : |\{(a,b) \in E_C\}| = d$;
while $\exists v \in N : I(v,t) \neq N_0$ **do**
    $T \coloneqq \{v \in N : \exists (a,b) \in E_C : I(a,t) \not\subseteq I(b,t)\}$;
    $d \coloneqq \max_{v \in T} |I(v,t)|$;
    $T \coloneqq \{v \in T : |I(v,t)| = d\}$;
    $w_{\max} \coloneqq -\infty$;
    foreach $P \in 2^T$ **do**
        $E \coloneqq \emptyset$;
        foreach $b \in N \setminus P$ **do**
            $D \coloneqq \{a \in P : (a,b) \in E_I\}$;
            if $|D| \neq 1$ **then** next;
            if $(a,b) \notin E_C$ **then** next;
            if $I(a,t) \subseteq I(b,t)$ **then** next;
            $E \coloneqq E \cup \{(a,b)\}$;
        if *collect* $\wedge\ I(v_{collect},t) \neq N_0$ **then**
            $w \coloneqq 0$;
            foreach $s \in N_0$ **do**
                $D \coloneqq \{v \in N : s \in I(v,t)\} \ \cup$
                $\{b \in N : \exists (a,b) \in E : s \in I(a,t)\}$;
                $w \coloneqq w - \min_{v \in D} h(v, v_{collect})$;
        else
            $w \coloneqq \sum_{(a,b) \in E} |I(a,t) \setminus I(b,t)|$;
        if $w > w_{\max}$ **then**
            $w_{\max} \coloneqq w$;
            $E_{\max} \coloneqq E$;
    $I(v,t+1) \coloneqq I(v,t) \quad \forall v \in N$;
    foreach $(a,b) \in E_{\max}$ **do**
        $S(a,t) \coloneqq TX$;
        $S(b,t) \coloneqq RX$;
        $I(b,t+1) \coloneqq I(b,t) \cup I(a,t)$;
    $t \coloneqq t + 1$;
    $S(v,t) \coloneqq IDLE \quad \forall v \in N$;
postProcessing(M, S, I);

flooding the whole network. Not all network topologies benefit from this aggregation. Thus the algorithm should be run twice (with and without the *collect* flag) to obtain best results.

In Lines 1 to 4, the cycle counter t, the schedule S and the information assignment I are initialized. In Lines 6 to 9, the central node $v_{collect}$ for the optional *collect* mode is selected as the node with minimal distance to all source nodes and maximum number of reachable neighbors. Each execution of the while-loop in Line 10 generates another schedule cycle, until all information is distributed to all nodes. In Line 11, a list T of potential transmitting nodes is selected. To keep the subsequent search-space as small as possible, T is restricted to those nodes carrying the most information (Line 13). Due to possible mutual interferences, it is not always the best solution to activate all selected transmitters in the same cycle. Therefore, the subset P of T actually maximizing a performance metric $w_{\max}$ is searched by the loop in Line 15. The performance metric varies depending on operating mode (*collect* true or false), and over the course of the search (see below).

In Lines 17 to 22, a list of edges E from the transmitters P to appropriate receivers is determined. A node b is an appropriate receiver if it is not included in the transmitters list (Line 17), if it is influenced by exactly one transmitter $a \in P$ (Line 19) and if it can actually receive additional information from a (Lines 20, 21).

In non-greedy mode (or in greedy mode, but with all information already aggregated at the collector node $v_{collect}$), the algorithm aims to maximize the overall appearance of new (previously unknown) information at receiver nodes. This computation is performed in Line 30 for the currently examined subset P. In greedy mode, the heuristic initially drives all information towards the collector node. Thus, until all information has arrived there ($I(v_{collect}, t) \neq N_0$), the current subset P is rated with regard to the minimal hop distance of each information s to the collector node $v_{collect}$, summed in the loop at Line 25 across all s. Once all information has arrived at $v_{collect}$ ($I(v_{collect}, t) = N_0$), the quality metric falls back to aiming for maximum information distribution.

The best-so-far solution is maintained in $w_{\max}$ and $E_{\max}$ (Line 31). This solution is used to schedule the nodes in the current cycle and expand the information assignment for the next cycle (Lines 34 to 40).

The resulting schedule is further optimized in a post-processing step (Line 41) by eliminating redundant data transfers. A data transfer from node a to node b in cycle t is redundant, if it is dominated by a later data transfer from node c to node b in cycle $k > t$, i.e., $\Delta I := I(b, t+1) \setminus I(b, t) \subseteq I(c, k)$ and b is not scheduled for transmission between cycles t and k. In this case, $S(b, t)$ is set to *IDLE* and $I(b, i)$ is reduced by ΔI for $t < i \leq k$. Afterwards, the transmission $S(a, t)$ may have become superfluous if b (now set to *IDLE*) was its *only* receiver in cycle t. Then, $S(a, t)$ is set to *IDLE*. This may result in a completely idle schedule cycle t, which has to be removed. Furthermore, removing a transmission for node a may make another reception of a redundant. Thus, these steps are performed repeatedly until no more redundant transfers can be found.

6.1.4 Evaluation

The difficulty to find an optimal schedule heavily depends on the network topology. In sparse networks, the number of possible information routes to choose from is smaller than in dense networks. However, in totally connected graphs the optimal solution is

straightforward. To evaluate the proposed scheduling algorithms on a large number of topologies, random test-cases are generated. For each test-case, a number of $|N|$ nodes is randomly placed in a two dimensional plane of a certain size l^2 by assigning $x, y : N \mapsto [0, l]$. Without loss of generality, the first $|N_0|$ nodes are selected as the source nodes. Two thresholds $d_C, d_I \in \mathbb{R}$ are used to derive the network-topology from the euclidean distance between two nodes $(a, b) \in N^2$:

$$d(a,b) = \sqrt{(x(a) - x(b))^2 + (y(a) - y(b))^2} \tag{68}$$

$$(a,b) \in E_C \Leftrightarrow d(a,b) \leq d_C \tag{69}$$

$$(a,b) \in E_I \Leftrightarrow d(a,b) \leq d_I \tag{70}$$

The inference and connectivity thresholds are derived from the 2-ray-ground radio propagation model [Rappaport2001]:

$$\frac{P_{RX}}{P_{TX}} = \frac{G_{TX} G_{RX}}{L} \frac{h_{TX}^2 h_{RX}^2}{d(TX, RX)^4} \tag{71}$$

where P_{TX} is the transmitted signal power, P_{RX} is the received signal power, $d(TX, RX)$ is the distance between sender and receiver, h_{TX} and h_{RX} are the vertical height of the transmitter and receiver antennas over ground. The system loss factor L and the antenna gains G_{TX}, G_{RX} are usually disregarded.

As a practical example, the TI [CC2530] RF-SoC of the HaLOEWEn3 provides a maximum P_{TX} of 4.5 dBm (2.82 mW) and may successfully receive −82 dBm (6.31 pW) signals while being subject to another −85 dBm (3.16 pW) noise signal (co-channel rejection). With an assumed antenna height of 0.24 m, the connectivity and interference ranges are calculated as

$$d_C = 0.24\,\mathrm{m} \cdot \sqrt[4]{\frac{2.82\ \mathrm{mW}}{6.31\ \mathrm{pW}}} = 35\,\mathrm{m} \tag{72}$$

$$d_I = 0.24\,\mathrm{m} \cdot \sqrt[4]{\frac{2.82\ \mathrm{mW}}{3.16\ \mathrm{pW}}} = 41\,\mathrm{m} \tag{73}$$

With these fixed settings, the density of the network topology can be adjusted by the number of network nodes and the size l^2 of the deployment area.

To evaluate the efficiency of the proposed optimal and heuristic scheduling algorithm, the resulting cost for various network configurations is compared against traditional and advanced flooding algorithms. These reference implementations share some common characteristics. In each cycle, a set of potential forwarding nodes $F \subseteq N$ is selected by a protocol specific mechanism (see below). Out of these, a subset $T \subseteq F$ is selected by CSMA collision avoidance and a hidden terminal detection, i.e., $\nexists a, \hat{a} \in T : (a, \hat{a}) \in E_I \vee (\exists b \in N : (a, b) \in E_C \wedge (\hat{a}, b) \in E_I)$. All non-transmitting nodes in the interference range of any transmitter are counted as receivers $R = b \in N \setminus T : \exists a \in T : (a, b) \in E_I\}$. All other nodes will turn off their radio after clear channel assessment and thus do not consume a considerable amount of energy when compared to actually receiving data.

The main difference between the various reference implementations is the selection of the potential forwarding nodes F. In blind flooding, all nodes that have received, but not yet forwarded an information are included in F. This does not require any knowledge about the network topology. In contrast, the self-pruning (*sp*) [Lim2001], dominant-pruning (*dp*) [Lim2001], partial dominant-pruning (*pdp*) [Lou2002], total dominant-pruning (*tdp*) [Lou2002] and the history-based 2-hop dominant-pruning (*h2dp*) [Agathos2011] try to reduce F based on knowledge about the one- or two-hop neighborhood of each node. Please refer to the original work for further details.

For this evaluation, CPLEX 12.5.1 is used as ILP solver running up to 8 threads. The ILP formulation is passed to CPLEX in the LP file format. The solver is supplied with an upper bound for the objective function derived from a preceding run of the scheduling heuristic. All computations are executed on a compute server equipped with 128 GB RAM and 32 AMD Opteron 6134 CPUs running at 2.3 GHz.

For networks of $|N| = 20$ nodes, 10 network configurations are considered by varying the number of source nodes $|N_0| \in \{1, \ldots, 5\}$ and the size of the deployment area $l^2 \in \{100\,\mathrm{m} \times 100\,\mathrm{m}, 150\,\mathrm{m} \times 150\,\mathrm{m}\}$. For each configuration, 50 networks are generated at random (within the given bounds). For each network, the various algorithms (flooding, pruning, heuristic and ILP) are applied to generate a schedule. To simplify comparison, the schedule costs (Equation 44) of the enhanced algorithms are normalized to the outcome of the naive blind flooding. The resulting relative costs are averaged over the 50 random test-cases per network configuration.

Figure 37 shows the results for the larger deployment area (i.e., the sparse topology). For a single source node, the existing pruning algorithms reduce the communication costs by up to 35 % compared to the blind flooding baseline costs. These pruning algorithms assume a certain preferred direction of information expanding from a single source throughout the network. With a growing number of source nodes, this assumption is no longer valid, resulting in the degraded improvement of less than 30 % cost reduction for five source nodes. In contrast, the novel heuristic and ILP scheduler are designed to manage multiple source nodes interfering with each other. Thus, their performance advantage over blind flooding increases with the number of source nodes up to 83 % for five source nodes.

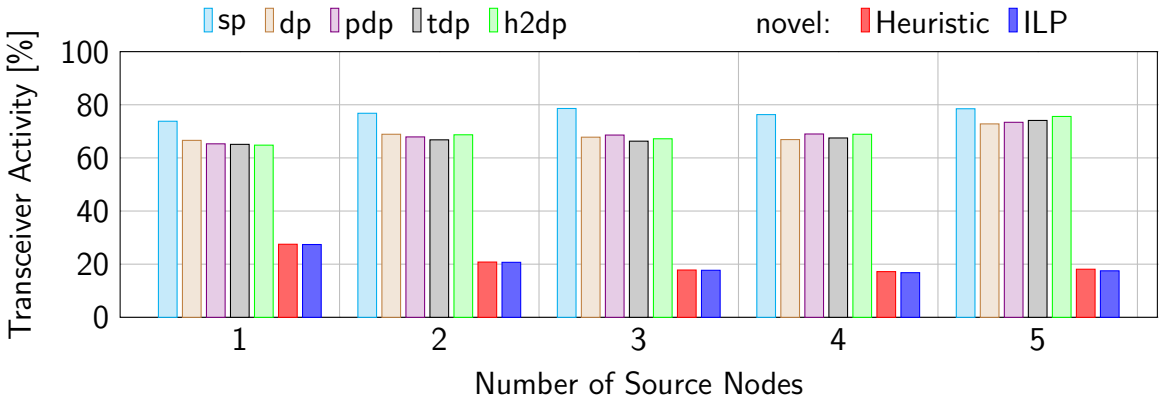


Figure 37: Averaged scheduling costs relative to blind flooding for 20 nodes in an **150 m × 150 m** deployment area

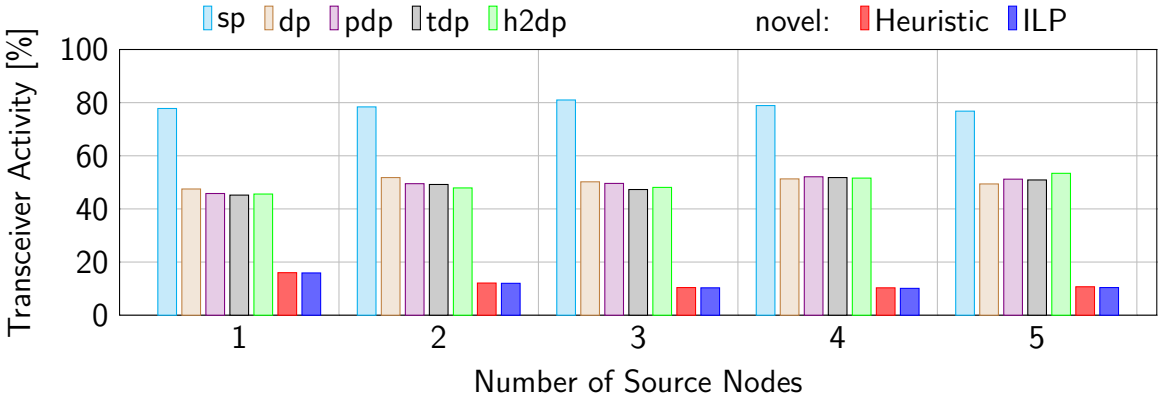


Figure 38: Averaged scheduling costs relative to blind flooding for 20 nodes in an **100 m × 100 m** deployment area

Similar observations can be made in Figure 38. Here, the smaller deployment area results in denser networks. The number of redundant transmissions and receptions performed in blind flooding thus increases and the relative cost of all other algorithms decreases. Furthermore, the knowledge of the pruning algorithms about the two-hop neighborhood becomes more valuable, as it now covers a larger part of the network. The pruning algorithms achieve up to 55 % cost reduction for a single source node, while the heuristic and optimal schedules computed by the novel methods achieve up to 90 % cost reduction for five source nodes.

Figure 39 examines the heuristic schedule costs (i.e., the required number of transmissions/receptions) normalized to the optimal schedule costs. The percentiles describe the range of heuristic schedule costs achieved by a certain percentage of the randomly generated topologies. For example, the 60 % to 95 % percentile of the 150 m × 150 m

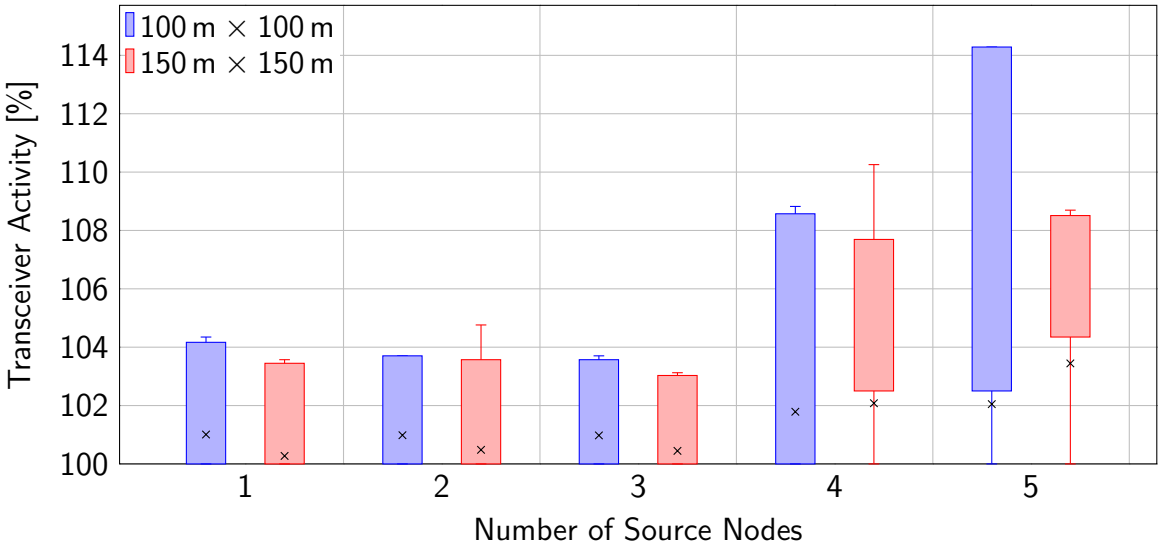


Figure 39: Heuristic scheduling costs relative to optimal scheduling costs: 0 % to 100 % percentile (⊢⊣), 60 % to 95 % percentile (▭), average (×) for the 50 randomly generated topologies.

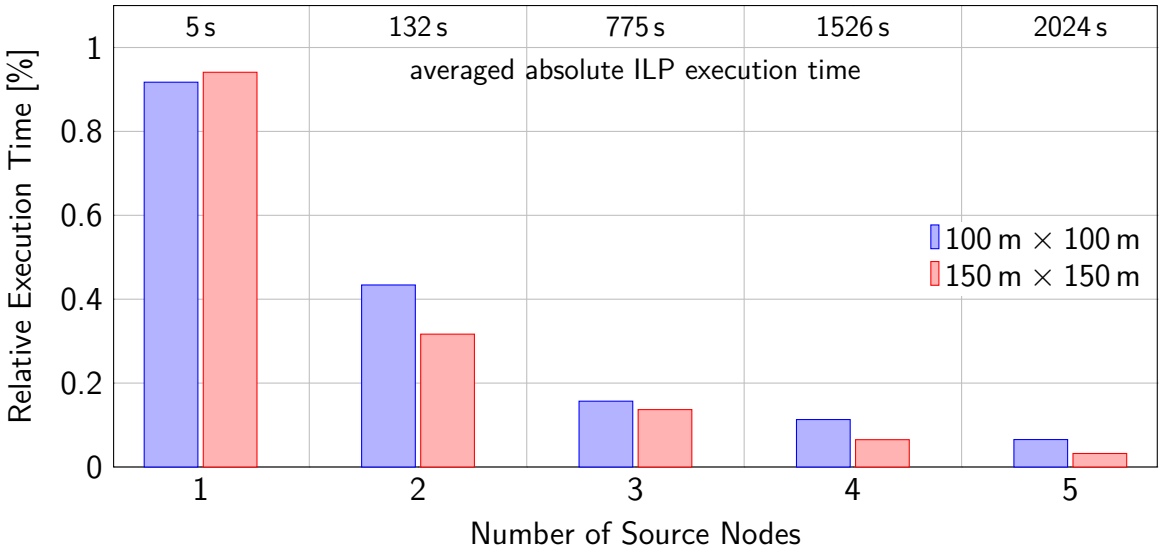


Figure 40: Averaged runtime of heuristic relative to runtime of ILP-solver (both executed on the same compute server)

topologies with 4 source nodes ranges from 102.5 % to 107.7 %. That is, for 60 % out of the 50 test topologies, the heuristic required not more than 102.5 % of the minimum achievable transmissions and receptions, while 95 % out of the 50 test topologies required not more the 107.7 % of the minimum schedule cost. For networks with up to three source nodes, the average heuristic results are less than 1 % worse compared to the optimal results. Furthermore, the heuristic found an optimal solution in at least 60 % of the generated test cases with less than four source nodes. With an increasing number of source nodes, the gap between the heuristic and the ILP solutions becomes larger, but does not exceed 4 % on average for five source nodes. This slight quality loss is the price for the highly accelerated computation, detailed in Figure 40. For four sources in the dense topology, the heuristic (run in two passes, with *collect* both enabled and disabled) executes in just 0.1 % of the time of the ILP solver. A speedup of more than 1000× can be observed for networks with five source nodes. Note that

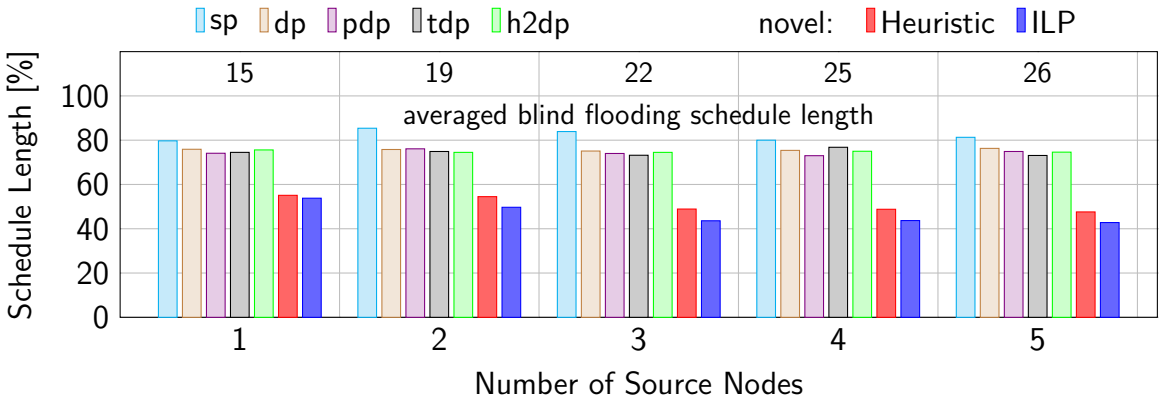


Figure 41: Averaged schedule length relative to blind flooding for 20 nodes in an **150 m × 150 m** deployment area

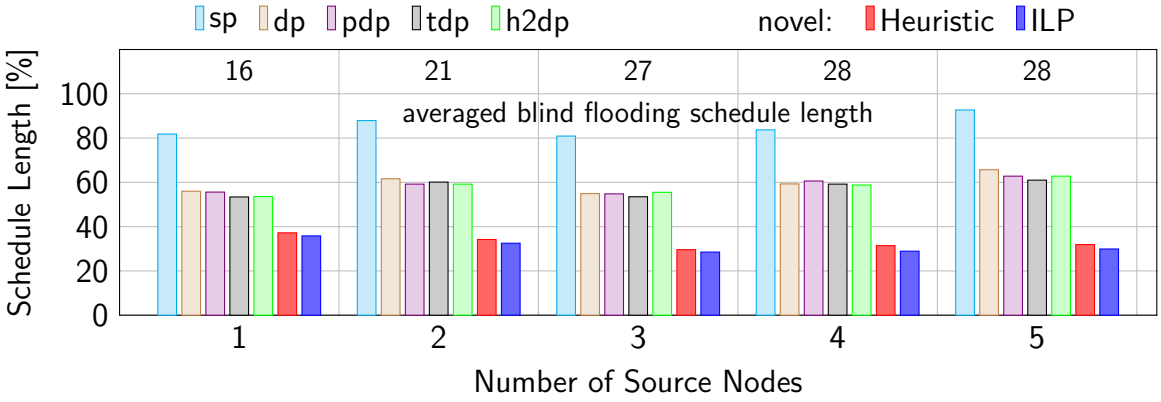


Figure 42: Averaged schedule length relative to blind flooding for 20 nodes in an **100 m** × **100 m** deployment area

all execution times are measured as pure wall-clock time, not counting the eight-core processing of the ILP solver as separate execution times.

The length of a schedule is the secondary optimization target defined by Equation 45. Figures 41 and 42 detail the average length of the schedules determined by the considered flooding schemes. As for the transceiver activity, the proposed ILP and heuristics clearly outperform the conventional pruning algorithms, especially for an increasing number of source nodes and denser networks.

6.2 Precise Wireless Time Synchronization

Time synchronization is required in WSN applications mainly for two reasons. First, data sampled from spatially distributed sensors cannot be properly interpreted without knowledge of the exact sampling time. The acceptable uncertainty is typically a small percentage of the application's sampling period, which may range from several seconds in environmental monitoring [Lazarescu2013] down to several milliseconds in vibration-based structural health monitoring (Section 7.3), or even shorter in acoustic localization applications [Astapov2014]. Thus, synchronization protocols with an accuracy of a few microseconds are required for the more demanding applications.

A second reason for precise time synchronization derives from the large power consumption of the radio transceiver when idly waiting for incoming messages. If the sender and receiver are synchronized, the radio protocol can define short periods of time in which transmissions can be initiated and received. Outside these windows, the power-hungry transceivers can be shut down.

6.2.1 Necessity of Clock Drift Compensation

The local time at a sensor node a at a certain absolute time t is represented as a timestamp $l_a(t) \coloneqq f_a \cdot (t - t_{a,0})$, which is the value of an oscillator driven counter with oscillation frequency f_a and start-up time $t_{a,0}$. If two nodes a and b exchanged

their timestamps $(l_a(t_e), l_b(t_e))$ at a time t_e, then node b can calculate the timestamp of node a at any subsequent time $t_s > t_e$ as

$$\begin{aligned} l_a(t_s) &= l_a(t_e) + f_a \cdot (t_s - t_e) \\ &= l_a(t_e) + \frac{f_a}{f_b}(l_b(t_s) - l_b(t_e)) \\ &= l_b(t_s) + l_a(t_e) - l_b(t_e) + \frac{f_a - f_b}{f_b}(l_b(t_s) - l_b(t_e)) \\ &= l_b(t_s) + \underbrace{l_a(t_e) - l_b(t_e)}_{\text{offset compensation}} + \underbrace{(f_a - f_b) \cdot (t_s - t_e)}_{\text{drift compensation}} \end{aligned} \tag{74}$$

Even if both nodes run at the same nominal frequency, the temperature and voltage dependency of the oscillators result in small relative frequency deviations $\frac{f_a - f_b}{f_b}$, typically in the range of a few parts per million (ppm), as shown in Figure 44. Manufacturers specify the maximum frequency uncertainty over the entire operating temperature range at even higher values (e.g., ±40 ppm for the TI CC2530). Without explicit drift compensation, the timestamps used for offset compensation have to be exchanged at a rate of at least $(f_a - f_b)/(\Delta t \cdot f_b)$ to keep the synchronization uncertainty below Δt. Even if the timestamp exchange can be piggybacked onto the application's regular communication payloads, the remaining additional communication effort is still undesirable for small Δt. The next section therefore shows how to efficiently integrate the clock drift compensation into a wireless synchronization protocol.

6.2.2 Organization of Timestamp Exchange and Clock Drift Estimation

As in the FTSP [Maroti2004], a single node r is selected as time reference, i.e., its local time l_r represents the global time g all other nodes try to synchronize with. More precisely, in addition to its local time l_a, every node a derives an assumption g_a about the global time. Under perfect synchronization, $g(t) = g_a(t)$ is achieved for any time t and all nodes a. As this section focuses on the LR-based drift compensation, dynamic reselection of the reference node, as it is described by [Maroti2004], is not considered here further.

The synchronization layer should be inserted between the MAC and the network layer to ensure that all nodes get synchronized, even if they are just routing packets. This violates the compliance with standardized protocols like [ZigBee]. However, interoperability with COTS [ZigBee] devices is not the primary design goal of the HaLOEWEn mote. The synchronization layer adds up to 6 B to the radio packet. The first byte indicates whether or not the sender has already been synchronized to the reference, i.e., it is the reference or it received a sufficient number of timestamps to perform the offset and drift compensation between its local clock and the reference clock, as described below. If the sender a is synchronized, a 5 B timestamp $g_a(t_{TX})$ is appended to represent the senders assumption of the global time of the current start of frame delimiter (SFD) transmission.

To calculate this timestamp, the sender a first inserts the PHY and the MAC header and the first byte of the synchronization header into the radio buffer and initiates the transmission. After the SFD was transmitted at time t_{TX}, the sender calculates $g_a(t_{TX})$ from the captured local time $l_a(t_{TX})$. At the reference node, the local time is just

Algorithm 4: Conversion from local to global time

Input: Local timestamp l
Input: Last exchanged synchronization point $(\hat{g}, \hat{l})$
Input: Clock drift represented as *num* and *den*
Output: Global timestamp g corresponding to l

1 $t \coloneqq l - \hat{l}$;
2 $g \coloneqq \hat{g} + t + (t \cdot (num - den))/den$; // Equation 74

used as global time, while all other nodes apply Algorithm 4 to the local timestamp. This algorithm realizes Equation 74 based on the outcome of the linear regression (Section 5.3) applied to the last exchanged synchronization points. More specifically, the ratio between *num* and *den* equals the ratio between the oscillator frequencies of the reference node and the local node, i.e., $num/den = f_r/f_a$. To avoid a radio buffer underrun, Algorithm 4 must be finished before the previous header data was transmitted (i.e., $8\,\mathrm{B}/250\,\mathrm{kbit/s} = 256\,\mu\mathrm{s}$).

Upon reception of the SFD of a radio packet by node b at time t_{RX}, the receiver timer captures the local timestamp $l_b(t_{RX})$. If the sender a was synchronized to the reference, the global timestamp $g_a(t_{TX})$ is included in the synchronization header of the packet. Node b than is provided with its local and a global timestamp nearly generated at the same time. The remaining difference between t_{RX} and t_{TX} results from the propagation delay between sender and receiver (i.e., 3.3 ns/m) and should not be significant for node distances of a few meters. In practice, however, $t_{RX} - t_{TX} = (3.51 \pm 0.04)\,\mu\mathrm{s}$ was measured by observing the SFD sampling events automatically generated by the transceiver hardware with an oscilloscope. A similar systemic offset was observed by [Akhlaq2013]. It is compensated for by subtracting $\Delta_{SFD} \coloneqq 3.51\,\mu\mathrm{s} \cdot 32\,\mathrm{MHz} = 112$ from $l_b(t_{RX})$, where 32 MHz is the timer frequency of the HaLOEWEn3 motes. After this error compensation, the receiver b uses the global and the local timestamp as new data point for the sliding window linear regression to update its regression slope with Algorithm 1.

In this manner, the synchronization information is flooded over multiple hops into the network. To perform a certain action (e.g., sensor sampling) simultaneously, all synchronized nodes must agree on a certain global time, convert this time into their local time by applying Algorithm 5, and configure their timers to generate an interrupt at the calculated local time.

Algorithm 5: Conversion from global to local time

Input: Global timestamp g
Input: Last exchanged synchronization point $(\hat{g}, \hat{l})$
Input: Clock drift represented as *num* and *den*
Output: Local timestamp l corresponding to g

1 $t \coloneqq g - \hat{g}$;
2 $l \coloneqq \hat{l} + t - (t \cdot (num - den))/num$; // Equation 74

6.2.3 Evaluation

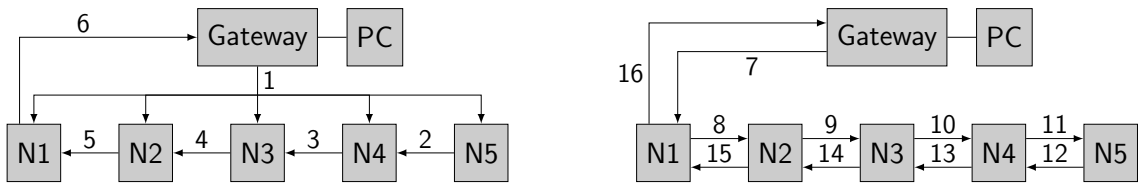
In this section the synchronization accuracy of the protocol described in Section 6.2.2 is analyzed for different configurations of the synchronization period and the regression table size. The synchronization accuracy is affected by hardware details of the utilized TI CC2531 radio transceiver (e.g., oscillator stability, timestamp accuracy and radio-triggered timestamp capturing) as well as the network topology and other environmental conditions (e.g., temperature stability). After restricting the design space to reasonable configurations for the synchronization period and the regression table size, the resource requirements of the various LR implementations are compared against each other.

The network setup shown in Figure 43 was used to evaluate the achievable synchronization accuracy. Five receiver nodes are located in the broadcast range of a gateway node. The latter dumps measurement results to the PC over a serial connection and acts as the time reference in the synchronization protocol. All nodes are not more than 1 m apart from each other, so the propagation delay is smaller than 3 ns and can be ignored. Note that the synchronized nodes are required to be within the broadcast range of the gateway only to simplify the test of the synchronization accuracy. The synchronization protocol itself relies on exchanging timestamps along the linear multi-hop chain and does not require any broadcasts.

Once per second, the gateway initiates a new measurement consisting of two phases. The *test* phase is started by a broadcast from the gateway to all other nodes (Message 1 in Figure 43a). The gateway captures the broadcast transmission time $g(t_{TX})$. Each receiving node i captures its local broadcast reception time $l_i(t_{RX})$, derives the local broadcast transmission time as $l_i(t_{TX}) \approx l_i(t_{RX}) - \Delta_{SFD}$, and calculates the corresponding assumed global time $g_i(t_{TX})$ using offset and drift compensation. These timestamps are reported back to the gateway in a linear chain starting at node 5 (Messages 2 to 6 in Figure 43a). The actual synchronization accuracy A_i and the actual clock drift D_i of node i relative to the reference are derived from the received timestamps as

$$A_i(t_{TX}) \coloneqq g_i(t_{TX}) - g(t_{TX})$$
$$D_i(t_{TX}) \coloneqq \frac{g(t_{TX}) - g(t_{TX} - 1\,\mathrm{s})}{l_i(t_{TX}) - l_i(t_{TX} - 1\,\mathrm{s})} - 1$$

where $t_{TX} - 1\,\mathrm{s}$ denotes the broadcast event of the previous *test* phase.



(a) *Test* phase (b) *Update* phase

Figure 43: Network setup for timestamp capturing

Immediately after the *test* phase, the *update* phase is started by the gateway transmitting a unicast request (Message 7 in Figure 43b) to node 1, which is forwarded in a linear chain (Message 8 to 11) to node 5. Each node captures its local request reception time and the subsequent local request transmission time and reports them back to the gateway (Message 12 to 16). The actual synchronization update (i.e., insertion of the exchanged timestamps into the regression table and update of the slope parameters) is only performed in every tenth *update* phase. Thus, the effective synchronization period of the whole network is 10 s. However, the timestamps captured during the *update* phase allow for an offline simulation of the whole synchronization protocol. This simplifies the analysis of the impact of the regression table size and the synchronization period on the synchronization accuracy. All data presented below for a regression table size other than two, or a synchronization period other than 10 s, result from these simulations.

The left side of Figure 44 shows the clock drift of the five receiver nodes relative to the reference clock. For the first 55 min, all nodes were kept at a fixed temperature of about 21 °C. Under this condition, the standard deviation of the drift amounts to 0.1 ppm at all nodes. Afterwards, node 4 was cooled down to 7 °C resulting in a drift drop of about 3 ppm. In the same time, node 3 was heated up to 52 °C before cooling down to 40 °C again. This resulted in a drift step of 4.5 ppm and 2.5 ppm respectively. The right side of Figure 44 aggregates the clock drift characteristic into a histogram for each receiver node. The insights gained from this measurement are twofold. First, without clock drift compensation, node 1 would have to exchange timestamps with the reference node at least every 200 ms to keep the synchronization inaccuracy below 1 µs. Thus, high precision time synchronization has to provide drift compensation to keep the communication overhead manageable. Second, even if the sensor nodes in typical WSN applications will not be exposed to sudden large temperature changes in an outdoor scenario, a significant variation of the clock drift can be expected if the subset of nodes, that are exposed to full sunlight, is changing.

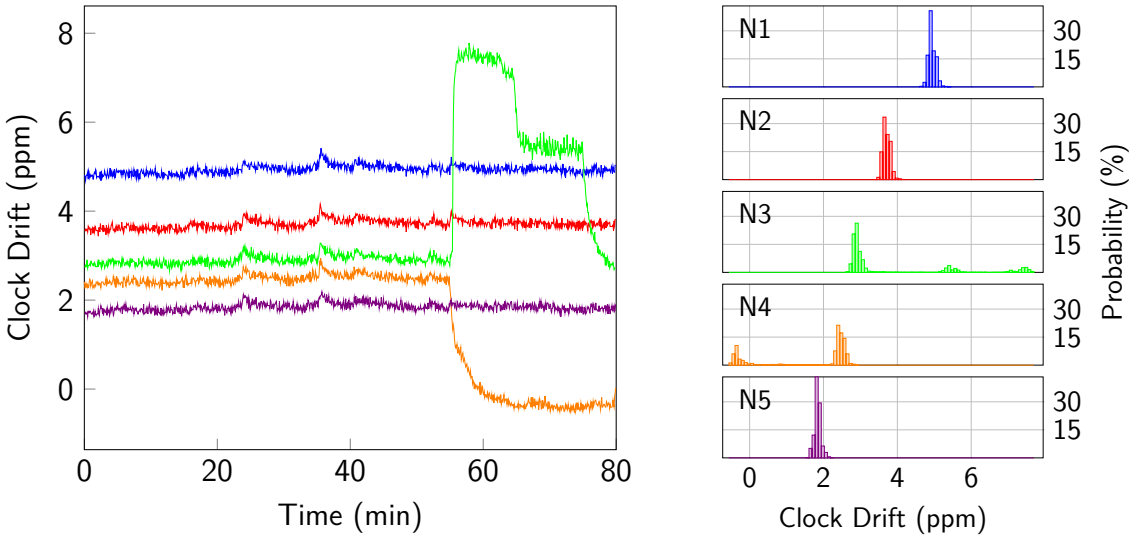


Figure 44: Observed clock drift relative to gateway under temperature variation (0 min to 55 min: 21 °C, 55 min to 80 min: 7 °C at N4, 55 min to 65 min: 52 °C at N3, 65 min to 75 min: 40 °C at N3)

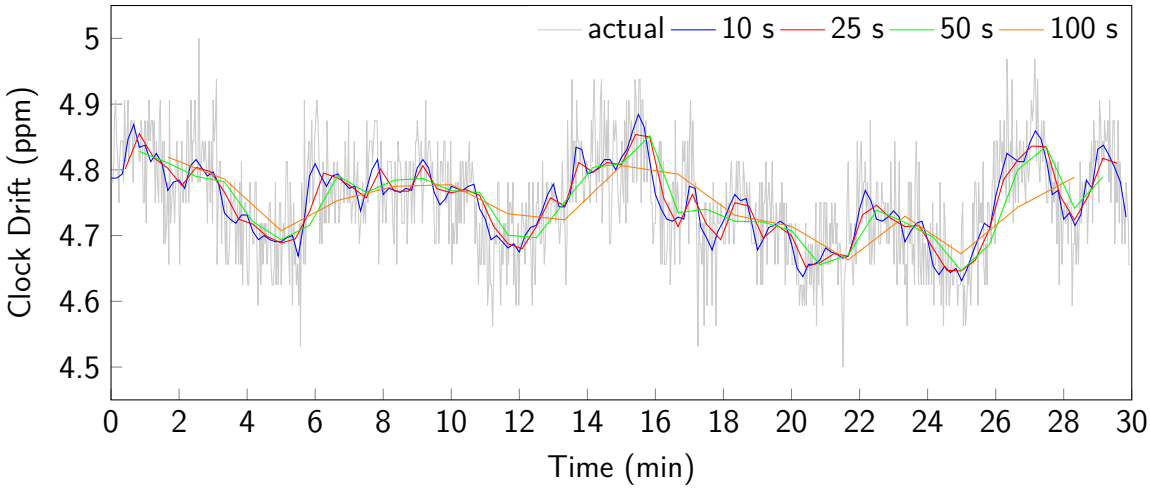


Figure 45: Impact of the synchronization period on estimated clock drift at N1 (regression table size = 2)

The influence of the node's supply voltage on the clock drift was also investigated as the supply of battery-powered sensor nodes cannot be assumed to be stable. However, due to the voltage converters inside the MCU, the reduction of the supply voltage from 3 V to 2 V did not significantly influence the clock drift at the corresponding node.

Figure 45 shows the actual clock drift of node 1 relative to the reference (captured at 1 Hz as described above) and its estimation about its own clock drift derived by LR with a synchronization period ranging from 10 s to 100 s and a fixed regression table size of two. As expected, the estimation becomes more inaccurate with longer synchronization periods. Figure 46 shows how this inaccuracy in drift estimation translates into synchronization inaccuracy. The maximum absolute clock deviations for the different synchronization periods are 0.7 µs, 2.4 µs, 5.7 µs and 8.3 µs respectively. In general, the

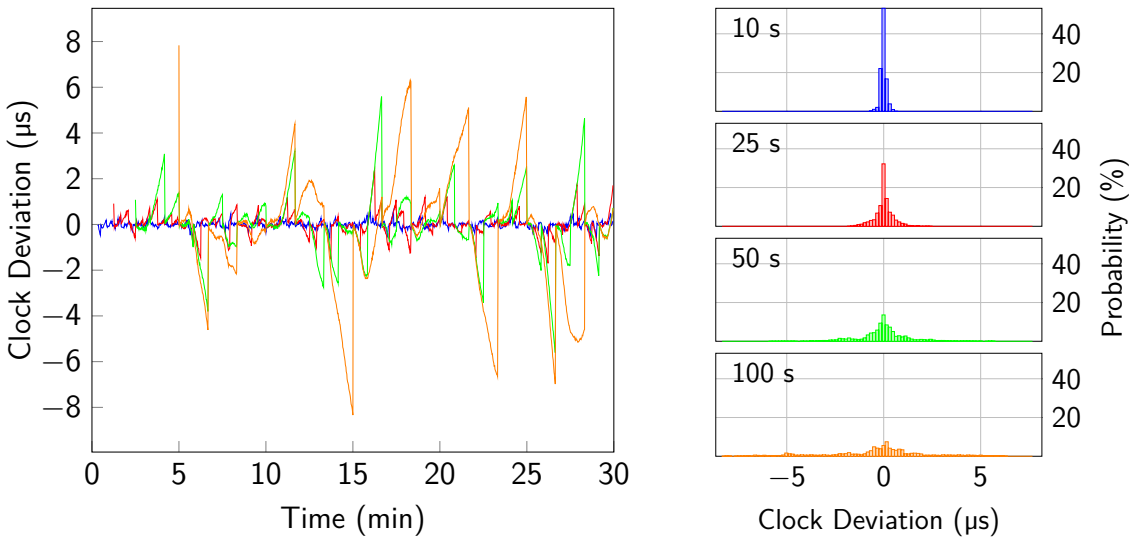


Figure 46: Impact of the synchronization period on the synchronization accuracy at N1 (regression table size = 2)

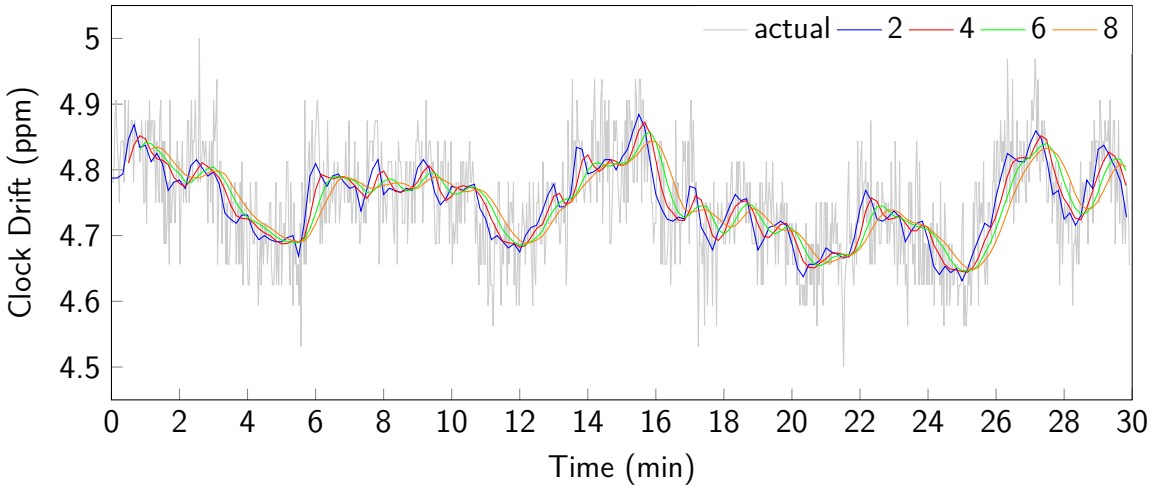


Figure 47: Impact of the regression table size on estimated clock drift at N1 (synchronization period = 10 s)

appropriate synchronization period depends on the concrete accuracy demands of the application.

Figure 47 shows the actual clock drift of node 1 relative to the reference and its estimation about its own clock drift derived by LR with a regression table size ranging from 2 to 8 elements at a fixed synchronization period of 10 s. Larger regression tables have an effect similar to the smoothing of the drift estimation occurring for larger synchronization periods. Indeed, larger regression tables actually do not improve the average synchronization accuracy, as shown in Figure 48. However, even for a table size of 8, the maximum absolute clock deviation does not exceed 1 µs.

The real benefit of larger regression tables becomes obvious only if the actual clock drift is more spiky, e.g., due to temporary temperature fluctuations, as shown in Figure 49. While the average clock deviation is not improved by the larger regression tables

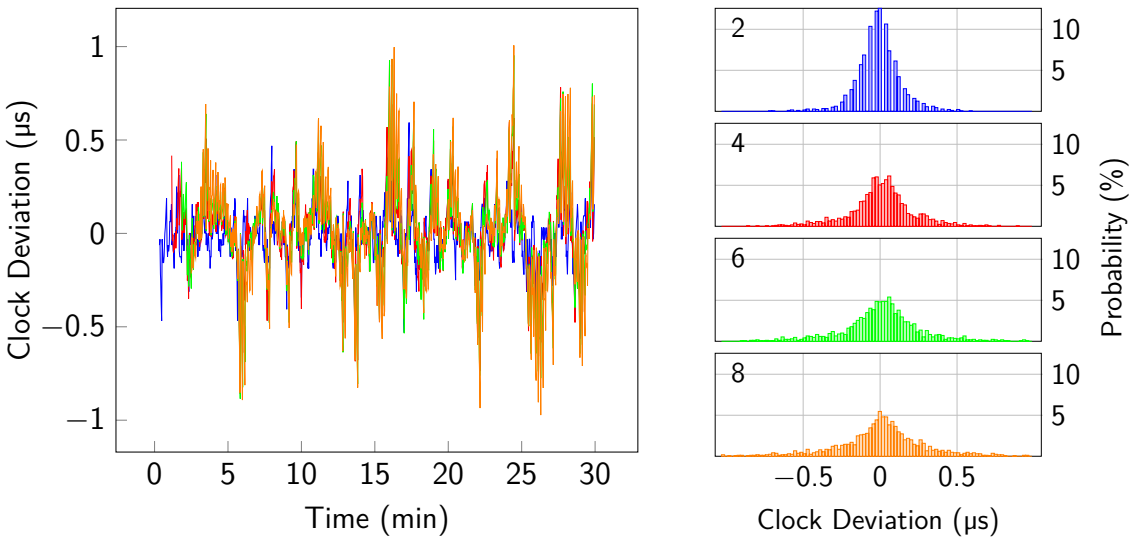


Figure 48: Impact of the regression table size on the synchronization accuracy at N1 (synchronization period = 10 s)

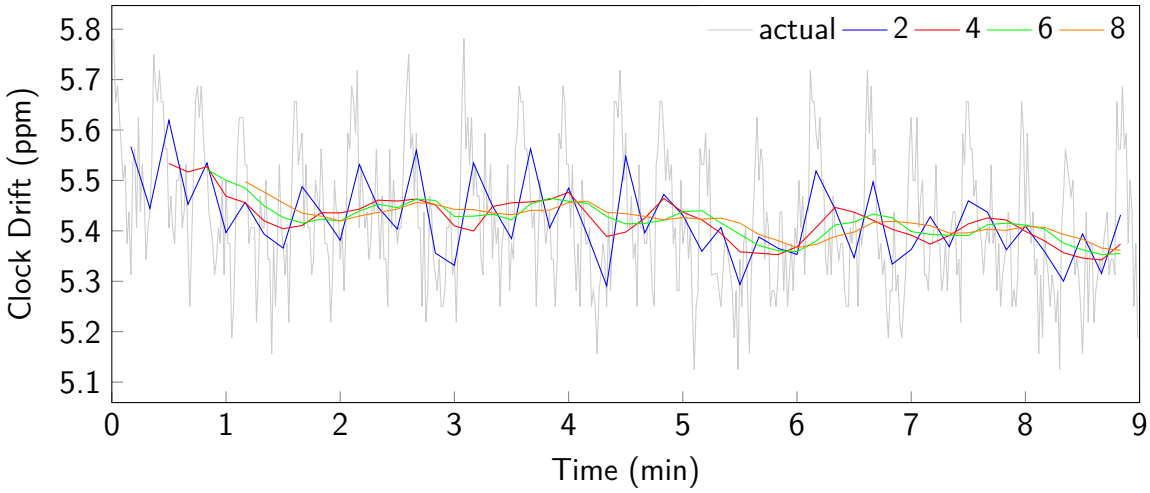


Figure 49: Impact of the regression table size on estimated clock drift at N3 (heated up to 40 °C, synchronization period = 10 s)

as shown in Figure 50, the maximum absolute error is reduced from 2.8 µs to 1.7 µs when choosing a table size of 8 instead of 2.

Finally, the synchronization accuracy over multiple hops is shown in Figure 51a. Even at the fifth hop, the maximum absolute clock deviation can be kept below 1 µs. However, the average deviation is increased by about 30 ns per hop. This might be caused by the fixed compensation time for the SFD delay as described in Section 6.2.2. The accuracy of this compensation is limited by the timestamp resolution of $1/32\,\mathrm{MHz} = 31\,\mathrm{ns}$. When simulating the synchronization with $\Delta_{SFD} = 111.5$, the mean synchronization error is kept stable over multiple hops as shown in Figure 51b. Although not yet implemented in the HaLOEWEn system, this error compensation with sub-timestamp precision can be realized by a pulse-wide modulated correction

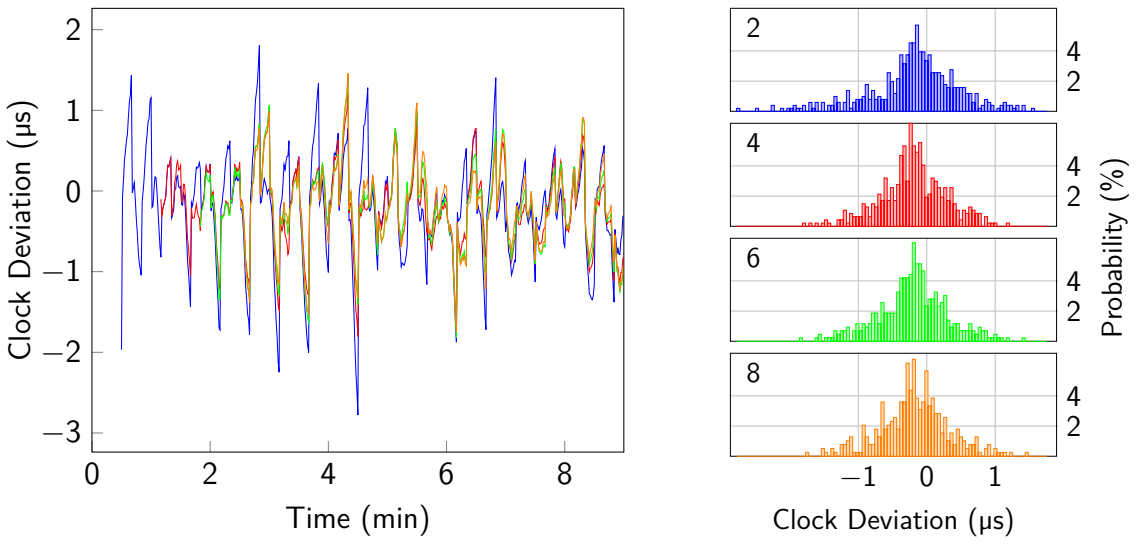
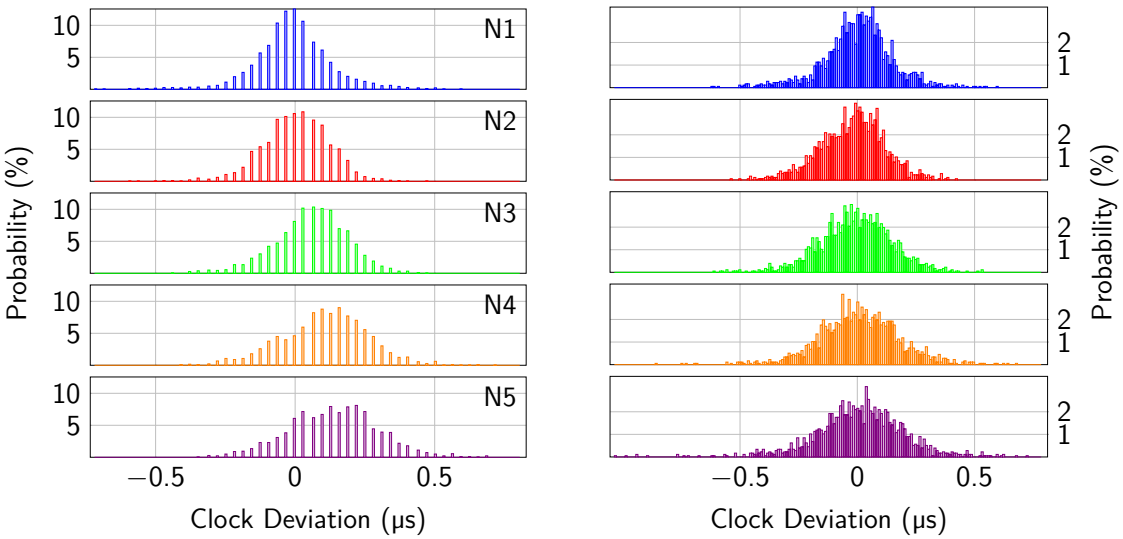


Figure 50: Impact of the regression table size on the synchronization accuracy at N3 (heated up to 40 °C, synchronization period = 10 s)



(a) Measured with $\Delta_{SFD} = 112$ (b) Simulated with $\Delta_{SFD} = 111.5$

Figure 51: Impact of the hop distance to the reference node on synchronization accuracy (synchronization period = 10 s, table size = 2)

term, i.e., toggling between $\Delta_{SFD} = 111$ and $\Delta_{SFD} = 112$ with each synchronization period.

As shown in Figure 46, the synchronization period should not exceed 10 s to achieve a synchronization accuracy of 1 µs. At a timestamp resolution of 1/32 MHz, the differences between successive timestamps inserted into the RLRCT (Algorithm 1) can thus be represented with 29 bit. Furthermore, regression tables with more than 8 entries do not improve the synchronization accuracy. More precisely, a regression table size of 4 is advisable as a trade-off between the achievable accuracy during stable environmental conditions (Figure 48) and the outlier reduction during dynamic temperature variations (Figure 50).

The arithmetic operators required for the various LR implementations described in Section 5.3 were optimized based on these assumptions (i.e., 8 entries per regression table, 29 bit sufficient to represent the difference between successive timestamps) and implemented on the HaLOEWEn3. The three different hardware accelerator architectures were synthesized for the Microsemi IGLOO M1AGL1000 FPGA with Synplify Pro (ME I-2014.03M-SP1). The resulting resource requirements are summarized in Table 11. The *Full Parallel* architecture does not fit on the target device, so its us-

Implementation	Fully Parallel	Single MAC	µArchitecture
Core Cells [%]	141	63	50
BRAM [%]	12	6	15
$f_{\max}$ [MHz]	8.7	7.1	8.5
Regression [cycles (µs)]	7 (0.8)	21 (3.0)	29 (3.4)
Conversion [cycles (µs)]	27 (3.1)	29 (4.1)	32 (3.8)

Table 11: Ressource requirements of hardware accelerator architectures on M1AGL1000

age is restricted to larger FPGAs. Both other architectures occupy slightly more than half of the FPGA logic resources, mainly due to the usage of combinatorial multipliers. Compared to the linear regression (executed once per synchronization period), the timestamp conversion is performed more often (typically once per sampling period). As the *µArchitecture* implements this critical operation faster than the *Single MAC* implementation, the former is used for the hardware-accelerated RLRCT in the following comparisons with the software solutions.

Figure 52 shows the overall execution time for the different regression implementations measured with an oscilloscope. All $\mathcal{O}(1)$ implementations (i.e., RLR, RLRCT, HW, OPT) outperform the $\mathcal{O}(n)$ implementation (i.e., LR) even for $n = 2$. Furthermore, the proposed RLRCT is 22 % faster than the RLR implementation. For $n > 2$, the hardware accelerator outperforms the best software implementation by 66 %. If a regression table of size 2 satisfies the application accuracy requirements, the optimized software implementation (referred to as OPT in Figures 52 to 54) actually computes the fastest regression.

Note that 95 % of the time required by the hardware accelerator is spent transferring the 80 bit timestamp pair from the MCU to the FPGA. Replacing the combinatorial by a sequential multiplier in ALU of the µArchitecture implementation (Figure 35) can thus be used to reduce the logic resources required by the hardware accelerator without significantly increasing its overall execution time.

All software implementations of the time conversion require 118 µs for the offset and drift compensation. The hardware accelerator requires only 72 µs, including the interprocessor communication. The sensor node thus benefits from the hardware accelerator even for the smallest regression tables.

Beyond execution time, memory is another limited resource on embedded systems. As shown in Figure 53, the proposed RLRCT clearly outperforms the RLR in terms of required RAM, as its regression table requires only $2n \cdot 29$ bit, while the RLR

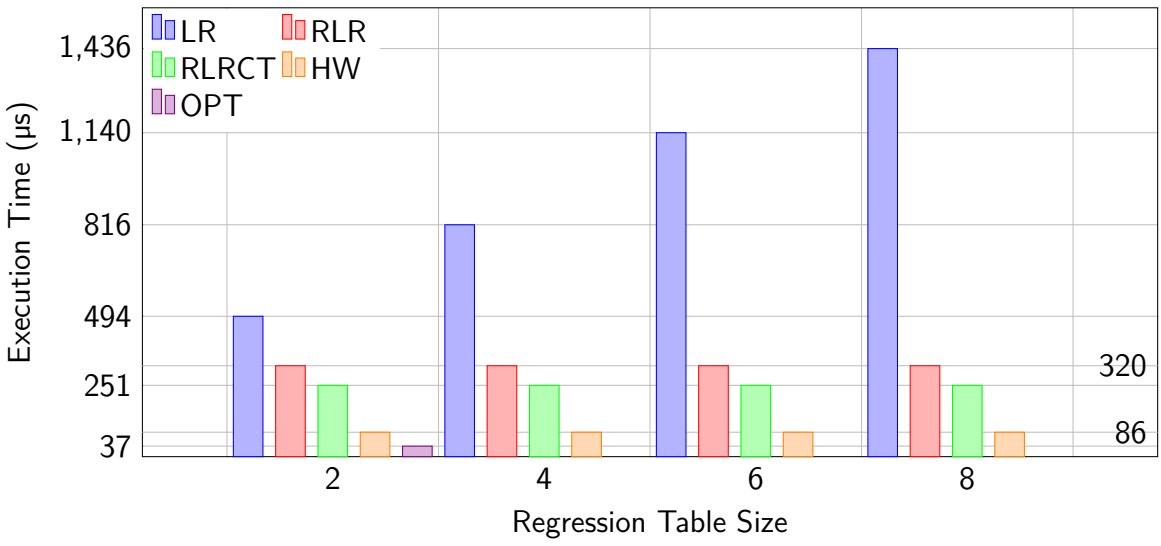


Figure 52: Overall execution time for the LR implementations (LR = Equation 15, RLR = Algorithm 2, RLRCT = Algorithm 1, HW = Accelerated µArchitecture, OPT = Equation 18 optimized for $n = 2$)

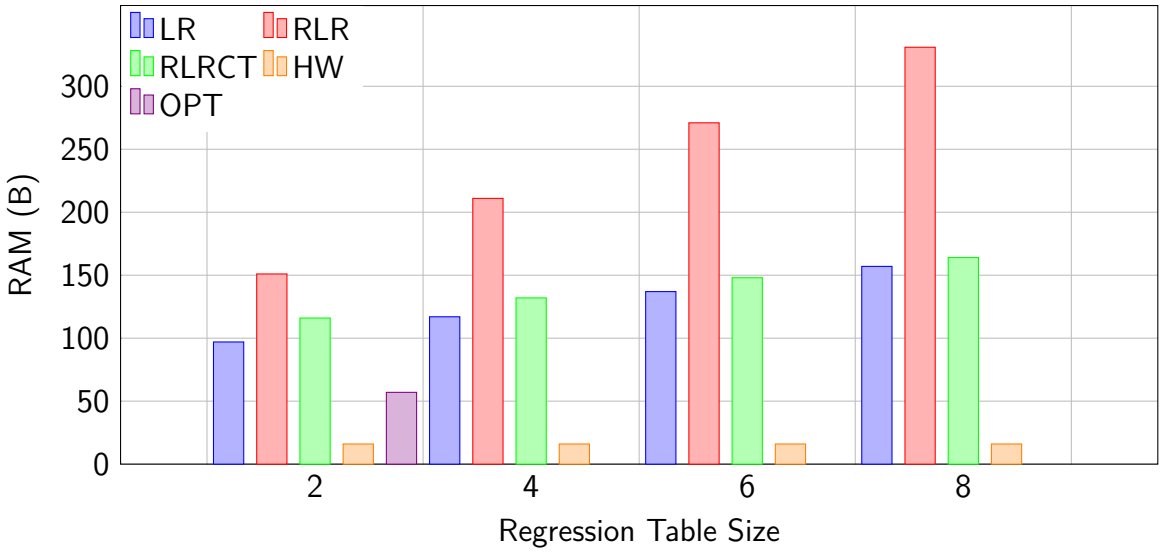


Figure 53: MCU-RAM requirement of the LR implementations (LR = Equation 15, RLR = Algorithm 2, RLRCT = Algorithm 1, HW = Accelerated µArchitecture, OPT = Equation 18 optimized for $n = 2$)

buffers $2n \cdot (40 + 80)$ bit. For regression tables larger than 12 entries, the RLRCT also requires less RAM than the LR implementation and is thus favorable in memory constrained systems. Although such large tables are not required for the wireless time synchronization, other LR applications like RSSI-based node localization [Vanheel2011] rely on larger tables and might thus benefit from the RLRCT approach.

As the assembler-based implementation of the arithmetic operations are optimized for execution speed, the instruction memory required by the different LR algorithms is relatively large. As shown in Figure 54, RLRCT requires 15 % less ROM for storing instructions than RLR, although the RLRCT has to perform more arithmetic operations.

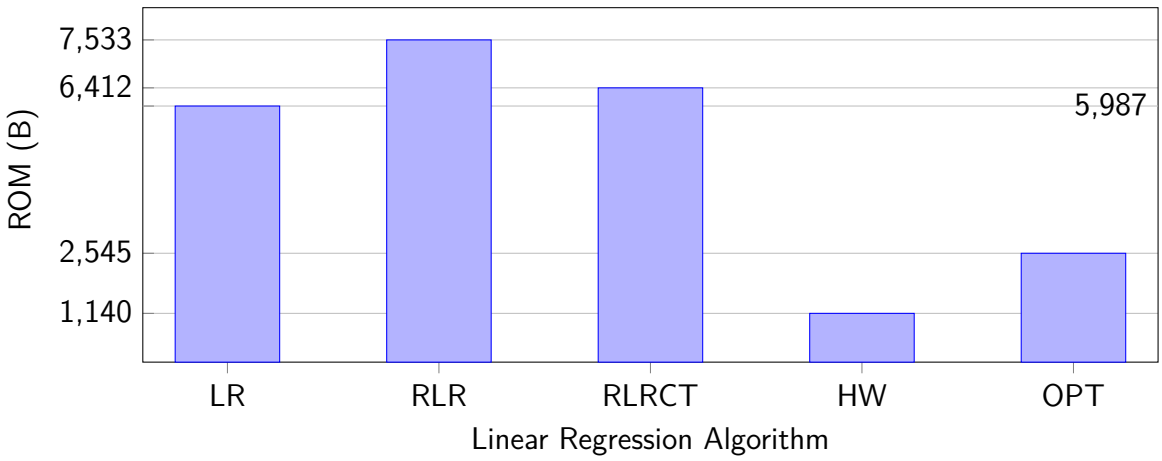


Figure 54: MCU-ROM requirement of the LR implementations (LR = Equation 15, RLR = Algorithm 2, RLRCT = Algorithm 1, HW = Accelerated µArchitecture, OPT = Equation 18 optimized for $n = 2$)

CHAPTER 7

Targeted Applications

After describing the HaLoMote architecture and some application-independent building blocks for computation and communication, three specific monitoring applications are presented in this chapter. The first two of the presented applications benefit from the energy-efficient lossless data compression capabilities of the HaLoMote. Both applications restrict the affordable sample block size either by critical delay or memory constraints. The third application requires a specific hardware accelerator for the decentralized feature extraction. The third application will thus be discussed more in detail, and evaluated at system level in the next chapter.

7.1 Monitoring Neural Activities of Primates

At the German Primate Center in Göttingen, the neural activities of primates solving different tasks are measured by a micro-electrode inside the probands brains [Rafflenbeul2012]. As the apes have to move freely over a wide testing area, wired instrumentation is impractical, and thus the sensor data sampled with 16 bit resolution at a frequency of 24.414 kHz has to be transmitted wirelessly to a control station. Figure 55 shows an sample block extracted from the neural data. The data rate of 391 kbit/s required for the transmission of the raw data stream exceeds the capability of the low-power IEEE [802.15.4] transceiver specification. The captured data stream thus has to be compressed by about 50 %, before it can actually be transmitted by an IEEE [802.15.4] transceiver.

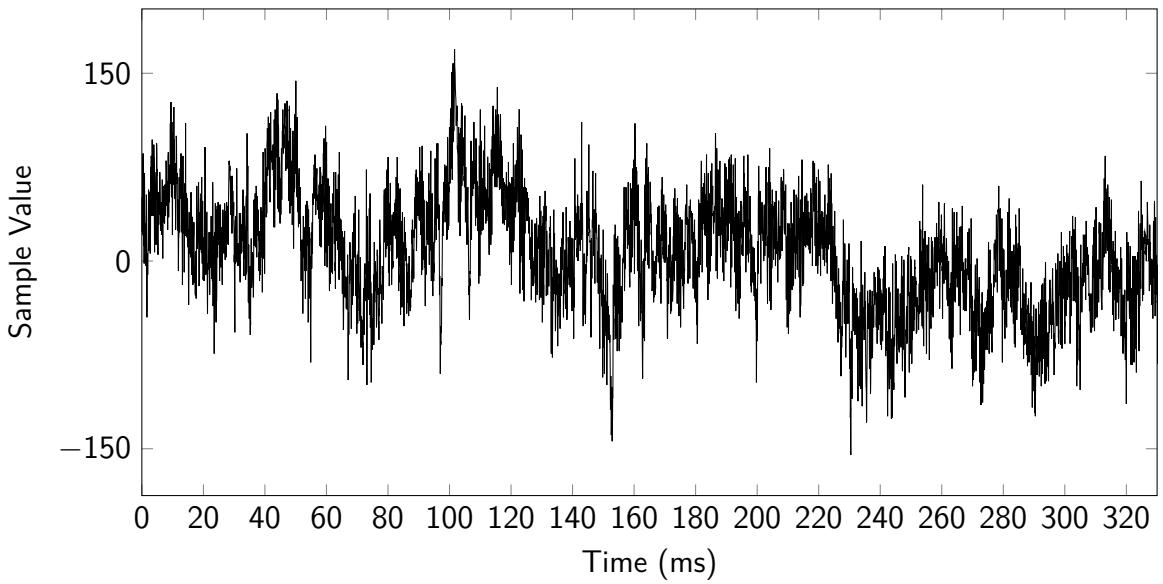


Figure 55: 8192 samples of neural activity data (min = -155, max = 169, mean = 6.3, stddev = 45.5)

Based on the received neural data, the variable penetration depth of the microelectrode is controlled remotely by an operator at the control station. In order to allow timely interactive manipulation of the probe depth, the maximum end-to-end latency is restricted by the human response time of about 100 ms. Thus, the maximum block size used by a two pass encoder may not exceed 2400 samples at 24.414 kHz.

In this section, the several lossless compression schemes are applied to the neural data. Based on a trade-off between data reduction and computational effort, a specific compression scheme is selected and the energy efficiency of its hardware-accelerated implementation on the HaLOEWEn mote is evaluated. This analysis has already been published in [Engel2014a].

7.1.1 Compressibility of the Monitored Data

As a baseline analysis, various compression *algorithms* have to be applied to the neural data to figure out a well suited candidate for an embedded implementation. The raw data sample stream was split into blocks of different size and fed into the compression algorithms listed in Table 12 with their specific run-time options. Static overhead (i.e., file headers not related to compression parameters) generated by these tools was disregarded when calculating compression ratios. In addition to using these off-the-shelf dictionary-based and predictive encoders, a software implementation of the ADPCM compression scheme with downstream Rice encoding was used, as presented in Section 5.2. This allows for a fine-grained trade-off between encoder complexity and compression ratio. Audio encoders also relying on Rice encoding (e.g., *MP4ALS*, *ALAC*, or *FLAC*), optimize the Rice parameter (W in Equation 10) for each sample block. In contrast, the ADPCM implementation employed in this section uses a fixed Rice parameter ($W = 4$) found by static optimization on the entire sensor channel.

The compression ratios (compressed data size / uncompressed data size) achieved by the different compression algorithms are shown in Figure 56. Compression ratios larger than 100 % are clipped. As expected, the compression ratios of all encoders improve with increasing block sizes. With the exception of *MP4ALS*, the predictive audio encoders clearly outperform the dictionary-based compression schemes. Given the audio-like characteristics of the neural activity data, this in itself is not surprising. However, the additional encoder complexity necessary for adapting the higher order linear predictor coefficients in the algorithms used in *MP4ALS*, *ALAC*, or *FLAC* does not improve the compression ratios significantly compared to the static first-order DPCM predictor. For instance, at a block size of 2048 samples, the *FLAC* encoder achieves

Codec	Options	Version	Algorithm	Overhead
BZIP2	-9	1.0.6	RLE + BWT + MTF + Huffman	24 B
RAR	-m5 -en	5.00	proprietary	55 B
ZP	c3	1.00	context modeling + arith. coding	221 B
MP4ALS	-7e	RM23	adapt. linear prediction + Rice	34 B
FLAC	-8	1.3.0	adapt. linear prediction + Rice	8292 B
ALAC	ffmpeg	0.8.7-6	adapt. linear prediction + Rice	0 B

Table 12: Compression algorithms and options used in further evaluation

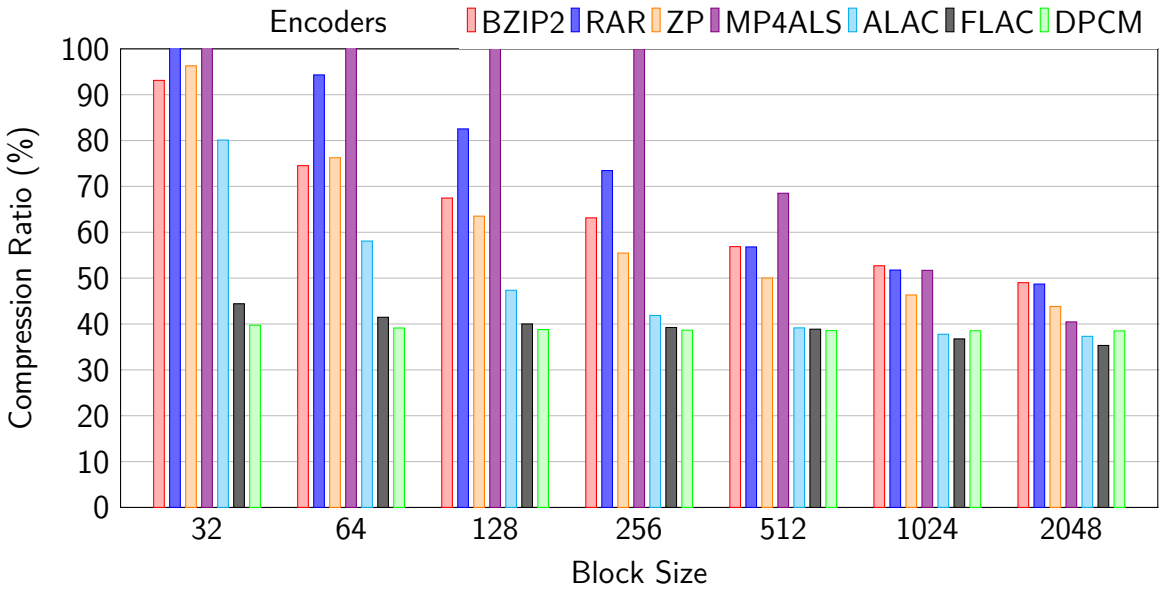


Figure 56: Reduction of neural activity data achieved by various compression schemes

only a 3 % improvement in compression ratios at the cost of double the execution time when compared to the DPCM encoder running on the same platform.

7.1.2 Energy Efficiency of Hardware-Accelerated Data Compression

The DPCM compression is sufficient to reach the target compression ration of 50 % even for small block sized, i.e., minimal end-to-end delay of the signal processing chain from the micro-electrodes to the human operator controlling their penetration depth. The ADPCM hardware accelerator presented in Section 5.2 configured for first-order prediction (either static or adaptive) is thus used for further evaluation of the overall energy efficiency of the in-sensor compression. Furthermore, the maximum block size of 2048 samples is used, derived from the delay-restrictions of the application. The test setup is detailed in Section 5.2.3.

Table 13 summarizes the results for processing the neural data. Both DPCM and ADPCM compression reduce the data size down to 1577 B (38.5 %), making the simpler DPCM algorithm a better choice for this application. The DPCM execution on the MCU takes more than double the 61.2 ms required for transmitting the compressed data stream. The MCU thus has to be kept active beyond the pure transmission time

		Compression			Compressed	Compression + Transmission			
		Time	Current	Energy	Data	Time	Current	Energy	
Scheme	On	[ms]	[mA]	[µJ]	[B]	[ms]	[mA]	[mJ]	[%]
None					4096	158.8	41.5	19.77	**100.0**
DPCM	MCU	132.2	11.8	4680	1577	184.6	21.6	11.96	60.5
ADPCM	MCU	249.4	11.8	8829	1577	310.4	17.5	16.30	82.4
DPCM	RCU	1.4	10.7	45	1577	62.6	41.9	7.87	39.8
ADPCM	RCU	2.6	10.7	83	1577	63.8	41.3	7.90	40.0

Table 13: Energy required to compress/transmit 2048 samples of neural activity data

of the prior packet in order to perform the compression of the current packet. As the current drawn by the system during compression exceeds the current drawn in low-power mode by about 120 times, the longer duty cycles lead to deteriorated overall energy efficiency. Furthermore, the MCU is not even able to support the sampling frequency required by the application. At the targeted 24.414 kHz, the processing of the 2048 samples must not take longer than 84 ms. This is achieved by the hardware-accelerated compression modules. In addition, the RCU to MCU data movement occurs in parallel to the ongoing data transmission, and the very short execution time of the compression stretches the duty cycle before going back to sleep only marginally. This leads to an almost complete translation of compression ratio (38.5 %) into system level energy savings (39.8 %) for the hardware-accelerated DPCM.

7.2 Condition Monitoring of Heavy Industrial Machinery

The second application used for evaluating the hardware-accelerated data compression deals with detecting damage or fatigue of the rotating parts of large industrial machines. This condition monitoring is used to schedule inspection and maintenance intervals before unanticipated major damage leads to high repair costs and downtime of the machinery. Since the monitoring algorithms observe long-term trends in the acquired data, this application is latency insensitive. However, multiple sensor channels have to be handled in parallel for this application, thus increasing the demand for processing power and memory for this application. Note that the sensor nodes are located on heavily vibrating parts of the machine, which would quickly wear out fixed cable connections, thus making low-power wireless communication preferable.

The raw data streams are gathered from a three channel MEMS sensor sampled at 1 kHz with a resolution of 16 bit per channel. Table 14 shows the relevant statistical characteristics of the captured signals. The actual waveforms cannot be shown here for confidentiality reasons. A Rice parameter of $W = 11$ was found to be most appropriate for all three sensor channels. The analysis described in this section has already been published in [Engel2014a].

Channel	1	2	3
Min	-3872	-4156	-22466
Max	5203	13104	16307
Mean	110	2919	-3339
Stddev	1213	2374	13041

Table 14: Statistical characteristics of 8192 samples of machinery condition monitoring data

7.2.1 Compressibility of the Monitored Data

Just as for the neural data monitoring application in Section 7.1, a PC-based baseline analysis of the compressibility of the condition monitoring data is carried out. The resulting overall compression ratios are shown in Figure 57. Again, the predictive audio codecs are most appropriate. At a block size of 2048 samples, *FLAC* produces results 6 % smaller than DPCM. This improvement comes however at the cost of a doubled execution time.

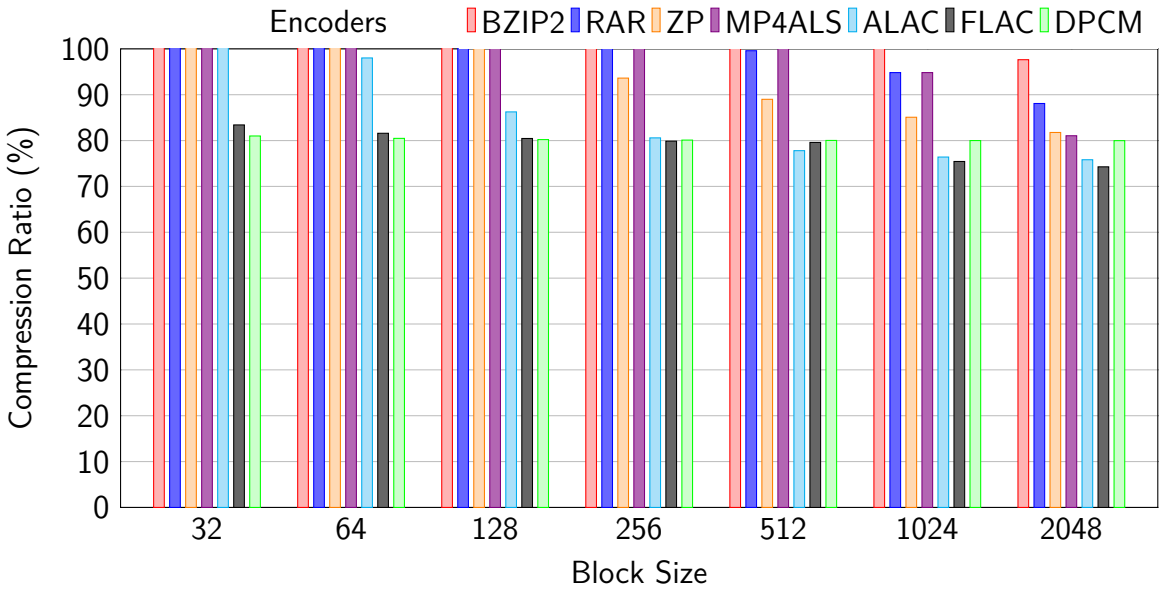


Figure 57: Reduction of condition monitoring data achieved by various compression algorithms

The ADPCM scheme is examined more in detail for this application. Figure 58 quantifies the impact of the prediction order. For small blocks, the size of the additional prediction parameters, which have to be encoded in the output stream, exceeds the benefit of improved compression ratios due to the reduced prediction error variance. For blocks of 2048 samples, the compression ratio strictly decreases with the prediction order, but only marginal improvements (about 1.2 %) can be achieved by increasing the prediction order above 4. The difference of the compression ratio between first and fourth order prediction amounts to 6 % for the largest block size. Thus, adaptive

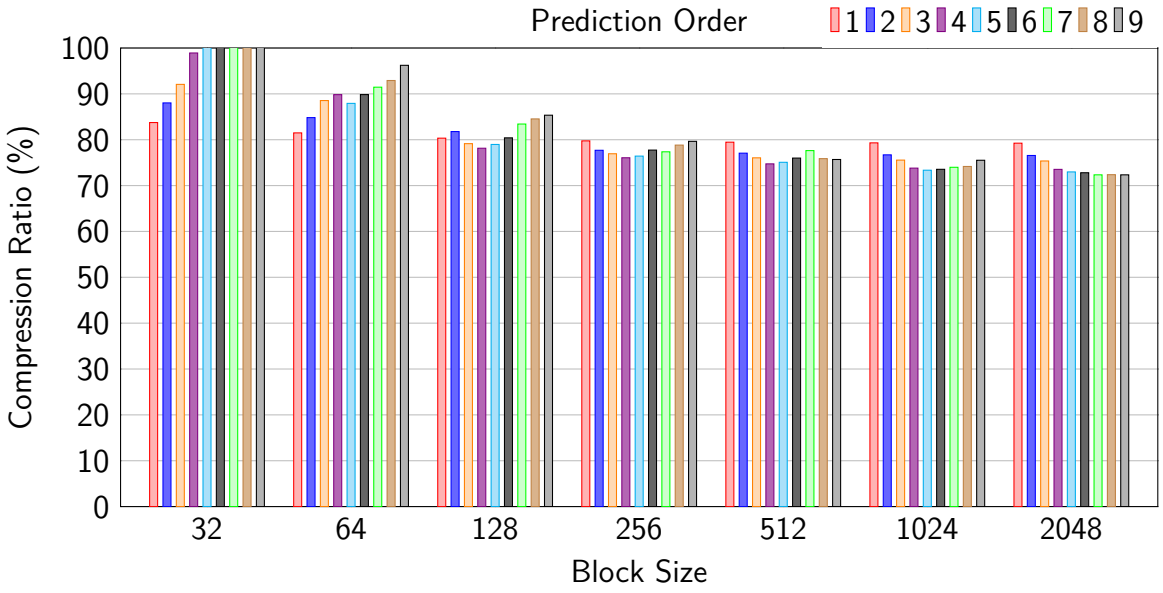


Figure 58: Impact of prediction order on ADPCM compression ratio

prediction should be used for condition monitoring if the target platform supports storing a sufficient amount of samples (recall that there are three data channels, and double-buffering may be necessary for parallel sampling and encoding).

In conclusion, the adaptive schemes reached the best compression ratios for larger sample blocks, while static DPCM proved superior on small blocks. Thus, for delay-sensitive applications or memory-constrained platforms, the simple DPCM scheme should be applied. In all other cases, FLAC can improve the compression ratio of DPCM by about 5 %. Note that *FLAC* is just a combination of ADPCM with an extensive search for the optimal prediction order. For data sources with known characteristics, a static selection of the prediction order may be sufficient. Compared to the static DPCM, the adaptation of the prediction coefficients in ADPCM only becomes worthwhile if the additional energy required for the second compression pass is smaller than the energy savings resulting from the 5 % data size reduction. Therefore, energy efficiency of a hardware implementation of the ADPCM algorithm is also examined in the next section.

7.2.2 Energy Efficiency of Hardware-Accelerated Data Compression

The ADPCM hardware accelerator presented in Section 5.2 configured for first-order prediction (either static or adaptive) is used for further evaluation of the overall energy efficiency of the in-sensor compression. Although the energy efficiency of higher order prediction would be worth examining, this feature is not yet supported by the HaLOEWEn implementation presented in Section 5.2. Furthermore, the maximum block size of 2048 samples is used, derived from the restricted on-chip memory of the HaLOEWEn3 (see Table 8). The test setup is detailed in Section 5.2.3.

Processing of the condition monitoring data is evaluated in Table 15. Here, the adaptive predictor improves the compression ratio over DPCM such that a complete radio packet is saved. On the MCU, however, compressing the three data channels takes so much time, that the energy required for compression cannot be amortized over the transmission savings, instead leading to an efficiency deterioration. The hardware-accelerated encoders fare much better: Since all channels can be compressed in parallel, the total execution time only slightly increases over that of the single channel encoder used in Table 13. As before, the encoders using the RCU convert nearly all of the data volume savings into actual energy savings, i.e., only 81.3 % of the baseline energy.

		Compression			Compressed	Compression + Transmission			
		Time	Current	Energy	Data	Time	Current	Energy	
Scheme	On	[ms]	[mA]	[µJ]	[B]	[ms]	[mA]	[mJ]	[%]
None					12288	474.4	41.6	59.21	**100.0**
DPCM	MCU	531.0	11.9	18957	9836	912.0	24.3	66.48	112.3
ADPCM	MCU	906.0	11.9	32344	9732	1281.0	20.6	79.17	133.7
DPCM	RCU	1.6	10.7	51	9836	381.6	42.5	48.62	82.1
ADPCM	RCU	2.9	10.7	93	9732	378.9	42.4	48.15	81.3

Table 15: Energy required to compress/transmit 3×2048 MEMS samples

7.3 Distributed Structural Health Monitoring

Civil infrastructures such as bridges are prone to fatigue and other load-induced damage. With increasing age, inspection intervals have to be scheduled more frequently to assure the secure operation of the infrastructure. As manual inspections are costly, they have to be complemented by automated Structural Health Monitoring (SHM) systems. By periodically observing modal properties such as the eigenfrequencies, mode shapes, or the damping of the structures, damage or fatigue can be identified by significant deviations of these properties from reference measurements.

As shown on the left hand side of Figure 59, modal parameters of an object are typically derived by observing the structural response to an artificial excitation. The Experimental Modal Analysis (EMA) requires the knowledge about the excitation and the response to derive the modal parameters. While large structures can be excited by appropriate equipment, a significant amount of energy is required to drive such large shakers and impact hammers. Furthermore, the excited structures often have to be taken out of service to assure safety and measurement accuracy. For continuous automated SHM, the identification of the modal properties thus has to be based on the

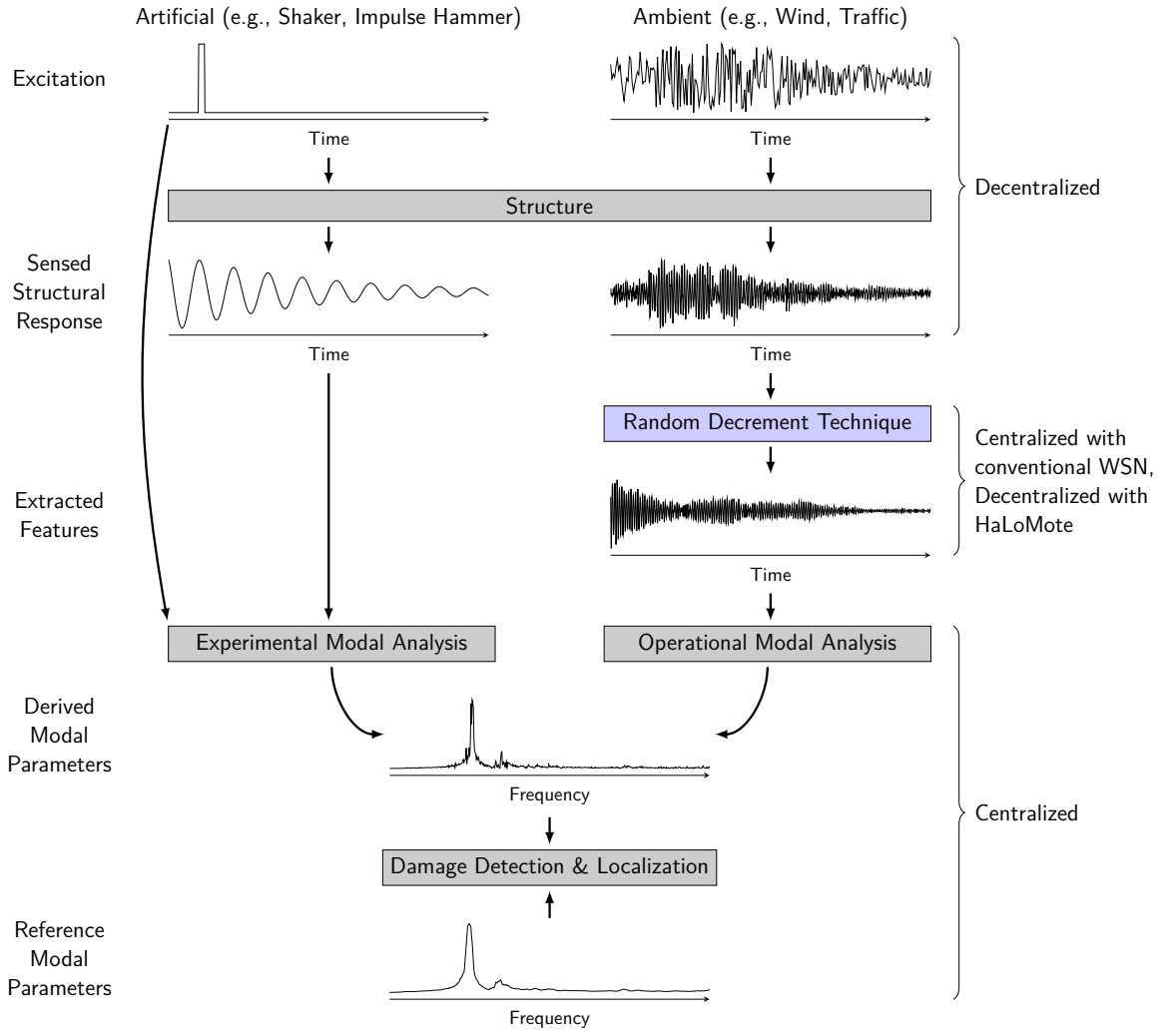


Figure 59: SHM dataflow based on experimental and operational modal analysis

natural ambient vibration of the structure caused by wind or traffic, as shown on the right hand side of Figure 59. The major challenge of this Operational Modal Analysis (OMA) is the extraction of the features within the structural response, which are only related to the structure but independent from the random excitation. The Random Decrement Technique (RDT) was proposed for this purpose [Cole1973]. Finally, by comparing the derived modal parameters with reference parameters identified for a well known (healthy) state of the structure, damage can be identified of even localized.

The modal analysis and damage detection requires information gathered from all over the structure to capture the global properties of the observed object. These algorithms are therefore typically executed on a centralized controller. On the other side, the sensing of the structural response has to be decentralized, as the sensors are distributed all over the structure. As will be shown in Section 7.3.2, the RDT requires only little information from other neighboring sensors while strongly aggregating the local sensor data. The RDT should therefore also be decentralized. Nevertheless, RDT-based SHM installations using conventional WSNs with small MCUs often avoid the decentralization of the RDT due to its computational complexity.

The remaining part of this section therefore shows, how the HaLoMote can be used to efficiently decentralize the RDT. The SHM application and the hardware-accelerated RDT implementation on the HaLOEWEn3 mote have already been published in [Engel2012a; Engel2014c; Engel2015c].

7.3.1 Random Decrement Technique

For the RDT, a set of sensors $S \subset \mathbb{N}$ is distributed all over the structure to acquire its vibrations in terms of acceleration or deflection as time series $(x_s : T \mapsto V)_{s \in S}$. For a finite sampling rate and measurement duration, the time domain is also finite and discrete, i.e., $T = \{0, \ldots, n_t - 1\} \subset \mathbb{N}$. For simplicity, $V \subset \mathbb{R}$ can be assumed by abstracting from the finite measurement accuracy.

As the OMA aims to discover the dynamic characteristics of the observed structure, the *static* components of the acquired signals (gravity or prestress) have to be eliminated by a high-pass filter, e.g., by applying an FIR filter of order $n_f \in \mathbb{N}$ with $n_f + 1$ appropriate coefficients $c_k \in \mathbb{R}$. A reformulation of Equation 5 for the time series input yields

$$\hat{x}_s(t) \coloneqq \sum_{k=0}^{\min(n_f,t)} c_k \cdot x_s(t-k) \qquad \forall (s,t) \in S \times T. \tag{75}$$

To eliminate the *random* signal components, the RDT selects a subset of the sensors as references $R \subseteq S$ and a trigger level $l_r \in V$ for each reference $r \in R$. In practice, sensors exposed to the largest vibration amplitudes are typically selected as references, to observe the structure's activity even under shallow excitation. Furthermore, the trigger levels have to be chosen above the noise floor of the acquired sensor signal. The points in time $t \in T$, at which a reference signal $\hat{x}_r$ crosses l_r, are referred to as trigger events

$$\begin{aligned} E_r \coloneqq \{t \in T : &(\hat{x}_r(t) \geq l_r \wedge \hat{x}_r(t-1) < l_r) \vee \\ &(\hat{x}_r(t) < l_r \wedge \hat{x}_r(t-1) \geq l_r)\} \qquad \forall r \in R. \end{aligned} \tag{76}$$

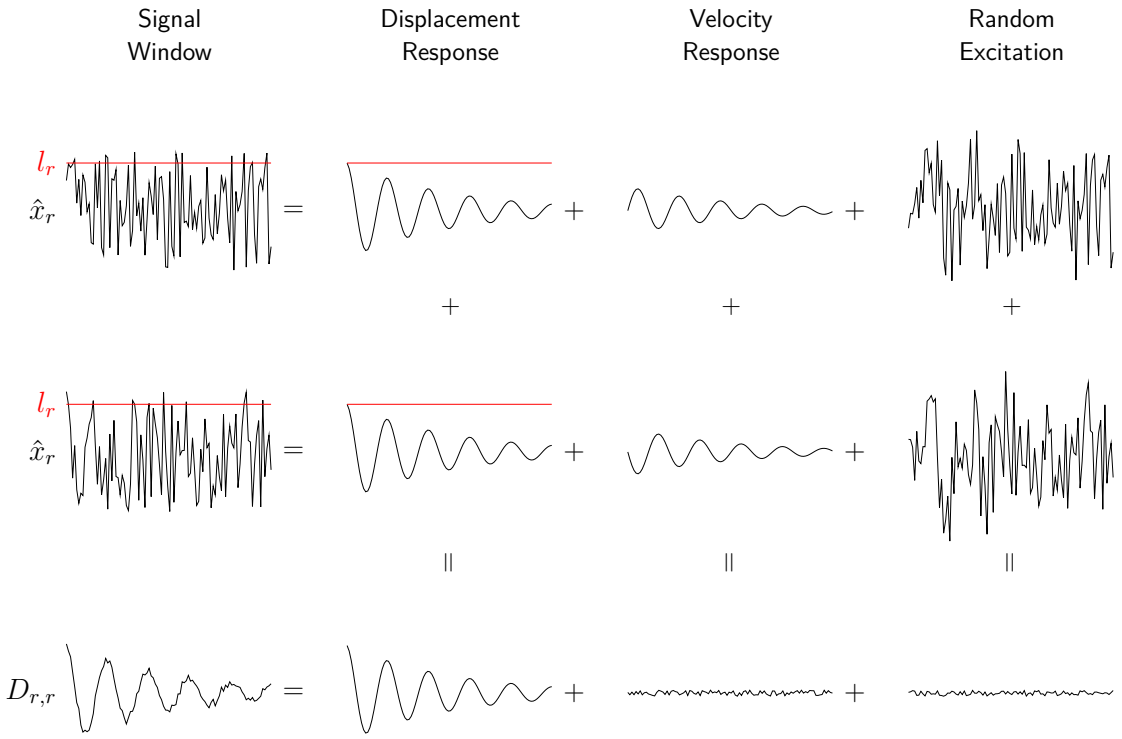


Figure 60: Accumulation of triggered signal windows for the Random Decrement Technique. Each window represents the qualitative behavior of the sensor signal (i.e., acceleration or deflection) over time.

A signal window $(\hat{x}_r(t+k))_{k=0}^{n_w-1}$ of fixed length $n_w \in \mathbb{N}$ starting at a trigger event is composed of the structure's response to its initial displacement $\hat{x}_r(t) = l_r$, its response to the initial velocity and the random ambient excitation, as shown in Figure 60. Assuming a zero-mean excitation, the random components are extinguished when accumulating a sufficient number of these triggered windows. The velocity response will also be eliminated, as each rising signal edge (with positive initial velocity) is followed by a falling signal edge (with negative initial velocity). Thus, the accumulated signal windows converge against the displacement response, which describes the structures free decay and can thus be used to derive its modal properties.

To estimate the mode shapes of the structure, spatial correlations between different sensor positions are required. Therefore, signal windows from all sensors are accumulated for each trigger event, resulting in $|S| \cdot |R|$ so called RD sequences

$$D_{s,r}(k) \coloneqq \sum_{t \in E_r} \hat{x}_s(t+k) \qquad \forall (s,r) \in S \times R,\ 0 \leq k < n_w. \tag{77}$$

7.3.2 Distributed Random Decrement Technique

The accumulation of the RD sequence $D_{s,r}$ requires an information transfer about the occurrence of the trigger events $t \in E_r$ from signal r to signal s. In a distributed WSN scenario, all sensors signals are captured by a dedicated sensor node. Although each sensor node may capture multiple nearby sensor signals, information about the trigger events have to be transmitted wirelessly from the reference nodes (capturing a reference

signal) to all other nodes in the network. When monitoring structures larger than the transmission range of the wireless transceivers (i.e., suspension bridges), the transport of the trigger information has to be organized by a multi-hop routing protocol.

In Section 6.1, a multi-source flooding mechanism was proposed for this purpose. The complete dissemination of the trigger events, i.e., the schedule length determined by the ILP or the corresponding heuristic, depends on the network size, its density, and the number of source nodes (i.e., $|R|$). To simplify the DPM of the network nodes, the sampling cycles are used as scheduling cycles, to avoid additional transitions between active and sleep states. Using this static scheduling approach, the maximum number of sampling cycles required to transfer trigger events from each reference node to all other nodes can by determined at the time of deployment. Let Δ denote this maximum communication delay.

In the multi-source flooding scheme, each trigger event is represented by the ID of its originating reference node and the age of the event. The latter starts at zero during the trigger event generation, and is incremented in every sampling period at all nodes that have received and stored the event. At the receiver nodes, the age information is used to properly calculate the RD sequences, as described in the next section. The reference ID as well as the age information can be represented by a few bits ($\leq 10\,\mathsf{bit}$ for both in realistic scenarios), thus fulfilling the prerequisites of the multi-source flooding mechanism stated in Section 6.1.

The trigger event flooding is only required as long as the RD sequences are accumulated. Once the number of processed trigger events fulfills the accuracy requirements of the application, the RD sequences have to be transmitted to a base station in addition to some statistical measures of the reference signals (i.e., channel variance and number of generated trigger events). The RDT thus aggregates the $|S| \cdot |T|$ raw sensor samples down to $|S| \cdot |R| \cdot n_w$ RD samples. The aggregation factor $\frac{|T|}{|R| \cdot n_w}$ increases with the measurement duration. n_w is typically chosen such that the RD sequences show the free decay of the structure, which may take several seconds for large bridges. After a measurement duration of several hours, which is often required to collect a sufficient number of trigger events, the aggregation factor typically exceeds two or more decades. For example, the laboratory-scale testbed presented in Chapter 8 uses $|R| = 2$ references to trigger signal windows, each consisting of $n_w = 256$ samples. After a 10 min measurement at 400 Hz sampling rate (i.e., $|T| = 2.4 \times 10^5$) the RD sequences that have to be transmitted to the subsequent OMA are 469× smaller then the the original raw sensor signals. In addition to eliminating the random parts of the sampled signals, this strong aggregation capability is the major benefit of the RDT for the distributed WSN implementation of SHM applications.

However, the RDT increases the demand for in-sensor preprocessing. The computational complexity of the RDT is dominated by the FIR filter and the memory accesses required for the accumulation of the RD sequences. The latter becomes particularly complex if these sequences cannot be stored in the few kilobytes of RAM provided by most WSN processing units, thus requiring access to external memory. RDT preprocessing scales linearly with the number of sensor channels to be processed by each sensor node. Each sensor typically provides three channels to capture the multidimensional movement of the structure, and multiple nearby sensors may be connected to a single sensor node to simplify the deployment. Thus, assuming three to twelve sensor channels per mote is not unrealistic. As the sensor channels can be processed indepen-

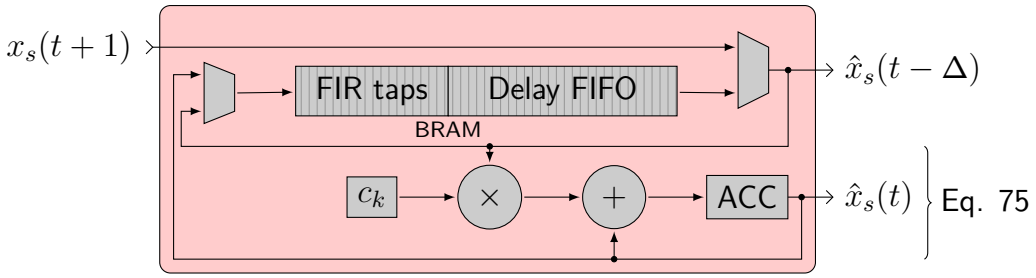


Figure 61: FIR and FIFO in single BRAM

dently of each other, most of the RDT preprocessing can be parallelized. Section 7.3.3 details the RDT implementation on the HaLoMote RCU.

7.3.3 Hardware-Accelerated Random Decrement Technique

In this section, an RDT hardware kernel for the HaLoMote architecture is presented. The hardware signals and modules presented in Figure 61 to 65 are associated with Equations 75 to 80. More precisely, the labels refer to the channel index s of the captured sensor input and the current time slot (i.e., sampling cycle) t, in which $\hat{x}_s(t)$ is generated by the high-pass filter.

Figure 61 shows the FIR implementation of this high-pass filter. As described at the end of this section (Figure 66), the FIR is executed in parallel to the sensor sampling. At the end of time slot t, the unfiltered sensor sample $x_s(t+1)$ for the next time slot is thus already present at the FIR input, while the filtered value $\hat{x}_s(t)$ is present at the FIR output. The filter itself is executed sequentially, thus requiring only one MACC unit to combine the constant coefficients c_k with the FIR taps buffered in BRAM.

The filtered value $\hat{x}_s(t)$ is passed through an additional FIFO buffer, which is integrated into the same BRAM as the FIR taps. Delaying the filtered samples by Δ time slots is required to compensate the maximum time required for the trigger event flooding over the entire network, as described in the previous section.

Figure 62 shows the computational logic required for each sensor channel. A sensor specific control module requests the samples over the digital sensor interface. Although most of the control lines of the sensor interfaces (e.g., SPI or I^2C) could be shared between multiple sensors, each sensor channel is controlled by a dedicated interface to allow for parallel independent sensor sampling.

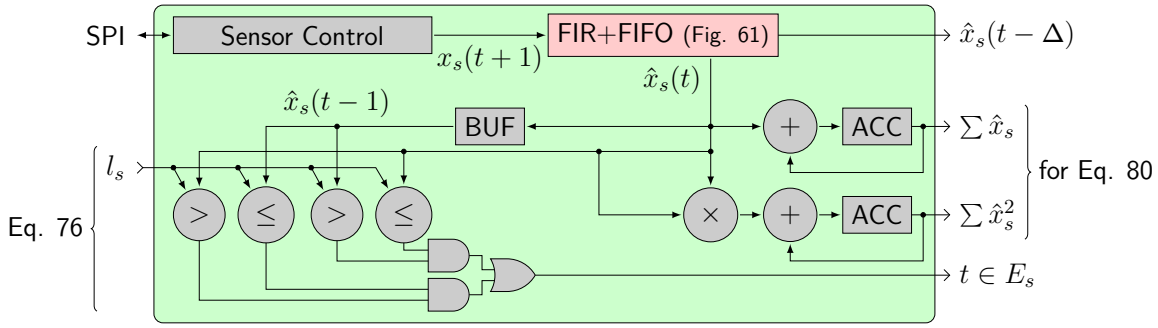


Figure 62: Sensor interface, trigger event detection and precomputation required for calculating the standard deviation

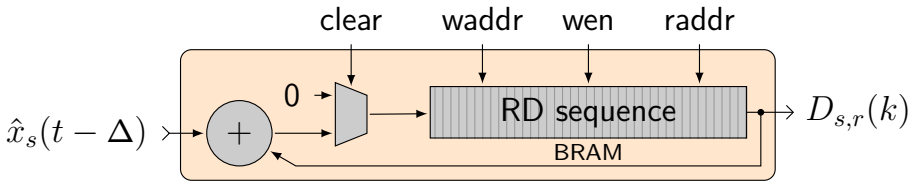


Figure 63: Accumulation of RD sequence (Equation 77)

The sensor samples are filtered and delayed by the module shown in Figure 61, as described above. The filtered samples are used to detect trigger events by comparing the current $\hat{x}_s(t)$ and the last $\hat{x}_s(t-1)$ value with the trigger level l_s as described by Equation 76. Furthermore, the filtered values and their squares are accumulated as running sums. As will be shown in Equation 80, these sums are required to derive the standard deviation of the sensor channel for a normalization step of the final OMA. Both, the trigger event detection and the standard deviation calculation are only required for sensor channels configured as RDT references (i.e., if $s \in R$). To simplify the network configuration, the reference-specific hardware is provided for each sensor channel and the software processor decides which of the results to use for further processing.

Figure 63 shows the module used for accumulating the delayed samples to an RD sequence stored in BRAM. External trigger event-specific logic (see Figure 64) decides whether and which memory position to modify. Additional clear logic is required to initialize the buffer at the start of each measurement.

Figure 64 shows the handling of trigger events for the reference channel r. This logic is required at all sensor nodes, not only at the sensor node sampling the reference signal. A trigger event $e_{r,k} \in E_r$ defined by Equation 76 is characterized by its current age $t - e_{r,k}$, i.e., the number of sampling cycles since it was generated by the logic shown in Figure 62. New events can be inserted into the list, and the age of each event in the list is incremented in every sampling cycle. As soon as $t - e_{r,k} = \Delta$, the sample at the output of the delay buffer $\hat{x}_s(t-\Delta) = \hat{x}_s(e_{r,k})$ corresponds to the filtered sample of sensor s from the time slot $e_{r,k}$, in which the trigger was generated at sensor r. Therefore, trigger events at least as old as Δ cause an accumulation of the delayed sample to the appropriate RD sequence, i.e., $D_{s,r} \mathrel{+}= \hat{x}_s(t - e_{r,k} - \Delta)$. Two RCU clock cycles are required for reading the old and writing the new value. However, due to the dual port BRAM, subsequent accumulations can be interleaved, thus resulting in a

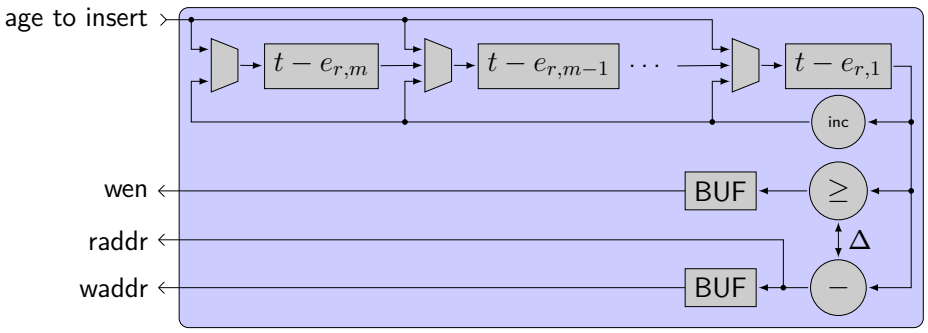


Figure 64: Handling of trigger events $E_r = \{e_{r,1}, \ldots, e_{r,m}\}$ for reference $r \in R$

throughput of one accumulation per sampling cycle. Trigger events older than $n_w + \Delta$ are removed from the event list.

The main difficulty of the trigger event management is that the sequence of event insertions does not necessarily have to match the sequence of event removals. For example, a trigger event generated at the local node will be inserted immediately with an age of 0. In the next sampling cycle, another trigger event generated at a remote node may arrive that already traveled for 10 sampling cycles on a multi-hop path to the local node. This second trigger event will thus be inserted *after* the first event, but it will be removed *before* the first event. To avoid the fragmentation of the data structure storing the trigger events, a shift register based queue is used. In each sampling cycle, each trigger event is dequeued and enqueued again after its incrementation unless it is old enough to be removed. New events are enqueued afterwards. The actual length of the queue is managed by dedicated logic and corresponds to the number of currently active (i.e., overlapping) signal windows triggered by the reference channel r.

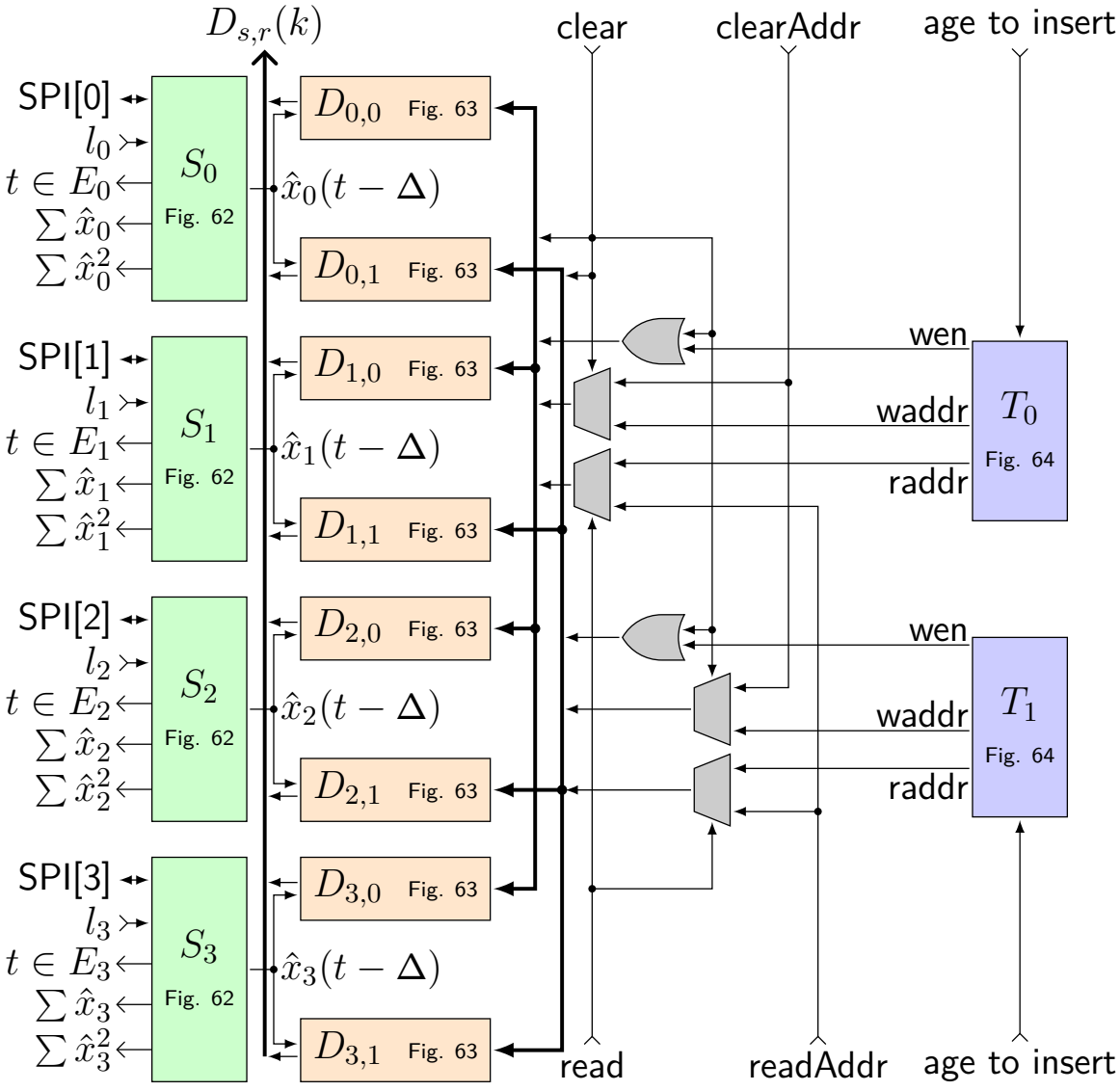


Figure 65: RDT kernel for four sensor channels and two reference signals

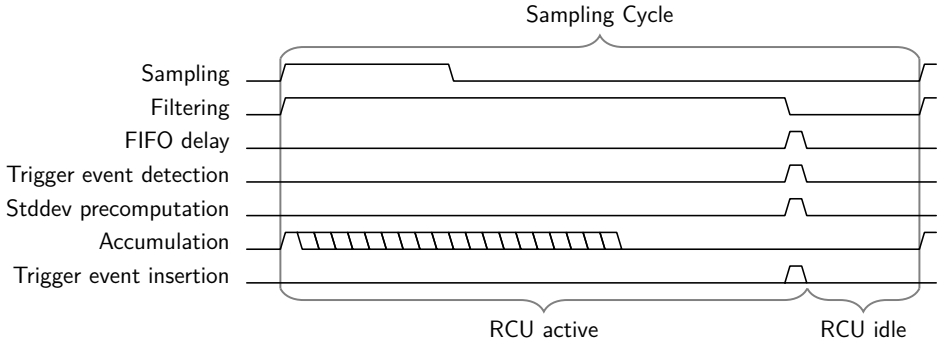
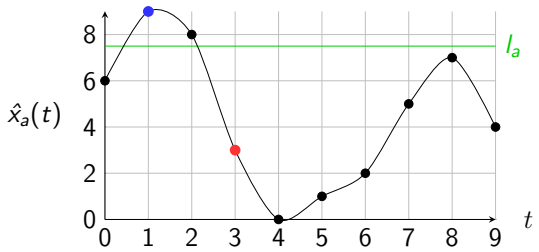


Figure 66: Scheduling of parallel computations

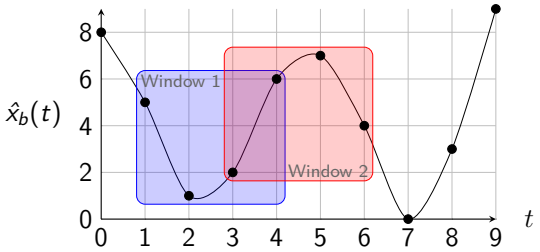
Figure 65 shows the combination of all these modules to realize an RDT kernel for $|S| = 4$ sensor channels and $|R| = 2$ reference signals. The SPI ports of this module are connected to the discrete digital sensors. All other ports are controlled by the HaLoMote MCU using the communication infrastructure described Section 4.3. The BRAM addressing for the RD sequences are overruled for clearing and reading of individual RD sequences. As all RD sequences of a single reference channel share the same addressing signals, BRAM can be shared among those modules.

Figure 66 details the RDT kernel execution sequence. At the start of each sampling cycle, the MCU wakes up the RCU, which starts requesting the next sensor samples. The number of RCU cycles required for this operation depends on the number of bits to read, which typically do not exceed $3 \cdot 16\,\text{bit}$. The high-pass filter operates on the sample from the previous sampling cycle and can thus be executed in parallel to the sensor sampling. The filtering takes $n_f + 1$ RCU cycles. The trigger event detection, the computation for the standard deviation, and the handling of the delay FIFO require one additional RCU cycle after the filtering operation. The accumulation of the delayed sample value from the last sampling cycle to the RD sequences is performed in parallel to the filtering. It requires one RCU cycle per registered trigger event. If available, new trigger events are inserted just before the RCU is shutdown again. As long as the number of registered trigger events does not exceed the order of the FIR filter, the overall execution time of the RDT kernel is fixed.

A small example shown in Figure 67 illustrates the distributed processing principle and the delayed trigger handling of the RDT implementation described in this section. To keep it simple, only two sensor signals $S = \{a, b\}$ are considered and Figures 67a and 67b show the already low-pass filtered sensor samples. Signal a is chosen as the sole reference signal $R = \{a\}$ with a trigger level of $l_a = 7.5$. The first trigger level overshoot (undershoot) of $\hat{x}_a$ at Time 1 (3) triggers the first (second) signal window to be extracted from the response signal, as shown in Figure 67b. If the signals a and b are sampled at different nodes within the WSN, the trigger events $E_a = \{1, 3\}$ have to be forwarded from the reference to the response node. This is achieved by combining the reference index a with the age $t - e_{a,i}$ into a wireless packet flooded through the network, as described in Section 7.3.2. The red (blue) arrow in Table 67c illustrates this wireless event distribution assuming that the first (second) trigger event requires five (two) sampling cycles to reach response node. Although the flooding mechanism described in Section 6.1 assures a fixed delay between the reference and response node,



(a) Filtered reference signal



(b) Filtered response signal

t	0	1	2	3	4	5	6	6	6	7	7	7	7	8	8	8	8	9	9	9	9	10	10	10	10	11	11	11	12	12	12
RCU cycle	9	9	9	9	9	9	0	1	9	0	1	2	9	0	1	2	9	0	1	2	9	0	1	2	9	0	1	9	0	1	9
On Reference Node:																															
$\hat{x}_a(t)$	6	9	8	3	0	1	1	1	2	2	2	2	5	5	5	5	7	7	7	7	4	4	4	4							
$t \in E_a$	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
On Response Node:																															
age to insert						3	6	6	6											2											
$\lvert E_a \rvert$	0	0	0	0	0	0	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	0
$t - e_{r,1}$							3	4	4	4	6	5	5	5	7	6	6	6	8	7	7	7	9	8	8	8	9	9	9		
$t - e_{r,2}$										6	5	7	7	7	6	8	8	8	7	9	9	9	8								
wen							0	0		0	0	1		0	0	1		0	1	1		0	1	1		0	1		0	1	
raddr							-3	-2		-2	0	-1		-1	1	0		0	2	1		1	3	2		2	3		3		
waddr								-3			-2	0			-1	1			0	2			1	3			2			3	
$\hat{x}_b(t)$	8	5	1	2	6	7	7	7	4	4	4	4	0	0	0	0	3	3	3	3	9	9	9	9							
$\hat{x}_b(t - \Delta)$							8	8	8	5	5	5	5	1	1	1	1	2	2	2	2	6	6	6	6	7	7	7	4	4	4
$D_{b,a}(0)$	0	0	0	0	0	0	0	0	0	0	0	0	5	5	5	5	5	5	5	5+2	7	7	7	7	7	7	7	7	7	7	7
$D_{b,a}(1)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1+6	7	7	7	7	7	7	7
$D_{b,a}(2)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2+7	9	9	9	9
$D_{b,a}(3)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	6	6	6	6	6	6+4

(c) Control signals

Figure 67: Example for the main processing sequence of the distributed RDT for $S = \{a, b\}$, $R = \{a\}$, $l_a = 7.5$, $n_t = 10$, $n_f = 8$, $n_w = 4$, $\Delta = 6$.

the robustness of this RDT implementation against variable network delays can be demonstrated with these example delays.

For an assumed FIR length of $n_f = 8$ and an even smaller delay of the serial sensor sampling, the RCU executes ten clock cycles in every sampling period, as described by Figure 66. Out of these ten clock cycles, Table 67c shows only the cycles required to demonstrate the trigger event handling and RD sequence calculation. While the FIR output changes at the end of cycle 8 and thus becomes visible in cycle 9, the FIFO output changes at the end of cycle 9 and thus becomes visible in the first cycle of the next sampling period. After receiving the first (second) trigger event in the sampling period $t = 5$ (6) and inserting the event into the active trigger queue E_a, the active trigger events are handled in the first two or three RCU cycles of the sampling periods six to twelve. As described in Figure 64, this handling includes the derivation of the read and write signals for the RD sequence accumulator (i.e., wen, raddr, and waddr)

by comparing the queue head (i.e., $t - e_{a,1}$) with the length Δ of the delay FIFO. As $\Delta = 6$ is assumed for this example, the first (blue) trigger event activates the accumulator in the sampling periods seven to ten, while the second (red) trigger event activates the accumulator in the sampling periods nine to twelve. The $n_w = 4$ element wide RD sequence $D_{b,a}$ is thus generated by accumulating the output $\hat{x}_b(t - \Delta)$ of the delay FIFO to the appropriate memory locations. Trigger events that would reach an age of $\Delta + n_w = 10$ are removed from the event queue in the sampling cycles ten and twelve. The accumulated RD sequence $D_{b,a}$ is the RDT result and required for the subsequent modal analysis, as described in the next section.

7.3.4 Operational Modal Analysis and Damage Detection Principles

[Asmussen1997] proved that a connection between the RD sequence $D_{s,r}$ and the correlation sequence $R_{s,r}$ can be established by normalizing with the number of detected trigger events $|E_r|$, the trigger level l_r and the standard deviation of the reference signals:

$$R_{s,r}(k) \coloneqq D_{s,r}(k) \cdot \frac{\sqrt{\sigma_r^2}}{l_r \cdot |E_r|} \qquad \forall (s,r) \in S \times R \tag{78}$$

$$\sigma_r^2 \coloneqq \frac{1}{|T|} \sum_{t \in T} \left(\hat{x}_r(t) - \left(\frac{1}{|T|} \sum_{t \in T} \hat{x}_r(t) \right) \right)^2 \qquad r \in R \tag{79}$$

$$= \frac{1}{|T|} \sum_{t \in T} \hat{x}_r(t)^2 - \left(\frac{1}{|T|} \sum_{t \in T} \hat{x}_r(t) \right)^2 \qquad r \in R \tag{80}$$

The second formulation for the variance of the reference signal r results from applying the binomial formula. To calculate the variance, only the running sums of $\hat{x}_r$ and $\hat{x}_r^2$ thus have to be collected by the sensor nodes.

In the WSN testbed described in Chapter 8, the OMA and the subsequent damage detection is executed at a *central base station* running [MATLAB] and not at a dedicated WSN mote. The individual OMA steps are therefore just summarized briefly in this section. More details can be found in [Engel2014c].

Having received all RD sequences from the WSN, the base station derives the correlation sequences according to Equation 78. Afterwards, a single block Discrete Fourier Transformation (DFT), followed by a Frequency Domain Decomposition (FDD) for each frequency bin of the observed spectrum is executed [Brincker2000]. In particular, a Singular Value Decomposition (SVD) is followed by a peak-detection on the resulting singular values to find the eigenfrequencies of the observed structure.

The FDD also yields the mode shapes of the structure for each of the detected eigenfrequencies, i.e., the maximum deflection ϕ_s of the structure at the position of the sensor s. After normalizing the mode shape to

$$F_s \coloneqq \frac{\phi_s^2}{\sum_{k \in S} \phi_k^2} \qquad \forall s \in S, \tag{81}$$

it can be compared with a mode shape $(\bar{F}_s)_{s \in S}$ captured during a reference measurement. A damage index for each sensor position is derived as

$$DI_s := \frac{F_s + 1}{\bar{F}_s + 1} - 1 \qquad \forall s \in S \tag{82}$$

Based on the Modal Strain Energy (MSE) method [Stubbs1995], an increased DI_s is expected, if the structure's stability has changed between the reference and the current measurement near the sensor s.

CHAPTER 8

System Level Evaluation: Distributed Monitoring of a Railroad Bridge Model

The SHM demonstrator used to evaluate the capabilities of the HaLoMote architecture in a complete WSN application consists of a bridge model equipped with a network of HaLOEWEn motes. These motes are performing the RDT as described in Section 7.3, to act as a decentralized Data Acquisition (DAQ) system. In addition, an embedded PC running [MATLAB] is used for the centralized OMA and damage detection algorithms. The bridge can be excited by an impact hammer, or a train model. A wire-bound DAQ system can be installed to capture baseline ("ground truth") measurements. This demonstrator was developed in cooperation with the Smart Structures Research Division of the Fraunhofer Institute for Structural Durability and System Reliability (LBF) and already described in [Engel2014c; Engel2015c]. It was presented at the Hannover Fair in 2014 and now is a permanent exhibit at the Transfer Center for Adaptronics at the Fraunhofer LBF, as shown in Figure 68. It was also covered by some local media reports (e.g., [Yannick2015]).

Figure 68: SHM demonstrator exhibit located at the Transfer Center for Adaptronics (Fraunhofer LBF)

8.1 Demonstrator Setup

A warren truss railroad bridge was modeled by connecting 54 metal rods with 24 metal joints resulting in 51 kg overall weight and a span width of 246 cm. The test structure can be excited by an impulse hammer or a 2.3 kg G-scale railway model crossing the bridge.

The four outer joints (marked green in Figure 69) are connected to fixed pedestals. They can rotate in either directions without changing their position. Thus, to assess the dynamic movement of the structure, only the 20 inner joints (marked red and labeled as S_1 to S_{20} in Figure 69) have to be monitored. This is achieved by connecting a 3-axis [ADXL362] MEMS acceleration sensor to each of the inner joints. This sensor

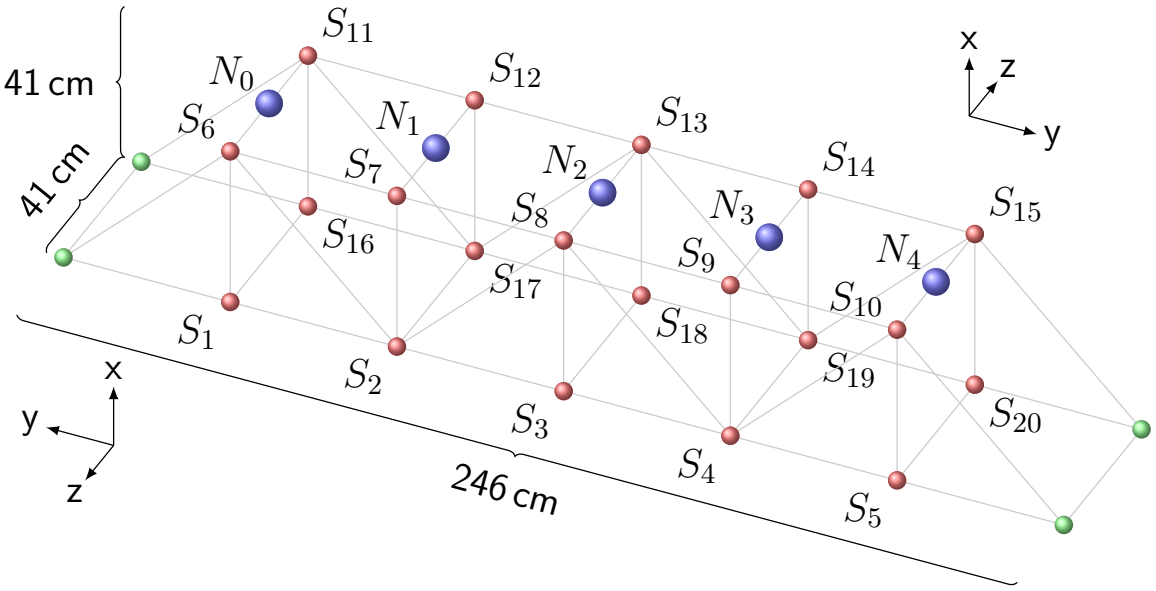
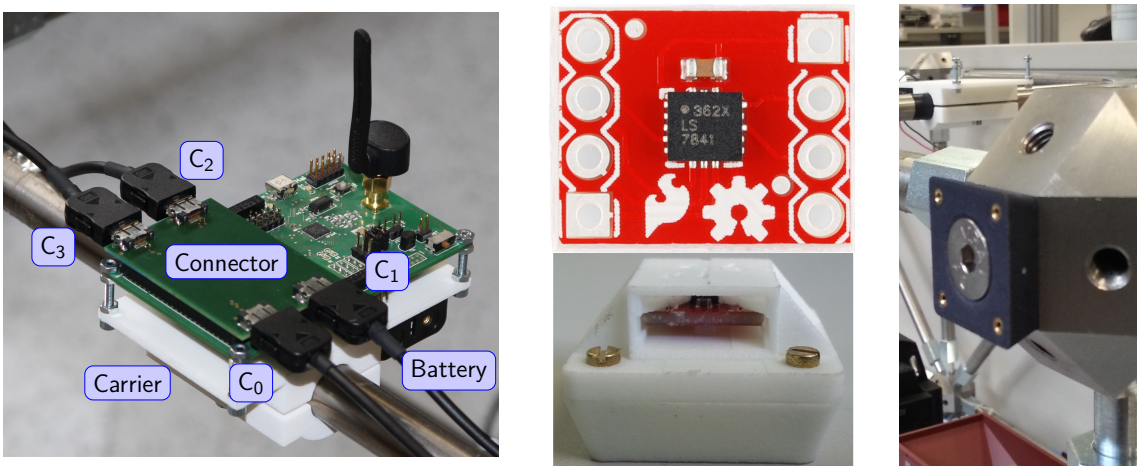


Figure 69: Sensor position and orientation on the truss bridge model

was chosen as a trade-off between power consumption (i.e., 6 µW at 400 Hz sampling rate) and measurement accuracy (i.e., about 1 mg resolution).

The COTS sensor breakout boards are sealed in a housing shown in Figure 70b. This housing was designed and additively manufactured at the Fraunhofer LBF. The housings are screwed to the joints facing outside the structure, as shown in Figure 70c. The resulting sensor channel orientation is shown in Figure 69 and differs for the front (S_1 to S_{10}) and the rear side (S_{11} to S_{20}) of the bridge. However, as the railway model excites the bridge mainly in vertical direction (i.e., orthogonal to the bridge deck), only the x-channel of each sensor is sampled for both sides of the bridge.

For the wireless DAQ system, five HaLOEWEn3 motes were mounted on the upper rods (marked blue and labeled as N_0 to N_4 in Figure 69). Special carriers were designed and additively manufactured at the Fraunhofer LBF. As shown in Figure 70a, these



(a) Mote mounting (b) Sensor housing (c) Sensor mounting

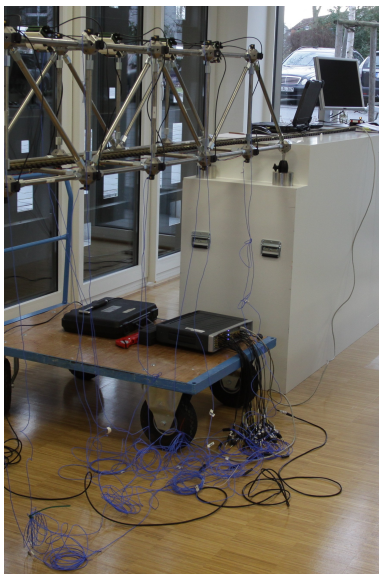
Figure 70: Mounting of sensor mote and sensors

carriers are clamped on the rods and comprise a battery compartment to power the HaLOEWEn screwed on top.

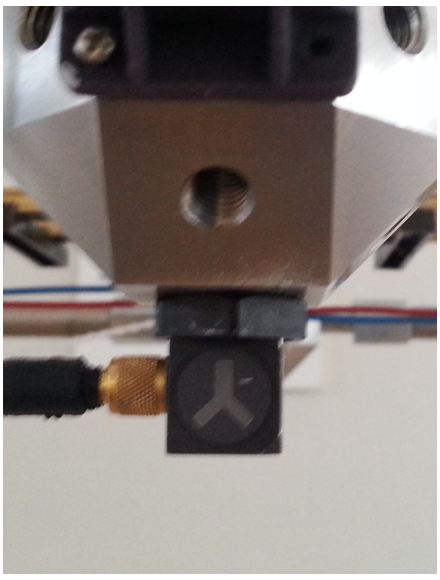
Finally, a connector board was designed and manufactured as PCB to connect the four sensors located in the same x-z-plane as the mote to its RCU. Four Hirose [ST60-10P] connectors (labeled as C_0 to C_3 in Figure 70a) are used to flexibly connect the sensors to the expansion headers.

Due to the relatively large stiffness of the small bridge model, the relevant structural modes to be observed are located between 50 Hz and 100 Hz. Thus, to safely meet the Shannon-Nyquist lower limit, a sampling rate of 400 Hz was chosen. A synchronization accuracy of a few microseconds between the HaLOEWEn motes, as it can be achieved by the approach described in Section 6.2, is sufficient for this sampling rate. The WSN-based DAQ system does not capture the actual excitation of the structure, as this would also not be possible for real-world SHM deployments. Thus, an OMA is required as described in Section 7.3. A $n_f = 64$ tap high pass filter with a cut-off frequency of 20 Hz is applied to each sensor channel. This configuration was chosen as a trade-off between computational complexity and the filter quality. Static acceleration is damped by 60 dB, while all frequencies above 40 Hz are damped by less than 4×10^{-3} dB. After the high-pass filtering, the RDT with up to three reference channels and runtime configurable trigger levels is applied. A fixed window length of $n_w = 256$ is used to capture the free decay of the structure within the first 640 ms after each trigger event.

A second wire-bound DAQ system was installed in parallel to the WSN for reference measurements, as shown in Figure 71a. It consists of 12 PCB [356A16] integrated circuit piezoelectric (ICP) accelerometers controlled by the LMS Test.Lab 14A (LMS) and attached to the joints of the bridge by an adhesive (wax), as shown in Figure 71b. Due to the limited number of input channels available at the [SCADAS] sensor front-end, only the front side of the bridge (i.e., S_1 to S_{10}) can be completely observed by the reference system. In principle, both sides of the bridge could be analyzed



(a) Cabling



(b) ICP Sensor

(c) Impulse hammer

Figure 71: Wire-bound reference DAQ systems (LMS Test.Lab 14A with 12 ICP accelerometers)

independently, but moving the sensors from one side to another is not practical for long term observations as required by SHM. Instead, only two additional ICP sensors are installed on the rear side (i.e., S_{13} and S_{18}) to assess the symmetry of the observed mode shapes. All sensors are sampled at 512 Hz with a resolution of 0.1 mg. The LMS system also captures the excitation provided by an impulse hammer (see Figure 71c), so an EMA can be performed thus providing more accurate results than the OMA-based WSN. Compared to the wireless data acquisition system, the cabling required for the LMS system becomes rather complex (Figure 71a) even though only 60 % of the structure is covered.

8.2 Accuracy of the Wireless Data Acquisition System

Choosing S_3 and S_{13} as reference channels provided the best results, as the center of the structure is excited the most. The trigger level of $l_3 = l_{13} = 200\,\mathrm{mg}$ was determined experimentally and corresponds to the peak excitation caused by the model train.

The manual excitation of the structure with the impact hammer for 10 s resulted in 60 trigger events registered at node 3 and 40 trigger events registered at node 13. The 40 resulting RD sequences $D_{1,3}, \ldots, D_{20,3}, D_{1,13}, \ldots, D_{20,13}$ were transmitted to a base station for the subsequent modal analysis. As shown in Figure 72 for $D_{3,13}$, these RD sequences characterize the free decay of the structure.

For the wired LMS measurement, five strokes on S_{17} were injected in vertical direction with the impulse hammer at intervals of about 3 s to excite the vertical bending modes of the bridge model. The EMA results are averaged over those five individual measurements. The resulting frequency response function at S_3 is shown in Figure 73. Below 40 Hz and above 110 Hz, the structure's characteristics cannot be captured properly by the WSN-based system. However, the two dominating modes are located outside of these inaccurate frequency bands and can be captured with an accuracy of at least 1 % as summarized in Table 16.

In addition to the eigenfrequencies, the actual mode shapes are of special interest for an SHM system, as minor damage will be reflected in the deformation of the mode

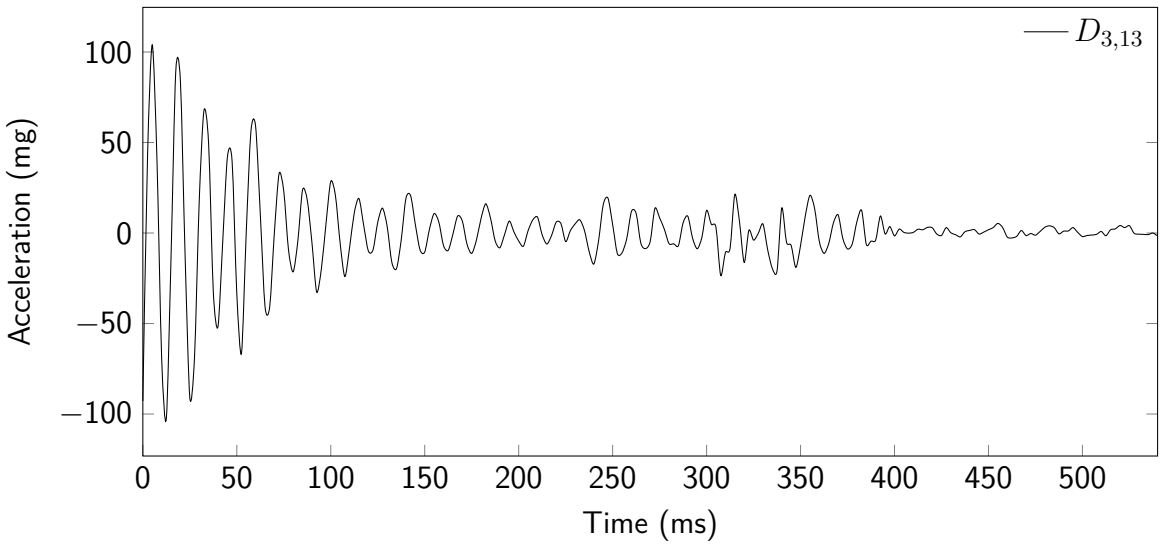


Figure 72: RD sequence $D_{3,13}$ captured by the WSN

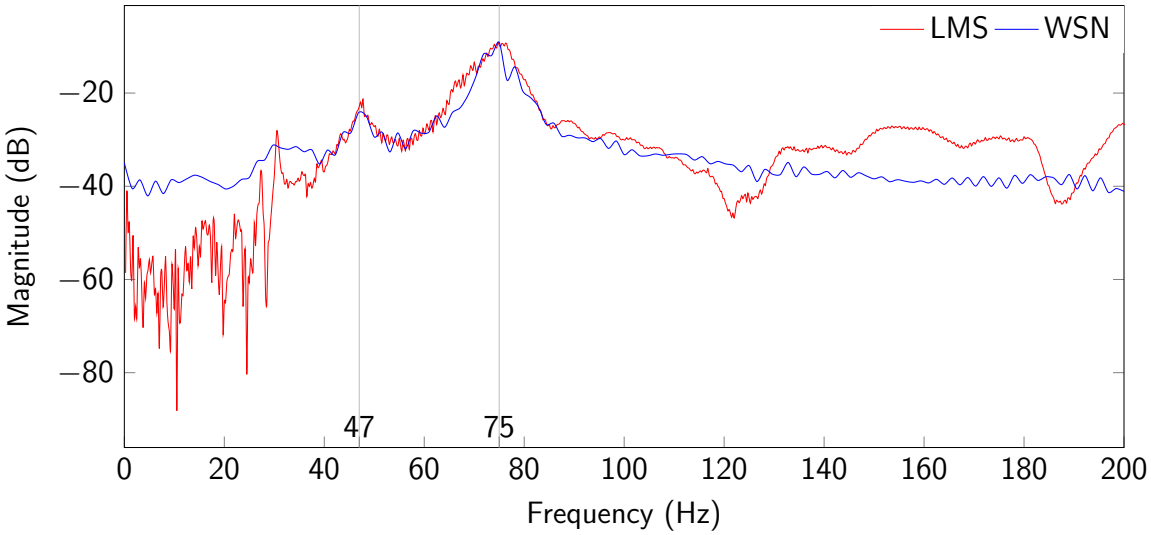
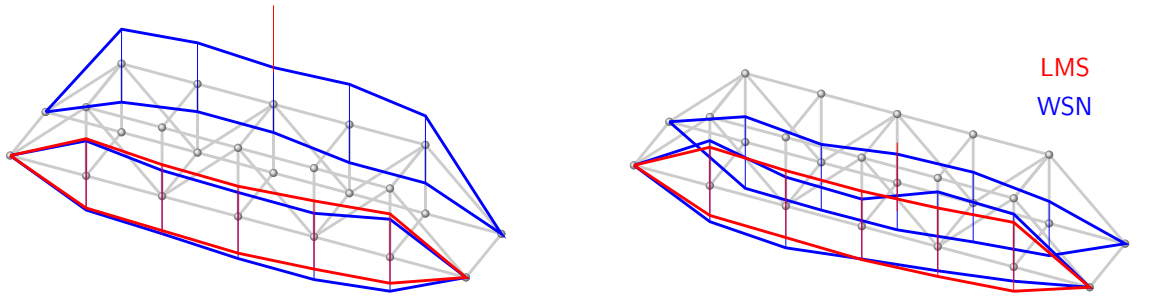


Figure 73: Frequency Response Function at S_3 captured with both DAQ systems

Mode shape	by LMS	by WSN	Relative deviation
Symmetric vertical bending	47.0 Hz	46.9 Hz	0.2 %
Asymmetric vertical bending	75.7 Hz	75.0 Hz	0.9 %

Table 16: Detected dominant mode shapes

shapes before significant changes in the eigenfrequencies can be detected. Figure 74a shows the asymmetric vertical bending mode captured by both monitoring systems. More specifically, the unexcited bridge is shown in gray, while the vertical movement of the horizontal bars is shown in red (for the wire-bound LMS DAQ system) and blue (for the WSN DAQ system). As only the position of the four outer bearing joints is fixed, the four lateral bars are also bending inwards. Remember that the wire-bound LMS system has a limited view on the rear side of the structure due to input channel restrictions. However, the two nodes on the rear side are sufficient to detect the asymmetric character of the mode, i.e., the rear side is bending up while the front side is bending down. But only the WSN system provides a detailed view on both



(a) Asymmetric vertical bending mode at 75 Hz

(b) Symmetric vertical bending mode at 45 Hz

Figure 74: Mode shapes captured with both DAQ systems

sides of the structure, which is essential for the subsequent SHM analysis. This also holds true for the symmetric vertical bending mode shown in Figure 74b. Although the reduced accuracy of the WSN system is clearly visible in the mode shapes, the principal behavior of the structure can still be observed without the need for extensive cabling and well-known controlled excitation.

After showing that the WSN system can identify structural properties such as eigenfrequencies and mode shapes, these properties are used to evaluate the damage detection capabilities of the HaLOEWEn network. For the selected configuration of the reference signals, each bridge crossing of the train generates 12 to 20 trigger events at each reference signal. The reference mode shape of the bridge was derived from a 600 s measurement, which generated 1103 triggers events at S_3 and 873 trigger events at S_{13}.

To demonstrate the reproducibility of the mode shape determined for the undamaged structure, measurements of reduced duration have been carried out. The resulting mode shapes of the first symmetric vertical bending mode and the corresponding damage indexes at the sensors along the horizontal axis of the structure (i.e., S_1 to S_5) are calculated by Equation 82 and shown in Figure 75. The reference mode shape is shown in red, while the actually determined mode shape is shown in blue. The number of accumulations performed for each RD sequence decreases with the measurement duration and the remaining noise in the RD sequences results into larger DI values. The possibility of false positive damage detection thus increases. For the experimentally determined damage detection threshold of 0.05, a 60 s measurement interval provides sufficiently low DI values.

To simulate a damaged structure, single screwed connections near specific sensors were loosened. Figure 76 shows the mode shapes and the damage indexes determined from the damaged structure for 3 different damage locations. In all three cases, the damage has been detected, i.e., the damage detection threshold of 0.05, determined during the reference measurements, is exceeded. The damage location, however, is not reflected by the DI diagram.

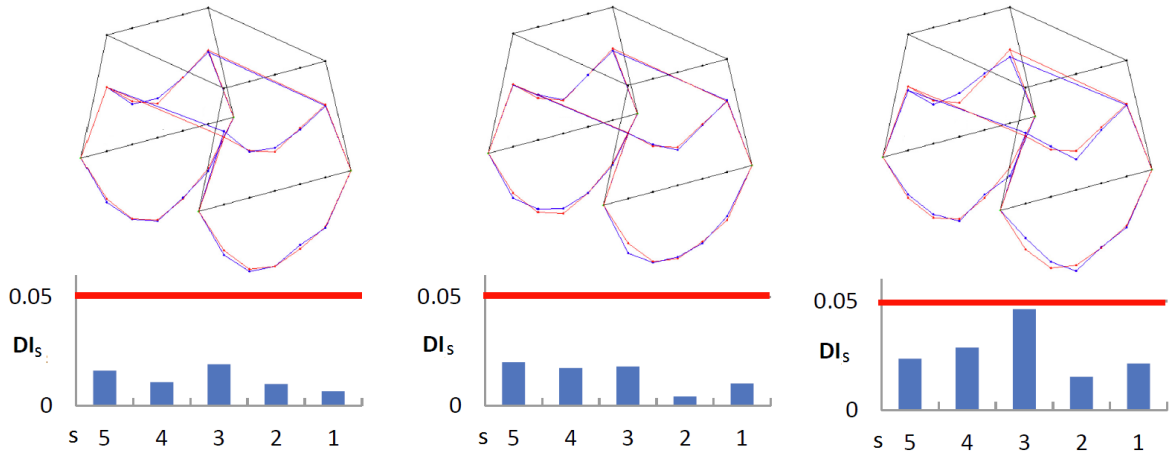


Figure 75: Mode-shapes and DI_s of undamaged structure obtained from 300 s (left), 60 s (center), and 30 s (right) measurements

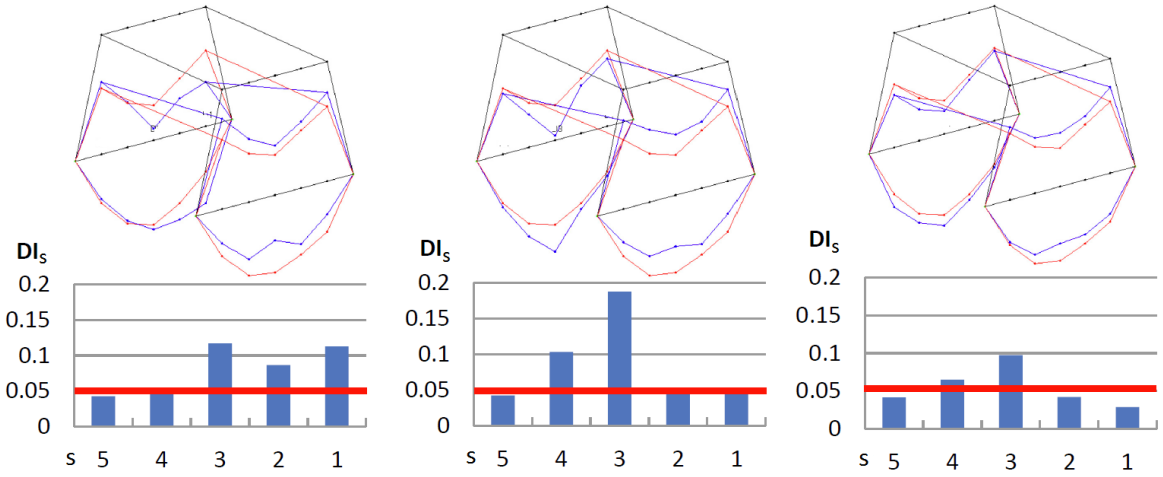


Figure 76: Mode-shapes and DI_s of damaged structure obtained from a 300 s measurement after loosening a screw at S_1 (left), S_3 (center) or S_5 (right) by a 5/6 rotation

8.3 Performance and Energy Evaluation

Finally, the runtime and energy required by the hardware accelerator of the HaLoMote architecture for the RDT computation is compared against software processors typically used in mobile and WSN applications. Namely, the TI [CC2530] and the Atmel [ATmega256RFR2] were chosen as representative 8 bit MCUs, as these RF-System-on-Chips (SoCs) have also been integrated into the HaLoMote. Furthermore, the TI [CC430] RF-SoCs was chosen as representative 16 bit MCU, as the MSP430 architecture is widely used in many WSN motes. Most recent software processors for mobile applications are based on the ARM Cortex-M architecture, so the [STM32F407] and the TI [CC2650] were chosen as particularly powerful and energy-efficient state-of-the-art references.

For a fair comparison, all systems were configured with the same RDT settings as described in Section 8.2, i.e., 400 Hz sampling frequency, four sensor channels, two reference channels, 64-tap FIR filter and 256 samples per RD sequence. As the actual workload heavily depends on the number of detected trigger events, all systems were fed with the same pre-recorded sensor samples stored in the processors code memories and the trigger levels were chosen such that the actual number of triggered events is 60 and 40, respectively, as observed in Section 8.2. The firmware for all systems was built with the most recent compilers, configured to optimize for execution speed. The average runtime per sampling cycle measured with peripheral timers was combined with data-sheet information about the power consumption and the wake-up time from sleep mode as shown in Table 17. The deepest sleep mode with memory retention and enabled real-time clock was chosen for each system respectively. To derive the overall energy spent per sampling cycle ($E_{overall}$), a power-consumption of P_{active} was assumed during wake-up, as the capacitors of the internal switching regulators have to be charged during the ramp-up. To better illustrate the consumed energy per sample by the different processor architectures, the corresponding system endurance (t_{alive}) achievable when supplied by a 1 W h energy buffer (e.g., a typical NiMH AAA cell) was derived. Note that this estimation takes only the processor into account,

Processing Unit	TI CC2530	ATmega256RFR2	TI CC430	STM32F407	TI CC2650	AGL1000
Architecture	8 bit	8 bit	16 bit	32 bit	32 bit	
	8051	AVR	MSP430	Cortex-M4F	Cortex-M3	IGLOO FPGA
Compiler	SDCC 3.4	AVR GCC 4.3.1	CL430 4.4.3	ARMGCC 4.9.3	ARMGCC 4.9.3	Synplify 2014
Main Clock [MHz]	32	16	20	128	48	8
V_{CC} [V]	2.0	1.8	2.4	1.8	1.8	1.2
I_{active} [mA]	6.5	3.7	4.6	40	2.9	25
P_{active} [mW]	13	6.7	11	72	5.2	30
sleep mode	LPM2	POWER-SAVE	LPM3	STOP	STANDBY	Flash*Freeze
I_{idle} [µA]	1	1.5	5.3	280	1	44
P_{idle} [µW]	2	2.7	12.7	156	1.8	53
t_{active} [cycles/sample]	257,795	43,387	24,985	5,463	7,150	68
t_{active} [µs/sample]	8,056	2,712	1,249	43	149	9
t_{wakeup} [µs/sample]	100	34	150	110	151	1
t_{idle} [µs/sample]	-	-	1,101	2,347	2,200	2,490
E_{active} [nJ/sample]	104,728	18,170	13,739	3,096	775	270
E_{wakeup} [nJ/sample]	1,300	228	1,650	7,920	785	30
E_{idle} [nJ/sample]	-	-	14	366	4	132
$E_{overall}$ [nJ/sample]	-	-	15,403	11,382	1,564	432
t_{alive} [d]	-	-	7	9	67	241

Table 17: Resources required for executing the RDT on various processing units

disregarding the sensors and the radio transceiver (which are assumed to be identical across the platforms examined).

As shown in Table 17, the 8 bit MCUs do not achieve the required sampling period of 2500 µs. The [CC430] requires about 50 % of the sampling period for the RDT computations while consuming 15.4 µJ per sampling cycle. The powerful Cortex-M4 device is nearly 30 times faster than the [CC430], but its comparatively large power draw in idle mode still results in 11.4 µJ consumed per sampling cycle. The TI [CC2650] proves to be the most energy-efficient software-processor under consideration as it requires only 1.6 µJ per sampling cycle. However, the FPGA on the HaLOEWEn4 requires only 28 % of the energy of the most efficient software processor. Note that the HaLOEWEn4 MCU causes an additional overhead, mainly due to its energy required during wake-up (228 nJ) and idle (6 nJ) periods. The combination of the hardware accelerator and the Atmel MCU, as used by the HaLOEWEn4 implementation, thus consumes only 44 % of the energy of the most efficient software processor.

The energy efficiency of the HaLoMote easily exceeds that of the other platforms when actively performing computations, but it suffers when the node has to remain powered, but stays idle. In that case, its idle power consumption is 53 µW, which is nearly 30× the power drawn by the CC2650 MCU in idle mode. As sensor motes spend most of their lifetime in idle mode, special care must be taken on the HaLoMote to address this issue. In the SHM application, this can be achieved by having a supervisory power manager suspend the sensor sampling when no traffic is present on the structure. For the specific use case of railway bridges, these times may be more than 95 % of the overall operating time. For these quiet periods, the non-volatile FRAM (see Section 4.4.2) can be employed to store the internal state of the hardware kernels, which has to be preserved once the supervisory manager completely powers down the hardware accelerator. Note that the FPGA configuration data is not affected by such a shutdown, as it is held on-chip in non-volatile Flash memory.

For the concrete SHM configuration discussed in this section, about 48 kbit of runtime state has to be preserved across shutdowns (i.e., 41 kbit for the RD sequences, 3 kbit for the FIR taps, 3 kbit for the delay FIFO, and 1 kbit for the trigger events). According to Equation 2 and 4, live variable swapping to the external SRAM pays off after approximately 2 s shutdown, while swapping to FRAM becomes attractive only after about 16 s. Beyond the railway bridge use case, such short idle-times occur even in many less frequently traveled automotive bridges. Thus, the capability of quickly and power-efficiently preserving the system state, while completely shutting down the accelerator, is attractive for a variety of applications.

8.4 Discussion of Results

In this section, the obtained results for the HaLOEWEn-based SHM system are compared with the related work described in Section 3.3.2.

The achieved measurement accuracy, i.e., the relative error of the detected eigenfrequencies compared to ground truth measurements of the wire-bound system ($\leq 0.9\,\%$) is comparable with the outcome of [Bocca2011] ($\leq 1.035\,\%$). All other wireless SHM systems missed the exact eigenfrequencies by up to 18 %. One possible cause of this observation might be that [Bocca2011] is the only of the considered reference projects also relying on a laboratory scaled testbed with well-controlled boundary conditions, such as the artificial excitation of the structure. However, while [Bocca2011] used a white-

noise generating shaker, the model train set used in this thesis might be considered more realistic.

Another comparison worth looking at is the achievable data-aggregation factor. [Kim2007] reported a 20× down-sampling, but this can not be understood as data aggregation, as the authors just increased the sampling rate to obtain a better signal quality (i.e., reduced noise floor). [Battista2013] reported a 96 % data aggregation (i.e., 25×) for 10 min measurements every 30 min. As stated in Section 7.3.2, the distributed RDT used by the HaLOEWEn system reduces the raw data stream by 469× for the same measurement duration.

Finally, the overall energy efficiency of the data acquisition systems must be compared against each other. It is hard to directly compare the energy or the power consumed by different systems against one another, without taking the quality and the information content of the gathered data into account (e.g., higher sampling rates result in higher power draw, but also in more accurate results). However, all systems considered in Section 3.3.2 were used to identify the modal properties of the observed structures. The smallest average power draw reported by the reference systems was 53 mW for a 60 s active measurement period [Bocca2011]. As analyzed in the previous section, the computational units of the HaLOEWEn4 consume 666 nJ per sample. At 400 Hz sampling rate, this equates to an average power draw of 266 µW just for processing. After the measurement period, $4 \cdot |R|$ =8 RD sequences per sensor node have to be transmitted to the base station, each consisting of $n_w \cdot 4 = 1024$ Byte. Even when assuming that only 10 % of the available IEEE [802.15.4] data rate is available for actual payload transfer (e.g., due to packet loss and addressing overhead), and the maximum transmission power of the HaLOEWEn4 RF-SoC is required for a specific transmission range (i.e., 36 mW power consumption during transmission), the overall average power of the HaLOEWEn4 is increased by

$$\frac{8 \cdot 1024 \cdot 8\,\mathrm{bit}}{250\,\mathrm{kbit/s} \cdot 10\,\%} \cdot \frac{36\,\mathrm{mW}}{60\,\mathrm{sec}} = 1.57\,\mathrm{mW} \tag{83}$$

The resulting average power drawn by the HaLOEWEn4 during a 60 s measurement is thus smaller than 2 mW or 29× times less the power required by [Bocca2011]. Note that in practice, the measurement duration will be much longer, thus even increasing the HaLOEWEn energy efficiency.

Finally, while all reference systems considered in Section 3.3.2 focus on system identification, the HaLOEWEn system also provides basic damage detection capabilities.

CHAPTER 9
Summary and Future Work

In this thesis, the Hardware-Accelerated Low Power Mote (HaLoMote) was proposed as a heterogeneous Wireless Sensor Network (WSN) architecture for hardware-accelerated energy-efficient distributed data aggregation. The Hardware-Accelerated Low Energy Wireless Embedded Sensor Node (HaLOEWEn) implementation of this architecture was detailed in conjunction with the platforms inter-processor communication and power management strategies. In addition to application-independent hardware accelerators for digital filters, lossless data compression, and linear regression, network communication primitives for flooding and time synchronization have been developed. These building blocks were evaluated in the context for three monitoring applications. One of these applications, i.e., a distributed Structural Health Monitoring (SHM) was analyzed more in detail, as an application-specific feature-extraction algorithm, namely the Random Decrement Technique (RDT), was hardware accelerated. A laboratory-scale SHM testbed was conducted for the system-level evaluation of the HaLoMote architecture. It could be shown that a HaLOEWEn-based wireless Data Acquisition (DAQ) system can identify the the dynamic characteristics relevant for the SHM application with an acceptable deterioration of the measurement accuracy compared to a wire-bound laboratory measurement system. At the same time, the energy efficiency of the HaLoMote platform was shown to outperform the energy efficiency of recent low-power microcontrollers by more than 2×.

9.1 Lessons Learned

Most of the research problems addressed in this thesis (see Section 1.2) can be answered clearly.

Which specific RCU is suited best to be integrated into a WSN mote?
FPGAs with a truly non-volatile configuration storage provide a low static power consumption with fast transitions between active and idle states. Even nowadays, the Microsemi [IGLOO] family provides the best trade-off between available logic resources and static power consumption.

How to integrate the RCU into the architecture of a WSN mote? To be more specific, how should the sensors, memories, processing and communication modules be arranged, and how should they communicate?
Using the FPGA as hardware accelerator next to a small RF-SoC is required, as a standalone FPGA cannot properly apply DPM-mechanisms during periods with reduced compute demands. However, two fundamental approaches for interconnecting the devices with each other and with the sensors were identified. The HaLoMote architecture connects all sensors to the FPGA and provides only a narrow communication link between the FPGA and the RF-SoC. The FPGA thus has to be woken up in every sampling cycle to query the next sensor samples. Although the DPM mechanism of the HaLoMote were designed to support

this fast state transitions, an alternative approach can be imagined. When connecting all sensors to the RF-SoC and interconnecting FPGA and RF-SoC with a shared memory, the hardware accelerator can be kept low until a sufficient amount of data is captured for processing. As this second approach was not evaluated throughout this thesis, this research question can not be clearly answered.

How does the RCU affect the DPM strategy of the WSN mote?

The clock supply and the external shutdown request required by the FPGA were identified as a major difficulty for the DPM of the entire platform. To decouple the sleep scheduling or FPGA and RF-SoC, two mechanisms were proposed, namely a hybrid clock supply and the self-controlled shutdown of the FPGA. To even save the power drawn by the FPGA in Flash*Freeze mode, the feasibility of live variable swapping to an external SRAM of FRAM was analyzed. For typical monitoring applications, this mechanism is, however, only useful as long as no active measurement is carried out.

How does the RCU affect the trade-off between in-sensor preprocessing and wireless data transmission? For which kinds of application do the improved data aggregation capabilities pay off in terms of overall energy consumption?

Due to the energy efficiency of the hardware accelerator, data aggregation pays off even if only a few percentage of the wireless communication bandwidth can be saved. The more relevant question thus is, which computational load justifies the usage of a hardware accelerator instead of a low-power MCU. For the SHM target application sampling four channels in parallel at 400 Hz, a recent ARM-based MCU was outperformed by the HaLOEWEn, however.

Once the RCU is integrated as application-specific accelerator, which generic WSN services can also benefit from the improved processing capabilities?

In this thesis, the hardware acceleration of a generic lossless data compression module and a linear regression solver used for improved time synchronization was proposed. Although not analyzed in this thesis, many other services like encryption or forward error correction are known to also benefit from hardware acceleration.

9.2 Remaining Research and Engineering Challenges

Although many algorithms and methods have been analyzed, implemented and tested for the HaLoMote platform, a lot of additional and complementary research and engineering challenges can be identified.

First of all, the following improvements can be achieved without major modifications of the HaLoMote design: Over-the-Air-Programming of the FPGA is useful to simplify in-field reconfiguration or the entire mote. For larger deployments and testbeds, this feature is almost mandatory. The HaLOEWEn4 MCU already has access to the JTAG port of the FPGA and Microsemi provides platform-independent firmware for the MCU-based JTAG-programming of their devices (namely DirectC and the Stable-Player). However, some additional effort should be spent on the bitstream transport throughout the network to efficiently support incremental updates or the delivery of only slightly differing bitstreams to different, but nearby network nodes.

Furthermore, the support for higher-order predictive compression can be implemented by just integrating higher-order linear equation solvers to the ADPCM kernel described in Section 5.2.2. The multi-channel conditional monitoring application (Section 7.2) was shown to benefits from a fourth order predictor in terms of compression ratio, but the energy required for this more complex encoder could not be analyzed in this thesis.

Two improvements for the multi-source flooding mechanism can be envisioned. Instead of assuming the knowledge of the entire topology (i.e., the connectivity and interference graph), some sophisticated protocols should be developed for an automated topology discovery. While the connectivity graph can be easily generated by well known neighbor-discovery algorithms [Dheap2003], capturing the interference graph is rather difficult. In addition, the flooding scheduler could exploit channel switching for congestion resolution and integrate energy-balancing to equally distribute the transceiver activities to all network notes, instead to building hot spots.

Only minor changes to the HaLOEWEn design, are required to support the live variable swapping described in Section 4.4.2. So far the core voltage supply of the FPGA can not be turned of completely, so an additional power FET controlled by the MCU is required.

If a major redesign of the HaLOEWEn is envisioned, than a new Flash-based FPGA device family is hopefully available. The [IGLOO] family is build with a quite outdated 130 nm CMOS process. The advantage of the hardware accelerator against the most recent MCUs has been reduced down to a factor of two. A break-even between the old FPGA device and new software processors can thus be expected in the near future.

Furthermore, the alternative hardware accelerator architecture based on shared memory between RCU and RF-SoC (Figure 13b) must be evaluated. It can be expected that this architecture is better suited for the higher sampling rates required by most condition monitoring applications, as the FPGA does not have to be power-cycled in every sampling cycle with this approach.

Most important, however, the HaLOEWEn network must be deployed in a real world scenario (e.g., bridge monitoring) to evaluate its long term robustness and energy efficiency under realistic conditions such as increased packet error rates caused by foggy weather.

Bibliography

Scientific Publications

Agathos2011 S. Agathos and E. Papapetrou. "Efficient broadcasting using packet history in mobile ad hoc networks". In: *IET Communications* 5.15 (2011), pp. 2196–2205.

Ahmed2009 I. Ahmed et al. "Implementation of Graph Algorithms in Reconfigurable Hardware (FPGAs) to Speeding Up the Execution". In: *Computer Sciences and Convergence Information Technology, 2009. ICCIT '09. Fourth International Conference on*. 2009, pp. 880–885.

Ahmed2010 N. Ahmed, S.S. Kanhere, and Sanjay Jha. "Mitigating the effect of interference in Wireless Sensor Networks". In: *35th IEEE Conference on Local Computer Networks (LCN)*. 2010, pp. 160–167.

Aiken1946 Howard H. Aiken and Grace M. Hopper. "The automatic sequence controlled calculator - I". In: *Electrical Engineering* 65.8-9 (1946), pp. 384–391.

Akbari2014 S. Akbari. "Energy harvesting for wireless sensor networks review". In: *Federated Conference on Computer Science and Information Systems (FedCSIS)*. 2014, pp. 987–992.

Akhlaq2013 M. Akhlaq and T.R. Sheltami. "RTSP: An Accurate and Energy-Efficient Protocol for Clock Synchronization in WSNs". In: *IEEE Transactions on Instrumentation and Measurement* 62.3 (2013). 5, pp. 578–589.

Alnuaimi2013 M. Alnuaimi et al. "Performance analysis of clustering protocols in WSN". In: *6th Joint IFIP Wireless and Mobile Networking Conference (WMNC)*. 2013, pp. 1–6.

Aoun2008 Marc Aoun, Anthony Schoofs, and Peter van der Stok. "Efficient Time Synchronization for Wireless Sensor Networks in an Industrial Setting". In: *6th ACM Conference on Embedded Network Sensor Systems*. SenSys '08. Raleigh, NC, USA: ACM, 2008, pp. 419–420.

Arango2004 J. Arango, M. Degermark, and A. Efrat. "An efficient flooding algorithm for ad hoc networks". In: *2nd Workshop on Modeling and Optimizations in Mobile Ad Hoc and Wireless Networks (WiOpt)*. 2004.

Araujo2012 A. Araujo et al. "Wireless Measurement System for Structural Health Monitoring With High Time-Synchronization Accuracy". In: *IEEE Transactions on Instrumentation and Measurement* 61.3 (2012), pp. 801–810.

Asada1998 G. Asada et al. "Wireless integrated network sensors: Low power systems on a chip". In: *Solid-State Circuits Conference, 1998. ESSCIRC '98. Proceedings of the 24th European*. 1998, pp. 9–16.

Asmussen1997 J. C. Asmussen. *Modal Analysis Based on the Random Decrement Technique*. Aalborg University. Department of Mechanical Engineering, 1997.

Astapov2014 S. Astapov, J. Ehala, and J.-S. Preden. "Collective acoustic localization in a network of dual channel low power devices". In: *21st International Conference on Mixed Design of Integrated Circuits Systems (MIXDES)*. 2014, pp. 430–435.

Battista2013 Nicholas de Battista et al. “Wireless structural monitoring of a multi-span footbridge with decentralised embedded data processing”. In: *6th International Conference on Structural Health Monitoring of Intelligent Infrastructure*. 2013.

Berder2010 Olivier Berder and Olivier Sentieys. “PowWow : Power Optimized Hardware/Software Framework for Wireless Motes”. In: *Architecture of Computing Systems (ARCS), 2010 23rd International Conference on*. 2010, pp. 1–5.

Bocca2011 Maurizio Bocca et al. “A Synchronized Wireless Sensor Network for Experimental Modal Analysis in Structural Health Monitoring”. In: *Computer-Aided Civil and Infrastructure Engineering*. Computer-Aided Civil and Infrastructure Engineering (CACAIE) 26.7 (2011), pp. 483–499.

Boonyakitmaitree2004 C. Boonyakitmaitree, K. Nandhasri, and J. Ngarmnil. “A low computational predictor coefficient algorithm for ADPCM implementation of portable recording devices”. In: *47th Midwest Symposium on Circuits and Systems (MWSCAS)*. Vol. 3. 2004.

Brincker2000 Rune Brincker, Lingmi Zhang, and P Andersen. “Modal identification from ambient responses using frequency domain decomposition”. In: *Proceedings of the International Modal Analysis Conference (IMAC 18)*. Vol. 18. 2000, pp. 625–630.

Carvalho2011 C.G.N. de Carvalho et al. “Multiple linear regression to improve prediction accuracy in WSN data reduction”. In: *7th Latin American Network Operations and Management Symposium (LANOMS)*. 2011, pp. 1–8.

Castillo-Secilla2013 J.M. Castillo-Secilla, J.M. Palomares, and J. Olivares. “Temperature-aware methodology for time synchronisation protocols in wireless sensor networks”. In: *Electronics Letters* 49.7 (2013), pp. 506–508.

Chae2012 M.J. Chae et al. “Development of a wireless sensor network system for suspension bridge health monitoring”. In: *Automation in Construction* 21 (2012), pp. 237–252.

Cho2010 Soojin Cho et al. “Structural health monitoring system of a cable-stayed bridge using a dense array of scalable smart sensor network”. In: *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace*. Vol. 7647. SPIE, 2010.

Chong2003 Chee-Yee Chong and S.P. Kumar. “Sensor networks: evolution, opportunities, and challenges”. In: *Proceedings of the IEEE* 91.8 (2003), pp. 1247–1256.

CMUPDCS1978 Carnegie-Mellon University Pittsburgh Department of Computer Science. *Proceedings of a Workshop on Distributed Sensor Nets*. Pittsburgh, Pennsylvania: Defense Technical Information Center, 1978.

Cole1973 Henry A Cole. *On-line failure detection and damping measurement of aerospace structures by random decrement signatures*. Nielsen Engineering and Research Inc., 1973.

Deng2012 Xi Deng and Yuanyuan Yang. “Online Adaptive Compression in Delay Sensitive Wireless Sensor Networks”. In: *IEEE Transactions on Computers* 61.10 (2012), pp. 1429–1442.

DePauw2010 T. De Pauw et al. “Resource-Aware Scheduling of Distributed Ontological Reasoning Tasks in Wireless Sensor Networks”. In: *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC)*. 2010, pp. 131–137.

Dheap2003 Vijay Dheap et al. “Parameterized neighborhood based flooding for ad hoc wireless networks”. In: *IEEE Military Communications Conference (MILCOM)*. Vol. 2. 2003, 1048–1053 Vol.2.

Djenouri2012 D. Djenouri. “R^4Syn : Relative Referenceless Receiver/Receiver Time Synchronization in Wireless Sensor Networks”. In: *IEEE Signal Processing Letters* 19.4 (2012), pp. 175–178.

Djenouri2014 D. Djenouri and M. Bagaa. “Synchronization Protocols and Implementation Issues in Wireless Sensor Networks: A Review”. In: *IEEE Systems Journal* PP.99 (2014), pp. 1–11.

Dondi2010 D. Dondi et al. “Shimmer: A wireless harvesting embedded system for active ultrasonic Structural Health Monitoring”. In: *Sensors, 2010 IEEE*. 2010, pp. 2325–2328.

Donoho2006 D.L. Donoho. “Compressed sensing”. In: *IEEE Transactions on Information Theory* 52.4 (2006), pp. 1289–1306.

Engel2011 Andreas Engel, Björn Liebig, and Andreas Koch. “Feasibility Analysis of Reconfigurable Computing in Low-Power Wireless Sensor Applications”. In: *Reconfigurable Computing: Architectures, Tools and Applications*. Ed. by Andreas Koch et al. Vol. 6578. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, pp. 261–268.

Engel2012a Andreas Engel, Björn Liebig, and Andreas Koch. “Energy-efficient heterogeneous reconfigurable sensor node for distributed structural health monitoring”. In: *Conference on Design and Architectures for Signal and Image Processing (DASIP)*. 2012, pp. 1–8.

Engel2012b Andreas Engel, Björn Liebig, and Andreas Koch. “HaLOEWEn: A heterogeneous reconfigurable sensor node for distributed structural health monitoring”. In: *Conference on Design and Architectures for Signal and Image Processing (DASIP)*. Ed. by Dr. Adam Morawiec and Jinnie Hinderscheit. Electronic Chips & Systems design Initiative, 2012.

Engel2014a Andreas Engel and Andreas Koch. “Hardware-Accelerated Data Compression in Low-Power Wireless Sensor Networks”. In: *Reconfigurable Computing: Architectures, Tools, and Applications*. Ed. by Diana Goehringer et al. Vol. 8405. Lecture Notes in Computer Science. 10th International Symposium (ARC). Vilamoura, Portugal: Springer International Publishing, 2014, pp. 167–178.

Engel2014b Andreas Engel and Andreas Koch. “An Energy-Efficient Wireless Routing Protocol for Distributed Structural Health Monitoring”. In: *7th IFIP Wireless and Mobile Networking Conference*. 2014.

Engel2014c Andreas Engel et al. “Hardware-accelerated Wireless Sensor Network for Distributed Structural Health Monitoring”. In: *Procedia Technology* 15 (2014). 2nd International Conference on System-Integrated Intelligence: Challenges for Product and Production Engineering, pp. 738–747.

Engel2014d Andreas Engel et al. “Hardware-Accelerated Embedded Controller for a Piezo-electric Haptic Feedback System”. In: *14th International Conference on New Actuators and Drive Systems*. 2014.

Engel2015a Andreas Engel and Andreas Koch. “Accelerated Clock Drift Estimation for High-Precision Wireless Time-Synchronization”. In: *40th IEEE Conference on Local Computer Networks (LCN)*. Tenth IEEE Workshop on Practical Issues in Building Sensor Network Applications (SenseApp). Clearwater Beach, USA, 2015.

Engel2015b Andreas Engel and Andreas Koch. “DEMO: The Need for Wireless Clock Drift Estimation and Its Acceleration on a Heterogeneous Sensor Node”. In: *40th IEEE Conference on Local Computer Networks (LCN)*. Clearwater Beach, USA, 2015.

Engel2015c Andreas Engel, Thomas Siebel, and Andreas Koch. “A Heterogeneous System Architecture for Low-Power Wireless Sensor Nodes in Compute-intensive Distributed Applications”. In: *40th IEEE Conference on Local Computer Networks (LCN)*. Tenth IEEE Workshop on Practical Issues in Building Sensor Network Applications (SenseApp). Clearwater Beach, USA, 2015.

Estrin1960 Gerald Estrin. “Organization of Computer Systems: The Fixed Plus Variable Structure Computer”. In: *Western Joint IRE-AIEE-ACM Computer Conference*. IRE-AIEE-ACM ’60 (Western). San Francisco, California: ACM, 1960, pp. 33–40.

Floyd1962 Robert W. Floyd. “Algorithm 97: Shortest Path”. In: *Communications of the ACM* 5.6 (1962), pp. 345–.

Gheorghe2010 L. Gheorghe, R. Rughinis, and N. Tapus. “Fault-Tolerant Flooding Time Synchronization Protocol for Wireless Sensor Networks”. In: *6th International Networking and Services Conference (ICNS)*. 2010, pp. 143–149.

Goh2012 K.M. Goh et al. “FPGA based wireless sensor node for distributed process monitoring”. In: *7th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. 2012, pp. 1934–1939.

Grassi2012 P.R. Grassi and D. Sciuto. “Energy-Aware FPGA-based Architecture for Wireless Sensor Networks”. In: *15th Euromicro Conference on Digital System Design (DSD)*. 2012, pp. 866–873.

Hameed2010 Rehan Hameed et al. “Understanding Sources of Inefficiency in General-purpose Chips”. In: *ACM SIGARCH Computer Architecture News* 38.3 (2010), pp. 37–47.

Hartenstein2007 Reiner Hartenstein. “The Future of Computing, essays in memory of Stamatis Vassiliadis”. In: ed. by K.L.M. Bertels et al. TU Delft, 2007. Chap. The von Neumann Syndrome.

Hill2002 J.L. Hill and D.E. Culler. “Mica: a wireless platform for deeply embedded networks”. In: *Micro, IEEE* 22.6 (2002), pp. 12–24.

Hoa2012 Tran Dang Hoa and Dong-Sung Kim. “Minimum latency and energy efficiency routing with lossy link awareness in wireless sensor networks”. In: *9th IEEE International Workshop on Factory Communication Systems (WFCS)*. 2012, pp. 75–78.

Hochberger2014 Christian Hochberger et al. “Synthilation: JIT-Compilation Targeting Microinstructions”. In: *Conference on Design and Architectures for Signal and Image Processing (DASIP)*. 2014, pp. 1–6.

Hu2013 Xiaoya Hu, Bingwen Wang, and Han Ji. “A Wireless Sensor Network-Based Structural Health Monitoring System for Highway Bridges”. In: *Computer-Aided Civil and Infrastructure Engineering* 28.3 (2013), pp. 193–209.

Jain2015 P.C. Jain. “Recent trends in energy harvesting for green wireless sensor networks”. In: *International Conference on Signal Processing and Communication (ICSC)*. 2015, pp. 40–45.

Jeong2010 Hyo-Cheol Jeong, Ki-Deok Kwon, and Younghwan Yoo. “Cross-Layer Counter-Based Flooding without Location Information in Wireless Sensor Networks”. In: *7th International Conference on Information Technology: New Generations (ITNG)*. 2010, pp. 840–845.

Kasirajan2012 Priya Kasirajan, Carl Larsen, and S. Jagannathan. “A new data aggregation scheme via adaptive compression for wireless sensor networks”. In: *ACM Transactions on Sensor Networks* 9.1 (2012), 5:1–5:26.

Kesel2009 Frank Kesel and Ruben Bartholomä. *Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs*. Oldenbourg Wissenschaftsverlag, 2009.

Kim2007 Sukun Kim et al. “Health monitoring of civil infrastructures using wireless sensor networks”. In: *6th International Conference on Information Processing in Sensor Networks*. IPSN ’07. Cambridge, Massachusetts, USA: ACM, 2007, pp. 254–263.

Kimura2005 N. Kimura and S. Latifi. “A survey on data compression in wireless sensor networks”. In: *International Conference on Information Technology: Coding and Computing (ITCC)*. Vol. 2. 2005, pp. 8–13.

Kohvakka2006 Mikko Kohvakka et al. “High-performance multi-radio WSN platform”. In: *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*. REALMAN ’06. Florence, Italy: ACM, 2006, pp. 95–97.

Kumar2001 Srikanta Kumar and David Shepherd. “SensIT: Sensor Information Technology For the Warfighter”. In: *Proceedings of the 4th International Conference on Information Fusion*. 2001.

Kung1982 H.T. Kung. “Why systolic architectures?” In: *Computer* 15.1 (1982), pp. 37–46.

Lazarescu2013 M.T. Lazarescu. “Design of a WSN Platform for Long-Term Environmental Monitoring for IoT Applications”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 3.1 (2013), pp. 45–54.

Lim2001 H Lim and C Kim. “Flooding in wireless ad hoc networks”. In: *Computer Communications* 24.3-4 (2001), pp. 353–363.

Lombardo2012 M. Lombardo et al. “Power management techniques in an FPGA-based WSN node for high performance applications”. In: *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*. 2012, pp. 1–8.

Lou2002 Wei Lou and Jie Wu. "On reducing broadcast redundancy in ad hoc wireless networks". In: *IEEE Transactions on Mobile Computing* 1.2 (2002), pp. 111–122.

Lueders2014 M. Lueders et al. "Architectural and Circuit Design Techniques for Power Management of Ultra-Low-Power MCU Systems". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.11 (2014), pp. 2287–2296.

Lynch2006 J. P. Lynch and K. J. Loh. "A Summary Review of Wireless Sensors and Sensor Networks for Structural Health Monitoring". In: *Shock and Vibration Digest* 38 (2006), pp. 91–128.

Mainetti2011 L. Mainetti, L. Patrono, and A. Vilei. "Evolution of wireless sensor networks towards the Internet of Things: A survey". In: *19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. 2011, pp. 1–6.

Maroti2004 Miklós Maróti et al. "The flooding time synchronization protocol". In: *2nd International Conference on Embedded Networked Sensor Systems*. SenSys '04. Baltimore, MD, USA: ACM, 2004, pp. 39–49.

Martinez2013 J. Martinez et al. "An accurate and fast technique for correcting spectral leakage in motor diagnosis". In: *Diagnostics for Electric Machines, Power Electronics and Drives (SDEMPED), 2013 9th IEEE International Symposium on*. 2013, pp. 215–220.

Mosterman2015 PieterJ. Mosterman and Justyna Zander. "Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems". English. In: *Software & Systems Modeling* (2015), pp. 1–12.

Mplemenos2012 G.-G. Mplemenos and I. Papaefstathiou. "Fast and power-efficient hardware implementation of a routing scheme for WSNs". In: *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*. 2012, pp. 1710–1714.

Neely1966 Peter M. Neely. "Comparison of Several Algorithms for Computation of Means, Standard Deviations and Correlation Coefficients". In: *Communications of the ACM* 9.7 (1966), pp. 496–499.

Neumann1945 John von Neumann. *First Draft of a Report on the EDVAC*. Tech. rep. 1945.

Ngau2012 C.W.H. Ngau, L.-M. Ang, and K.P. Seng. "Low memory visual saliency architecture for data reduction in wireless sensor networks". In: *IET Wireless Sensor Systems* 2.2 (2012), pp. 115–127.

Ni1999 Sze-Yao Ni et al. "The broadcast storm problem in a mobile ad hoc network". In: *5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*. MobiCom '99. Seattle, Washington, United States: ACM, 1999, pp. 151–162.

Nylaenden2014 Teemu Nyländen et al. "Low-Power Reconfigurable Miniature Sensor Nodes for Condition Monitoring". English. In: *International Journal of Parallel Programming* (2014), pp. 1–21.

OConnor2010 Sean O'Connor et al. "Fatigue Life Monitoring Of Metallic Structures By Decentralized Rainflow Counting Embedded In A Wireless Sensor Network". In: *ASME Conference on Smart Materials, Adaptive Structures and Intelligent Systems (SMASIS)*. Philadelphia, Pennsylvania, USA, 2010.

Pakzad2008 Shamim N. Pakzad et al. “Design and Implementation of Scalable Wireless Sensor Network for Structural Monitoring”. In: *Journal of Infrastructure Systems* 14.1 (2008), pp. 89–101.

Parks2014 A.N. Parks and J.R. Smith. “Sifting through the airwaves: Efficient and scalable multiband RF harvesting”. In: *IEEE International Conference on RFID*. 2014, pp. 74–81.

Patterson2012 David A. Patterson and John L. Hennessy. *Computer Architecture: A Quantitative Approach*. 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012.

Perkins1999 C.E. Perkins and E.M. Royer. “Ad-hoc on-demand distance vector routing”. In: *2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*. 1999, pp. 90–100.

Philipp2011 F. Philipp, F.A. Samman, and M. Glesner. “Design of an autonomous platform for distributed sensing-actuating systems”. In: *Rapid System Prototyping (RSP), 2011 22nd IEEE International Symposium on*. 2011, pp. 85–90.

Philipp2012 F. Philipp et al. “A smart wireless sensor for the diagnosis of broken bars in induction motors”. In: *Electronics Conference (BEC), 2012 13th Biennial Baltic*. 2012, pp. 119–122.

Piedra2012 Antonio de la Piedra, An Braeken, and Abdellah Touhafi. “Sensor Systems Based on FPGAs and Their Applications: A Survey”. In: *Sensors* 12.9 (2012), pp. 12235–12264.

Polastre2005 J. Polastre, R. Szewczyk, and D. Culler. “Telos: enabling ultra-low power wireless research”. In: *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*. 2005, pp. 364–369.

Putnam2015 A. Putnam et al. “A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services”. In: *IEEE Micro* 35.3 (2015), pp. 10–22.

Rabaey2000 J.M. Rabaey et al. “PicoRadio supports ad hoc ultra-low power wireless networking”. In: *Computer* 33.7 (2000), pp. 42–48.

Rafflenbeul2012 Lutz Rafflenbeul, Roland Werthschützky, and Alexander Gail. “Integrated Wireless Neural Recording and Electrode Positioning System”. In: *Procedia Engineering* 47 (2012). 26th European Conference on Solid-State Transducers, {EUROSENSOR} 2012, pp. 1097–1100.

Rappaport2001 Theodore Rappaport. *Wireless Communications: Principles and Practice*. 2nd. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

Raval2010 R. K. Raval, C. H. Fernandez, and C. J. Bleakley. “Low-power TinyOS tuned processor platform for wireless sensor network motes”. In: *ACM Trans. Des. Autom. Electron. Syst.* 15 (3 2010), 23:1–23:17.

Ravinagarajan2010 A. Ravinagarajan, D. Dondi, and T Simunic Rosing. “DVFS based task scheduling in a harvesting WSN for structural health monitoring”. In: *Conference on Design, Automation and Test in Europe*. DATE ’10. Dresden, Germany: European Design and Automation Association, 2010, pp. 1518–1523.

Rawat2014 Priyanka Rawat et al. “Wireless sensor networks: a survey on recent developments and potential synergies”. English. In: *The Journal of Supercomputing* 68.1 (2014), pp. 1–48.

Reinhardt2009 A. Reinhardt et al. "On the energy efficiency of lossless data compression in wireless sensor networks". In: *34th IEEE Conference on Local Computer Networks (LCN)*. 2009, pp. 873–880.

Rodriguez-Henriquez2007 Francisco Rodríguez-Henríquez et al. *Cryptographic Algorithms on Reconfigurable Hardware*. Signals and Communication Technology. Springer US, 2007.

Rosello2011 V. Rosello, J. Portilla, and T. Riesgo. "Ultra low power FPGA-based architecture for Wake-up Radio in Wireless Sensor Networks". In: *37th Annual Conference on IEEE Industrial Electronics Society (IECON)*. 2011, pp. 3826–3831.

Sayood2005 Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 2005.

Shahzad2014 Khurram Shahzad. "Energy Efficient Wireless Sensor Node Architecture for Data and Computation Intensive Applications". PhD thesis. Mid Sweden University, Department of Electronics Design, 2014, p. 112.

Sheng2005 Min Sheng, Jiandong Li, and Yan Shi. "Relative degree adaptive flooding broadcast algorithm for ad hoc networks". In: *IEEE Transactions on Broadcasting* 51.2 (2005), pp. 216–222.

Silverstein1978 H. Silverstein. "CEASAR, SOSUS, and Submarines: Economic and Institutional Implications of ASW Technologies". In: *OCEANS '78*. 1978, pp. 406–410.

Stelte2010 Björn Stelte. "Toward development of high secure sensor network nodes using an FPGA-based architecture". In: *6th International Wireless Communications and Mobile Computing Conference*. IWCMC '10. Caen, France: ACM, 2010, pp. 539–543.

Stojcev2009 M.K. Stojcev, M.R. Kosanovic, and L.R. Golubovic. "Power management and energy harvesting techniques for wireless sensor nodes". In: *9th International Conference on Telecommunication in Modern Satellite, Cable, and Broadcasting Services (TELSIKS)*. 2009, pp. 65–72.

Stubbs1995 N Stubbs, JT Kim, and CR Farrar. "Field Verification of a Nondestructive Damage Localization and Severity Estimation Algorithm". In: *Proceedings of the 13th International Modal Analysis Conference*. Vol. 2460. 1995, p. 210.

Tang2014 Y. Tang and N. Bergmann. "A Hardware Scheduler based on Task Queues for FPGA-based Embedded Real-Time Systems". In: *Computers, IEEE Transactions on* PP.99 (2014), pp. 1–1.

Vanheel2011 Frank Vanheel et al. "Automated linear regression tools improve RSSI WSN localization in multipath indoor environment". English. In: *EURASIP Journal on Wireless Communications and Networking* 2011.1, 38 (2011).

Vera-Salas2010 L.A. Vera-Salas et al. "Reconfigurable Node Processing Unit for a Low-Power Wireless Sensor Network". In: *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. 2010, pp. 173–178.

Voyles2010 R.M. Voyles et al. "RecoNode: A reconfigurable node for heterogeneous multi-robot search and rescue". In: *Safety Security and Rescue Robotics (SSRR), 2010 IEEE International Workshop on*. 2010, pp. 1–7.

Walravens2014 C. Walravens and W. Dehaene. “Low-Power Digital Signal Processor Architecture for Wireless Sensor Nodes”. In: *IEEE Transactions on Very Large Scale Integration Systems* 22.2 (2014), pp. 313–321.

Wang2012 Lu Wang, Kaishun Wu, and M. Hamdi. “Combating Hidden and Exposed Terminal Problems in Wireless Networks”. In: *IEEE Transactions on Wireless Communications* 11.11 (2012), pp. 4204–4213.

Watral2013 Z. Watral and A. Michalski. “Selected problems of power sources for wireless sensors networks”. In: *IEEE Instrumentation Measurement Magazine* 16.1 (2013), pp. 37–43.

Xu2009 Na Xu et al. “An Improved Flooding Time Synchronization Protocol for Industrial Wireless Networks”. In: *International Conference on Embedded Software and Systems (ICESS)*. 2009, pp. 524–529.

Yan2010 Li Yan et al. “Hardware Implementation of muC/OS-II Based on FPGA”. In: *Education Technology and Computer Science (ETCS), 2010 Second International Workshop on*. Vol. 3. 2010, pp. 825–828.

Yeh1991 Pen-Shu Yeh, Robert F. Rice, and Warner Miller. *On the optimality of code options for a universal noiseless coder*. Tech. rep. Jet Propulsion Laboratory, 1991.

Yi2013 Won-Jae Yi, S. Gilliland, and J. Saniie. “Wireless sensor network for structural health monitoring using System-on-Chip with Android smartphone”. In: *Sensors, 2013 IEEE*. 2013, pp. 1–4.

Yick2008 Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. “Wireless sensor network survey”. In: *The International Journal of Computer and Telecommunications Networking* 52 (12 2008), pp. 2292–2330.

Zhao2015 Yi Zhao, J.R. Smith, and A. Sample. “NFC-WISP: A sensing and computationally enhanced near-field RFID platform”. In: *IEEE International Conference on RFID*. 2015, pp. 174–181.

Zhiyong2009 Chao Hu Zhiyong et al. “A novel FPGA-based wireless vision sensor node”. In: *IEEE International Conference on Automation and Logistics (ICAL)*. 2009, pp. 841–846.

Hardware Datasheets

23A1024 Microchip. *1Mbit SPI Serial SRAM with SDI and SQI Interface*. 2011. URL: `http://www.microchip.com/wwwproducts/en/23A1024` (visited on 2016-11-14).

34411A Agilent. *34410A/11A Multimeter mit 6.5 Stellen - Benutzerhandbuch*. 2005.

356A16 PCB Piezotronics. *Triaxial, high sensitivity, ceramic shear ICP*. 2008. URL: `http://www.pcb.com/Products.aspx?m=356A16` (visited on 2016-11-14).

7-Series Xilinx. *7 Series FPGAs Overview*. DS180. 2015. URL: `http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf` (visited on 2016-11-14).

AC332 Actel. *AC332: Flash*Freeze Control Using an Internal Oscillator*. Tech. rep. Actel Corporation, 2009. URL: `http://www.actel.com/documents/FlashFreeze_Ctrl_using_IOSC_AN.pdf` (visited on 2016-11-14).

ADXL362 Analog Devices. *Micropower, 3-Axis, +-2 g/+-4 g/+-8 g Digital Output MEMS Accelerometer*. 2014. URL: http://www.analog.com/en/products/mems/mems-accelerometers/adxl362.html (visited on 2016-11-14).

ATmega128 Atmel. *8-bit Microcontroller with 128K Bytes In-System Programmable Flash*. URL: http://www.atmel.com (visited on 2016-11-14).

ATmega256RFR2 Atmel. *8-bit AVR Microcontroller with Low Power 2.4 GHz Transceiver for ZigBee and IEEE 802.15.4*. 8393B-MCU. 2013. URL: http://www.atmel.com/devices/atmega256rfr2.aspx (visited on 2016-11-14).

Axcelerator Microsemi. *Axcelerator Family FPGAs*. 2012. URL: http://www.microsemi.com/products/fpga-soc/antifuse-fpgas/axcelerator (visited on 2016-11-14).

BioMonitor BIOTRONIK. URL: http://www.heart-monitoring.com (visited on 2016-11-14).

BlackFin Analog Devices. *Blackfin Embedded Processor with Codec*. 2010. URL: http://www.analog.com/en/products/processors-dsp/blackfin.html (visited on 2016-11-14).

CC2530 Texas Instruments. *CC253x System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee Applications*. URL: http://www.ti.com/product/cc2530 (visited on 2016-11-14).

CC2650 Texas Instruments. *SimpleLink Multistandard Wireless MCU*. SWRS158B. 2016. URL: http://www.ti.com/product/cc2650 (visited on 2016-11-14).

CC430 Texas Instruments. *CC430 Family User's Guide*. URL: http://www.ti.com/corp/docs/landing/cc430 (visited on 2016-11-14).

CoolRunner-II Xilinx. *CoolRunner-II CPLD Family*. DS090. 2008. URL: http://www.xilinx.com/support/documentation/data_sheets/ds090.pdf (visited on 2016-11-14).

CPLDAPP Xilinx. *Xilinx CPLD Applications Handbook - Featuring CoolRunner-II and XC9500XL CPLDs*. 2006. URL: http://www.xilinx.com/publications/products/cpld/cpld_applications_handbook.pdf (visited on 2016-11-14).

HyperFlex Altera. *Stratix 10 Advance Information Brief*. 2015. URL: https://www.altera.com/products/fpga/stratix-series/stratix-10/features.html (visited on 2016-11-14).

iCE40 Lattice Semiconductor. *iCE40 LP/HX Family Data Sheet*. DS1040. 2015. URL: http://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40.aspx (visited on 2016-11-14).

iCE40ultra Lattice Semiconductor. *iCE40 Ultra Family Data Sheet*. DS1048. 2015. URL: http://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40Ultra.aspx (visited on 2016-11-14).

iCE65 Silicon Blue. *iCE65 Ultra Low-Power mobileFPGA Family*. 2010.

IGLOO Microsemi. *IGLOO Low Power Flash FPGAs with Flash*Freeze Technology*. 2015. URL: http://www.microsemi.com/products/fpga-soc/fpga/igloo-e (visited on 2016-11-14).

IGLOO2 Microsemi. *IGLOO2 FPGA and SmartFusion2 SoC FPGA*. 2015. URL: http://www.microsemi.com/products/fpga-soc/fpga/igloo2-fpga (visited on 2016-11-14).

IGLOOnano Microsemi. *IGLOO nano Low Power Flash FPGAs with Flash*Freeze Technology*. 2015. URL: http://www.microsemi.com/products/fpga-soc/fpga/igloo-nano (visited on 2016-11-14).

IGLOOplus Microsemi. *IGLOO PLUS Low Power Flash FPGAs with Flash*Freeze Technology*. 2012. URL: http://www.microsemi.com/products/fpga-soc/fpga/igloo-plus (visited on 2016-11-14).

Imote2 MEMSIC. *Imote2 High-performance Wireless Sensor Network Node*. URL: http://www.memsic.com (visited on 2016-11-14).

LTC3388 *20V High Efficiency Nanopower Step-Down Regulator*. 2010. URL: http://www.linear.com/product/LTC3388 (visited on 2016-11-14).

LTC6930 Linear Technology. *32.768 kHz to 8.192 MHz Presicion Micro-Power Oscillators*. 2011. URL: http://www.linear.com/product/LTC6930 (visited on 2016-11-14).

MachXO3 Lattice Semiconductor. *MachXO3 Family Data Sheet*. DS1047. 2015. URL: http://www.latticesemi.com/en/Products/FPGAandCPLD/MachXO3.aspx (visited on 2016-11-14).

Max5 Altera. *MAX V Device Handbook*. 2011. URL: https://www.altera.com/products/cpld/max-series/max-v/overview.html (visited on 2016-11-14).

MB85RS1MT Fujitsu. *Memory FRAM 1M (128 K x 8) Bit SPI*. 2013.

Mica2 MEMSIC. *MICA2 wireless measurement system*. 2010. URL: http://www.memsic.com (visited on 2016-11-14).

MicaZ MEMSIC. *MICAz wireless measurement system*. 2010. URL: http://www.memsic.com (visited on 2016-11-14).

MSP430 Texas Instruments. *MSP430x1xx Family User's Guide*. SLAU049F. 2006. URL: www.ti.com/lit/ug/slau049f/slau049f.pdf.

ProASIC3 Microsemi. *ProASIC3 Flash Family FPGAs with Optional Soft ARM Support*. 2015. URL: http://www.microsemi.com/products/fpga-soc/fpga/proasic3-e (visited on 2016-11-14).

ProASIC3L Microsemi. *ProASIC3L Low Power Flash FPGAs with Flash*Freeze Technology*. 2015. URL: http://www.microsemi.com/products/fpga-soc/fpga/proasic3l (visited on 2016-11-14).

ProASIC3nano Microsemi. *ProASIC3 nano Flash FPGAs*. 2015. URL: http://www.microsemi.com/products/fpga-soc/fpga/proasic3-nano (visited on 2016-11-14).

PXA270 Intel. *PXA270 Processor - Electrical, Mechanical, Thermal Specification*. 2006.

SCADAS Siemens. *LMS SCADAS Data Acquisition Systems*. 2011. URL: http://www.plm.automation.siemens.com/de_de/products/lms/testing/scadas/ (visited on 2016-11-14).

SST25WF010 Silicon Storage Technology. *512 Kbit / 1 Mbit / 2 Mbit / 4 Mbit 1.8V SPI Serial Flash*. 2011.

ST60-10P Hirose Connector. *Interface Connectors for Portable Terminal Devices*. 2014. URL: http://www.mouser.com/ds/2/185/ed_ST_20140920-515456.pdf (visited on 2016-11-14).

STM32F407 ST. *Reference Manual*. RM0090. 2016. URL: http://www.st.com/en/microcontrollers/stm32f407-417.html (visited on 2016-11-14).

STS4 Bridge Diagnostics Inc. *STS4 WIRELESS STRUCTURAL TESTING SYSTEM*. URL: http://bridgetest.com/wp-content/uploads/Products-STS42.pdf (visited on 2016-11-14).

TelosB MEMSIC. *TelosB mote platform*. URL: http://www.memsic.com (visited on 2016-11-14).

Thermostat Nest. *Learning Thermostat*. 2012. URL: https://nest.com/thermostat/meet-nest-thermostat/ (visited on 2016-11-14).

TIBPAL22V10 Texas Instruments. *TIBPAL22V10-10C - High-performance Impact-X Programmable Array Logic Circuits*. 2010. URL: http://www.ti.com/product/TIBPAL22V10-20M (visited on 2016-11-14).

TMoteSky Moteiv Corporation. *Ultra low power IEEE 802.15.4 compliant wireless sensor module*. June 2006. URL: http://www.snm.ethz.ch/snmwiki/pub/uploads/Projects/tmote_sky_datasheet.pdf (visited on 2016-11-14).

UltraScale Xilinx. *UltraScale Architecture and Product Overview*. DS890. 2015. URL: http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf (visited on 2016-11-14).

Waspmote Libelium. *Datasheet*. 2015. URL: http://www.libelium.com/products/waspmote/ (visited on 2016-11-14).

XP2 Lattice Semiconductor. *LatticeXP2 Family Data Sheet*. DS1009. 2014. URL: http://www.latticesemi.com/en/Products/FPGAandCPLD/LatticeXP2.aspx (visited on 2016-11-14).

Software Tools

IAR IAR Systems. *Embedded Workbench*. URL: https://www.iar.com/iar-embedded-workbench (visited on 2016-11-14).

iCEcube Lattice. *iCEcube2 2015-04 User Guide*. 2015. URL: http://www.latticesemi.com/iCEcube2 (visited on 2016-11-14).

Keil Keil Microcontroller Tools. *C51 Development Tools*. URL: http://www.keil.com/c51 (visited on 2016-11-14).

Libero-SoC Microsemi. *Libero SoC Design Suite*. 2015. URL: http://www.microsemi.com/products/fpga-soc/design-resources/design-software/libero-soc (visited on 2016-11-14).

MATLAB MathWorks. *Primer*. 2015. URL: http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf (visited on 2016-11-14).

Quartus Altera. *Quartus II Handbook*. QII5V1. 2015. URL: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/qts/quartusii_handbook.pdf (visited on 2016-11-14).

ROCCC Jacquard Computing. *ROCCC 2.0 User's Manual*. 2011. URL: http://www.cse.wustl.edu/~roger/565M.f12/UserManual-0.6.pdf (visited on 2016-11-14).

SDCC Sandeep Dutta. *SDCC Compiler User Guide*. 2016. URL: http://sdcc.sourceforge.net/doc/sdccman.pdf (visited on 2016-11-14).

SynphonyHLS Synopsys. *Synphony HLS Actel Edition User Guide*. 2010.

SynphonyModelCompiler Synopsys. *Synphony Model Compiler User Guide*. Microsemi Edition I-2013.09M. 2013. URL: http://www.synopsys.com/Tools/Implementation/FPGAImplementation/Pages/synphony-model-compiler.aspx (visited on 2016-11-14).

Vivado Xilinx. *UltraFast Design Methodology Guide for the Vivado Design Suite*. UG949. 2015. URL: http://www.xilinx.com/support/documentation/sw_manuals/ug949-vivado-design-methodology.pdf (visited on 2016-11-14).

Vivado-HLS Xilinx. *Vivado Design Suite User Guide - High-Level Synthesis*. UG902. 2014. URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf (visited on 2016-11-14).

Specifications and Standards

100.11a International Society of Automation. *ANSI/ISA-100.11a-2011 Wireless systems for industrial automation: Process control and related applications*. Standard. 2011. URL: https://www.isa.org/store/ansi/isa-10011a-2011-wireless-systems-for-industrial-automation-process-control-and-related-applications/118261 (visited on 2016-11-14).

6LoWPAN Internet Engineering Task Force. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. 2007. URL: https://tools.ietf.org/html/rfc4944 (visited on 2016-11-14).

802.11 IEEE Computer Society. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. 2012. URL: http://standards.ieee.org/about/get/802/802.11.html (visited on 2016-11-14).

802.15.1 IEEE Computer Society. *Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs)*. 2005. URL: http://standards.ieee.org/findstds/standard/802.15.1-2005.html (visited on 2016-11-14).

802.15.4 IEEE Computer Society. *IEEE Standard for Local and metropolitan area networks - Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. 2011. URL: http://standards.ieee.org/about/get/802/802.15.html (visited on 2016-11-14).

802.15.6 IEEE Computer Society. *Wireless Body Area Networks*. 2012. URL: http://standards.ieee.org/findstds/standard/802.15.6-2012.html (visited on 2016-11-14).

Thread Thread Group. *Thread Stack Fundamentals*. 2015. URL: http://threadgroup.org/ (visited on 2016-11-14).

UVM Accellera Systems Initiative. *Universal Verification Methodology (UVM) 1.2 Users Guide*. 2015.

WirelessHART HART Communication Foundation. *WirelessHART Overview*. 2008. URL: http://en.hartcomm.org/hcp/tech/wihart/wireless_overview.html (visited on 2016-11-14).

ZigBee ZigBee Alliance. *ZigBee Specification*. 2012. URL: http://www.zigbee.org/zigbee-for-developers/network-specifications/zigbeepro (visited on 2016-11-14).

Others

FrostSullivan2015-WSN *Wireless Sensor Networks See Uptake in Global Industries Due to Technological Breakthroughs*. Frost & Sullivan, 2015. URL: http://corpcom.frost.com/forms/NA_PR_ABrown_N92B-32_22Jan2015 (visited on 2016-11-14).

H2020-FA *Horizon 2020 Work Programme 2016 - 2017, Cross-cutting activities (Focus Areas)*. European Commision, 2015. URL: http://ec.europa.eu/programmes/horizon2020/en/h2020-section/cross-cutting-activities-focus-areas (visited on 2016-11-14).

H2020-ICT *Horizon 2020 Work Programme 2016 - 2017, Information and Communication Technologies*. European Commision, 2015. URL: http://ec.europa.eu/programmes/horizon2020/en/h2020-section/information-and-communication-technologies (visited on 2016-11-14).

TMR2015-FPGA *FPGA Market - Global Industry Analysis, Size, Share, Growth, Trends and Forecast, 2014 - 2020*. Market Report. Transparency Market Research, 2015. URL: http://www.transparencymarketresearch.com/field-programmable-gate-array.html (visited on 2016-11-14).

TMR2015-IoT *Internet of Things Market - Global Industry Analysis, Size, Share, Growth, Trends and Forecast, 2015 - 2021*. Market Report. Transparency Market Research, 2015. URL: http://www.transparencymarketresearch.com/internet-of-things-market.html (visited on 2016-11-14).

Yannick2015 Michel Yannick. *Thomas Schäfer besucht Fraunhofer-Institut*. 2015. URL: http://www.rheinmaintv.de/video/Thomas-Schaefer-besucht-Fraunhofer-Institut/090ff204f49d251f6c2b1fbd1185ed12 (visited on 2016-11-14).

Abbreviations

ACLK	auxiliary clock
ADC	Analog-to-Digital Converter
ADPCM	Adaptive Differential Pulse Code Modulation
ALAC	Apple Lossless Audio Codec
ALM	Adaptive Logic Module
ALU	Arithmetic Logic Unit
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
ASM	Auto-Sequencing Memory
BA	Back Annotation
BDI	Bridge Diagnostics Inc
BLE	Bluetooth Low Energy
BRAM	Block RAM
CGRA	Coarse-Grained Reconfigurable Architecture
CLB	Configurable Logic Block
CMem	Configuration Memory
COTS	commercial off-the-shelf
CPLD	Complex Programmable Logic Device
CPS	Cyber Physical System
CPU	Central Processing Unit
CSMA	Carrier Sense Multiple Access
DARPA	United States Defense Advanced Research Projects Agency
DAQ	Data Acquisition
DC/DC	Voltage Level Converter
DFT	Discrete Fourier Transformation
DMA	Direct Memory Access
DPCM	Differential Pulse Code Modulation
DPM	Dynamic Power Management
DSN	Distributed Sensor Networks
DSP	Digital Signal Processor
DVFS	Dynamic Voltage and Frequency Scaling

EMA Experimental Modal Analysis

FB Functional Block

FDD Frequency Domain Decomposition

FET Field-Effect Transistor

FF Flip-Flop

FFT Fast Fourier Transformation

FIFO First In, First Out

FIR Finite Impulse Response

FLAC Free Lossless Audio Codec

FPGA Field Programmable Gate Array

FRAM Ferroelectric RAM

FTSP Flooding Time Synchronization Protocol

GAG Generic Address Generator

GPIO General Purpose Input/Output

GPU Graphics Processing Unit

HaLOEWEn Hardware-Accelerated Low Energy Wireless Embedded Sensor Node

HaLoMote Hardware-Accelerated Low Power Mote

HDL Hardware Description Language

HLS High-Level Synthesis

HMI Human-Machine Interface

HPC High-Performance Computing

HW hardware

IC Integrated Circuit

ICP integrated circuit piezoelectric

ICT Information and Communication Technology

IDE Integrated Development Environment

IEEE Institute of Electrical and Electronics Engineers

I^2C Inter-Integrated Circuit

ILP Integer Linear Program

IO Input/Output

IOB Input/Output Block

IoT Internet of Things

IP	Intellectual Property
ISA	International Society of Automation
ISM	Industrial, Scientific and Medical
JTAG	Joint Test Action Group
LBF	Institute for Structural Durability and System Reliability
LC	Logic Cell
LE	Logic Element
LED	Light-Emitting Diode
LMS	LMS Test.Lab 14A
LR	Linear Regression
LR-WPAN	Low Rate WPAN
LUT	Lookup Table
LZW	Lempel-Ziv-Welch
MAC	Medium Access Control
MACC	Multiply-Accumulate
MCLK	main clock
MCU	microcontroller
MEMS	micro-electro-mechanical
MMIO	memory-mapped IO
MSE	Modal Strain Energy
NFC	Near Field Communication
NiCd	Nickel-Cadmium
NiMH	Nickel-Metal Hydride
NRE	non-recurring engineering costs
NVM	non-volatile memory
OMA	Operational Modal Analysis
OSC	Oscillator
OSI	Open Systems Interconnection
PAL	Programmable Array Logic
PAR	Place and Route
PC	Personal Computer
PCB	Printed Circuit Board

PE	Processing Element
PEG	Piezoelectric Generator
PLA	Programmable Logic Array
PLD	Programmable Logic Device
PLL	Phase Locked Loop
ppm	parts per million
PROM	Programmable Read-Only Memory
RAM	Random Access Memory
RCU	Reconfigurable Compute Unit
RD	Random Decrement
RDT	Random Decrement Technique
RFID	Radio-frequency Identification
RF-SoC	Radio Frequency System-on-Chip
RLR	Rolling Linear Regression
RLRCT	Rolling Linear Regression with Coordinate Transformation
RMS	Root Mean Square
ROM	Read-Only Memory
RSSI	Received Signal Strength Indication
RTL	Register Transfer Level
RTOS	Real-Time Operating System
RTSP	Recursive Time Synchronization Protocol
SDF	Standard Delay Format
SFD	start of frame delimiter
SHM	Structural Health Monitoring
SIMD	Single Instruction Multiple Data
SM	Switch Matrix
SoC	System-on-Chip
SPI	Serial Peripheral Interface
SPLD	Simple Programmable Logic Device
SRAM	Static RAM
SRD	Short Range Device
SVD	Singular Value Decomposition

TEG	Thermoelectric Generator
TI	Texas Instruments
UC	University of California
USART	Universal Synchronous and Asynchronous Serial Receiver/Transmitter
USB	Universal Serial Bus
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLIW	Very Long Instruction Word
WAN	Wide Area Network
WBAN	Wireless Body Area Network
WINS	Wireless Integrated Network Sensors
WISP	Wireless Identification and Sensing Platform
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network
WPAN	Wireless Personal Area Network

List of Figures

List of Tables

List of Algorithms

List of Code Listings