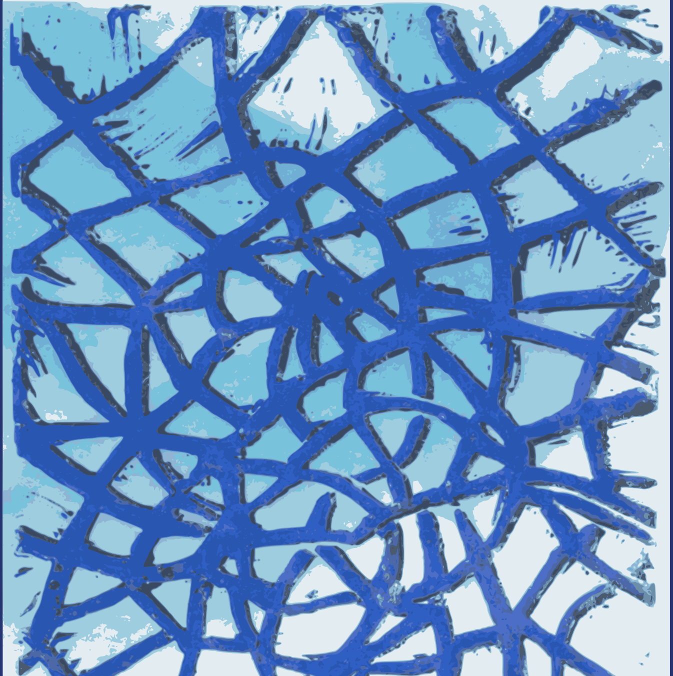
Runtime Hardware Reconfiguration in Wireless Sensor Networks for Condition Monitoring

Vom Fachbereich Elektrotechnik und Informationstechnik
zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation von Dipl.-Ing. François Philipp aus Forbach, Frankreich
Tag der Einreichung: 16. Juni 2014, Tag der Prüfung: 1. September 2014
Darmstadt — D 17 — 2014

1. Gutachten: Prof. Dr. Dr. h. c. mult. Manfred Glesner
2. Gutachten: Prof. Dr.-Ing. Christian Hochberger
3. Gutachten: Prof. Dr. Eduardo de la Torre



TECHNISCHE
UNIVERSITÄT
DARMSTADT

et:it

Runtime Hardware Reconfiguration in Wireless Sensor Networks for Condition Monitoring

Genehmigte Dissertation von Dipl.-Ing. François Philipp aus Forbach, Frankreich

1. Gutachten: Prof. Dr. Dr. h. c. mult. Manfred Glesner
2. Gutachten: Prof. Dr.-Ing. Christian Hochberger
3. Gutachten: Prof. Dr. Eduardo de la Torre

Tag der Einreichung: 16. Juni 2014
Tag der Prüfung: 1. September 2014

Darmstadt — D 17

Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-41597
URL: http://tuprints.ulb.tu-darmstadt.de/4159

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

Die Veröffentlichung steht unter folgender Creative Commons Lizenz:
Namensnennung – Keine kommerzielle Nutzung – Keine Bearbeitung 2.0 Deutschland
http://creativecommons.org/licenses/by-nc-nd/2.0/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 24. September 2014

(François Philipp)

en mémoire de mon grand-père Pierre

Acknowledgments

A Ph.D. thesis is certainly a long, challenging but also rewarding work that cannot be accomplished without a high quality supervision and close accompaniment. In this sense, I believe that Prof. Manfred Glesner was thanks to his long and rich experience and his close relationships with his research assistants, one of the greatest source of advice and inspiration for the development of relevant scientific work. Therefore, my very first acknowledgment goes to him for the wise guidance and all opportunities he offered me during the time of my stay in the Microelectronic Systems Research Group, which resulted in the contributions presented in this document.

Similarly, I would like to sincerely thank Prof. Christian Hochberger from the Computer Systems Group of TU Darmstadt and Prof. Eduardo de la Torre from the Center for Industrial Electronics in Madrid for being the co-referees of my Ph.D thesis. I express my gratitude to all members of the examination committee for taking the time to consciously and rightly evaluate this work.

Nowadays, scientific innovations are often triggered by multi-disciplinary research projects. With the European project Maintenance on Demand and the LOEWE-Zentrum AdRIA, this requirement was largely fulfilled by offering exciting application scenarios in the domain of mechanical and structural engineering. I would like to thank Prof. Thilo Bein who was coordinating these projects and guiding my work towards a successful completion of their objectives. I thank all the persons who collaborated with me during these projects and particularly Dominik Elsberkirch and Manuel Andreu in MoDe as well as Andreas Engel in AdRIA. Thank you to Prof. Saman Halgamuge and all his coworkers for the fruitful partnership built during the DAAD collaborative project with the University of Melbourne. A large part of this work would not have been possible without the help of Javier Martinez, with whom I started one of the most productive collaborative work.

A Ph.D. thesis is also an everyday fight that could not be won without a constant support of coworkers and colleagues. This time at the Microelectronic Systems Research Group allowed me developing new friendships and constructive collaborations which were the source of many parts of this work. I thank Heiko Hinkelmann for his wise supervision during my diploma thesis and the opportunity he gave me by recommending me to work on the Maintenance on Demand project. The early stages of the thesis were the results of a great collaborative work with Faizal Arya Samman and Ping Zhao. I am thankful to my colleagues Ramkumar Ganesan, Sebastian Pankalla, Leandro Moller and particularly Surapong Pongyupinpanich and Christopher Spies who share many memories and good times with me. Thank you to Yuan Fang, Lufei Shen, Alex Schönberger and all colleagues from the Integrated Electronic Systems group who started they journey towards the doctoral degree with me. I also thank Pablo Guerrero from the Databases and Distributed Systems group, Kristof Van Laerhoven from the Embedded Sensing Group and all co-organizers of the Wireless Sensor Networks labs and seminars, which always were an interesting and enriching experience. Administrative work would have been much more difficult without the help of Silvia Hermann and Brigitte Kuntzsch. I of course thank all the students who worked with me during the time of the thesis. I am very proud that I had the chance to accompany them during their studies.

Outside the work, the support of the close family and friends is primordial to keep the motivation level and the enthusiasm high. Many thanks to my mother who illustrated the cover page of this document, to my father for his support and his constant care about the advancement of the work and all other members of my family who shared with me the happy and hard events of life which occurred during these last couple of years. Finally, I address my last and strongest thank you to Lang Yu, who illuminated my daily life with love and cheerfulness.

Abstract

The integration of miniaturized heterogeneous electronic components has enabled the deployment of tiny sensing platforms empowered by wireless connectivity known as wireless sensor networks. Thanks to an optimized duty-cycled activity, the energy consumption of these battery-powered devices can be reduced to a level where several years of operation is possible. However, the processing capability of currently available wireless sensor nodes does not scale well with the observation of phenomena requiring a high sampling resolution. The large amount of data generated by the sensors cannot be handled efficiently by low-power wireless communication protocols without a preliminary filtering of the information relevant for the application. For this purpose, energy-efficient, flexible, fast and accurate processing units are required to extract important features from the sensor data and relieve the operating system from computationally demanding tasks. Reconfigurable hardware is identified as a suitable technology to fulfill these requirements, balancing implementation flexibility with performance and energy-efficiency.

While both static and dynamic power consumption of field programmable gate arrays has often been pointed out as prohibitive for very-low-power applications, recent programmable logic chips based on non-volatile memory appear as a potential solution overcoming this constraint. This thesis first verifies this assumption with the help of a modular sensor node built around a field programmable gate array based on Flash technology. Short and autonomous duty-cycled operation combined with hardware acceleration efficiently drop the energy consumption of the device in the considered context.

However, Flash-based devices suffer from restrictions such as long configuration times and limited resources, which reduce their suitability for complex processing tasks. A template of a dynamically reconfigurable architecture built around coarse-grained reconfigurable function units is proposed in a second part of this work to overcome these issues. The module is conceived as an overlay of the sensor node FPGA increasing the implementation flexibility and introducing a standardized programming model. Mechanisms for virtual reconfiguration tailored for resource-constrained systems are introduced to minimize the overhead induced by this genericity.

The definition of this template architecture leaves room for design space exploration and application-specific customization. Nevertheless, this aspect must be supported by appropriate design tools which facilitate and automate the generation of low-level design files. For this purpose, a software tool is introduced to graphically configure the architecture and operation of the hardware accelerator. A middleware service is further integrated into the wireless sensor network operating system to bridge the gap between the hardware and the design tools, enabling remote reprogramming and scheduling of the hardware functionality at runtime.

At last, this hardware and software toolchain is applied to real-world wireless sensor network deployments in the domain of condition monitoring. This category of applications often require the complex analysis of signals in the considered range of sampling frequencies such as vibrations or electrical currents, making the proposed system ideally suited for the implementation. The flexibility of the approach is demonstrated by taking examples with heterogeneous algorithmic specifications. Different data processing tasks executed by the sensor node hardware accelerator are modified at runtime according to application requests.

Kurzfassung

Die Integration miniaturisierter heterogener elektronischer Bauteile hat der Einsatz winziger drahtlos angebundener Erfassungsplattformen, sogenannte drahtlose Sensorknoten, ermöglicht. Der Energieverbrauch dieser Komponenten kann dank optimierter Betriebszeiten soweit reduziert werden, dass eine batteriebetriebene Laufzeit über mehrere Jahre möglich ist. Allerdings skaliert die Rechenkapazität der bisher vorliegenden Sensorknoten bei Beobachtung von Ereignissen, die eine hohe Abtastauflösung benötigen, schlecht. Eine große Menge von Sensordaten kann ohne eine Vorfilterung der für die Anwendung relevanten Merkmale nur ineffizient durch drahtlose stromsparende Kommunikationsprotokolle verarbeitet werden. Daher werden energieeffiziente, flexible, schnelle und genaue Recheneinheiten benötigt, um wichtige Merkmalen aus den Sensordaten lokal zu extrahieren, ohne das Betriebssystem mit rechenintensiven Tasks zu belasten. Rekonfigurierbare Hardware ist als eine geeignete Zieltechnologie anerkannt, um diese Anforderungen durch eine Balance zwischen Implementierungsflexibilität, Performanz und Energieeffizienz zu erfüllen.

Obwohl der statische und dynamische Leistungsverbrauch der *Field Programmable Gate Arrays* (FPGAs) sich oft als nachteilig für Anwendungen mit sehr niedrigem Energieverbrauch darstellte, kommen neueste, auf einem nichtflüchtigen Speicher basierende, programmierbare Logikbausteine als mögliche Lösung, die diese Nachteile aufheben, in Frage. Diese Doktorarbeit überprüft zuerst diese These mit Hilfe eines modularen Sensorknotens, der auf einem Flash-basierenden FPGA aufgebaut ist. Die Kombination einer kurzen und autonomen Betriebsperiode mit Hardwarebeschleunigung ergibt eine effiziente Reduzierung des Leistungsverbrauchs für den betrachteten Anwendungsbereich.

Allerdings weisen Flash-basierende Bausteine unter Beschränkungen wie lange Konfigurationszeiten und begrenzte Ressourcen auf, die ihre Tauglichkeit für komplexe Datenverarbeitungsaufgaben reduziert. Eine Vorlage einer dynamisch rekonfigurierbaren Architektur, die auf grob-granularen Funktionseinheiten basiert, wird in einem zweiten Teil dieser Arbeit als potentielle Lösung vorgestellt. Das Rechenmodul wird als ein Overlay des FPGAs realisiert und verbessert die Implementierungsfreiheit mit Hilfe eines standardisierten Programmierungsmodells zur Laufzeit. Für ressourcenbeschränkte Geräte angepasste Mechanismen werden eingeführt, um das Modul virtuell zu rekonfigurieren und den durch diese Konfigurierbarkeit erzeugten Zusatzaufwand zu minimieren.

Diese Architekturvorlage erschließt den Entwurfsraum und vereinfacht die anwendungsspezifische Anpassung der Plattform. Dennoch müssen diese Funktionalitäten durch entsprechende Designwerkzeuge für automatische Erzeugung von Designdateien auf niedriger Ebene unterstützt werden. Hierzu wird ein Softwarewerkzeug eingeführt, um die Architektur und den Ablauf der Hardwarebeschleuniger graphisch zu konfigurieren. Zusätzlich wird ein Middleware-Dienst im Betriebssystem des Sensornetzwerks integriert, um die Kluft zwischen der Hardware und den Designwerkzeugen zu schließen. Dies ermöglicht eine Umprogrammierung aus der Ferne und Ablaufplanung der Hardwarefunktionalität zur Laufzeit.

Zuletzt wird diese Hardware- und Software-Werkzeugkette mit relevanten Anwendungen im Bereich der Zustandsüberwachung bewertet. Die komplexe Analyse von Signalen im betrachteten Abtastfrequenzbereich, wie Schwingungen oder elektrische Ströme, wird für solche Anwendungen oft benötigt. Das vorgestellte System ist somit für die Implementierung bestens geeignet. Die Flexibilität des Konzepts wird mit Hilfe von Algorithmen mit heterogenen algorithmischen Anforderungen gezeigt. Auf Anwendungsanfrage können unterschiedliche Datenverarbeitungsaufgaben auf der Sensorknotenhardware zur Laufzeit beschleunigt werden.

Contents

Affidavit I

Acknowledgments V

Abstract VII

Kurzfassung IX

Table of Contents XI

List of Symbols XV

List of Abbreviations XVII

List of Figures XXI

List of Tables XXV

I Context and Motivation for Reconfigurable Hardware in Wireless Sensor Networks 1

1 Introduction 3

- 1.1 Background . . . 3
- 1.2 Research scope and objectives . . . 4
- 1.3 Thesis outline . . . 4

2 High-Bandwidth Sensing Wireless Networks 9

- 2.1 Origins of wireless sensor networks . . . 9
  - 2.1.1 Bell's law . . . 9
  - 2.1.2 Moore and more . . . 10
  - 2.1.3 Ubiquitous computing . . . 11
- 2.2 The design space of wireless sensor networks . . . 12
  - 2.2.1 Operation of a wireless sensor network . . . 12
    - 2.2.1.1 General principle . . . 13
    - 2.2.1.2 System-level metrics . . . 14
  - 2.2.2 Sensing . . . 15
  - 2.2.3 Processing . . . 17
    - 2.2.3.1 Processing hardware . . . 17

2.2.3.2 In-network processing . . . 19
2.2.3.3 Operating systems for wireless sensor networks . . . 20
2.2.4 Wireless communication in wireless sensor networks . . . 21
2.2.5 Energy consumption of a wireless sensor node . . . 23
2.2.5.1 Limits of wireless communication and in-networking processing for high-bandwidth sensing on standard motes . . . 27
2.3 Conclusion . . . 29

3 Reconfigurable Hardware for Low-Power Embedded Systems 31

3.1 Features of reconfigurable hardware systems . . . 32
3.1.1 Technology . . . 33
3.1.2 Granularity . . . 34
3.1.3 Reconfiguration processes . . . 36
3.2 Estimating the power consumption of reconfigurable hardware devices . . . 37
3.3 Low-power duty cycling for FPGAs . . . 39
3.4 Conclusion . . . 41
3.4.1 Summary of the considerations on reconfigurable hardware . . . 41
3.4.2 Outlook on the following part . . . 41

II Design of a Framework Enabling Reconfigurable Hardware Acceleration in Wireless Sensor Networks 43

4 FPGA-based Hardware Acceleration for Wireless Sensor Nodes 45

4.1 Related work . . . 45
4.1.1 Wireless sensor nodes using an FPGA for SoC prototyping . . . 46
4.1.2 Wireless sensor nodes with standalone FPGA . . . 48
4.1.3 Wireless sensor nodes with a co-processing unit based on programmable logic . . . 49
4.1.4 General considerations on related work . . . 51
4.2 Design of a modular FPGA-based low-power mote . . . 53
4.2.1 Core architecture . . . 53
4.2.2 The HaLOEWEn platform . . . 54
4.2.2.1 Main board . . . 54
4.2.2.2 Extensions . . . 55
4.2.2.3 Software . . . 57
4.2.3 MoDe LPSIP . . . 58
4.3 Performance evaluation . . . 58
4.3.1 Power consumption . . . 60
4.3.2 Autonomous control of sleep mode . . . 61
4.3.3 Hardware abstraction layer . . . 62
4.3.4 Application examples . . . 64
4.3.4.1 FFT processing . . . 64
4.3.4.2 Localization . . . 68
4.3.4.3 Combined channel coding and cryptography . . . 70
4.4 Conclusion . . . 72

5 Design of a Virtually Reconfigurable FPGA Overlay Architecture for Resource-Constrained Devices 73

5.1 Introduction . . . 73

5.2 Related work . . . 76
5.2.1 Low-power coarse-grained reconfigurable architectures . . . 77
5.2.2 Virtually reconfigurable hardware . . . 77
5.2.3 Dynamic reconfiguration for Flash-based devices . . . 78
5.3 Template architecture . . . 78
5.3.1 Overview . . . 78
5.3.2 Producer-consumer transactions and interconnect . . . 79
5.3.3 Sensor interfaces . . . 80
5.3.4 Memory elements . . . 82
5.3.5 Processing elements . . . 83
5.3.5.1 Reconfigurable multiply-accumulate unit . . . 83
5.3.5.2 CORDIC unit . . . 84
5.3.5.3 ALU unit . . . 85
5.3.5.4 Other units . . . 88
5.3.6 Reconfiguration layer . . . 88
5.3.6.1 Difference-based reconfiguration . . . 89
5.3.6.2 Context flow . . . 91
5.3.6.3 Meta-reconfiguration . . . 93
5.3.7 Clock and power management . . . 93
5.4 Evaluation . . . 94
5.4.1 Resources . . . 94
5.4.2 Performance . . . 96
5.5 Conclusion . . . 98

6 Tools for Generation and Online Dissemination of Dynamically Reconfigurable Hardware Accelerators 99

6.1 General overview . . . 100
6.2 Graphical configuration interface . . . 101
6.2.1 Related work . . . 101
6.2.2 Architecture editor . . . 102
6.2.3 Data flow editor . . . 104
6.2.4 Evaluation . . . 106
6.2.5 Methodology for application-specific customization and programming of the architecture . . . 106
6.3 Middleware for configuration management . . . 110
6.3.1 Related work . . . 110
6.3.2 General overview . . . 110
6.3.3 Middleware components . . . 111
6.3.4 Event-based reconfiguration . . . 112
6.3.5 Middleware commands . . . 114
6.3.6 Implementation . . . 115
6.3.7 Application examples . . . 116
6.4 Conclusion . . . 118

III Application of Wireless Sensor Networks Strengthened with Reconfigurable Hardware to Condition Monitoring Systems 121

7 Condition Monitoring of a Shock Absorber for Predictive Maintenance 123

7.1 Concept . . . 124
7.1.1 General overview . . . 124
7.1.2 Condition monitoring of the damping system . . . 125
7.2 Motivation for on-demand reconfiguration . . . 129
7.3 Implementation and results . . . 129
7.4 Conclusion . . . 134

8 Diagnosis of Induction Motors 135

8.1 Concept . . . 135
8.1.1 Detection of broken bars . . . 135
8.1.2 Detection of dynamic eccentricity . . . 139
8.1.3 Detection of inter-turn short circuit . . . 139
8.2 Motivation for on-demand reconfiguration . . . 142
8.3 Implementation and results . . . 142
8.3.1 Implementation on HaLOEWEn . . . 143
8.3.2 Diagnosis results . . . 145
8.4 Conclusion . . . 146

9 Conclusion 147

9.1 Contributions of the work . . . 147
9.2 Outlook . . . 148
9.3 Final conclusion . . . 149

IV Appendix 151

A HaLOEWEn Design Files 153

B Details of implemented algorithms 159

B.1 High-Diffusion . . . 159
B.2 Range-extended CORDIC . . . 162
B.3 Corrected fixed-point Fast Fourier Transform . . . 163

References 167

List of Own Publications 185

Supervised Theses 189

Curriculum Vitae 191

List of Symbols

Symbol	Unit	Definition
a	-	Constant angle $e^{\frac{2\pi i}{3}}$ in Fortescue transformation
a_i	-	Address value in address generators
A_i	A	Wavelet approximation at level i of the current signal
C	-	Cost function of a state estimation using a wireless sensor network
C_D	-	Damping ratio of the shock absorber
d_{WSN}	s	Delay of the wireless sensor network
D_i	A	Wavelet details at level i of the current signal
E	J	Energy consumption
f	Hz	Clock frequency
f_bb	Hz	Frequency of spectral components induced by broken bars
f_{dyn}	Hz	Frequency of spectral components induced by dynamic eccentricity
f_s	Hz	Machine supply frequency
I	A	Electric current
I_a, I_b, I_c	A	Phases of the electrical motor
I_0, I_1, I_2	A	Symmetrical components
H	-	Frequency response
K_S	N·m^{-1}	Spring stiffness
K_T	N·m^{-1}	Tire radial stiffness
K_{xy}	-	Correlation factor
L	-	Number of physical phenomenons observed by the wireless sensor network
M	-	Number of nodes in the wireless sensor network
M_S	kg	Sprung mass of a vehicle
M_U	kg	Unsprung mass of a vehicle
n_{b1}, n_{b2}	-	Broken bars harmonic indexes
n_d, n_r, n_{st}	-	Harmonic indexes related to dynamic eccentricity
N	-	Number of sensor nodes cluster
p	-	Number of pole pairs
P	W	Power consumption
R	-	Number of rotor bars
s	-	Slip of the machine
S	-	State of a physical phenomenon
S_{xy}	m^2·s^{-3}	Cross spectrum of two acceleration sequences
S_{yy}	m^2·s^{-3}	Power spectrum of a sequence of acceleration values
$\hat{S}$	-	Estimated state of an observed physical phenomenon
t_S	s	Time when a state was observed
$t_{\hat{S}}$	s	Time when a state has been estimated
$T_{S_i \rightarrow S_j}$	s	Duration of a transition between states
T_{WSN}	s	Time resolution of the wireless sensor network
T_{cyc}	s	Duration of one processor cycle
T_{12}	-	Transmissibility of the damping system
U	-	Set of raw sensor values
V	-	Set of external values provided to a sensor node

Table continued

Symbol	**Unit**	**Definition**
V_{DD}	V	Supply voltage
X	-	Set of values sent by a sensor node
X_1	m	Vertical displacement of the vehicle body
X_2	m	Vertical displacement at the wheel hub
α	-	Switching activity factor
ε_{WSN}	-	Approximation error of the wireless sensor network
λ	$\mathrm{bit \cdot s^{-1}}$	Sensing rate
ξ	m	Road excitation
ω	$\mathrm{rad \cdot s^{-1}}$	Radian frequency

List of Abbreviations

ACM	**A**ctive **C**urrent **M**easurment
ADC	**A**nalog-**D**igital **C**onverter
AdRIA	**Ad**aptronics, **R**esearch, **I**nnovation, **A**pplications
AES	**A**dvanced **E**ncryption **S**tandard
ALU	**A**rithmetic **L**ogic **U**nit
API	**A**pplication **P**rogramming **I**nterface
ASIC	**A**pplication-**S**pecific **I**ntegrated **C**ircuit
BLE	**B**luetooth **L**ow **E**nergy
CGRA	**C**oarse **G**rained **R**econfigurable **A**rchitecture
CMOS	**C**omplementary **M**etal-**O**xide-**S**emiconductor
CORDIC	**CO**ordinate **R**otation **DI**gital **C**omputer
COTS	**C**ommercial **O**ff-**T**he-**S**helf
CPLD	**C**omplex **P**rogrammable **L**ogic **D**evice
CPU	**C**entral **P**rocessing **U**nit
CRC	**C**yclic **R**edundancy **C**heck
DSP	**D**igital **S**ignal **P**rocess-ing ,-or
DWT	**D**iscrete **W**avelet **T**ransform
ECG	**E**lectro**c**ardio**g**raphy
EMG	**E**lectro**m**yo**g**raphy
FEC	**F**orward **E**rror **C**orrection
FFT	**F**ast **F**ourier **T**ransform
FIFO	**F**irst-**I**n **F**irst-**O**ut
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FPU	**F**loating-**P**oint **U**nit
GECO^2	**G**raphical **E**nvironment for **CO**nfiguration and **GE**neration of bitstreams for a **CO**arse-grained dynamically reconfigurable architecture
GPS	**G**lobal **P**ositioning **S**ystem
GUI	**G**raphical **U**ser **I**nterface

HAL	**H**ardware **A**bstracion **L**ayer
HaLOEWEn	**H**ardware-**a**ccelerated **LO**w **E**nergy **W**ireless **E**mbedded Sensor-Actuator **n**ode
HDL	**H**ardware **D**escription **L**anguage
I/O	**I**nput/**O**utput
I^{2}C	**I**nter-**I**ntegrated **C**ircuit
IC	**I**ntegrated **C**ircuit
IoT	**I**nternet **o**f **T**hings
IP	**I**ntellectual **P**roperty
JTAG	**J**oin **T**est **A**ction **G**roup
LPSIP	**L**ow-**P**ower **S**ensor **I**nterface **P**latform
LSB	**L**ess **S**ignificant **B**it
MAC	**M**edium-**A**ccess **C**ontrol
MCU	**M**icro**c**ontroller **U**nit
MEMS	**M**icro**e**lectro**m**echanical **S**ystems
MoDe	**M**aintenance **o**n **De**mand
MSB	**M**ost **S**ignificant **B**it
OS	**O**perating **S**ystem
PAN	**P**ersonal **A**rea **N**etwork
PDR	**P**artial **D**ynamic **R**econfiguration
PE	**P**rocessing **E**lement
PER	**P**acket **E**rror **R**ate
PLD	**P**rogrammable **L**ogic **D**evice
PLL	**P**hase-**L**ocked **L**oop
PWM	**P**ulse **W**ave **M**odulation
RAM	**R**andom-**A**ccess **M**emory
RF	**R**adio **F**requency
ROM	**R**ead-**O**nly **M**emory
RSSI	**R**eceived **S**ignal **S**trength **I**ndicator
RX	**R**eception
SoC	**S**ystem-**o**n-**C**hip
SPI	**S**erial **P**eripheral **I**nterface

SRAM	**S**tatic **R**andom **A**ccess **M**emory
TX	**T**ransmission
UART	**U**niversal **A**synchronous **R**eceiver-**T**ransmitter
UWB	**U**ltra-**W**ide **B**and
VCD	**V**alue **C**hange **D**ump
VHDL	**V**ery-High-Speed Integrated Circuits **H**ardware **D**escription **L**anguage
Wi-Fi	**Wi**reless **Fi**delity
WSN	**W**ireless **S**ensor **N**etwork

List of Figures

1.1 Overview of the thesis structure . . . 5

2.1 Evolution of computer classes according to Bell's law [Bel08] Wireless sensor networks emerge in the 2000's . . . 10
2.2 Generic architecture of a mote enabled by the technological trends for integrated electronic systems . . . 12
2.3 Schematic representation of the general functionality of a sensor network [GK07] . . . 13
2.4 Design of wireless sensor networks for low and high bandwidth sensing . . . 17
2.5 Design alternatives to handle high-bandwidth sensing in wireless sensor networks [Mar+06] 20
2.6 Energy efficiency, data rate and peak power consumption of selected wireless communication standards . . . 23
2.7 Typical current consumption profile of a *duty-cycled* mote [Wan+13a] . . . 24
2.8 State transition latency and power [SC01] . . . 26
2.9 Example for high-bandwidth sensing . . . 28
2.10 Schematic representation of the effect of high-bandwidth sensing and enhanced computational power on WSN metrics . . . 29

3.1 Generic cells of a reconfigurable hardware system . . . 33
3.2 Static power consumption and logic density of recent FPGA devices . . . 34
3.3 Generic architecture of a coarse-grained reconfigurable architecture (derived from [KM11]) 35
3.4 Design flow for accurate FPGA power estimation . . . 38
3.5 Power consumption breakdown of a memory-intensive application on different devices . . 38
3.6 Low-power duty cycling solutions for FPGAs . . . 39

4.1 Architecture alternatives for FPGA-based wireless sensor nodes . . . 46
4.2 Applications of FPGA-based wireless sensor nodes . . . 52
4.3 Design space of FPGA-based wireless sensor nodes . . . 53
4.4 Generic architecture of the FPGA-based sensor node . . . 54
4.5 Block diagram of the HaLOEWEn main board . . . 55
4.6 Photo of HaLOEWEn version 3 and HaLOEWEn logo . . . 56
4.7 Task distribution on HaLOEWEn . . . 58
4.8 Block diagram of the LPSIP platform . . . 59
4.9 Photos of LPSIP modules . . . 60
4.10 Setup for the measure of power consumption . . . 60
4.11 FPGA startup delay . . . 61
4.12 Concept for internal wake-up process on FPGAs with Flash*Freeze technology . . . 62
4.13 Low-level placement of ring oscillators components on an Igloo FPGA . . . 63
4.14 MCU-FPGA Interface . . . 64
4.15 FPGA design for simple FFT processing . . . 65
4.16 Comparison of different implementations for FFT processing on the wireless sensor node . 66
4.17 Board-level distribution of power consumption . . . 68
4.18 FPGA design for localization accelerator . . . 69
4.19 FPGA design for high-diffusion accelerator . . . 71

5.1 Levels of hardware dynamic reconfiguration . . . 76

5.2 Architecture of the coarse-grained overlay based on two clusters 79
5.3 Levels of reconfiguration enabled by the virtually reconfigurable architecture 80
5.4 Example of producer-consumer interconnect 81
5.5 Architecture of reconfigurable Multiply-Accumulate unit 84
5.6 Architecture of the CORDIC unit 86
5.7 Architecture of a $\{2, 2, 1\}$ ALU unit 87
5.8 Block diagram of the configuration controller 89
5.9 Threshold percentage of the number frames to reconfigure to make difference-based reconfiguration more efficient than full reconfiguration 91
5.10 Example of reconfiguration process for a filtering task 92
5.11 Task-level reconfiguration 93
5.12 Customization of the architecture template for three types of applications 95
5.13 Performance evaluation of the overlay architecture 97

6.1 General overview of the configuration tools 101
6.2 $(\text{GECO})^2$ concept for HDL parameterization 103
6.3 Screenshot of the $(\text{GECO})^2$ graphical user interface 105
6.4 Methodology for architecture and algorithm design 109
6.5 Block diagram of the middleware components 112
6.6 Event and configuration tables 113
6.7 Duration and energy costs of configuration data transmission 117
6.8 Tool-flow for generating and running hardware reconfigurable tasks on the sensor node . 119

7.1 Typical flowchart of a distributed condition monitoring application running on a sensor node (inspired from [Kul10]) 124
7.2 Overview of the MoDe condition monitoring infrastructure [Bei12] 125
7.3 Concept for transmissibility measurement 126
7.4 Transmissibility function of the damping system during the test drive [Bei+14] 128
7.5 Test measurements on a passenger vehicle 130
7.6 FPGA overlay for the condition monitoring of the shock absorber with LPSIP 131
7.7 LPSIP hardware with Continental BSZ04D wheel acceleration sensors 131
7.8 Computation of Welch's modified periodogram on the vertical acceleration time series . . 132
7.9 Time-space partitioning of the damper condition monitoring tasks on the overlay architecture 133
7.10 Breakdown of time spent for the damper condition monitoring tasks (time is given in milliseconds) 134

8.1 Spectral analysis of the induction motor currents (Part 1) 136
8.2 Spectral analysis of the induction motor currents (Part 2) 137
8.3 Pictures of the experimental setup 139
8.4 Alternative techniques for the detection of broken bars (Part 1) 140
8.5 Alternative techniques for the detection of broken bars (Part 2) 141
8.6 Concept for motor Condition Monitoring 143
8.7 Time-space partitioning of the motor condition monitoring tasks on the overlay architecture 144
8.8 Breakdown of time spent for the motor condition monitoring tasks (time is given in milliseconds) 145

9.1 Miniaturized implementation of the HaLOEWEn platform 149

A.1 HaLOEWEn schematics - FPGA page 1/2 154
A.2 HaLOEWEn schematics - FPGA page 2/2 155
A.3 HaLOEWEn schematics - Power 156
A.4 HaLOEWEn schematics - RF SoC 157

B.1 Flow chart of the *High Diffusion* algorithm 160
B.2 Results of NIST pseudo-randomness tests for the High-Diffusion algorithm 161
B.3 8-FFT decimation in frequency with perfect shuffle address pattern 163
B.4 Error correction for fixed-point FFT 165

List of Tables

2.1 Evolution of CPU specifications for microcontrollers from selected general purpose motes 10
2.2 Type of COTS sensors commonly used in wireless sensor networks applications 16
2.3 Amount of data generated by different sensors [WC02] 16
2.4 Specifications of CPUs for high-bandwidth sensing motes 18
2.5 Selected communication standards for wireless sensor networks 22
2.6 Comparative overview of energy costs for basic operations on TmoteSky[1] [Meu+08] 25
2.7 Power consumption and threshold active time for selected MCUs 26

3.1 Power consumption and threshold shutdown times for SRAM and Flash FPGAs 40

4.1 Research works using FPGAs for mote architecture prototyping 47
4.2 Research works using FPGAs as standalone processing unit 48
4.3 Research works using FPGAs for CPU extension 50
4.4 Main features of the HaLOEWEn platform 56
4.5 Main features of the LPSIP platform 59
4.6 Power consumption of HaLOEWEn in different operation modes 61
4.7 Transition metrics for the FPGA on HaLOEWEn 63
4.8 Resource consumption of the 1024-FFT design 67
4.9 Resource consumption of the localization accelerator 69
4.10 Performance evaluation for the localization process 69
4.11 Resource consumption of the High-Diffusion core 71
4.12 Performance evaluation for combined encryption and forward error correction 71

5.1 Resource utilization of selected cores for Igloo AGL1000 FPGAs 75
5.2 Configuration of the interconnect 80
5.3 Resource consumption for one cluster interconnect with identical amount of producers and consumers on each line 81
5.4 Configuration of sensor interfaces 82
5.5 Configuration of address sequencers 83
5.6 Resource consumption for one address sequencer 83
5.7 Configuration of reconfigurable Multiply-Accumulate unit 84
5.8 Resource consumption for the reconfigurable Multiply-Accumulate unit 85
5.9 Configuration of CORDIC 85
5.10 Resource consumption for the CORDIC unit 86
5.11 Configuration of the ALU array 87
5.12 Resource consumption for the ALU unit 88
5.13 Resource consumption for the configuration controller 90
5.14 Instructions of the configuration controller 92
5.15 Resource utilization of the example architectures 96

6.1 Components available in the (GECO)^2 HDL library 107
6.2 List of elementary functions available in the (GECO)^2 DSP library 108
6.3 Summary of the middleware commands 114

7.1 Resource consumption of the FPGA overlay used for damper condition monitoring 130

7.2 Performance metrics for the damper condition monitoring tasks 133

8.1 Nameplate and constructive parameters of the studied motor 138
8.2 Resource consumption of the FPGA overlay for the motor condition monitoring 143
8.3 Performance metrics of the motor condition monitoring tasks 144
8.4 Estimation results for a motor suffering from dynamic eccentricity 145
8.5 Estimation results for a motor with broken bars 146
8.6 Estimation results for a motor suffering from inter-turn short circuit 146

B.1 CORDIC modes of operation 162

Part I.

Context and Motivation for Reconfigurable Hardware in Wireless Sensor Networks

1 Introduction

1.1 Background

Hundreds of tiny electronic devices capturing physical phenomena and collaborating to monitor complex systems and interact with the environment: wireless sensor networks are an important step towards an ubiquitous world where electronics component are vanishing in our surroundings while constantly observing and evaluating them. As a result of numerous major innovations in the domains of microelectronics, communication engineering and sensor technology, the emergence of this new class of computing systems created a big wave in the scientific community by offering a new range of multi-disciplinary challenges to solve. From energy harvesting systems to routing protocols via novel integrated sensors, a whole spectrum of topics has been deeply investigated by the scientific community during the past decade with a focus on optimizing *energy-efficiency*. The restrictions on the energy budget of such platforms have created a need for very low power systems which can complete complex tasks with a minimum amount of resources. Not only the underlying technology of wireless sensor networks has attracted much interest, but also their application potential in domains as various as geology, biomedicine or agriculture. Wireless sensor networks have become a universal topic where almost all domains of engineering can find a benefit or give a new contribution.

While the technology has now reached a level of maturity suitable for large-scale commercialization as it has been proven by the success of recently founded companies such as Libelium [Liba] or Memsic [Mem], there is still room to improve the capability and the reliability of these systems for opening new application domains. Large-scale monitoring of low-rate environmental parameters such as temperature, light, humid or gas concentration are mastered with installations lasting for several years of operation using a simple battery power supply. Standardized communication protocols such as ZigBee [Zig08] have facilitated the deployment of networks in all kinds of environment, varying from mobile wearable sensors to harsh industrial constraints.

However, a problem which is often pointed out as the bottleneck of wireless sensor networks is the high energy costs of the wireless communication. Although wireless connectivity is the main powerfulness of these systems, maximizing the energy-efficiency of a wireless sensor node often requires a minimization of the radio activity. The low throughput and low range of low-power radios become particularly critical when the sensed signal is getting more complex and the amount of data to transmit is more important. While recent progress has been made for improving the energy-efficiency of wireless transceivers with the introduction of standards such as Bluetooth Low Energy [Blu10], a more viable solution for long term installations resides in distributing the analysis of sensor data among the sensor nodes and finding an optimal communication-computation trade-off.

Simple microcontrollers are however not suitable to efficiently handle complex data processing tasks. Although multi-processor systems are becoming necessary in general purpose high-performance computing, they do not scale well for very low power systems such as wireless sensor nodes. A preferable solution to minimize energy consumption is the integration of application-specific hardware accelerators. The time and energy spent for data processing tasks implemented with specialized digital logic can be reduced to several orders of magnitude when compared to pure software solutions. Nevertheless, the broad range of wireless sensor networks applications implies that a certain level of flexibility is available in the underlying hardware, so that the same infrastructure can be reused for multiple purposes. Recent advances in the design of programmable logic devices have drastically reduce the power consumption of these chips, which can be arbitrarily configured with complex data processing circuits. Reduced static and dynamic power consumption combined with high logic density make these devices a promising option for flexible hardware acceleration. Reconfigurable hardware emerges then as a potential solution to provide a sufficient level of computation power to the sensor node while leaving enough freedom for the implementation of applications in fundamentally different domains.

1.2 Research scope and objectives

This thesis investigates the potential of integrating low-power programmable logic in the architecture of wireless sensor nodes. The utilization of reconfigurable hardware is considered at several level of abstractions, from the technology choice up to the utilization in real-world applications and the development of appropriate development tools. As such, the thesis covers multiple complementary disciplines such as low-power embedded systems design, reconfigurable computing, computer-aided design, signal processing or cryptography. In particular, this thesis gives answers to the following questions:

- **What range of wireless sensor networks applications could benefit from reconfigurable hardware acceleration ?** Concrete issues with current deployments and potential improvements need to be identified. The impact of hardware acceleration on system-level metrics will be evaluated and the different types of wireless sensor nodes based on reconfigurable hardware are categorized.
- **How to include reconfigurable hardware in the architecture of a wireless sensor node while keeping a minimal power consumption ?** The power consumption overhead typically introduced by reconfigurable hardware is often seen as critical for very low-power applications. To deal with this problem, low-power design techniques are applied to define suitable modes of operation where this overhead stays negligible. The approach will be validated by building hardware prototypes implementing typical tasks from wireless sensor networks applications and taking on-board power measurements. It is foreseen to reduce the power consumption to a level that has not yet been reached by state-of-the-art wireless sensor nodes using reconfigurable hardware.
- **Which programming model is suitable to obtain the best profit from the reconfigurable hardware ?** Solutions to overcome the arduous task of application-specific hardware description are required. The multiplicity of wireless sensor networks applications implies the availability of a programming environment where the hardware and software components can be rapidly deployed. A new generic model which can be customized for the desired data processing algorithms is introduced. This model combines the design techniques for reducing power consumption with an overall flexibility enabling the implementation of a large range of data processing tasks. This level of flexibility is notably achieved by supporting dynamic hardware reconfiguration at runtime: hardware accelerated tasks running on the sensor node can be replaced and activated at any time. Special mechanisms are introduced to make this feature compatible with the restrictions induced by wireless sensor networks. This framework is supported by a special design flow reducing the overall development time and system complexity.
- **How does a generic framework for reconfigurable hardware acceleration scale with real-world applications ?** Application scenarios relevant for an industrial context are taken to illustrate how reconfigurable hardware acceleration can improve existing installations or enable new types of deployments that were previously not feasible. Even if it is not only limited to this domain, the thesis is focusing on applications related to the condition monitoring of electro-mechanical systems since they can benefit from the proposed infrastructure particularly well.

1.3 Thesis outline

To answer these questions, this thesis has been split into three main parts addressing respectively a more detailed description of the background and motivation of this thesis (Part I), the development of the set of

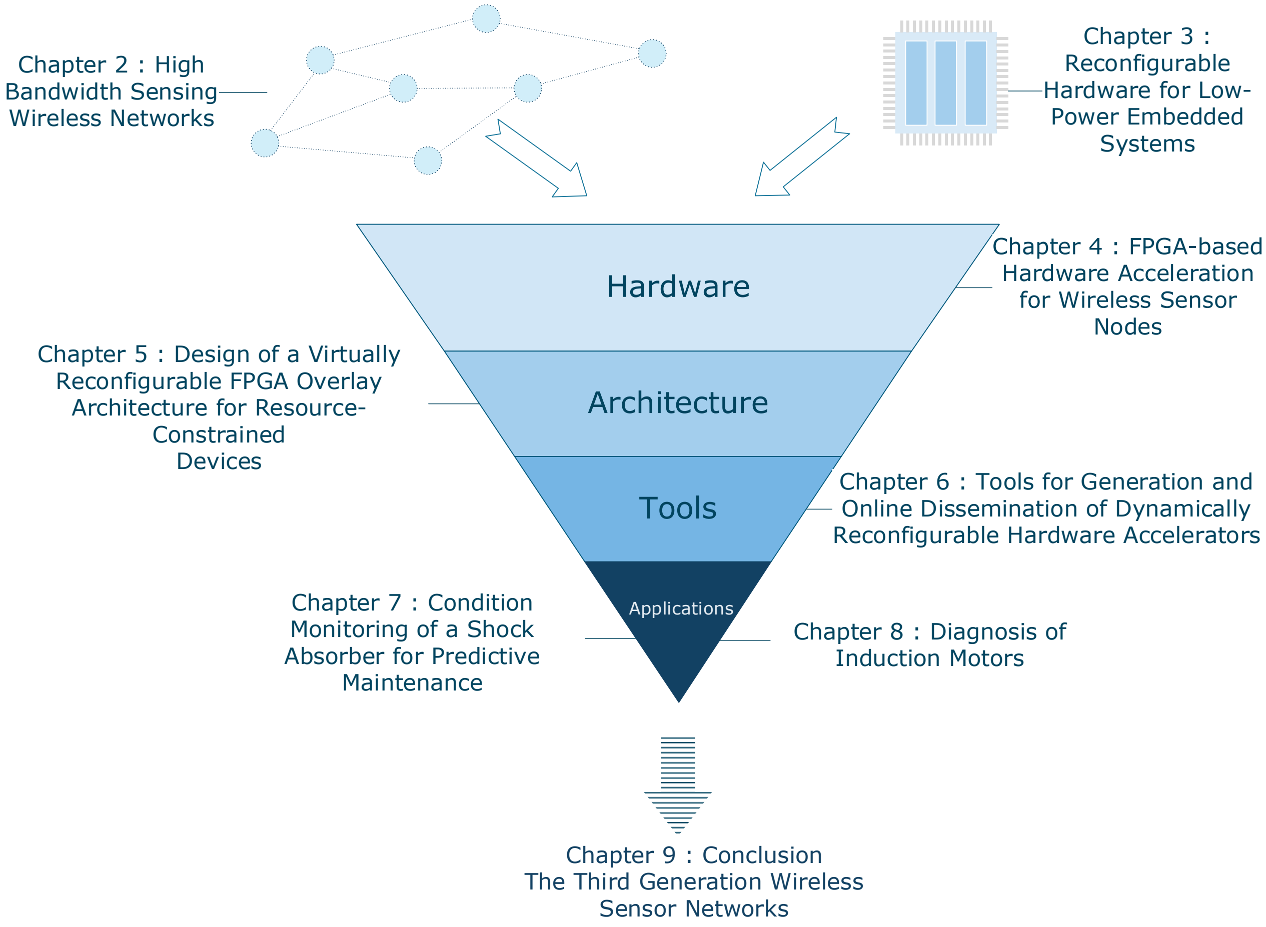

Chapter 2 : High Bandwidth Sensing Wireless Networks
Chapter 3 : Reconfigurable Hardware for Low-Power Embedded Systems
Hardware
Chapter 4 : FPGA-based Hardware Acceleration for Wireless Sensor Nodes
Chapter 5 : Design of a Virtually Reconfigurable FPGA Overlay Architecture for Resource-Constrained Devices
Architecture
Tools
Chapter 6 : Tools for Generation and Online Dissemination of Dynamically Reconfigurable Hardware Accelerators
Chapter 7 : Condition Monitoring of a Shock Absorber for Predictive Maintenance
Applications
Chapter 8 : Diagnosis of Induction Motors
Chapter 9 : Conclusion The Third Generation Wireless Sensor Networks

Figure 1.1.: Overview of the thesis structure

hardware concepts and tools supporting the usage of reconfigurable hardware on a wireless sensor node (Part II) and the application and evaluation of the complete framework to concrete scenarios. The overall structure of the thesis is illustrated by Figure 1.1: based on the different constraints and features of wireless sensor networks, a modular hardware is built as the foundation of the proposed infrastructure. This base is abstracted by a generic architecture reusable with application-specific customizations. Configuration and deployment tools are then introduced to fully exploit this architecture and the underlying hardware. Based on these components, complex applications can be efficiently implemented, deployed and evaluated.

The following paragraphs give a short summary of the chapters composing these different parts:

Part I: Context and Motivation for Reconfigurable Hardware in Wireless Sensor Networks

- Chapter 2 introduces the area of wireless sensor networks by describing its operation in a generic way. The main constraints and requirements of such systems are identified. Being the core aspect of this thesis, the chapter is focusing on state-of-the-art features related to processing and duty cycled operation. Challenges and open issues related to distributed processing and high sensor data bandwidth are emphasized.
- Chapter 3 reports recent advances in the domain of reconfigurable hardware applied to low-power embedded systems. Development and architectural challenges are identified and confronted with the ones from wireless sensor networks, notably in terms of duty-cycled operation.

Part II: Design of a Framework Enabling Reconfigurable Hardware Acceleration in Wireless Sensor Networks

- Chapter 4 describes the development of a wireless sensor node hardware with embedded programmable logic. Unexplored architectural features of state-of-the-art wireless sensor nodes based on field-programmable gate arrays (FPGAs) are identified and new platforms are designed to exploit their potential benefit. The hardware is evaluated for different scenarios typical for the considered range of applications.
- Chapter 5 introduces a template design to overcome certain technological restrictions of the proposed sensor node programmable hardware. A generic architecture and systematic mechanisms exploiting dynamic reconfiguration of coarse-grained reconfigurable operators are implemented to maximize the re-utilization of the available hardware resources while keeping a high-level of energy-efficiency. The chapter describes the different types of operators and components available to customize the hardware accelerator as well as the reconfiguration mechanisms targeting a minimization of the resource usage, reconfiguration delay and amount of configuration data. Example architectures are generated to illustrate and evaluate the template design.
- Chapter 6 presents the software development and management tools that have been developed to reduce the design time of the hardware accelerator. At first, the functionalities of a graphical interface assisting the static and dynamic configuration of the template architecture are described. A design flow based on the composition of elementary function blocks is proposed. A second software component which is running on the main processor embedded on the platform is then introduced. This component implements a set of generic services for the management of configuration data on the sensor node and within the network.

Part III: Application of Wireless Sensor Networks Strengthened with Reconfigurable Hardware to Condition Monitoring Systems

- Chapter 7 addresses the implementation of a set of signal processing algorithms with the proposed infrastructure for the condition monitoring of a road vehicle's shock absorber. Developed in the frame of a European research project with industrial partners, this chapter illustrates the capability of the proposed architecture to be adapted for scenarios with a high computational complexity and requiring a high reliability.

- Chapter 8 describes another application in the domain of condition monitoring of electrical machines. A system able to detect faults of an induction motor by analyzing features of the current signal in both frequency and time domain is implemented. The reconfigurability of the platform is exploited to analyze the sensor signals with different types of algorithms which can be dynamically selected according to the current operation of the motor. It is shown that the framework provides enough flexibility to implement algorithms producing results with high accuracy.

At last, Chapter 9 concludes this thesis by summarizing the important contributions and giving directions for future work.

2 High-Bandwidth Sensing Wireless Networks

> *All emphasize low-cost components operating on shoestring power budgets for years at a time in potentially hostile environments without hope of human intervention.*
>
> J. Hill & al., “The Platforms Enabling Wireless Sensor Networks”. In: *Communications of the ACM*, June 2004, Vol. 47, No. 6

Contents

1.1 Background 3
1.2 Research scope and objectives 4
1.3 Thesis outline 4

This introductory chapter describes the current development status of wireless sensor networks by first giving an historical perspective explaining their rising popularity. This section is followed by a description of the general operation principle of these systems. This analysis will emphasize aspects where computational power plays a particularly important role, notably in cases where an important amount of sensor data is generated.

2.1 Origins of wireless sensor networks

Different perspectives exist to explain the emergence of WSNs as a fascinating research topic in multidisciplinary domains as well as a fast growing market with a wide range of application areas. In this section, some of these views are reviewed and aspects relevant for the motivation of this work are identified.

2.1.1 Bell’s law

In 2007, Gordon Bell established his law on the evolution of *computer classes* [Bel08]. These classes are defined as sets of computing technologies in a common price range supported by “ *a programming environment* [...] *, a network, and a user interface*”. Every decade, a new class of computers appears from a fractional set of an older class, which reached a technological maturity through fast-evolving performance improvement (Figure 2.1). The newly created class benefits from lower prices and form factors, which enable a new market penetration. Thus, tiny networked sensors called wireless sensor nets appeared as a new class around 2002. They include miniaturized devices known as *Motes*. Millimeter-scaled systems known as *Smart Dust* appeared already in 2001 [War+01] but they are still costly and undergoing intensive research to develop the technologies suitable for a broader usage [Lee+13]. If production costs can be minimized, they will follow Bell’s law as the most recent computer class.

A second trend of Bell’s law is the performance improvement at a constant price level within a computer class. However, this technological evolution is slow compared to the penetration of a newly introduced class. To stay competitive and attractive, a technology must undergo constant improvement while sustaining its price range. Eventually, a class will disappear to benefit the newer class, which offers a

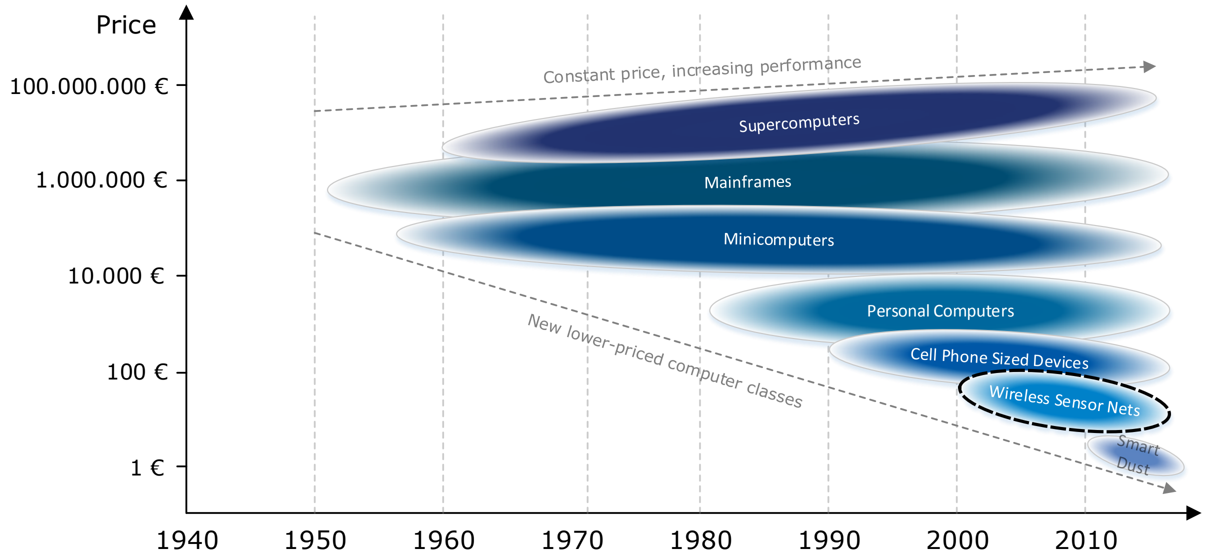


Figure 2.1.: Evolution of computer classes according to Bell's law [Bel08]
Wireless sensor networks emerge in the 2000's

broad new market. The performance improvement within the class of WSNs is one of the main aspect tackled by this thesis. Developing new generations of high-performance motes at low price will sustain the technological relevance of this computer class and eventually open directions for new applications.

2.1.2 Moore and more

The main trigger of the technological improvement within all computer classes is Moore's Law [Moo98]. The integration of a huge number of transistors in smaller silicon areas allows a simultaneous reduction of the cost and size of chips while their computational capability is improved. For WSNs, Moore's law enabled the integration of small-scaled microcontrollers able to implement complex signal processing tasks and communication protocols on a tiny, autonomous device. The progress in Moore's law will continue to reduce the size, price and power consumption of such microcontrollers while improving their information processing capability. Table 2.1 shows the improvement of central processing unit (CPU) characteristics of selected general-purpose motes. These motes are either historically relevant, or popular, commercially available devices. In one decade, each characteristic is tendentially improved.

Mote	CPU	Word Size	Flash	RAM	Clock (MHz)	Year
weC [Hil00]	Atmel AT90LS8535	8-Bit	8K	512	4	1999
Mica [HC02]	Atmel Atmega 163	8-Bit	16K	1K	8	2000
Mica2 [Mica]	Atmel Atmega 128L	8-Bit	128K	4K	8	2002
Telos [PSC05]	TI MSP430F149	16-bit	60K	2K	8	2004
TelosB [Tel]	TI MSP430F1611	16-Bit	48K	10K	8	2005
IRIS [Iri]	Atmel Atmega1281	8-Bit	128K	8K	8	2007
AVR Raven [Avr]	Atmel Atmega1284P	8-Bit	128K	16K	8	2008
Zolertia Z1 [Z1]	TI MSP430F2617	16-Bit	92K	8K	16	2010
XM1000 [Xm1]	TI MSP430F2618	16-Bit	116K	8K	8	2011

Table 2.1.: Evolution of CPU specifications for microcontrollers from selected general purpose motes

Beyond the miniaturization of digital information processors, wireless sensor networks also benefited from the technological trend known as *More than Moore* [Ard+10; Int05]. Within this movement, the increasing heterogeneity of integrated circuits (ICs) is addressed. The diversification of the types of circuits available in a single chip or package open a very large spectrum of applications. In particular two types of microsystems played a crucial role for the emergence of WSNs. Advances in low-power complementary metal-oxide-semiconductor (CMOS) radio frequency (RF) ICs made the realization of a broad family of new low-power radio transceivers possible. Technologies which are nowadays commercially successful like ZigBee or Bluetooth are the best examples of the important role that CMOS RF ICs played for this penetration. Such RF transceivers are now directly available in mixed-signal System-on-chips (SoCs). Popular examples of such chips include Texas Instruments's CC2530 [Cc2b] or Atmel's ATMEGA128RFA1 [Atm], combining respectively an 8051 central processing unit (CPU) and an AVR microcontroller with an IEEE 802.15.4 transceiver. Thanks to a tight coupling between the radio and the CPU, the power consumption and the form factor of such chips are further reduced.

Similarly, the integration of sensing elements into chips contributed to the elaboration of such heterogeneous systems. Analog-Digital Converters (ADCs) and temperature sensors are standard elements for WSN-enabled SoCs [Cc2b]. Sensors based on microelectromechanical systems (MEMS) such as accelerometers, microphones, gyroscopes or pressure sensors are widely used in WSN applications since they provide rich context about the environment at the cost of a few square millimeters. Analog Devices's ADIS16229 [Adi] is for example one of the few available products integrating a MEMS accelerometer, a signal processing digital core and an RF transceiver in a single package.

Integrated power electronics also largely contributed to the success of WSNs. Optimizing energy and power consumption on a battery-powered sensor node is one of the most critical challenges in the WSN research. Power management circuits are essential components of motes to achieve ultra-low-power consumption. CMOS RF transceivers notably require reliable and efficient power amplifiers to meet their stringent performance requirements. In some cases, energy harvesting modules can be used in synergy with the mote power unit to potentially provide a lifetime over several years of operation. The original Smart Dust project was for instance integrating a millimeter-scaled solar cell [War+01]. Motes with piezo-MEMS based vibrational energy harvesters are nowadays commercially available [Dre10].

All together, these technological advances build the hardware skeleton of a mote (Figure 2.2). Around the digital brain, sensors, RF circuitry and a power management unit enable the functionality of autonomous wireless sensing networks [Aky+02]. Integrating these elements into a single chip at minimal cost is the road-map to the RF *Smart Dust* [CLP06] as it was defined in 2000 [Rab+00] and updated in 2006 [Rab+06] by Rabaey *et al.* for the PicoRadio project. Eventually, all features of a mote will fit on a few square millimeters chip enabling a seamless and ubiquitous deployment in our environment.

2.1.3 Ubiquitous computing

A further perspective supporting the rise of WSNs is the concept of ubiquitous computing. In his article *"The Computer for the 21st Century"* [Wei99], Mark Weiser foresaw in 1999 the invisible penetration of a huge number of a new generation of computing systems in our daily life. Wireless connectivity coupled to a reduced size and cost will help to make these devices vanish in our background environment. This technology will serve our everyday purposes with specialized components without the frustration of interfacing a complex machine.

WSNs and the infrastructure known as the Internet of Things (IoT) are today the best examples fulfilling this vision. Intelligent, connected objects are completely integrated in our environments. From smart houses, to wearable computing, not only we do not notice these embedded devices any more, but their size make them unnoticeable. A large number of modern systems are automated by sensor and actuators networks, but there is still a limit for a certain range of applications whose complexity is too high to be supported by tiny embedded devices. It is then a worthy research goal to raise this limit and distribute always more intelligence in the electronic devices being part of our surroundings.

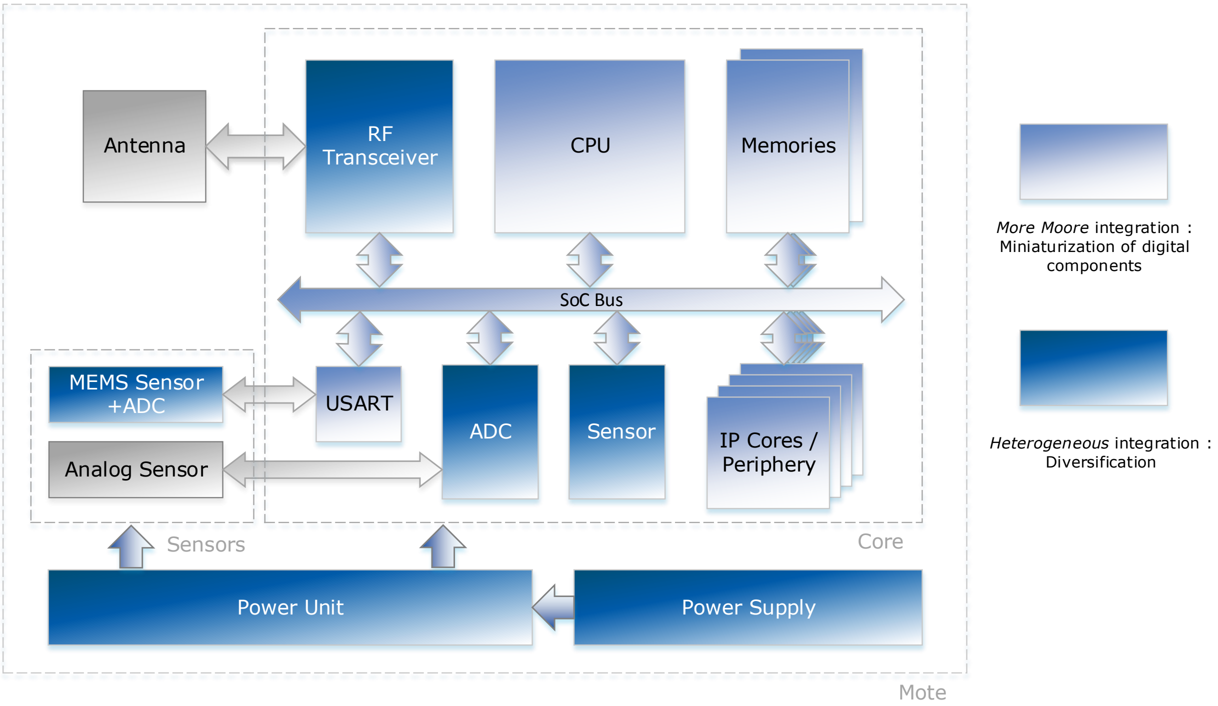


Figure 2.2.: Generic architecture of a mote enabled by the technological trends for integrated electronic systems

Making the underlying technology "invisible" is one aspect of ubiquitous computing defined by Mark Weiser that this thesis takes closely into account. Abstracting the features of the hardware and enabling the possibility to autonomously adapt or change its behavior without a manual human intervention is a key aspect for a large scale adoption of this technology.

2.2 The design space of wireless sensor networks

Emerging from these paradigms, wireless sensor networks appear as a large congregation of tiny embedded devices communicating over wireless links. The key elements of each device are sensors, a processing engine, a communication interface and a power supply unit. Each electronic unit is a *node* of the network. Low-power communication protocols and data processing algorithms are jointly used to collect and process the sensor data acquired by each node. A sensor network will then provide heterogeneous information about the physical or environmental condition of the observed system. In the simplest form of a wireless sensor network, one or several physical quantities are measured across time and space. Typical examples of physical quantities monitored by WSNs include temperature, humidity, barometric air pressure, gas concentration, light or moisture.

In this section, the general aspects of WSNs are described while emphasizing the main challenges relevant for the scope of this thesis. We start from a generic approach to go into more detailed aspects. An information processing point of view is adopted rather than considerations on communication networks as it is frequently the case in the domain of WSN, since it is fundamentally more relevant for the original purpose of a WSN. Beside the definition of founding principles, metrics and properties are introduced to enable evaluation and comparison of WSN deployments.

2.2.1 Operation of a wireless sensor network

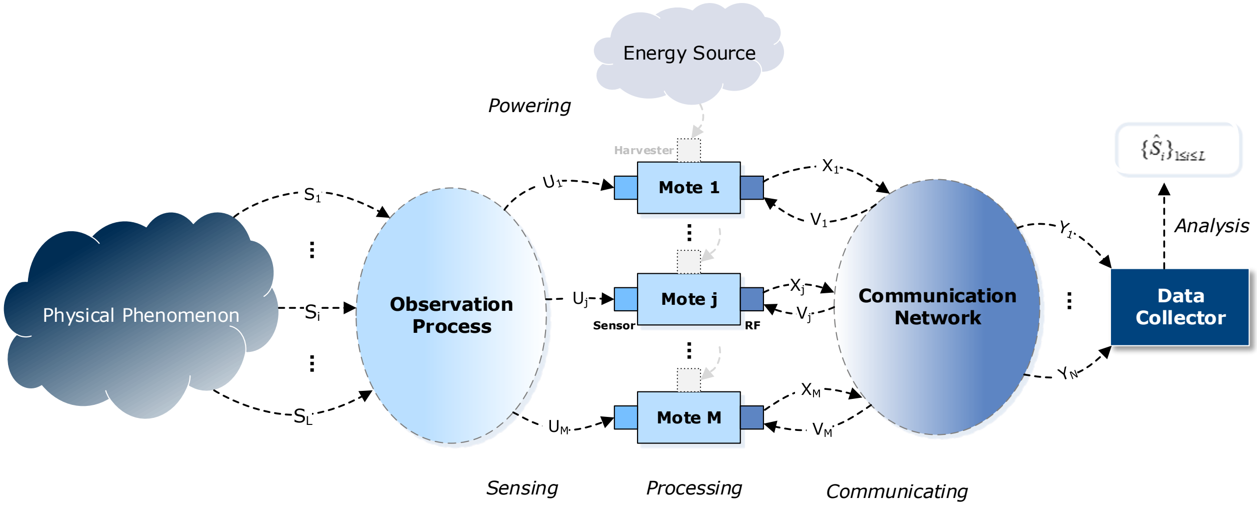


Figure 2.3.: Schematic representation of the general functionality of a sensor network [GK07]

2.2.1.1 General principle

The general functionality of a WSN can be described by the schematic representation of Figure 2.3 [GK07]. Three core aspects govern the operation of the system : *sensing*, *processing* and *communicating*. Thereafter, the role of each of these functionalities is individually abstracted. The primary target of any WSN is to observe or characterize a physical phenomenon. This phenomenon is describable by a set of variables $\{S_i\}_{1\leq i\leq L} = S$. Each variable S_i is a numerical indicator of the phenomenon or more generally represents a given state of this phenomenon. The variables $\{U_j\}_{1\leq j\leq M}$ represent the raw values *sensed* by each mote through an observation process. Each U_j is a disturbed observation of a combination of the phenomenon state variables $\{S_i\}_{1\leq i\leq L}$.

In addition to the raw measurements, each mote receives a variable set V_j including information provided by other motes of the network and application parameters. A combination of U_j and V_j is *processed* by each mote to create a measurement X_j. $\{X_j\}_{1\leq j\leq M}$ are shared within the *communication* network to collaboratively create new information. Groups of motes sharing their observations are denoted as clusters. By fusing the data, a cluster of motes will output the set of variables $\{Y_k\}_{1\leq k\leq N}$. The values Y_k are collected on a central node of the network known as the *sink*. Eventually, when information is available on a sufficiently long time, the data accumulated on the sink is analyzed to retrieve the physical phenomenon states $\{S_i\}_{1\leq i\leq L}$. The proximity between the WSN estimated $\{\hat{S}_i\}_{1\leq i\leq L}$ and the original $\{S_i\}_{1\leq i\leq L}$ defines the **accuracy** of the WSN. The closeness of these variables represent the ability of the WSN to observe the considered phenomenon correctly. In this sense, a WSN can be considered as a measuring tool, whose **approximation error** can be expressed as:

$$\varepsilon_{WSN} = ||\hat{S} - S||. \tag{2.1}$$

The optimal operation of a WSN will then allow determining all $\{\hat{S}_i\}_{1\leq i\leq L}$ at minimal costs while sustaining an accuracy fulfilling the requirements of the application. Real costs induced by a WSN are specific to each application. In general, when considering wireless communication and autonomous operation of the motes, these costs include energy consumption, infrastructure costs, delay, etc.

The physical phenomenon observed is usually changing over time. S, respectively $\hat{S}$, can therefore be expressed as a discrete sequence $S[n], n \geq 0$, respectively $\hat{S}[n], n \geq 0$, whose indexes n are increasing with time. This implies that the process of estimating $\hat{S}$ has to be repeated over time in order to follow the fluctuations of the physical phenomenon on long time periods. If $C[n]$ is defined as the cost function for estimating $\hat{S}[n]$, the general optimization problem of a WSN can be formulated as:

$$\begin{aligned} &\min && \sum_{n=0}^{\infty} C[n] \\ &\text{subject to} && \forall n \geq 0, ||\hat{S}[n] - S[n]|| \leq \varepsilon_{WSN}[n] \end{aligned} \tag{2.2}$$

As each estimation may have different costs and accuracy requirements, a WSN must be evaluated on its complete operation time, leading to consider the sum of all costs. Improving the **efficiency** of a WSN involves reducing the cost $C[n]$ of estimating $\hat{S}[n]$ for all $n \geq 0$.

The general ideal operation of a WSN is to retrieve $\hat{S}$ with the smallest approximation error while spending for example the minimal amount of energy, in the shortest time and with the minimal amount of hardware costs.

This optimization problem leads to two fundamental directions in WSN research: one one hand, the minimization of the cost function is the center of interest. The multiplicity of parameters impacting this function result in a large number of sub-problems to solve. Optimizing the operation of the communication network was a particularly important focus in academia [KW07a]. On the other hand, one wish to obtain a description as accurate as possible from the underlying physical phenomenon, *i.e.* maintaining the approximation error low [MKS12]. This approach is more application-centric but leads to an even high number of technical challenges.

2.2.1.2 System-level metrics

For all the different kinds of application scenarios that can be built on this model, these relatively abstract optimization goals can be turned into measurable figures of merits defining the quality of service of the WSN [KW07b]. These parameters are commonly used in the literature to evaluate a WSN [Xia08]. Although a concrete formulation is only possible with a given application, a generic definition can be given based on the previous model. One of the most fundamental metric is the **accuracy**. It can be defined as the average approximation error on observations made during a certain period of time:

$$\varepsilon_{WSN} = \frac{1}{m} \sum_{n \geq 0}^{m} \varepsilon_{WSN}[n] = \frac{1}{m} \sum_{n \geq 0}^{m} ||\hat{S}[n] - S[n]||. \tag{2.3}$$

The accuracy first depends on the ability of the sensing infrastructure (type, number, resolution, accuracy and location of sensors) to reflect the state of the observed physical phenomenon. In terms of information processing, the accuracy of the WSN is affected if the raw data are not directly transmitted to the sink, *e.g.* because the amount of data is too important. In this case, the accuracy depends on the ability of the processing infrastructure to extract the relevant features from the acquired data. This can vary according to the complexity of the data processing algorithms and the numerical resolution of the processing unit. Efficient processing capability plays here also a central role for the improvement of this metric.

The **delay** $d_{WSN}[n]$ of the WSN represents the time difference between the observation time of $S[n]$, $t_S[n]$, and the time $t_{\hat{S}}[n]$ when $\hat{S}$ has been determined :

$$d_{WSN}[n] = t_{\hat{S}}[n] - t_S[n] \tag{2.4}$$

The delay usually depends on the time required to process and route the sensor data to a central node. For time-critical applications, it is desirable that this delay stays minimum. Intrinsically, this delay depends on the amount of data that needs to be processed or transfered within the network. In particular, the delay is strongly related to the size of the data sets X_j,V_j and Y_j as the bandwidth and the throughput of WSN is limited. Reducing this amount of data is then a mandatory steps towards a minimum delay. On the other hand, the delay might be largely dominated by the time spent for processing if not enough computational power is available. In both cases, an efficient processing infrastructure is necessary to limit the amount of data flowing through the network and minimize computation times.

Similarly, the **time resolution** T_{WSN} of the WSN corresponds to the minimum achievable time interval between two successive evaluations of S.

$$T_{WSN} = \min_{n} \{ t_S[n+1] - t_S[n] \} \tag{2.5}$$

Here again, this metric depends on the processing capability and throughput of the network. If a node is too busy with handling data at time step n, the data corresponding to time step $n+1$ needs to be queued and processed at a later point in time. This metric can be related to the **jitter**, which is generally defined as the variation in delay [Xia08].

The WSN **lifetime** is the time during which it is possible to obtain estimations $\hat{S}$ while keeping a set of other system-level metrics over a specific threshold, *e.g.* when the WSN is not able to deliver results with a sufficient accuracy or under a certain amount of time. Typically, the lifetime of the WSN is over when nodes playing a central role in the network fail or run out of energy. In a global sense, the **energy-efficiency** of the WSN is the average amount of energy which is spent by battery-powered nodes to obtain an estimate $\hat{S}$. In order to improve both metrics, reducing the average energy consumption of each node, *i.e* the power consumption, is crucial. This requires a maximum efficiency for all components of the node, from the radio transceiver to the sensors via the processing core. More details on energy consumption models for a wireless sensor node will be given in the section 2.2.5 of this chapter.

Further metrics can be defined for a WSN [KW07b] such as **reliability**, *i.e.* the percentage of estimations staying over a given accuracy threshold, the **trustworthiness**, *i.e.* the probability that the delivered data has not been distorted by a malicious component, or the **scalability**, *i.e.* the ability to maintain a level of accuracy and a long lifetime independently from the size of the network. For almost all metrics, two fundamental aspects are playing a major role: the amount of data wirelessly exchanged within the network and the capability of the node to process information efficiently. If these features are optimized, the WSN is likely to reach a high quality level.

Now that the essential features of WSNs are defined, the following sections will address distinct aspects of WSN design, notably with regard to the key elements composing a wireless sensor node, *i.e.* the sensors, the processing unit and the communication unit.

2.2.2 Sensing

Sensors are the interface between the mote processing unit and the observed phenomenon. The sensing operation will deliver digitalized information from a physical effect. The size and price requirements of motes usually implies that low-cost, off-the-shell sensors are preferred to high-end devices. Lower quality sensor signals can be compensated by a preprocessing of the data on the mote. Operations such as de-noising and spike removal are therefore common in WSNs [GK07]. An aspect playing a fundamental role in the design of a WSN is the bandwidth of the sensing process. The amount of sensed data highly depends on the nature of the observed phenomenon and on the desired accuracy of the WSN. High amount of data will impact the type of hardware used and the approach for wireless communication since the data has to be processed and transmitted within the network [WC02].

Table 2.2 gives a non-exhaustive overview of commercial off-the-shelf (COTS) sensors used in typical WSN applications. The separation in the table marks a differentiation between general-purpose, low-bandwidth sensing devices and high-bandwidth sensing devices [Hil+04]. This distinction is taken relatively to the bandwidth of the communication network used to transmit the data. For low-power WSNs, the bandwidth of the RF channel is usually in the range of 100 kilobits per second up to 1 Megabit per second. When sensor data is sampled at a range within one order of magnitude of this bandwidth, the communication channel is likely to be overloaded and processing the data becomes mandatory.

Sensor	Sampling rate	Application example
Photodiode	0.1 Hz - 1 Hz	Intelligent light control
CMOS Light	0.1 Hz - 1 Hz	Color sensing, object detection
Temperature	0.1 Hz - 1 Hz	Environment monitoring, heat control
Humidity	0.1 Hz - 1 Hz	Environment, weather monitoring
Barometric Pressure	0.1 Hz - 1 Hz	Altitude estimation, weather monitoring
Soil Moisture	0.1 Hz - 1 Hz	Smart irrigation
GPS	0.1 Hz - 1 Hz	Animal tracking
Gas	0.1 Hz - 1 Hz	Fire detection, biometrics
Push button	0 - 1 Hz	Remote control
Infrared	1 Hz - 10 Hz	Proximity sensor, intruder detection
Capacitive touch	5 Hz - 10 Hz	Pad, human machine interface
Compass	1H z - 500 Hz	Motion recognition, orientation
Accelerometer	10 Hz - 10 kHz	Gesture recognition, condition monitoring
Piezoelectric	100 Hz - 10 kHz	Machine condition monitoring
Flex	1 Hz - 100 Hz	Condition monitoring
Gyroscope	10 Hz - 500 Hz	Motion recognition
Seismic	1 Hz - 200 Hz	Volcano monitoring
EMG electrode	100 Hz - 5 kHz	Biometrics, device control
ECG electrode	100 Hz - 5 kHz	Biometrics
Current	10 Hz - 10 kHz	Motor condition monitoring, power meter
Acoustic	1 kHz - 100 kHz	Target tracking
Ultrasound	1 kHz - 100 kHz	Localization
RF Wave	1 Hz - 1 kHz	Identification, Localization
Video Camera	0.1 Hz - 20 Hz	Intruder detection

Table 2.2.: Type of COTS sensors commonly used in wireless sensor networks applications

In this context, sensors requiring a sampling rate higher than 10 Hertz can already be considered as **high-bandwidth** sensing devices.

Figure 2.4 illustrates the two different situations where low and high bandwidth sensing is applied. In general, networks using sensors requiring high sampling rates are less dense since the data includes already sufficient information to describe the observed phenomenon. In Table 2.3, the amount of data generated by different types of sensors during one hour is given. There is almost a difference of three orders of magnitude between each category, showing that the same networking or processing approach can not be used for each of this sensor.

	Temperature	Vibration	Image
Sampling frequency	1 Hz	1 kHz	20 frames/s 25 kpixels/frame
Bit rate	7 bits/sample	12 bits/sample	8 bits/pixel
Amount of data for one hour sampling	25 kb	43 Mb	14 Gb

Table 2.3.: Amount of data generated by different sensors [WC02]

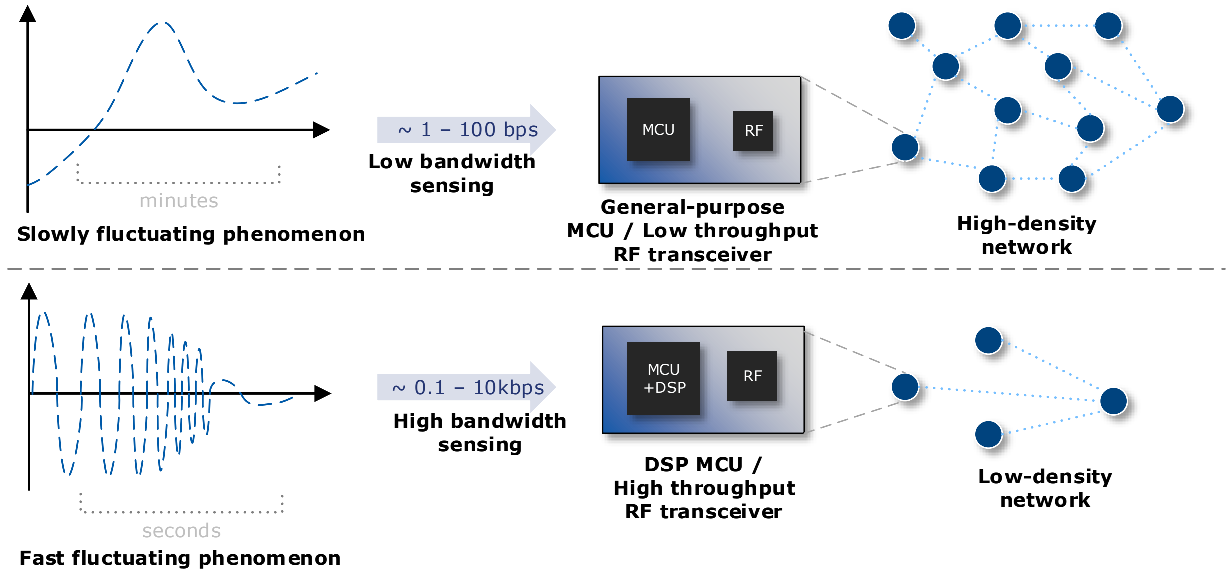

minutes
Slowly fluctuating phenomenon
~ 1 – 100 bps
Low bandwidth sensing
MCU
RF
General-purpose MCU / Low throughput RF transceiver
High-density network
seconds
Fast fluctuating phenomenon
~ 0.1 – 10kbps
High bandwidth sensing
MCU +DSP
RF
DSP MCU / High throughput RF transceiver
Low-density network

Figure 2.4.: Design of wireless sensor networks for low and high bandwidth sensing

2.2.3 Processing

The processing unit is the core of the mote and provides the necessary intelligence to process the sensor readings and transfer them to the communication network while managing the power consumption of the whole device. The main tasks of the processing core can thus be subdivided into three categories: **signal processing**, **communication protocol** and **resources management**.

2.2.3.1 Processing hardware

The choice of the processing core for a mote is a trade-off between flexibility, performance, power consumption and cost. It is also highly related to the complexity of the tasks assigned to the mote. The nature and amount of these tasks generally defines an application profile. Neither should it be oversized, which would result in low efficiency and overconsumption of energy, nor undersized, which would make it unable to perform the required tasks in a reasonable amount of time. In line with the two categories of sensing bandwidth introduced in Section 2.2.2, processor devices listed in Table 2.1 are appropriate for general-purpose, low-rate tasks. For high-bandwidth sensing, processors with higher computational power (larger word length, additional memory, larger data path) are preferred to implement data processing tasks more efficiently. Such processors usually support larger word sizes and include cores for digital signal processing (DSP). Example of such chips on selected motes are presented in Table 2.4.

Processors for WSNs rarely integrate floating-point units (FPUs). For instance, none of the processors listed in Table 2.4 and 2.1 support floating-point operations in hardware. Even if floating-point arithmetic can be emulated as a software library, this results in significant memory and processing costs to implement algorithms requiring high-precision results or numerical stability. However, processors used in WSN are often associated to a number of intellectual property (IP) cores able to support or accelerate typical tasks, as illustrated in Figure 2.2.

Among the essential components of WSN microcontroller units (MCUs), timers allow a precise schedule and synchronization of operations between motes of a network. Specialized units for standard serial communication protocols such as Universal Asynchronous Receiver-Transmitter (UART), Inter-Integrated

Mote	CPU	Word size	Flash	RAM	Clock (MHz)	Year
μAMPS [Min+00]	Intel StrongARM SA-1100	32-Bit	512K	24k	59-206	2000
iMote [Nac+05]	Zeevo TC2001P	32-Bit	512K	64K	12	2005
Stargate [Sta]	Intel PXA255 XScale®	32-Bit	32M	32K	400	2006
iMote2 [Imo]	Intel PXA271 XScale®	32-Bit	32M	64K	13-416	2008
Sun SPOT [Sun]	Atmel AT91SAM9G20	32-Bit	8M	32K	9-60	2010
Lotus [Lot]	ARM Cortex®M3	32-Bit	64M	64k	10-100	2011

Table 2.4.: Specifications of CPUs for high-bandwidth sensing motes

Circuit (I^2C) or Serial Peripheral Interface (SPI) are common for interfacing sensors or other peripheral chips present on the mote. The instruction and data memory must be sized to limit the area and the static power consumption of the chip while holding a complete operating system and large data sets.

An important feature of WSN MCUs is their ability to switch in low-power mode. As energy is often a critical cost for a mote, the MCU must be programmed to limit its period of activity as much as possible, and switch for the rest of the time in a *sleep* mode. The *depth* of the sleep mode vary according to the number of functionalities that stay active, *e.g.* timers, peripheral components or oscillators. Recent MCU support several depths of sleep in order to fit to specific application requirements. Very low power consumption in sleep mode and short transition times are as important as low power consumption in active mode. This aspect will be addressed with more details in section 2.2.5.

Developing custom processor or microcontroller architecture for WSNs is a prolific research area. New application-specific integrated circuits (ASICs) are regularly introduced with a focus on low-power consumption like the Phoenix processor, reaching $30pW$ in sleep mode [Han+09], or integrating hardware accelerators to handle networking tasks like the Charm processor [She+06] or the ASIC developed by Hempstead & *al.* [HBW11]. Even if these processors show significant improvement towards COTS MCUs and highlight architecture solutions for future WSN chips, they are rarely deployed in a large range of applications because of costs reasons, ASIC development being not affordable for the deployment of a WSN.

A reasonably priced alternative is to prototype WSN processors on reconfigurable logic, *e.g.* FPGAs. This is the solution adopted by Hinkelman & *al.* [Hin+08] and Lu & *al.* [Lu+09] to evaluate their custom processor architectures. The Hyperion [Hil10] and the HogthrobV0 [Leo07b; Leo07a] platforms use FPGAs to test, evaluate and compare different architectures. Using reconfigurable logic requires a description of the core in a hardware description language (HDL), such as VHDL or Verilog, or using *opaque* IP cores. The usage of FPGAs in WSNs will be largely covered in the chapter 4 dedicated to the design of the mote developed in the frame of this thesis.

When the computational power of the main processor is not sufficient to handle a specific task, a second processor or a co-processor is added to the mote. It can be integrated on the main board of the mote, *e.g.* the AVR Raven includes a second microcontroller (Atmel ATmega3290P) to handle sensors and user interface (LCD screen) [Avr]. In the ZebraNet application dedicated to wildlife tracking, the motes combined a Texas Instruments MSP430F149 MCU with a μ-blox GPS-MS1E chip including an Hitachi RISC CPU SH-7020 [Zha+04] for GPS localization. Alternatively, standard motes are extended with add-on processing modules, *e.g* the Mica2 mote [Mica] is extended with a Xilinx Spartan II FPGA to handle audio data processing in a counter-sniper system [Sim+04].

In general, these examples show that a general purpose processing hardware is quickly limited when the implementation of application-specific or computationally intensive tasks is needed. Specialized hardware is then required to replace or extend the main processor. This is particularly valid in the case of high-bandwidth sensing wireless networks where intensive in-mote processing is often required.

2.2.3.2 In-network processing

The main purpose of processing the data directly on the motes, denoted *in-network processing*, is to create data sets holding selective information relevant for the purpose of the WSN. Obtaining this data sets within the network must provide a cost reduction for the operation of the WSN, usually in terms of reduction of the network traffic. Using the symbols defined in Section 2.2.1, in-network processing correspond to the series of transformations of the raw observations $\{U_j\}_{1 \leq j \leq M}$ into $\{Y_k\}_{1 \leq k \leq N}$. As illustrated by Figure 2.5, two main approaches for data processing can be distinguished here:

- **Single node algorithms** cover all processes that a mote can perform without using information from the sink or from neighbour nodes, *i.e.* without using wireless communication. Examples of such algorithms include:
  - *Data characterization*: based on its own readings, a mote can polish the sensor signal and extract relevant features.
  - *Data interpolation*: the trend of the data is identified, allowing to represent the process with function parameters and predict future behaviour.
  - *Data compression*: the size of data is reduced set by eliminating statistical redundancy. *Lossy* data compression implies removing unnecessary information in addition.
  - *Difference-based approach*: only the difference between successive observations is considered. This is particularly suitable when the observations have a low fluctuation rate.
  - *Confidence interval approach*: the sensor signal is described with its statistical characteristics.
- **Distributed algorithms** implies the collection and combination of data and parameters from other members of the network. This data is denoted $\{X_j\}_{1 \leq j \leq M}$ in Section 2.2.1. Examples of such algorithms include:
  - *Data fusion*: sensor readings from multiple nodes are combined into a single data set by eliminating for example spatial or temporal redundancy.
  - *Data aggregation*: observations are reduced to statistical characteristics computed among the data sets provided by each node.
  - *Incremental analysis*: measurements from neighbors or past measurements are used to limit the computational burden of regenerating a new observation.
  - *Distributed analysis*: the analysis of the WSN data to estimate $\hat{S}$ is directly executed on the motes.

In addition to the processing of sensor data, motes may have to handle further computationally intensive tasks related to the organization of the network or to the improvement of the wireless communication. A non-exhaustive list of such tasks include:

- *Localization*: in networks where the positions of motes is unknown, a positioning process is necessary to localize the sensor readings. Such process can involve the solving of least squares problems using QR Factorization as it is the case for multilateration algorithms for example [KW07c]. Texas Instruments' CC2431 RF SoC includes a location engine to accelerate a positioning algorithm based on received signal strength indicators (RSSI) [Cc2a].

- *Cryptography*: by nature, wireless communication is vulnerable as it uses the air as a shared communication medium. In critical applications, it is therefore necessary to authenticate and encrypt packets to guarantee the security and the validity of the data. Cryptographic operations usually require a significant amount of computations and must be repeated for each packet exchange.

However, as communication standards are often associated with a predefined cryptographic algorithm, *e.g.* the advanced encryption standard (AES) for IEEE 802.15.4 [IEE11], modern transceivers integrate a dedicated hardware accelerator to handle these tasks, *e.g.* the AES engine in the CC2431 [Cc2a].

- *Error detection and correction*: wireless communication is also unreliable and packets are likely to be received with errors such as simple bit flips or burst errors. Receivers must therefore detect and eventually correct such errors. Two processes can be distinguished here:
  - Automatic repeat request (ARQ) where a node is asked to resend a packet if the receiver detected errors with a checksum algorithm. Cyclic redundancy checks (CRC) are for example used by IEEE 802.15.4 [IEE11]. SoCs such as the CC2431 [Cc2a] integrate dedicated hardware accelerators to implement them.
  - Forward Error Correction (FEC) schemes uses error correction codes to directly correct errors on the receiver side without retransmission. In [How+06], the authors compare the hardware implementation of various decoders for WSNs and stress the need for low-power consumption to make the coding process energy-efficient.

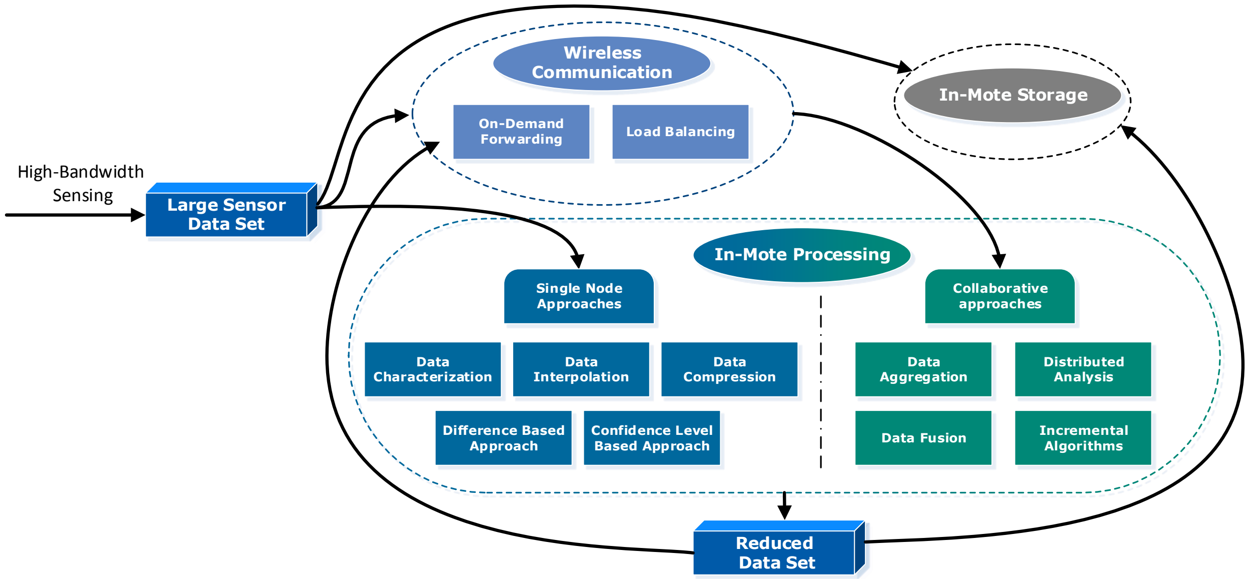


Figure 2.5.: Design alternatives to handle high-bandwidth sensing in wireless sensor networks [Mar+06]

In conclusion, the variety of tasks that a mote processor is susceptible to implement generally speaks for general-purpose processing hardware. On the other hand, the resource and energy constraints are rather promoting the usage of application-specific hardware. Solving this trade-off becomes the main challenge when designing an advanced sensor node with a good capability for application-specific customization.

2.2.3.3 Operating systems for wireless sensor networks

As autonomous and resource-constrained embedded systems, motes require a lightweight operating system (OS) for proper operation. Among the main tasks of the OS on a mote, scheduling the activity of the hardware components is one of the most important. This should be obtained by using as low memory as possible while maintaining energy-efficient operation. Several types of programming models for WSN OS can be distinguished to achieve this purpose [KW07d; FK11]:

- **Multithreading programming**: The CPU supports the execution of concurrent threads. Although this model is common in modern OSs, it can suffer from a large overhead for switching between threads on systems with low capability such as motes. Memory management is particularly challenging as each thread require its own stack. WSN OSs based on this model offer than specific mechanisms to reduce this overhead. MANTIS OS [Bha+05] and Nano-RK [ERR05] are example of such OSs.

- **Event-based**: A kernel manages the execution of pieces of code based on events. Each event is completely handled before waiting or handling the next one. This fits well to the reactive nature of motes, which typically have to react to events such as the arrival of a packet or the interrupt of a timer. Event-based OSs consume significantly less resources than multithreading OSs in terms of dynamic memory utilization and size of code [LSR03]. However, the programmability is lower since developers are more used to a sequential model of execution.

The currently most popular WSN OSs, Tiny OS [Lev+05] and Contiki [DGV04], are then inherently event-based. They however both support multithreading as a lightweight library built on top of the kernel (TOS threads for TinyOS, Protothreads for Contiki). Thus, they combine the best of both approaches.

When considering in-network processing tasks with a large computational overhead implemented on the main MCU, multithreading becomes mandatory since their large delay might block other vital functionalities of the mote such as the radio. The OS should then switch between the handling of application events and the time consuming data processing tasks, validating the choice of a mixed approach.

One of the main challenges for WSN OS is hardware compatibility or portability. They are numerous types of WSN MCUs and successfully porting an OS to a new MCU is a complex task. This partially explains why platforms to which popular OS have been ported remain used for long time periods, even if the hardware is out of date. This is for instance the case of the TelosB/TmoteSky [Tel] and MicaZ [Micb] motes, which support all TinyOS, Contiki, Mantis and Nano-RK.

Most of WSN OSs are programmed in the C language. TinyOS is based on a language called *nesC*, which reuses the basic constructs of C while offering new constructs useful to benefit from the OS features [Gay+03]. Some OSs are based on high-level programming languages or virtual machines, *i.e.* Java. This is for example the case of IBM's MoteRunner [Car+09] or Java Micro Edition for the Sun SPOT [Sun]. While such approaches are offering a better programmability, they also introduce a slight overhead reducing their energy-efficiency when compared to OSs close to the hardware [Car+11]. In general, defining a generic hardware abstraction layer (HAL) is crucial for an OS in order to improve its portability and programmability.

Besides providing a programming paradigm, OSs offer libraries of functionalities which ease the development of applications significantly. Even if a large number of libraries are available for supporting wireless communication or interfacing the mote hardware, few OSs provide support for in-network processing. For example, Contiki only includes a function to implement integer Fast Fourier Transform (FFT). It is worth highlighting this lack, since improving support for in-network processing within the OS of a mote is one of the main contribution of this thesis. Chapter 6 will develop this aspect in details.

2.2.4 Wireless communication in wireless sensor networks

The powerfulness of WSNs is their ability to remotely communicate without an underlying cable infrastructure, *i.e.* wirelessly. This is enabled by CMOS RF ICs which achieve combining very low consumption and medium data rates for short range communication. Beyond optimizations on the physical layer such as the transceiver architecture, the antenna efficiency or the modulation scheme, the efficiency of WSN communication protocols arise from *duty-cycled* operation. The energy-efficiency of a mote largely depends on its capacity to avoid idle listening, overhearing or packet collisions. In order to minimize the uptime of the transceiver, a mote should acquire knowledge about the radio activity of its neighbors

Name	Frequency band	Datarate	Max. node count	Indoor range
IEEE 802.15.4 [IEE11]	868/915 MHz 2.4 GHz	250 kbps	> 1000	50 m
Bluetooth Low Energy [Blu10]	2.4 GHz	1 Mbps	8	30m
ANT[ANT13]	2.4GHz	1 Mbps	2^8	10m
EnOcean [IEC12]	868MHz	125 kbps	2^{32}	30 m
ONE-NET[One]	868/915MHz	30.4 kbps	4096	50 m
Wi-Fi [IEE12]	2.4 GHz	54 Mbps	> 10000	100m
UWB [IEE03]	2.4 GHz	114 Mbps	245	10 m

Table 2.5.: Selected communication standards for wireless sensor networks

and coordinate the sending and reception of packets. These challenges lead to the investigation and standardization of a large numbers of medium-access control (MAC) protocols. A non-exhaustive list of popular WSN communication standards is given in Table 2.5.

IEEE 802.15.4 [IEE11] is the most used communication standard in WSNs. It is the basis for numerous routing protocols such as ZigBee [Zig08] for applications in building automation, smart energy or health care, WirelessHART [Wir] for industrial process monitoring and predictive maintenance, ISA-100.11a [Int11] for industrial process automation or 6LoWPAN [She+12] for IPv6 enabled networks (IoT). Another common standards is IEEE 802.15.1, also known as Bluetooth. The latest specification of the Bluetooth protocol (v. 4.0) notably introduced a low energy variation of the traditional standard, which tackles the high power consumption of the transceivers when transmitting at low data rates. This standard is known as Bluetooth Low Energy (BLE).

Selecting the right wireless communication protocol for an application usually depends on the size, the range and the topology of the network as well as the reliability, the delay requirements and the application throughput. Figure 2.6 illustrates the data rate of selected wireless communication standards with regard to their energy efficiency, *i.e.* the amount of energy required to receive one Megabit over the air, and the peak power consumption, *i.e.* the power consumed by a state-of-the art transceiver customized for this standard in receive mode. The energy-efficiency metric should be considered with respect to the physical layer since it does not take the protocol overhead into account. Additional costs should be taken into account when considering packet sizes, carrier-sense mechanisms or acknowledgments.

High-speed communication protocols, Wi-Fi [IEE12] and ultra wide band (UWB)(IEEE 802.15.3) [IEE03], outperform low-power standards in terms of energy-efficiency from one to two orders of magnitude. They however suffer from a very high peak power consumption, which makes them unsuitable for battery-powered devices, unable to deliver such current. ANT [ANT13], a technology specialized for fitness devices and personal area networks (PAN), and BLE demonstrate very similar performance metrics. However, they are both limited in terms of transmission range compared with IEEE 802.15.4 based networks. ANT has additional MAC mechanisms that prevent application throughput higher than 60 kilobits per second, whereas BLE has some restrictions with respect to the network topology and number of nodes being part of the network. At last, 802.15.4 based networks suffer from a low energy-efficiency and low data rates but offer more flexibility in terms of network size, since both range and maximum node count are higher.

For applications with a large throughput, BLE is a good choice when the range and the size of the network are small. When the number of nodes or the covering area increase, solutions reducing the application throughput by in-network processing should be coupled to a low-rate protocol such as IEEE 802.15.4. A further argument supporting this statement is the higher reliability requirement for high-bandwidth sensing applications. In order to improve the efficiency of a MAC protocol, *i.e.* the ratio of the

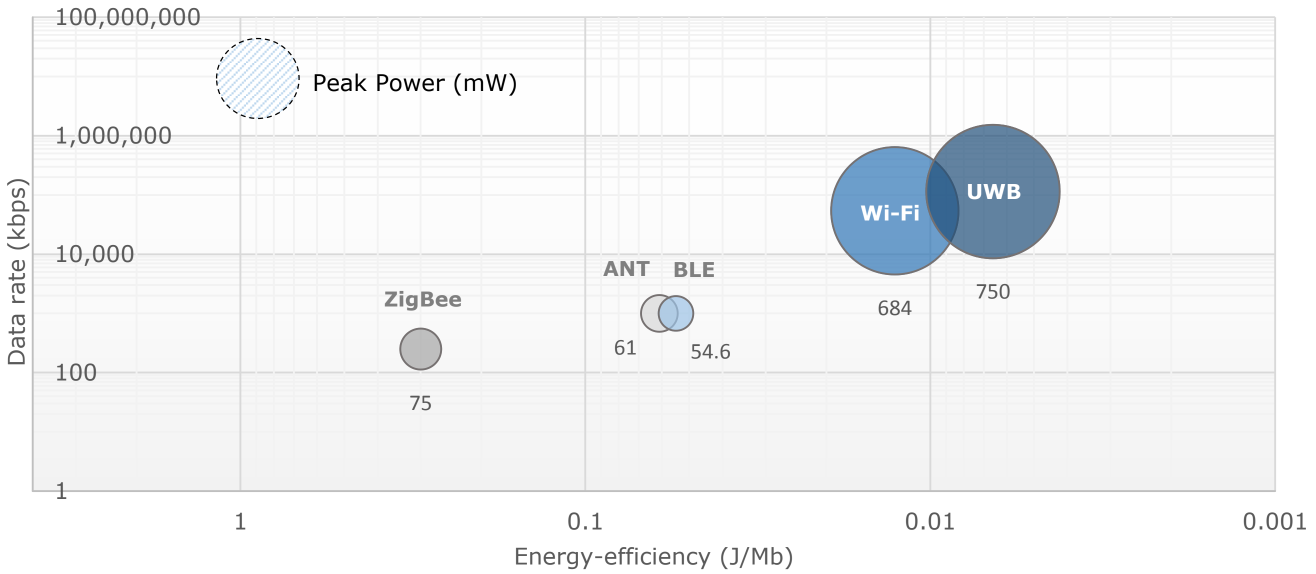


Figure 2.6.: Energy efficiency, data rate and peak power consumption of selected wireless communication standards

energy required to send the complete packet, including headers and overhead to access the link, over the energy costs required to send the useful payload alone, maximizing the size of a packet is recommended.

This efficiency will have a bigger impact in networks with high throughput requirements since the number of sent packets will be higher. Packets with larger payload are however more important at the application layer. The loss of large packets will create large gaps in the data, which is often send as streams in high-bandwidth sensing applications. While the lost of single value in a stream is usually acceptable since it can be retrieved using for example interpolation, losing a series of samples is likely to make the data inaccurate and unusable. The ANT protocol [ANT13] uses small packet sizes to avoid such losses at the cost of a lower efficiency.

These standards hide a considerable amount of MAC protocols developed in an academic context. In [Bac+10], a non exhaustive list of 75 MAC protocols are classified. The proliferation of MAC protocols can be partially explained by the variety of WSN applications, each application having various topologies and performance requirements (delay, throughput, energy-efficiency, . . .). Each protocol aims to specifically improve the performance metrics in given sectors, to fit to a particular network architecture or to fit a particular data delivery scheme.

2.2.5 Energy consumption of a wireless sensor node

A generally accepted energy consumption model for wireless sensor nodes is based on the activity periods of each subcomponent owning a significant share of the global power consumption [Dun+07][1]. It distinguishes when components are active, when they are temporarily unused and set in a low-power sleep mode, or when they are unused for a long time period and completely shutdown. More complete models include energy costs of transiting from one state to the other. This transitions periods might have a significant cost on the overall energy consumption if they are repeated too frequently [Min+01]. The energy consumption of a mote on a given time period can be expressed as:

$$E_{Tot} = \sum_{i} \left(P_i^{Act} \cdot t_i^{Act} + P_i^{Sleep} \cdot t_i^{Sleep} + E_i^{Transitions} \right) \tag{2.6}$$

---

[1] Such a model has been studied in the frame of the bachelor theses [Bol11] and [Man12a]

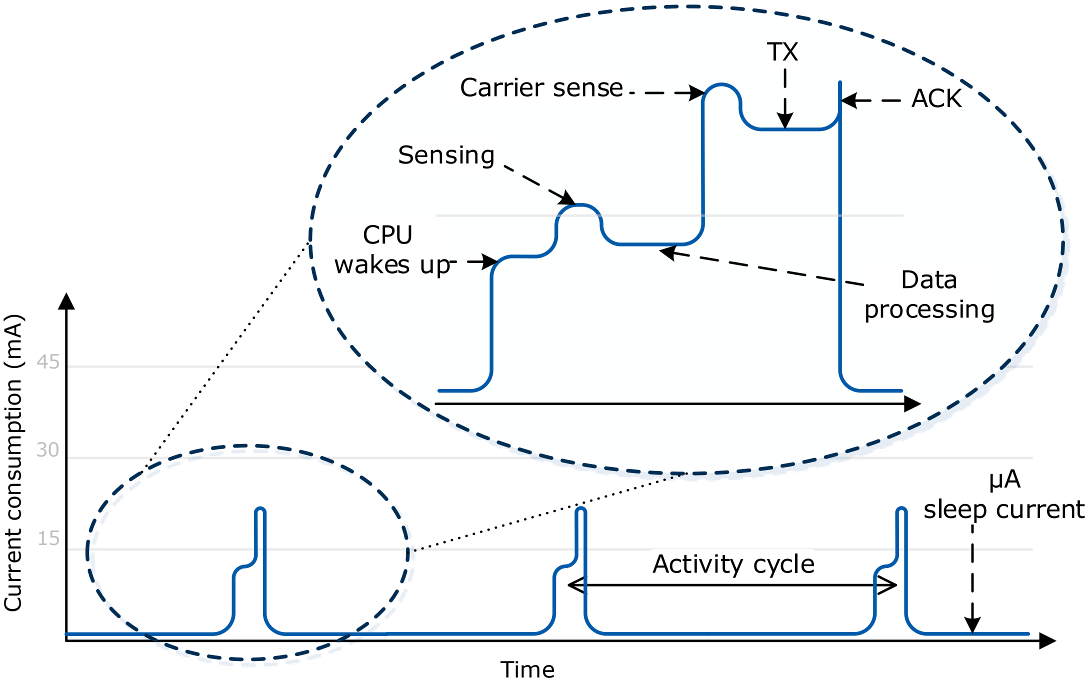


Figure 2.7.: Typical current consumption profile of a *duty-cycled* mote [Wan+13a]

with $i \in \{$ CPU, Radio, Sensor, Memory, LEDs, $\ldots\}$. P_i^k and t_i^k are respectively the average power consumption of the component i in mode k and the time spent in this mode. The energy spent for transiting between different states can be expressed as:

$$E_i^{Transitions} = \sum_{\{A,B\} \in S^2} P_i^{S_A \to S_B} t_i^{S_A \to S_B} \tag{2.7}$$

where S is the set of possible states, *i.e* active, sleep, shutdown, etc.

Individual active energy consumption terms can be further subdivided according to internal sub-states. Radio and nonvolatile memory active energy can for example be expressed as

$$E_{Radio}^{Act} = P_{TX_{0\mathrm{dBm}}} \cdot t_{TX_{0\mathrm{dBm}}} + P_{TX_{-10\mathrm{dBm}}} \cdot t_{TX_{-10\mathrm{dBm}}} + P_{RX} \cdot t_{RX} \tag{2.8}$$

$$E_{Mem}^{Act} = P_{Erase} \cdot t_{Erase} + P_{Read} \cdot t_{Read} + P_{Write} \cdot t_{Write}. \tag{2.9}$$

The superposition of these individual power consumption states results in typical current consumption profiles as depicted in Figure 2.7.

Thus, each mote can be characterized by the set of power consumption values of its individual subcomponents. Roughly evaluating the energy consumption of the mote is then reduced to measuring the time during which the subcomponents are staying in their respective sub-states. This is for instance the solution followed by the WSN energy estimation tools Energest [Dun+07] (see next subsection) or Avrora.

In general, the activity of a wireless sensor node can be described with cycles: the node is waking-up from a sleep state at regular time intervals to perform a set of tasks, including sensor sampling, processing and RF communication operations (channel sense, packet transmission, etc.) before going back to sleep. As the wake-up period is usually fixed by application and communication protocol parameters such as

Operation	Energy consumption	Equivalent CPU cycles
Compute for 1 T_{cyc}	1.2 nJ	1
Transmit 1 bit	0.72 μJ	600
Receive 1 bit	0.81 μJ	680
Listen for 1 T_{cyc}	15 nJ	13
Sleep for 1 T_{cyc}	9 pJ	7.5×10^{-3}

Table 2.6.: Comparative overview of energy costs for basic operations on TmoteSky[2][Meu+08]

sampling rate, packet size or channel check rates, minimizing the uptime of each component during the active part of the duty cycle is the main path towards lowest energy consumption.

In the context of high-bandwidth sensing, the average uptime of the sensors, processing unit, radio is intuitively a monotonically increasing function of the sampling rate when no in-network processing scheme is applied. By processing the data locally on the node, the processing time will further increase but may result in lower radio activity. On the other hand long processing times may be costly in terms of energy consumption if a significant reduction of radio activity is not achieved. Table 2.6 shows the energetic equivalent number of CPU cycles for basic operations of a wireless sensor node. This table gives a general indicator of how much computational power can be invested to save wireless communication energy.

The cost of transiting between states must be carefully taken into account as well. Intuitively, a component is not doing anything useful during these transition states. As a consequence, if the transition delay is non negligible, these transitions will have a significant impact on the overall energy consumption of the mote if they occur frequently (see equation 2.6). This is particularly relevant for MCUs: coming back from a deep sleep state requires the execution of power-up sequences where the internal voltage regulators and oscillators must stabilize. A long wake-up delay implies that there is a specific duty-cycling period for each MCU for which it is more energy-efficient to stay idle than to switch in low-power mode. This observation is particularly important for applications where high sampling rates are required since it implies that the MCU can never switch in a low-power mode, which significantly increases the average power consumption of the platform.

This problem can be formulated in the following mathematical terms [SC01]: if S_i describes the different modes of operation of the MCU, S_0 being the active mode, and S_j $(j > 0)$ the different levels of sleep supported by the device, t_1 is the time when it switches to a lower power mode and t_2 the time when it starts recovering from this mode, the total energy saved by switching in low-power mode S_k can be expressed as:

$$E_{save,k} = \left(\frac{P_0 - P_k}{2}\right) T_{S_0 \to S_k} + (P_0 - P_k)(t_2 - t_1 - T_{S_0 \to S_k}) - \left(\frac{P_0 + P_k}{2}\right) T_{S_k \to S_0} \tag{2.10}$$

Figure 2.8 illustrates these transition periods for different levels of sleep. Intuitively, the transition is only useful if the saved energy $E_{save,k}$ is positive, which results in

$$t_2 - t_1 > \frac{1}{2}\left[T_{S_0 \to S_k} + \frac{P_0 + P_k}{P_0 - P_k} T_{S_k \to S_0}\right]. \tag{2.11}$$

This last equation defines a time threshold upon which it is not energy-efficient to switch in low-power mode. Such a value can be estimated for each type of system, as long as average power consumption and wake-up delay values are available. Table 3.1 gives these threshold times for the different modes of operation of selected MCUs used in WSNs. In this table, sleep mode corresponds to the simplest low-power mode supported by the device, usually corresponding to a gating of the main clock. From

---

[2] These values are estimated for a CPU running at 4 MHz and a radio transmitting at -5 dBm

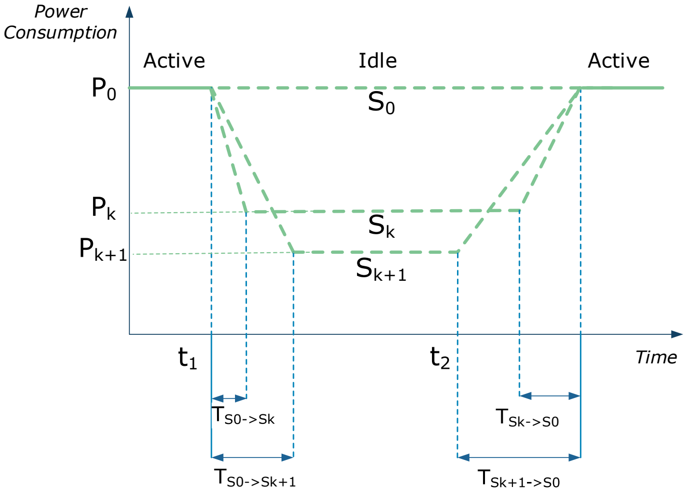


Figure 2.8.: State transition latency and power [SC01]

MCU	Mode	Power	Threshold
CC2531	Active	19.5 mW	-
[Cc2b]	Sleep	0.6 mW	$< 1\ \mu$s
(32 MHz | 3V)	Deep Sleep	3 μW	3 ms
StrongARM SA 1100	Active	1,040 mW	-
	Sleep	400 mW	8 ms
[SC01]	Deep Sleep	270 mW	20 ms
PXA271 XScale	Active	264 mW	-
(Imote2 [Imo])	Sleep	4.5 mW	1.08 ms
(104 MHz | 4V)	Deep Sleep	1.5 mW	$>$ 1.3 ms
NXP LPC1758	Active	138.6 mW	-
(LOTUS [Lot])	Sleep	6.6 mW	$< 1\ \mu$s
(100 MHz | 3.3V)	Deep Sleep	0.8 mW	$> 765\ \mu$s

Table 2.7.: Power consumption and threshold active time for selected MCUs

this mode, operation can be resumed within a very short time, usually a few clock cycles. In deep-sleep mode, the main oscillator or phase-locked loop (PLL) is stopped and must be restarted after waking-up. Although power consumption is significantly reduced, the start-up sequence in deep-sleep mode is usually much longer.

The values from Table 2.7 are showing that deep-sleep mode cannot be applied if the duty-cycle of the mote has a very short period. For instance, if a sampling frequency of 1 kHz is required, none of the reported MCUs can efficiently switch in deep-sleep mode and must stay in a state where power consumption stays relatively high. This observation in particularly important in the context of high-bandwidth sensing, since it implies that low-power sleep techniques cannot be applied when the sampling frequency increases. Not only is this observation valid for very low-power MCUs, but also for chips with larger word length or DSP extensions. In addition, these chips with higher computational capability still have a relatively high power consumption in their first level of sleep mode. Finally, the oscillator stabilization time might be rather unpredictable: time intervals from 0.5 up to 271 milliseconds can for example be expected for the PXA271 XScale [Imo].

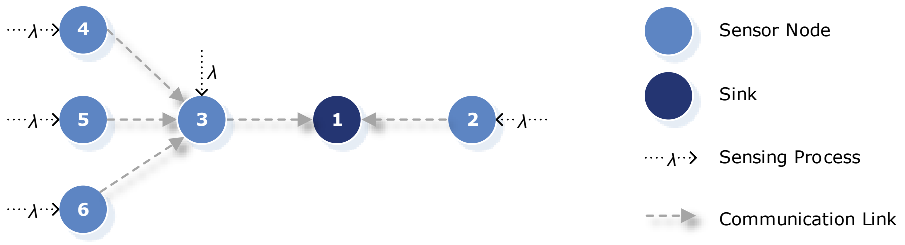
2.2.5.1 Limits of wireless communication and in-networking processing for high-bandwidth sensing on standard motes

In this section, an example is taken to illustrate the limits of a standard mote MCU and MAC protocol when dealing with high-bandwidth sensing. The topology depicted in Figure 2.9a is considered. Node 1 is the sink of the network and collects the data generated by all other nodes at a rate λ. Nodes 4,5 and 6 cannot communicate directly with the sink. Their data is therefore routed via node 3.

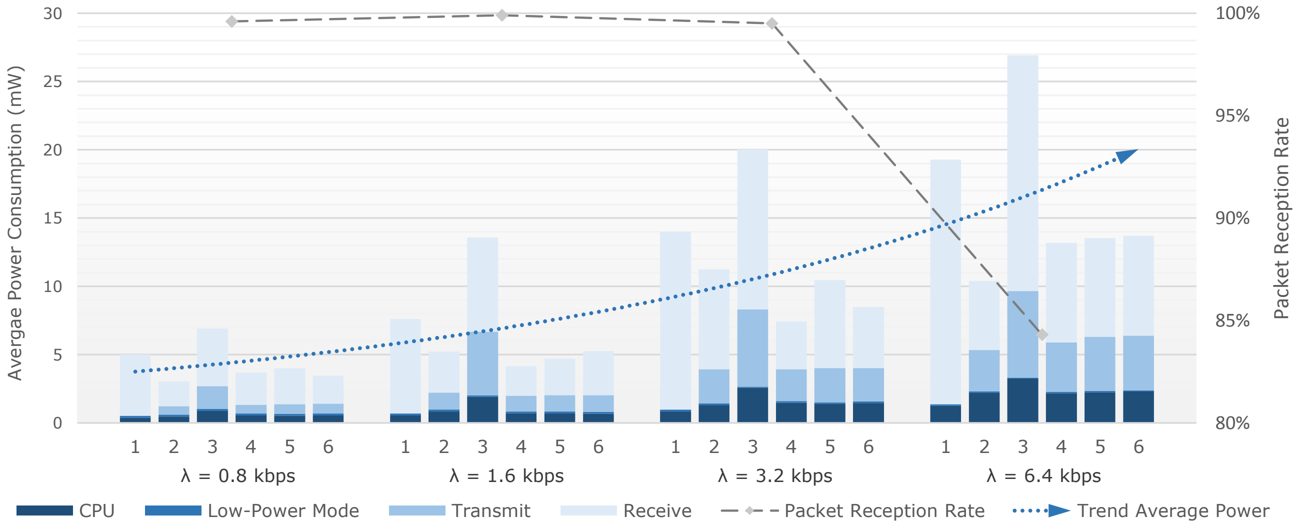
This scenario is evaluated in terms of average power consumption and reliability, *i.e* proportion of packets that successfully arrived at the sink. For this purpose, the network is simulated in COOJA, a Java-based simulation framework for WSNs using the Contiki OS [Ost+06]. Contiki integrates a software tool for online estimation of energy consumption called *energest* [Dun+07]. *Energest* measures the active periods of the motes in a modular basis. Every time a module of the mote, *e.g.* the radio, is activated or deactivated, timestamps are saved to keep track of the activity. The network is simulated with Telos-B / Tmote-Sky motes [Tel]. COOJA integrates the CPU simulator MSPSIM in order to simulate the activity of the MSP430F1611 MCU of the the Telos-B at the instruction level [Eri+09]. The communication between nodes is based on the semi-asynchronous Contiki-MAC protocol [Dun11] working at a cycle rate of 32Hz and the IEEE 802.15.4 [IEE11] physical layer. Data is clustered in packets of 100 Bytes in order to maximize the efficiency of the MAC protocol. The behavior of the network is simulated for 30 minutes of operation.

The chart 2.9b depicts the results of a scenario where the data generated by the nodes is directly sent to the sink without any processing. The energy consumption and the packet reception rate on the sink are evaluated for different data generation rates λ. Until 3.2 kbps, the network is operating in a reliable way with packet reception rates over 99%. The power consumption of the motes is increasing for both CPU and radio, with a large ratio for radio operations. Leaf nodes 2, 4, 5 and 6 have similar power consumption histograms since they have the same role in the network. Node 1 spends most of its energy for reception purposes while node 3 has the largest power consumption since it needs to receive and forward the data from 4, 5 and 6 in addition to its own data. For λ equals 6.4 kbps, the packet reception rate drops because the communication links are overloaded. The MAC protocol is not able to handle the transmission of all packets, which leads to packet drops and makes the data unusable. As a matter of comparison, a mote equipped with a 12-bit 3-axial accelerometer sampling at a frequency of 100 Hz would generate 3.6 kbps. The maximum reachable throughput with this configuration, *i.e.* the amount of data per time unit received by the sink, is then close to 25 kbps, which is about one tenth of the physical capability of the link. A large disparity can be observed between nodes 1 and 3 and other nodes. In average, they respectively consume 49% and 129% more energy than leaf nodes. At equal battery capability, this ratio is critical since the failure of node 1 or 3 will result in the failure of the complete network. The battery level of these nodes is therefore an indicator of the remaining useful lifetime of the network.

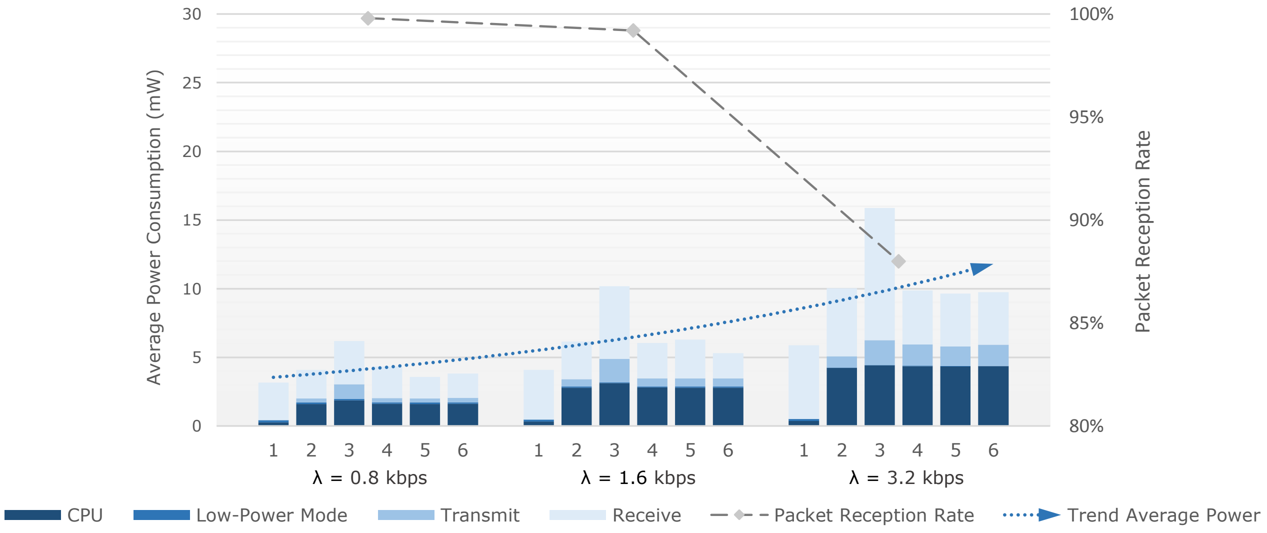
The results of a second scenario where data is processed by the motes CPU is depicted in chart 2.9c. In this case, data is processed by the 512 points integer FFT available in the Contiki OS, without overlapping window. The raw data is assumed to be real, so that the amplitude spectrum is symmetric. Therefore, only 50% of the amount of generated data needs to be transmitted over the air. In this case, the packet reception rate starts to drop for λ equals 3.2 kbps. The reason is that the execution of the FFT is keeping the CPU busy for long periods. Even if the operation of algorithm can be interrupted by multi-threading mechanisms, the CPU is unable to handle the high radio traffic and the high computational burden of the FFT at the same time. For higher λ, not only does the packet error rate (PER) drop, but the leaf nodes are unable to complete the FFT within the data generation cycle, leading to packet drops at the source. In terms of power, the average consumption is approximately the same for both scenarios with $\lambda \leq 1.6$kbps. The consumption is however better balanced in the second case where node 3 consumes 64% more power than leaf nodes while node 1 has now the lowest energy consumption. The energy spent for wireless communication in the first scenario has been converted into CPU energy. For leaf nodes, only 20% of the energy was spent by the CPU in the first scenario against 48% in the second one.



(a) Network Topology



(b) Chart for wireless communication approach



(c) Chart for in-network processsing approach

Figure 2.9.: Example for high-bandwidth sensing

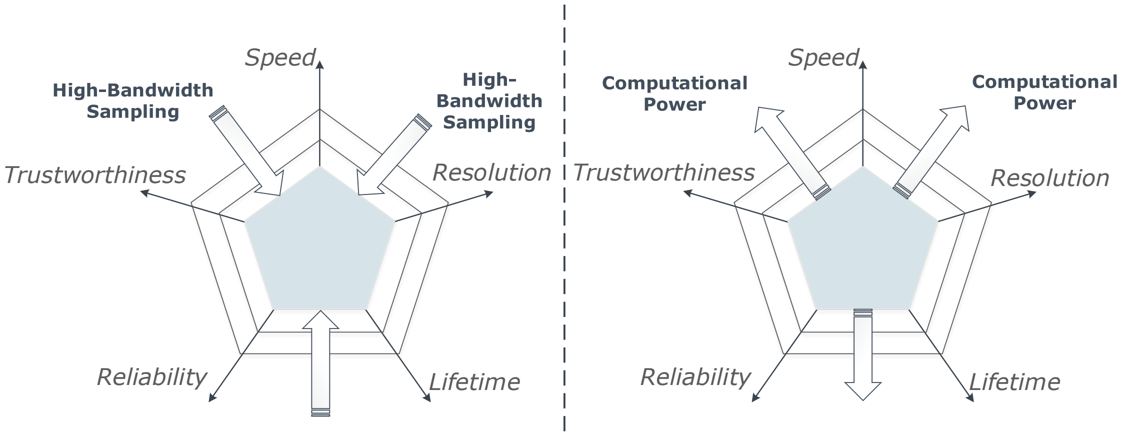


Figure 2.10.: Schematic representation of the effect of high-bandwidth sensing and enhanced computational power on WSN metrics

This example clearly shows that the main limitation of standard motes in high-bandwidth sensing application is their computational capability. By improving the speed and the energy-efficiency of the FFT processing, motes will be able to handle additional radio traffic while decreasing their energy consumption. From example 2.9c, it is also advisable to decouple data processing from the communication protocol.

Converting processing efforts for wireless communication into processing energy is not straightforward. A limited CPU can rapidly be overloaded and becomes unable to handle data processing tasks and MAC protocol at the same time. Works such as [Nac+08] have shown that more advanced chips such as DSPs achieve better energy-efficiency when the computational load is higher. However, an efficient low-power duty-cycling is hardly achievable with these devices, making them less suitable for long-term deployments.

2.3 Conclusion

Wireless sensor networks have a very large design space where a large number of parameters have a significant impact on the quality of service and costs of the system. However, a large amount of data exchanged within the network and low processing power on individual nodes have been identified as two critical aspects limiting the main system-level metrics. Either the communication channel or the processor becomes quickly overloaded and the data cannot be handled completely. This situation occurring typically when high bandwidth sensors are used, a special hardware is required to cover this range of applications. Even if DSPs or larger CPUs have been successfully introduced to improve the processing efficiency of motes, they still suffer from very large power consumption figures, which limits the energy-efficiency improvement. Raising the operating frequency of the sensor node induces high costs, which are not always compatible with the very low power consumption expected from these devices. In addition, as the sampling frequency increases, these chips must stay continuously active and will start to consume significantly more energy since power management techniques cannot be applied.

Therefore, alternative solutions must be found to increase the computational power of the sensor node while keeping its power consumption as low as possible in all modes of operation. Increasing the number of processors is not a very scalable solution as it does not solve the problem of long wake-up delays. A solution closer to the hardware has a better potential as the best optimizations and customizations can be made at this level, both in terms of power management and acceleration of computationally intensive tasks.

3 Reconfigurable Hardware for Low-Power Embedded Systems

> *"The future for embedded systems that integrate ever-increasing portions of the system is becoming reality. And the availability of FPGAs with a broad range of sophisticated digital and analog capabilities is driving that future"*
>
> Yankin Tanurhan, "Processors and FPGAs Quo Vadis?" *IEEE Computer*, November 2006, Vol. 39, No. 11

Contents

2.1 Origins of wireless sensor networks 9
  2.1.1 Bell's law 9
  2.1.2 Moore and more 10
  2.1.3 Ubiquitous computing 11
2.2 The design space of wireless sensor networks 12
  2.2.1 Operation of a wireless sensor network 12
    2.2.1.1 General principle 13
    2.2.1.2 System-level metrics 14
  2.2.2 Sensing 15
  2.2.3 Processing 17
    2.2.3.1 Processing hardware 17
    2.2.3.2 In-network processing 19
    2.2.3.3 Operating systems for wireless sensor networks 20
  2.2.4 Wireless communication in wireless sensor networks 21
  2.2.5 Energy consumption of a wireless sensor node 23
    2.2.5.1 Limits of wireless communication and in-networking processing for high-bandwidth sensing on standard motes 27
2.3 Conclusion 29

Alternatives to approaches relying on software running on a general-purpose processor which improve the performance and the energy-efficiency of a system are numerous. In terms of architecture, this enhancement is driven by a specialization for a specific range of applications. From domain-specific processors such as DSPs to application specific processors (ASICs), the design space is vast. In the end, a solution finding a trade-off between implementation flexibility, performance, costs and development time is selected. Towards this goal, reconfigurable hardware has been often identified as suitable technology, equitably balancing these characteristics [KM11; TSV07; Bob07].

Along with the *More Moore* technological trend, the number of transistors per die is constantly increasing [Int05; Moo98]. Being based on homogeneous patterns, which are copied and distributed all over the chip, programmable logic devices (PLDs) are often among the first large-scale applications of newer technology processes. This results in chips with an always higher density, where designs with very high degree of complexity can be built. On the other hand, the operating frequency of digital circuits has reached a threshold, where it is no longer scalable with the transistor density. Reaching this so-called frequency wall lead into a growing interest for parallel computing, where computation is not only distributed in time, but also in space [Sut05]. This trend is intrinsically at the advantage of reconfigurable hardware devices, which are inherently more scalable than MPSoCs or NoCs where the benefit is much more difficult to extract [Sun10].

However, this increase in size and density conducted to another limitation related to power consumption [Sut05]. Switching and leakage current is becoming so important that temperature and power are becoming critical restrictions to achieve an always increasing performance. In particular, the static power consumption, which has been often neglected in the past, is becoming as equally important as the dynamic power consumption. This problem resulted in the development of new technologies and design techniques where low power consumption was particularly relevant. This trend is however not only applicable to systems for high-performance computing, but it is also beneficial for very low-power systems as wireless sensor networks.

This chapter intends to give an overview on the main features of systems based on reconfigurable hardware. In particular, state-of-the art techniques and devices achieving very low-power consumption are identified and presented.

Section 3.1 gives a general description on current technologies based on reconfigurable hardware. In section 3.2, methods used to estimate the power consumption of programmable logic devices are described. Finally, section 3.3 identifies to what extend can devices based on reconfigurable hardware be used for duty-cycled operation as required by wireless sensor networks applications.

3.1 Features of reconfigurable hardware systems

A generic digital reconfigurable hardware system can be defined as a combination of digital logic blocks, configuration signals and an interconnection network. Different reconfigurable hardware architectures can be created by modifying the nature, the number and the organization of these elements. The basic concept of reconfigurability can be defined as the ability to modify the operation of the logic block and its connection by modifying the state of the configuration signals. In PLDs, logic cells are usually organized in a two-dimensional array surrounded by Input/Output (I/O) cells for external connectivity and specialized cores enabling heterogeneous functionalities. However, one-dimensional structures can also be used to limit the overhead introduced by a too flexible interconnection network. In the end, the interconnect of reconfigurable arrays are based on an heterogeneous mix of one and two dimensional routing networks [TSV07].

Reconfigurability is not only limited to the digital world: reconfigurable analog components can also be implemented by following similar design patterns. A combination of analog and digital reconfigurable elements create a customizable mixed-signal chip (*More than Moore* trend [Int05]). Examples of such devices among COTS include Cypress PSoCs [Pso] or Microsemi Smart Fusion [Iglb]. As this thesis is focusing on the improvement of digital computational power, reconfigurable analog parts are not considered in details, even though they could be used to implement *e.g.* pre-filtering or ADC functionalities. Therefore, the rest of this chapter emphasizes reconfigurable digital logic.

In addition to the spatial organization of logic blocks and their interconnect, reconfigurable hardware devices can be mainly distinguished by their underlying technological characteristics, their granularity, and their reconfiguration model. The following sections are investigating these aspects.

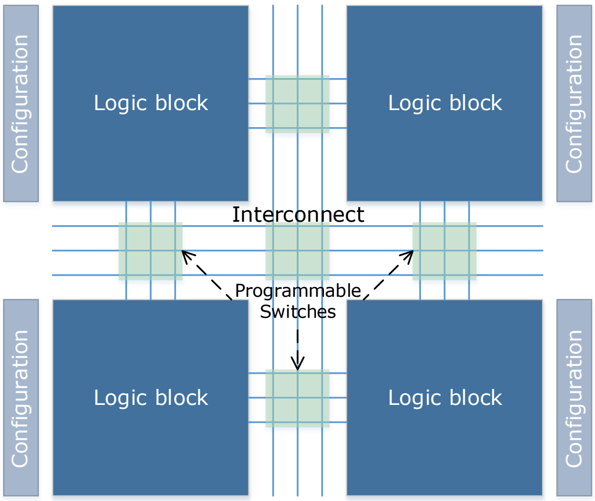


Figure 3.1.: Generic cells of a reconfigurable hardware system

3.1.1 Technology

Different PLD technologies first differ from their programming technology, *i.e.* the way the configuration signal are set. As the configuration logic may be dominating the area of the die, selecting an appropriate technology is of utmost importance for the global performance. Following technologies are usually distinguished:

- **Antifuse** technology: these devices are usually one-time programmable, but their size can be significantly reduced. The configuration is intrinsically non-volatile. Example: Microsemi SX-A family.

- **Static RAM** (SRAM): configuration information is saved in SRAM cells. This technology is nowadays the most commonly used. The device can be programmed a very high amount of times but the configuration is volatile. Example: Xilinx Spartan, Altera Cyclone families.

- **Nonvolatile** memory such as EEPROM or Flash cell: these devices combine the non-volatility of the anti-fuse technology with the reconfigurability of SRAM. The number of reconfigurations is however limited to several thousand times. Example: Microsemi Igloo family.

Certain devices combine these two technologies such as the Lattice iCE40 FPGA, which can be either configured as an SRAM FPGA or the configuration data can be loaded from a one-time programmable nonvolatile memory. As the SRAM technology is implementable with a smaller process technology than the antifuse and non-volatile counterparts, the density and performance achieved by SRAM devices is higher.

The logic density and the static power consumption of different families of FPGAs is represented in Figure 3.2. The data has been extracted from device datasheets and vendor-specific power estimation spreadsheets. The chart shows a clear trend towards higher density and higher static power consumption. However, recent devices featuring a higher amount of logic gates but lower leakage (Virtex 6 to Virtex 7 for example) are inverting the static power consumption trend. This trend conveys the fact that static power consumption has become a critical design limitation for chips with very dense logic. Much efforts have been invested to reduce this figure in the newest technology processes. At the bottom of the chart, FPGAs based on Flash technology demonstrate more than two to three orders of magnitude lower static power consumption than equivalent SRAM FPGAs. This makes this family of devices very suitable for

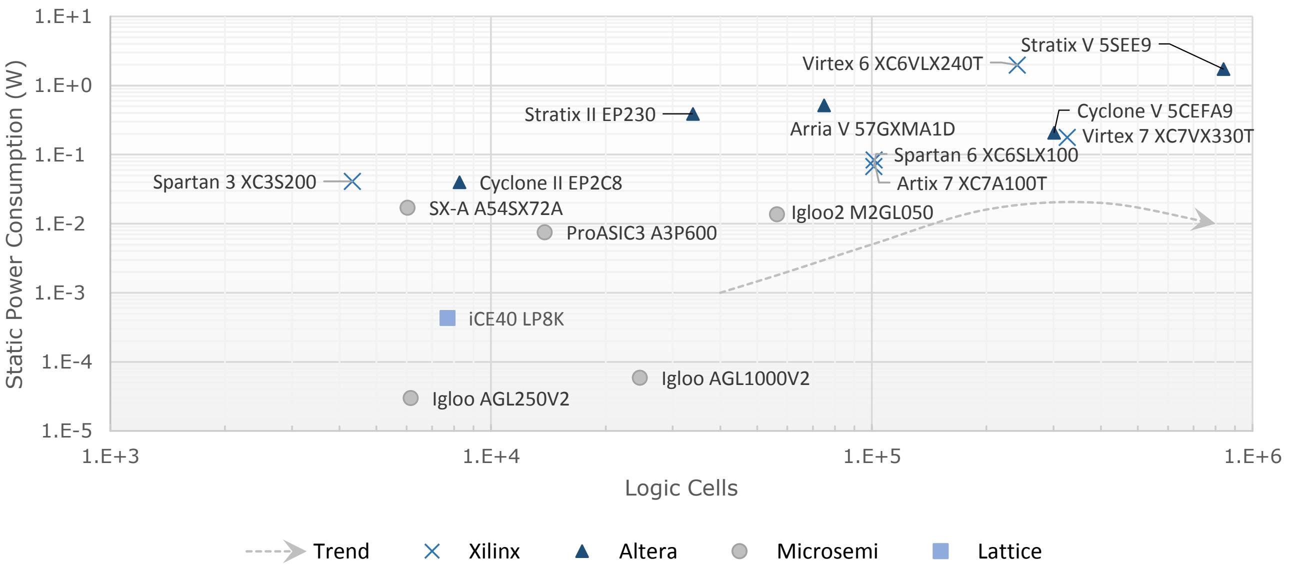


Figure 3.2.: Static power consumption and logic density of recent FPGA devices

low-power applications. However, using Flash technology results in a significant loss of performance : Igloo FPGAs are based on a 130 nm technology whereas Xilinx Spartan 6 uses 45 nm, which significantly lowers propagation delays and the overall density.

3.1.2 Granularity

The *granularity* of a PLD represents the complexity of the logic blocks forming the basis of the architecture. For instance, the base logic elements of FPGAs are implementing gate level operations (**fine granularity**) while coarse-grained reconfigurable arrays (CGRAs) are based on word operations (**coarse granularity**). In typical processing elements of CGRAs, a set of synchronous elements such as FIFOs or register files are combined with ALUs, multiplexers and simple configurable controllers [TSV07]. This contrasts with FPGA logic cells where a flip-flop is associated with a look-up table. If approximately the same amount of configuration logic is used for cells with different granularity, it can be easily deduced that a CGRA requires a significantly less amount of configuration resources than an FPGA to implement a similar functionality. However, fine granularity offers more flexibility and may implement low-level functions more efficiently. As a consequence, CGRAs are usually demonstrating higher performance for domain specific applications such as DSP or image processing [TSV07; Bob07]. In terms of power and energy efficiency, CGRAs are dominating FPGAs in their specialization domain since their overhead for reconfigurability is much lower [KM11].

CGRAs have emerged at the end of the 90's as a potential solution to accelerated patterned signal processing tasks in multimedia and cryptographic systems. Since then, a large number of architecture variants has been described in the literature. In general, all designs are derived from the generic architecture depicted in Figure 3.3. An array of interconnected processing elements configurable at the word level are used by a main processor to accelerate performance-critical tasks. Thanks to a low number of reconfiguration bits, the configuration of the CGRA can be hold in a single register. Even multiple configurations bitstreams called contexts can be hold in underlying registers for configuration switching within a few clock cycles. The flow of these configuration contexts is controlled by a sequencer, which selects which configuration should be currently active.

Among the most popular and most complete CGRAs in terms of design flow, one can cite the **ADRES** (Architecture for dynamically reconfigurable embedded systems) framework [Mei+03]. A CGRA is tightly integrated into a VLIW processor in order to reduce the communication overhead induced by traditional

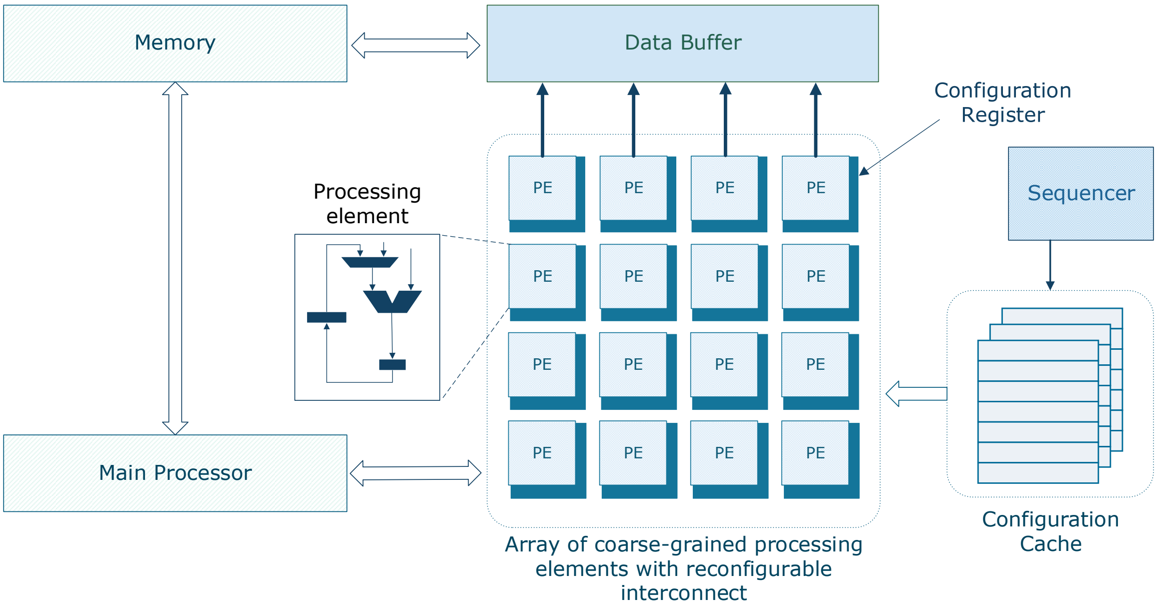


Figure 3.3.: Generic architecture of a coarse-grained reconfigurable architecture (derived from [KM11])

interaction with a RISC processor. This combination leads to a simplified programming model and a better support to describe parallelism. The architecture is further supported by a complete tool flow [Bou+07] where trade-offs between energy consumption and performance can be found. Co-developed with the IMEC research institute, this CGRA is targeting high performance multimedia and digital signal processing applications.

The Pact XPP (eXtreme Processing Platform) [Bau+03] is another CGRA template targeting high-performance DSP applications. Pact-XPP utilizes complex mechanisms to handle packet-like data structures and sophisticated configuration scheduling protocols. The architecture is also supported by a compiler to map C code to the processing elements.

RaPiDs (Reconfigurable pipelined datapaths) [EGF96] are one-dimensional CGRAs with multiple functional units communicating through a simple programmable interconnect. Data caches and complex crossbar switches are not used in order to save area and reduce processing delays. This simplification has an impact on the flexibility of the architecture, since tasks with irregular address patterns or complex control flows will not map well on the architecture.

Another type of reconfigurable datapath is implemented by the **PipeRench** architecture [Gol+99], which is composed of successive pipelined stages. A virtualization layer splits a static data flow configuration into smaller pieces corresponding to individual pipeline stages. The functionality of each stage can be reconfigured at runtime according to a time and space multiplexing process.

The **Pleiades** is another architecture template where heterogeneous satellite processing elements are connected to a main processor [Wan+01]. Low energy consumption is achieved by mixed granularity (satellites can be embedded FPGAs or ASIC modules) and application-specific customization. The overhead introduced by the interconnection network is reduced by using a mesh structure. The Pleiades template is customized for an application by analyzing C and C++ software code.

Despite their inherent advantages compared to FPGAs for domain-specific computing, most of pure CGRAs are still undergoing research for the optimization of architectures and development tools. Only a few architectures such as the ADRES and Pact-XPP have been already integrated in chips for commercial applications [Mei+03; Bau+03]. For very low-power applications, the suitability of CGRAs with regard to acceleration of data processing tasks has been demonstrated in previous works [Kim+12; Hin11]. However, these already specialized CGRAs often lack a certain level of flexibility, which make them unsuitable for large-scale tape-out. This significantly increases the price of single chips implementing this technology and make them unsuitable with the low-priced feature of wireless sensor nodes.

In order to profit from both fine and coarse granularity, recent reconfigurable hardware systems tend to mix them into heterogeneous architectures. For instance, Xilinx Virtex-7 FPGAs combine generic programmable logic with DSP slices, Gigabit transceivers, block RAMs, PCIe interfaces, high speed memories interfaces, ADCs, etc. An even higher level of granularity can be achieved by directly integrating CPU cores in the architecture. Typical examples include Xilinx Zynq devices, Microsemi Smart Fusion 2, or Altera Stratix 10. The family of Cypress PSoCs is also excellent example of devices with mixed granularity [Pso]. A CPU core is combined with PLD macrocells (fine granularity) and configurable datapaths (coarse granularity). The low amount of configurable blocks available in these chips makes only the implementation of elementary functions possible. This includes low-level communication interfaces and simple data processing tasks, *e.g.* filtering. This aspect restricts the suitability of the PSoC to implement custom hardware accelerators or complex data processing blocks.

3.1.3 Reconfiguration processes

Different approaches can be adopted to reconfigure the functionality implemented by the hardware system. Except one-time programmable PLDs, all reconfigurable systems support **static** reconfiguration. The **configware**, *i.e.* the data stored in the cells' reconfiguration memory, is loaded into the device during system initialization. The data is read from an external source such as a PC or a memory chip through a programming interface such as a vendor-specific programming device or a specialized programming chip. The duration of this process depends on the size, the architecture and the technology of the device. Static reconfiguration of Flash-based PLDs takes a significantly longer time than SRAM FPGAs because of the delay induced by the erasing and writing of Flash cells. Configuring SRAM FPGAs is completed in the order of hundreds of milliseconds while it takes up to several minutes to program Flash FPGAs [Igla; HWH12].

In **dynamically** reconfigurable systems, the reconfiguration process is part of the application and can be triggered at runtime by a configuration controller in response to internal or external events [RM10]. In general, dynamic reconfiguration is applied when the resources of the device are not sufficient to hold accelerators for each functionality required by the application or when the desired functionality was unknown at programming time. The PLD is in these cases a customizable hardware accelerator, which can be adapted *on-demand* to application requirements. This approach requires additional control logic to be implemented in a dependable fashion. The reconfiguration process includes access to the configuration memory, which can only be done when the device operation is suspended.

A subclass of dynamically reconfigurable systems comprises PLDs supporting **partial** dynamic reconfiguration (PDR). Here, only a specific part of the configware is modified during the reconfiguration process. This implicitly reduces the size of the configware and the reconfiguration time. A further advantage is the possibility to leave the rest of the device active. The operation of active logic is not interrupted, which may improve the overall speed of the design when compared to full reconfiguration. This enables **self** reconfigurable systems where the PDR control logic is implemented as a static part of the device itself. On the other hand, PDR implies stringent restrictions in terms of resource usage. A partially reconfigurable hardware block is limited to a predefined area of the device, which gives less flexibility for placing and routing the design [RM10].

Another notable approach is multi-context reconfiguration. Systems supporting this type of reconfiguration holds within each cell multiple versions of the configuration data while only one of them is active. The FPGA functionality can thus be modified in a very short amount of time by simply multiplexing these configuration vectors. Designs implementing this solution are running in a time-multiplexed manner with very short activity periods. Commercial devices implementing this functionality include Tabula's ABAX2P1 with the so called Spacetime architecture. Most of CGRAs are also based on this type of reconfiguration process since they have lower requirements than FPGAs in terms of configuration memory, which drastically reduces the overall overhead.

During the reconfiguration process, the system is usually not able to do anything useful, so that it can be considered as a pure loss when estimating the power and energy efficiency of the system. Multi-context approaches accelerate this task but have a large overhead in terms of resources utilization whereas classical reconfiguration is slow but needs only a minimal amount of additional logic. Finding a good trade-off depends mainly on how frequently the architecture needs to be reconfigured and the time criticality of the application. In general, frequent reconfiguration is only suitable if a significant gain in terms of processing efficiency is reached.

3.2 Estimating the power consumption of reconfigurable hardware devices

Precisely estimating the power consumption of a reconfigurable hardware device is an arduous task because of the heterogeneity of the components, the size of the chip and the high dependency with the input data. In general, the power consumption P_{Total} of a reconfigurable hardware device can be split into static (leakage) and dynamic (switching activity) power as [Igla]

$$P_{Total} = P_{Stat} + P_{Dyn}. \tag{3.1}$$

Due to the heterogeneity of FPGA architectures, static and dynamic power consumption can be further split into component-based values. This includes the core logic and routing resources but also I/O banks, individual pins, block RAMS and other coarse-grained elements included in the architecture.

$$P_{Stat} = P_{Stat_{Core}} + P_{Stat_{Bank}} + P_{Stat_{IO}} + P_{Stat_{RAM}} \cdots \tag{3.2}$$

$$P_{Dyn} = P_{Logic} + P_{Routing} + P_{RAM} + P_{Clock} + P_{IO} + \ldots \tag{3.3}$$

Static power is mainly coming from sub-threshold conduction and tunneling currents such as

$$P_S = I_S V_{DD} \tag{3.4}$$

where V_{DD} is the supply voltage and I_S the static current. It the static current globally depends on the size of the device, other parameters such a supply voltage and ambient temperature also have a significant impact. In general, it is hardly possible to reduce the static contribution without an intervention at the technology level [RCN96]. On the other hand, the dynamic power consumption of a component is related to the switching activity of the underlying transistors. This power consumption can be expressed as

$$P_D = \alpha C V_{DD}^2 f \tag{3.5}$$

where C is the load capacitance of the circuit, f the clock frequency and α the activity factor of the gate, *i.e.* the probability that the transistor will switch during a clock cycle. α is a parameter which is very difficult to estimate precisely since it depends on the data itself (repartition of 1's and 0's), on the circuit architecture and on the upper level operation activity. Spurious logic transitions may also occur within a single clock cycle due to different propagation times in the signal paths.

The dynamic power consumption adds up for each basic component of the architecture. Therefore, the fastest way to obtain an estimation of power consumption is to give a rough approximation of the global switching activity and the resource usage of the device. FPGA vendors provide power estimation spreadsheets based on this concept to obtain these values. Nevertheless, this method is often considered as inaccurate because of the erroneousness of the switching activity and the absence of a precise estimate of the interconnect contribution.

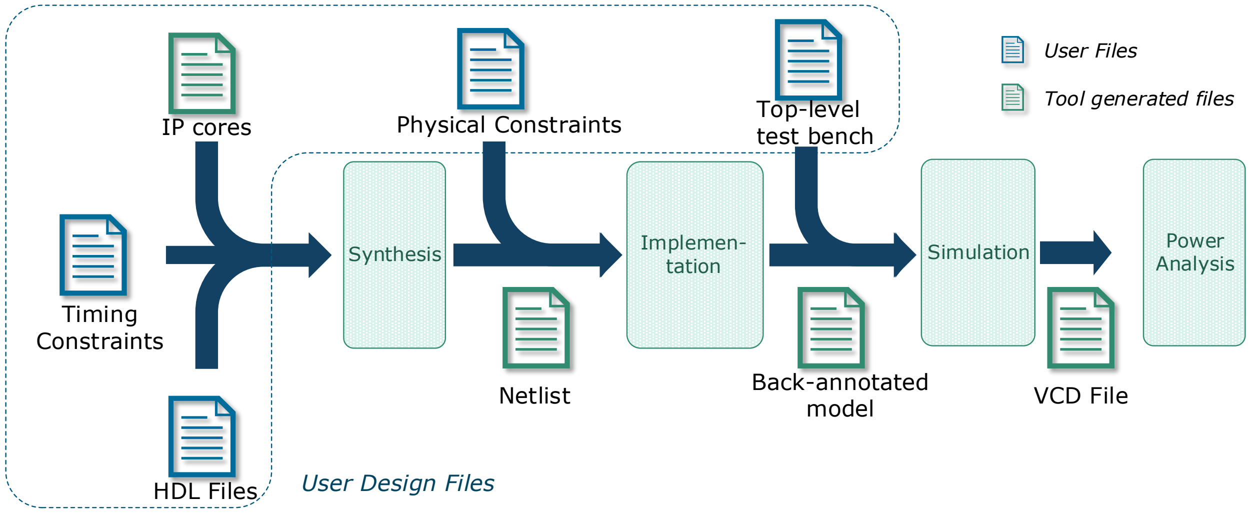
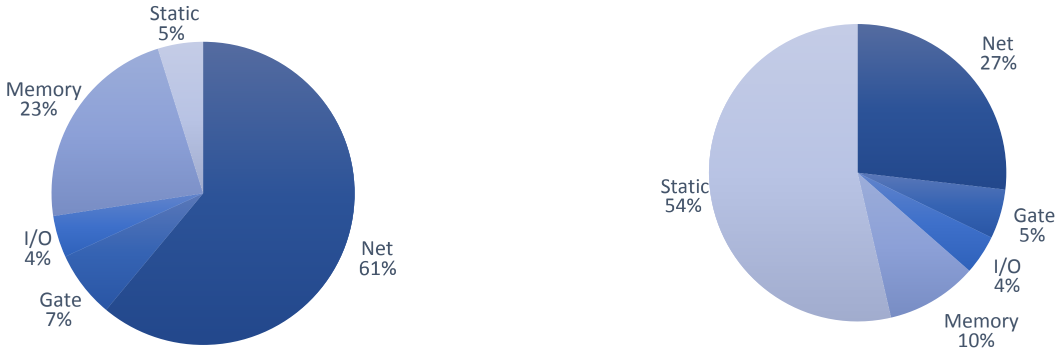


Figure 3.4.: Design flow for accurate FPGA power estimation



(a) Igloo FPGA

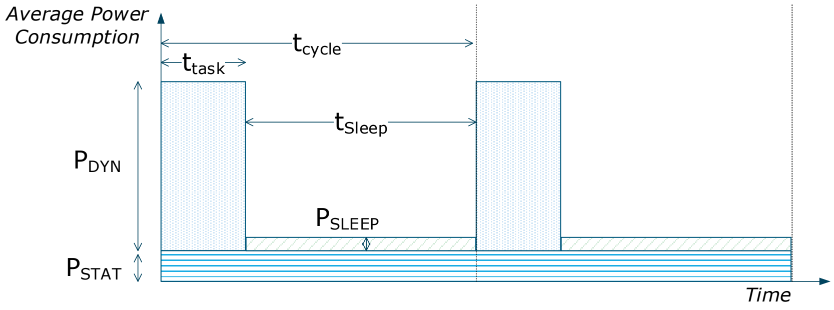
(b) Spartan6 FPGA

Figure 3.5.: Power consumption breakdown of a memory-intensive application on different devices

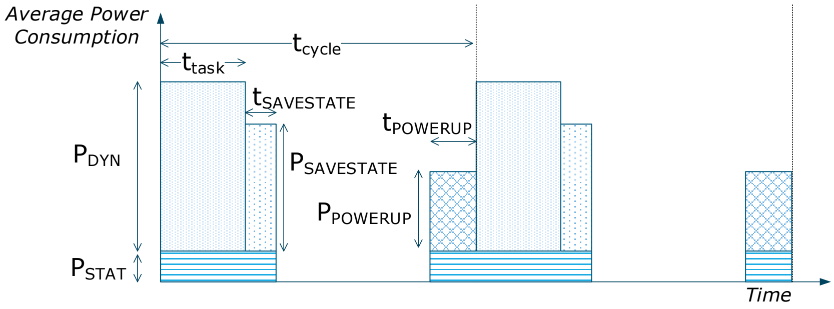
A more accurate estimate can be obtained with a post Place & Route simulation. Indeed, once the position and the functionality of each logic cell is known, the activity factor can be precisely estimated by counting the numbers of transitions induced by injecting stimulus data in the FPGA model. If the simulation model is sufficiently accurate in terms of architecture and propagation delays, a more reliable estimate can be obtained. This functionality is usually included into FPGA development environments (Xilinx XPower, Microsemi Smart Power, etc.). The complete flow using this method is illustrated by Figure 3.4.

When looking at a detailed breakdown of the power consumption on an FPGA as shown with a typical example in Figure 3.5, the most important share of the dynamic power comes from the interconnect. The consumption of the functional logic only represents a tenth of the consumption from the interconnect, which is one of the main reason why ASICs achieve about one order of magnitude better energy-efficiency than FPGAs. Similarly, CGRAs achieve better power efficiency because less resources are required to route the data signals, which are more constrained. When comparing the Flash-based Igloo FPGA with the SRAM-based FPGA, it can be noticed that the latter has a significantly higher share for static power consumption. Because of the low clock frequency considered for this test (10 MHz), the static power consumption is even dominating.

Dynamic power consumption is however only relevant when the device is active and processing a task. During inactivity times, power saving schemes must be introduced in order to save energy.



(a) Sleep



(b) Shutdown

Figure 3.6.: Low-power duty cycling solutions for FPGAs

3.3 Low-power duty cycling for FPGAs

In general, reconfigurable hardware devices achieve better energy efficiency than MCUs for the execution of a task because they allow an implementation with custom logic which is significantly faster than software, *i.e* the device stays active for a shorter time. However, this gain must be counterbalanced with the energy spent after the execution of the task, *i.e.* when the device is idle and waiting for a new task. This problem can be modeled by a duty-cycled activity similar to the one of typical WSN applications (see section 2.2.5).

In general, two different scenarios can be considered:

- In scenario 3.6a, the device goes into a low power sleep mode where the clock to the core of the design is gated. Optionally, a minimal part of the device can stay active to monitor the arrival of a new task (timer, interrupt handler, etc.). The dynamic power consumption of the device is then minimized but the static current must still be taken into account. The internal state of the logic and memory stays unchanged. In this scenario, the normal operation of the device can be resumed in almost no delay. Flash FPGAs such as Igloo support this mode inherently and have a dedicated pin to activate it. During this so called *Flash*Freeze* mode, the I/Os of the device are *frozen*, so that no signal, in particular the clock, is driving the logic within the FPGA. For SRAM FPGAs, the supply voltage can usually be lowered down until a threshold where the internal state of the registers and block memories is kept. Hybrid devices such as PSoCs support low-power modes with configuration retention but lost of the internal logic state [Pso]. This implies that data processing tasks must be fully executed during each activity cycle. In addition, these devices have a wake-up delay from sleep in the range of 200 microseconds before they are operational.

- In scenario 3.6b: the supply voltage of the device is shutdown, effectively zeroing both static and dynamic power consumption. As a consequence, the internal content of the registers and memories

FPGA	$P_{POWERUP}$	$t_{POWERUP}$	P_{STAT}	Threshold
Xilinx XC6SLX150 (Full)	2.4 mW	1.044 sec	61.2 mW	1.085 sec
Xilinx XC6SLX150 (Compressed)	4.8 mW	127 msec	61.2 mW	137 msec
Microsemi AGL1000V2	0 μW	315 μsec	112 μW	315 μsec

Table 3.1.: Power consumption and threshold shutdown times for SRAM and Flash FPGAs

is lost. If this data is critical, the device must undergo a preliminary save-state sequence where the important information is transfered to a nonvolatile memory. Similarly, this data must be restored when the device is powered on again. More importantly, the configuration data of SRAM FPGAs is lost when powered down. This implies that the complete stream of configuration data must be reloaded prior to resuming operation. Furthermore, additional delays caused by clock or voltage regulator stabilization must be taken into account.

Intuitively, the second scenario is only efficient if the down time is sufficiently long. A threshold time for which this solution is preferable to the sleep mode can be established in a similar way as it has been done for an MCU in section 2.2.5: the energy spent during the save state and the recovery can be considered as pure overhead. The energy saved in shutdown mode compared to the energy spent in sleep mode can then be expressed as:

$$E_{Saved} = (P_{SLEEP} + P_{STAT})t_{SLEEP} - (P_{SAVESTATE} + P_{STAT})t_{SAVESTATE} - (P_{POWERUP} + P_{STAT})t_{POWERUP}. \quad (3.6)$$

This results in a threshold time of

$$t_{SLEEP} > \frac{(P_{SAVESTATE} + P_{STAT})t_{SAVESTATE} + (P_{POWERUP} + P_{STAT})t_{POWERUP}}{P_{SLEEP} + P_{STAT}}. \quad (3.7)$$

In order to evaluate this threshold time, it is important to analyze the different delays and current draws for different types of FPGA technologies. For simplification, it is considered that only the reconfiguration process is required during the power-up sequence and that no data need to be saved or restored. If it is the case, the FPGA technology plays here only a minor role and no notable difference will emerge. Table 3.1 gives estimates of this threshold time for a Xilinx Spartan6 FPGA and a Microsemi Igloo FPGA. Values for the Xilinx Spartan 6 are extracted from the work realized in [Lom+12] where such a scenario is considered. Reconfiguration times is given for reconfiguration with a full and compressed bitstream. For the Igloo FPGA, no power-up sequence is required, so that the device is almost instantaneously ready to operate. A value of zero is then considered for $P_{POWERUP}$, although the current is progressively increasing to the active value. The power-up time corresponds only to the startup delay of external components (power gate, oscillator) and internal regulators.

Switching off SRAM-Based FPGAs is therefore only useful if long waiting periods are expected. On the other hand, Igloo FPGAs can be already switched off at very early stages. However, the long startup time of the supply voltage and clock regulators prevent a shutdown at very high frequencies. The low-power static consumption in sleep mode stays however in a range acceptable for WSN applications. Igloo devices are ready for operation within one microsecond when recovering from the low-power Flash*Freeze mode [Igla] with a quiescent supply current as low as several micro-amperes.

Considering that Flash-based FPGAs and SRAM-based FPGAs have a similar level of dynamic power consumption when applying the same clock frequency, Flash FPGAs are preferable for both types of duty-cycled activity. The lower static power and the negligible startup delays of Flash FPGAs make SRAM FPGAs clearly out of the race to achieve the lowest energy consumption.

3.4 Conclusion

3.4.1 Summary of the considerations on reconfigurable hardware

By enabling a higher level of parallelism and the implementation of dedicated processing cores with digital logic, reconfigurable hardware easily outclass CPUs and DSPs with regard to speed. Thanks to a relatively patterned and homogeneous architecture, FPGAs can also benefit from the latest process technologies and achieve a high cell density while vendors invest significant efforts to reduce the power consumption of their devices. However, this characteristic remains a costly overhead for SRAM FPGAs applied to very low-power systems.

Nevertheless, a novel class of FPGAs has emerged in the recent years. Thanks to very promising figures in terms of static power consumption and a nonvolatile configuration memory, Flash FPGAs are constituting a convincing alternative to traditional SRAM-based devices. These features make these chips ideal candidates for integration in computationally demanding wireless sensor nodes. Minimal activation times are compatible with the high sampling rates typical for the targeted application domain. The lower performance is not critical as it is usually not expected that the device runs at its maximum operating frequency. Resources necessary to maintain high frequency oscillators can be saved and the overall power consumption reduced.

On the other hand, Flash devices lose the ability for runtime reconfiguration, which is much longer and energy costly as the similar procedure for the SRAM counterparts. This intrinsically reduces the flexibility of Flash FPGAs, which become more difficult to adapt to changing application specifications or environmental conditions. Innovative solutions must therefore be found to extract the maximum profit from a static FPGA configuration.

3.4.2 Outlook on the following part

Based on the preliminary analysis developed in these two introductory chapters, the practical investigation of this thesis will address the following problems in the next part:

- A general purpose hardware implementation of an FPGA-based wireless sensor node is potentially improving energy efficiency for computationally demanding applications. This property must be verified not only for the processing capability, but also for long term operation with appropriate low-power modes. In particular, it must be investigated how such a solution scales with high sampling rates compared with traditional MCU-based approaches.

- A solution overcoming the flexibility loss inherited by the lack of support for dynamic reconfiguration with Flash FPGAs is required. Wireless sensor networks are typical applications requiring flexible adaptivity mechanisms after deployments. This ability must be generalized in order to cover multiple application domain in a scalable manner.

- Deploying such an infrastructure requires the appropriate tools and drivers to fully support application deployment in a limited amount of time. The design of the FPGA architecture must not complexify the application unnecessarily. A solution integrating support for hardware acceleration and dynamic reconfiguration within the operating system is required.

Part II.

Design of a Framework Enabling Reconfigurable Hardware Acceleration in Wireless Sensor Networks

4 FPGA-based Hardware Acceleration for Wireless Sensor Nodes

Contents

3.1 Features of reconfigurable hardware systems 32

3.1.1 Technology 33

3.1.2 Granularity 34

3.1.3 Reconfiguration processes 36

3.2 Estimating the power consumption of reconfigurable hardware devices 37

3.3 Low-power duty cycling for FPGAs 39

3.4 Conclusion 41

3.4.1 Summary of the considerations on reconfigurable hardware 41

3.4.2 Outlook on the following part 41

The design of motes architecture can benefit from reconfigurable hardware technology with regard to multiple aspects. Not only is the inherent design flexibility exploited to develop custom hardware accelerators, but also to prototype new processor architectures or to explore the hardware design space. However, the overhead introduced by this implementation freedom is not always bearable for real-world applications, often restricting the created platform to experimental purposes. Therefore, it is important to first distinguish in the related work the projects with sensor nodes based on reconfigurable hardware for a usage in long-term deployments and projects staying at the prototype level. This chapter first reviews the literature with works following this approach and proposes several classifications of the existing platforms. Based on this analysis and the results from the preliminary chapters, the design of a mote based on reconfigurable hardware and suitable for real-world deployments is presented. The main features, performance metrics ans limitations of a mote implementing this architecture are presented and illustrated with selected application scenarios[1].

4.1 Related work

This section reviews recent research works where the idea of embedding reconfigurable hardware in a wireless sensor node is considered. The concept of integrating reconfigurable hardware into distributed sensor systems already emerged at the very early stages of wireless sensor networks with the Caμs system [SFN00] and the PicoRadio mote architecture [Rab+00]. The potential of reconfigurable hardware to accelerate computationally demanding signal processing tasks was already identified. Nevertheless it has never been fully exploited since then despite a high number of implementations.

A good and recent overview of sensor systems based on FPGAs has been given in [PBT12]. This work served as a preliminary basis for elaborating the tables reported in the following subsections, along with personal literature review and experiences from conference meetings. Because of their higher amount of resources, FPGAs are more commonly used in the design of hardware reconfigurable sensor nodes as complex programmable logic devices (CPLDs). However, because motes are intrinsically low-resourced

---

[1] The work presented in this chapter is related to the publications [PSG11a; PSG11b]

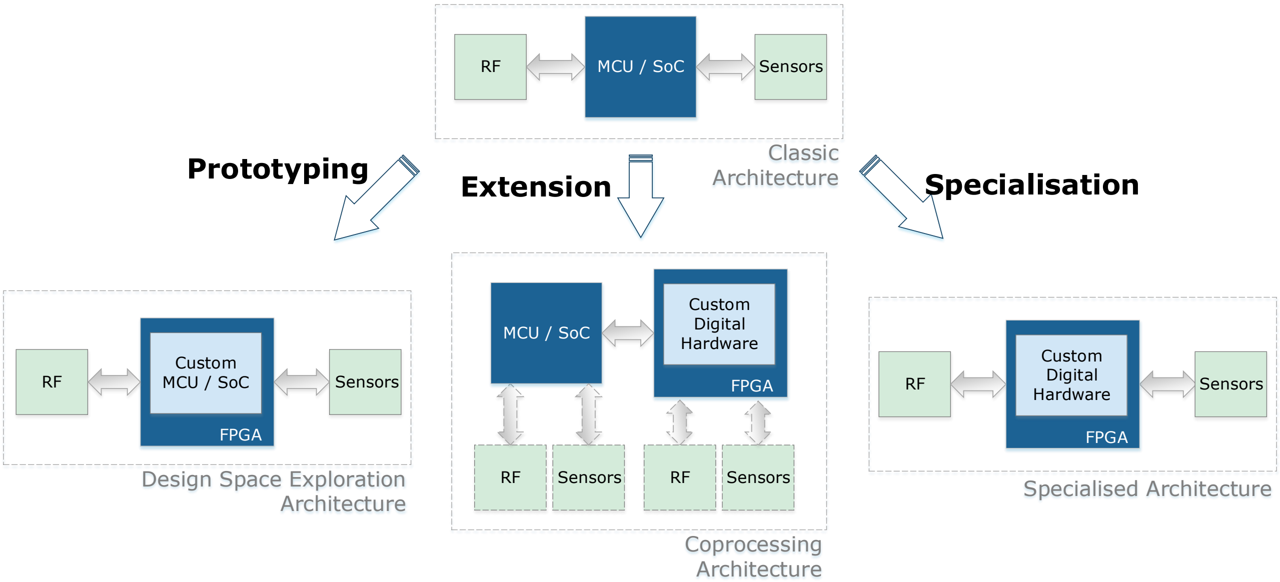


Figure 4.1.: Architecture alternatives for FPGA-based wireless sensor nodes

devices, CPLDs are in some cases a good alternative [LPZ07; MCP09; Bro+11]. Therefore, CPLD-based sensor nodes are also considered for this review work. Throughout the state-of-the-art analysis, three main types of approaches motivating the usage of reconfigurable hardware emerged:

- **FPGA as prototype for SoC design**: the reconfigurability of the FPGA is used to implement and evaluate different SoC architectures customized for motes (Section 4.1.1).
- **PLD as MCU coprocessing unit**: an FPGA is extending the MCU to accelerate specific tasks (Section 4.1.3).
- **FPGA as standalone processing unit**: the traditional mote MCU is completely replaced by custom hardware. No CPU is implemented (Section 4.1.2).

The generic architectures of these three design alternatives are illustrated by Figure 4.1. The next subsections will first consider these different approaches independently.

4.1.1 Wireless sensor nodes using an FPGA for SoC prototyping

Modern FPGAs have sufficient capacity to host a complete SoC architecture including CPU, memory and peripheral components based on digital logic. Specific design flows and tools have been developed by all majors FPGA vendors to support this feature (Xilinx Embedded Development Kit (EDK) [Xila], Altera's Nios II Embedded Design Suite [Alt] or the Microsemis's Libero SoC IDE [Libb]). Using these devices to evaluate different design alternatives is therefore a straightforward, cheap and fast approach. Table 4.3 reports the most relevant works based on this approach along with the main features of the proposed mote.

In general, the works are focusing on custom peripheral modules in the SoC. The CPU is kept unchanged and only application-specific hardware accelerators are investigated. In [Hil10] and [Ple+03], custom combinations of CPU and hardware accelerators (sensor interfaces) are analyzed. Design space exploration methods are applied to optimize the design with regard to power consumption and execution time. The application range is very wide, with cores accelerating routing algorithms [CSM08] up to image processing [Lu+09]. Certain works explore custom CPU architectures [Hin+08; HRG08; RFB10; AGP09]. In these works, new processor instructions are introduced to accelerate frequent operations executed by motes.

Mote - Reference	Year	FPGA	CPU	Radio	Application	Comment
WURM [Ple+03]	2003	Xilinx Virtex	LEON 32-Bit	Bluetooth	Networking Wearable Computing	PDR with OS support
[Hin+08; HRG08]	2007	Xilinx Spartan 3	LEON2 32-Bit	Xemics 868 MHz	General purpose	Reconfigurable Datapath
[Vol+07]	2007	Xilinx Spartan 3	Pico-Blaze	IEEE 802.15.4 + Bluetooth	Acoustic signal processing	Multi-channel
Nokia ASP [Aho+07]	2007	Altera Cyclone II	Nios II	Bluetooth	Wearable computing	Smart watch
[CSM08]	2008	Altera Cyclone	Nios II	Bluetooth	General purpose	Protocol optimization
[MR08]	2008	Altera Cyclone II	Nios II	Bluetooth	Temperature monitoring	Protocol optimization
[H.+09]	2009	Altera Cyclone II	Nios II	nRF2401 2.4 GHz	Image processing	
EasiSoC [Lu+09]	2009	Xilinx Spartan 3E	MC8051	Unknown	Image processing	Over-the-air programming
FemtoNode [AGP09]	2009	Xilinx Virtex II	FemtoJava	IEEE 802.15.4	Temperature monitoring	Java VM
[Ton+09]	2009	Xilinx Spartan 3E	Microblaze	IEEE 802.15.4	Cryptography	Petalinux OS
[Wei+09]	2009	Altera Cyclone II	OpenRISC	IEEE 802.15.4	Temperature monitoring	
BlueDot [RFB10]	2010	Xilinx Virtex 4	BlueCore	nRF2401 2.4 GHz	General purpose	Instruction set optimization
Hyperion [Hil10]	2010	Xilinx Spartan 3	Plasma 32-Bit	300 - 400 -800 MHz	General purpose	Design Space Exploration
[Vol+10]	2010	Actell Igloo	ARM Cortex-M1	Unknown	General purpose	Radio interface
[Zha11]	2011	Altera Cyclone II	Nios II	nRF2401 2.4 GHz	Vibration monitoring	
[Li+12]	2012	Xilinx Virtex 4	LEON3 / 8051	IEEE 802.15.4	Cryptography	PDR
[AM12]	2012	Xilinx Virtex 4	MiniMIPS NoC	Unknwon	DSP	MPSoC solution
HiReCookie [Val+12a]	2012	Xilinx Spartan 6	Microblaze	IEEE 802.15.4	Cryptography	Cookies [Por+06a] extension
[Hay+12]	2012	Xilinx Spartan 3	8051	WiFi	Vibration Monitoring	Safety applications
[FGV13]	2013	Cypress PSoC 5	Cortex M3	IEEE 802.15.4	General purpose	Smart transducer interface
Marmote SDR [Szi+13]	2013	SmartFusion	Cortex M3	MAX2830	Low-power SDR	Modular mote

Table 4.1.: Research works using FPGAs for mote architecture prototyping

Even though this approach gives good design hints to optimize processing tasks on motes, most of the works do not consider low-power operating modes. Duty-cycled operation is not taken into account even though it is one of the most efficient design aspect to achieve low energy consumption. By focusing on processing aspects only, other important mote-level features such as power management are neglected. In the end, the works presented here are not used in real-world applications, mainly because of a too high energy consumption. Even if this can be explained by the intrinsic nature of prototypes, the choice of the FPGA technology has also an impact on this observation. The devices are usually selected with high gates number in order to benefit from the maximum flexibility when evaluating the SoC architectures. By nature, these chips do not support well very low power operation modes since they are not targeting this range of applications. As a consequence, only a few works report the actual active current consumption of the mote (up to 1.1 W for [Hin+08], 330 mW for [Aho+07], 221 mW for [Wei+09], up to 400 mW for [Hil10], up to 1.2 W for [Li+12]). Using large devices is also taking the risk that all the available resources are not exploited and that a part of the power consumption and the infrastructure are wasted.

The only exception is the HireCookie node [Val+12a; Lom+12] where a MicroBlaze-based SoC with PDR capability is evaluated. Specialized hardware accelerators can be loaded at runtime to accelerate algorithms on-demand. When unused, the FPGA is switched off in order to save power. A second low-power microcontroller is controlling the activity and the reconfiguration of the main FPGA at regular time intervals. The authors reported an average active current consumption of 140 mA, which must be counterbalanced with the overhead introduced by the reconfiguration process (between 0.5 and 1 second at 50 mA for each power-up sequence).

Although these works show in general that the performance of motes can be largely improved by using tailored hardware accelerators and CPU architectures, they do not represent a reliable solution for real deployments. Most of the designs are targeting ASIC implementation, which is often a costly solution for WSNs where low prices and development times are important.

4.1.2 Wireless sensor nodes with standalone FPGA

Mote - Reference	Year	FPGA	Radio	Application	Comment
RCH [CTA08]	2008	Xilinx Virtex II	Sub-1-GHz	Data aggregation	PDR
VAPRES [GGRG09]	2009	Xilinx Virtex 4	Unknwon	Target tracking	Kalman filters
AEPod [Led+09]	2009	Actel Igloo	2.4 GHz radio	SHM	High sampling rate on multilple channels
[Kad+10]	2010	Xilinx Virtex 5	Unknown	Image processing	ASIC prototyping
[Gas+11b]	2010	Actel Igloo AGL600	IEEE 802.15.4	Image processing	Low power
[VS+10]	2010	Actel Igloo AGL250	IEEE 802.15.4	Vibration analysis	Industrial monitoring
[CSS11]	2011	Xilinx Virtex 5	Unknown	Image processing	ASIC prototyping
[TD11]	2011	SiliconBlue iCE65L08	144 MHz	Image processing	Biomedial applications
WMSN [PA11]	2011	Xilinx Spartan 3	IEEE 802.15.4	Image processing	
[Lia+13]	2013	Xilinx Spartan 3	IEEE 802.15.4	Network tasks	Protocol acceleration

Table 4.2.: Research works using FPGAs as standalone processing unit

Another notable design alternative for motes based on reconfigurable hardware is to completely remove the CPU. Existing works following this approach are reported in Table 4.2. Even though this solution intrinsically reduces the flexibility, it also impacts the energy-efficiency of the mote by focusing on the core functionalities. For this category, one can further distinguish between works using FPGAs for ASIC prototyping ([CSS11; Kad+10]) and works using the FPGA-based mote in real-world deployments with application specific configurations [Gas+11b; VS+10; TD11]. While the first approach is suffering from similar symptoms as mote SoC prototypes (Section 4.1.1), the latter is more suitable for very low-power application scenarios. [Gas+11b] and [TD11] respectively reported an average consumption of 8 mW and 5.9 mW for motes performing image processing while [VS+10] reported 8 mW for a mote implementing vibration analysis algorithms. It must be emphasized that these motes are based on FPGAs with non-volatile memory and a low number of logic cells.

Except in [Lia+13], the motes from this category have almost no support for communication tasks. This limits severely the networking capability of the nodes and restricts the number of supported topologies and the complexity of the MAC protocol, thus reducing the number of potential applications. In general, motes without CPU are already very focused on the target application and lack a genericity, which would make them suitable to a broader range of implementation scenarios.

4.1.3 Wireless sensor nodes with a co-processing unit based on programmable logic

The last and largest category is comprised of motes where MCUs or SoCs which are commonly found in motes architecture are extended with reconfigurable hardware devices. A list of works based on this approach is given in Table 4.3. In this case, the task of the reconfigurable hardware device is to relieve the CPU from specific duties that cannot be handled efficiently in software. Unlike FPGA-based motes designed for prototyping, most wireless sensor nodes from this category are expected to perform better than MCUs-only solution, in particular in terms of energy consumption and execution speed. That is one of the reason why the chips used for these motes are in the lower range of reconfigurable hardware devices in terms of gate count. In order to limit the power consumption, the co-processing unit is carefully scaled to the application requirements. In most of the cases, the extension is implementing a task that is not supported at all by the CPU because of resources restrictions, *e.g.* image processing or vibration analysis.

In the earliest stage, this solution has been studied with the PicoNode, a prototype mote for the PicoRadio project [BMR02]. Although the used technology is not up-to-date anymore, the architecture and the design flow were already very advanced and can still be used as a reference today. The FPGA co-processor could notably be programmed with a set of standardized functions, which could be selected out of a library and parameterized by the processor over different levels of application programming interfaces (APIs). Another founding work is the countersniper project [Sim+04] where an FPGA has been used to perform acoustic signal processing. This work is still used as standard reference for numerous WSN applications. Later on, numerous motes were created with the similar objectives. Among the most advances projects, it is worth citing the modular sensor node created by the Tyndall research institute [Bel+05]. Two different types of FPGAs with different capability can be plugged to a main CPU board and diverse sensor, communication or power supply modules. To the best of my knowledge, this mote is now the only FPGA-based sensor node which is commercially available. A very similar approach has been adopted for the design of the Cookie mote [Por+06b], which enables a modular combination of FPGA processing boards with different CPU, radio and sensing modules. Among other relevant projects, multiple recent motes are based on the low-power Igloo FPGA [Cen; Tan+08; Sch+08; Kos+10; BS10; Khu+11]. Like in the examples from section 4.1.2, an average active power consumption in the range of 5 to 20 mW is achieved. Works using CPLDs [Bro+11; MCP09] accelerate simple functionalities related to networking such as coding or cryptography. The mote developed by Microsoft research [LPZ07] uses CPLDs for inter-board communication in a modular sensor node architecture.

Mote - Reference	Year	FPGA	CPU	Radio	Application	Comment
PicoRadio [BMR02]	2002	Xilinx XC4000XLA	Strong ARM 1100	Bluetooth	DSP - Network Tasks	PicoNode SoC
Countersniper [Sim+04]	2004	Xilinx Spartan II	Atmega 128L	Sub-1-GHz	Acoustic signal processing	MICA2 extension
25 mm Tyndall [O'F+05; Bel+05]	2005	Xilinx Spartan II-E Xilinx Virtex 4	Atmega 128L 8051	nRF2401 2.4 GHz IEEE 802.15.4	Neural networks sensor connectivity	Small and modular
RANS-300 [Cal+05]	2005	Xilinx Spartan II-E	TI MSP430	Sub-1-GHz	Fire detection	Dynamic reconfiguration
Hogthrob [Vir+05]	2005	Xilinx Spartan 3	Atmega 128L	nRF2401 2.4 GHz	Animal monitoring	CPU prototyping
Square [KK06]	2006	Xilinx Spartan II	MicroBlaze (second FPGA)	RFM 433.92 MHz	Image processing	Dynamic reconfiguration
Cookies [Por+06b; Kra+11]	2006	Xilinx Spartan 3	ADuC841	Bluetooth IEEE 802.15.4	General purpose	Dynamic reconfiguration
Cookies [Cen]	2012	Actel Igloo AGL125	TI MSP430	Bluetooth IEEE 802.15.4	General purpose	Wake-up radio
Parrotfish [EKD06]	2006	Unknown	ATmega162	Bluetooth	General purpose	Dynamic reconfiguration
Micreleye [Nah+07]	2007	Atmel FPSLIC	Atmel AVR	Unknown	Image processing	Dynamic reconfiguration
Microsoft mPlatform [LPZ07]	2007	Xilinx CoolRunner-II	TI MSP430	IEEE 802.15.4	Acoustic signal processing	Stacked motes CPLD based
Rewise [Wil+07]	2007	Xilinx Virtex	TI MSP430	IEEE 802.15.4	General purpose	Dynamic reconfiguration
[Tan+08]	2008	Actel Igloo AGL600	ATmega644p	Unknown	FFT	Module slots
[Sch+08]	2008	Actel Igloo AGL250	TI MSP430	Unknown	Precise timing	Low-power
[MCP09]	2009	Xilinx CoolRunner-II	ATmega 1281	IEEE 802.15.4	Network tasks	CPLD based
[Kos+10]	2010	Actel Igloo	Atmel AVR	868 MHz	Vibration analysis	Industrial application
PowWow [BS10]	2010	Actel Igloo AGL125	TI MSP430	IEEE 802.15.4	Networking tasks	Contiki OS
[SLL11]	2011	Xilinx Spartan 3E	Atmel SAM7X	IEEE 802.15.4	Networking tasks	High bandwidth
[Khu+11]	2011	Xilinx Spartan 6 Actel Igloo AGL600	Atmel AVR32	IEEE 802.15.4	Image Processing	FPGA comparison
MasliNET [Jel+11]	2011	Actel ProASIC 3	ATmega 1281	IEEE 802.15.4	Image Processing	Energy harvesting
RESENSE [Bro+11]	2011	Xilinx CoolRunner-II	ATmega 1281	IEEE 802.15.4	Networking tasks	CPLD based
CES [HWH12]	2012	Actel SmartFusion	ARM Cortex-M3	IEEE 802.15.4	General purpose	Reconfigurable SoC

Table 4.3.: Research works using FPGAs for CPU extension

As the application scenarios differ for each of these motes, a direct comparison in terms of performance is not fair and not possible. Except CPLDs-based architecture and the PowWow node [BS10], hardware accelerators were used to improve sensor data processing, since it is usually the domain where highest gain can be obtained.

4.1.4 General considerations on related work

Among this high amount of motes selected from the literature analysis (more than 50), only a few works have reached an advanced level of development where a hardware prototype and a development environment is available. This includes :

- The PicoNode testbed as described in the previous section [BMR02]. Unfortunately, the reported power consumption is over 400 mW in all modes of operation.
- The 25mm sensor node from the Tyndall institute [Bel+05; O'F+05], which combines a Spartan-IIE or a Virtex-4 FPGA with different radio and sensor modules into a modular platform with a very small form factor. The nodes were used in applications where high sensor connectivity was required and for wearable computing applications. Even though the form factor of the node is very advantageous, no or very few performance and energy consumption metrics are given.
- The Cookies platform, which supports several types of FPGA (Igloo and Spartan 3), radio (Bluetooth and IEEE 802.15.4) and CPU (MSP430 and ADuC841) [Cen]. The mote was successfully deployed in an application for industrial environment monitoring [Val+12b]. Custom sensor interfaces and hardware accelerators were also implemented on the node. The authors show notably a significant gain in terms of speed when compared to software approaches.
- The Marmote SDR [Szi+13] with its modular architecture where the reconfigurable is principally used for SDR purposes. The authors show here a very large benefit for the usage of SoC embedding Flash-based reconfigurable logic when compared to standard SDR systems. This show the potential of these devices for high-speed and complex digital signal processing. The authors emphasize however that their approach with a SoC is not as good as what one could expect with an FPGA, as the low-power mode of the SoC is still power hungry.
- The platform developed by Hinkelmann et al. for the evaluation of different processor architectures on a Spartan-3 FPGA [Hin+08]. Unlike most of the existing works. Hinkelmann exploited the potential of a coarse-grained architecture to improve the energy-efficiency of the node. Even if his approach demonstrated that CGRAs are a good trade-off between software and ASIC, the acceleration capability is limited to a specific domain of applications (error correction and cryptography) where most of modern WSN MCUs or transceivers already integrate dedicated IP cores. The proximity to the processor limits also the level of parallelism that could be achieved with a custom design implemented on reconfigurable hardware.
- The Hyperion platform [Hil10] used for SoC design space exploration. This platform gives a great potential to test new architectures as it comes with a power estimation framework and a flexible connectivity. The benefit of selected SoC architectures for image processing could be demonstrated. However, the platform is not suitable for real deployments as the consumed power is too high.

None of these works was fully exploiting the low-power consumption of Flash-based FPGAs for sensor data processing. In particular, all the existing works using Igloo FPGAs are mainly focusing on the processing only part and do not take the power management issues into account.

The majority of the works are based on the WSN standard IEEE 802.15.4 radio. Bluetooth and Sub-1-GHz radios could also be commonly found but no motes was using a protocol supporting over-the-air

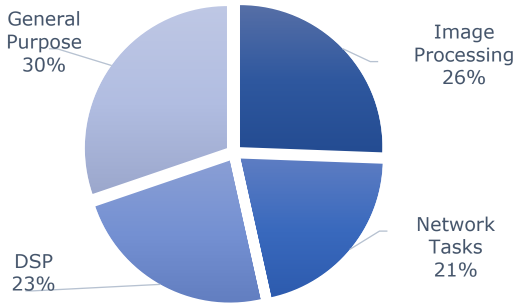


Figure 4.2.: Applications of FPGA-based wireless sensor nodes

throughput larger as 1 Mbps. In general, only a few works are making detailed considerations on the communication protocol.

The application range of the selected works is very wide, although three main domains could be identified: image processing, digital signal processing and networking tasks (including cryptography, channel coding and routing). The distribution of these application domains is shown in the pie chart from Figure 4.2. This shows that the additional processing power is not only used for sensor data processing but also for tasks related to communication.

In parallel to the mote architecture, three main trends are emerging to motivate the utilization of FPGAs on motes:

- **Application-specific acceleration** : One or several computation-intensive tasks need to be accelerated to achieve higher energy-efficiency. FPGAs with very low power consumption are selected to implement the algorithms. In general, the design is limited to the specific application domain, *e.g.* image processing, but is suitable for real-world deployments.

- **Design space exploration** : Large FPGAs are used to test and evaluate different CPU or SoC architectures. The larger power consumption makes the node not suitable for real-world deployments.

- **Dynamic reconfiguration** : Several works are investigating concepts to take profit from the dynamic reconfiguration capability of certain FPGAs [Ple+03; Hin+08; Li+12; Cal+05; KK06; EKD06; Nah+07; Kra+11; Lom+12]. Although this aspect will be addressed with more details in chapter 5, it is worth identifying this trend at this point since it describes a category of nodes with particular features.

Selected works have been classified according to these trends and placed on the graph 4.3. When considering the FPGA size and the implementation flexibility enabled by each design, the three respective zones can clearly be distinguished. The diagonal of the graph results from the logical relationship between the amount of available logic and the implementation freedom. The dynamic reconfiguration approach gives more flexibility while using less resources, placing the corresponding zone towards the right of the graph. Some motes share characteristics from several zones, like the WURM project, which combines dynamic reconfigurability features with design space exploration. In the lower left part of the graph, one can find existing works using Flash-based FPGAs, which are relatively limited for an utilization in different applications.

A zone corresponding to lower-resourced FPGAs but enabling more design flexibility stays however empty. There is here a good potential to develop a mote with limited resources allowing real deployments but sufficiently generic to cover a large range of applications and explore different types of processing solutions. By taking motes application-specific acceleration as a reference, this could be achieved by slightly increasing the size of the FPGA device. These additional resources can be used to implement a

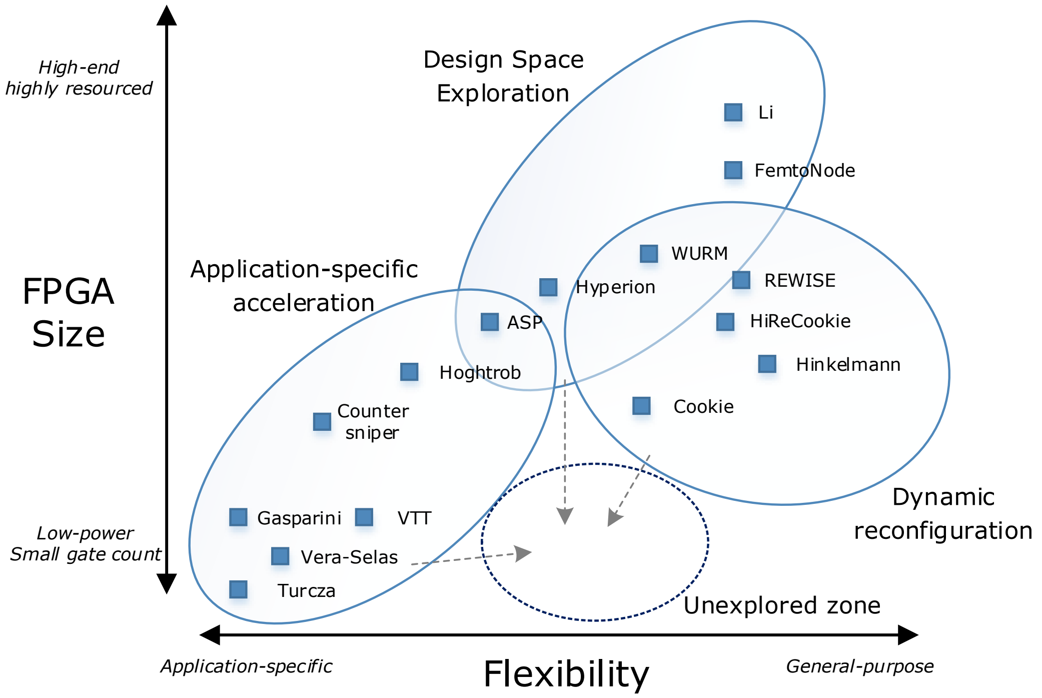


Figure 4.3.: Design space of FPGA-based wireless sensor nodes

framework for dynamic reconfiguration, which would place the mote further to the right of the graph. Finally, more resources and more flexibility implies better support for design space exploration.

Such a platform would then be suitable for deployments in real-world applications and benefit from the other main design features of reconfigurable hardware: resource sharing and architecture prototyping. This approach is promoting the utilization of the FPGA as an extension of a main microcontroller in order to maximize the CPU energy-efficiency. Implementing CPUs as softcores is a good solution for SoC design space exploration but stays a costly solution in terms of resource consumption. The rest of this chapter is endeavoring to describe the design of a platform architecture meeting these objectives.

4.2 Design of a modular FPGA-based low-power mote

The design requirements and the functionalities of a low-power FPGA-based wireless sensing platform have been identified in previous chapters and sections. The core architecture of the mote will be first described and metrics of two hardware implementations developed in the frame of two research projects related to this thesis. These are respectively HaLOEWEn (**H**ardware-**a**ccelerated **LO**w Energy **W**ireless **E**mbedded Sensor-Actuator **n**ode) developed in cooperation with the TU Darmstadt group for Embedded Systems and Applications (ESA) for the research center AdRIA [Adr] and LPSIP (**L**ow-**P**ower **S**ensor **I**nterface **P**latform) developed in cooperation with the Wireless Sensor Network research group from the Fraunhofer Institute for Integrated Circuits (IIS) for the EU FP7 MoDe (Maintenance on Demand) project [Mod].

4.2.1 Core architecture

A major feature driving the design of general-purpose mote and sensing-actuating systems in general is **modularity**. Directly integrating sensors on the main board of the platform is too restrictive. Each application requires the utilization of specific components which can be attached to the mote in a plug-and-play fashion. The high I/O connectivity of FPGAs is a further argument giving the board a natural

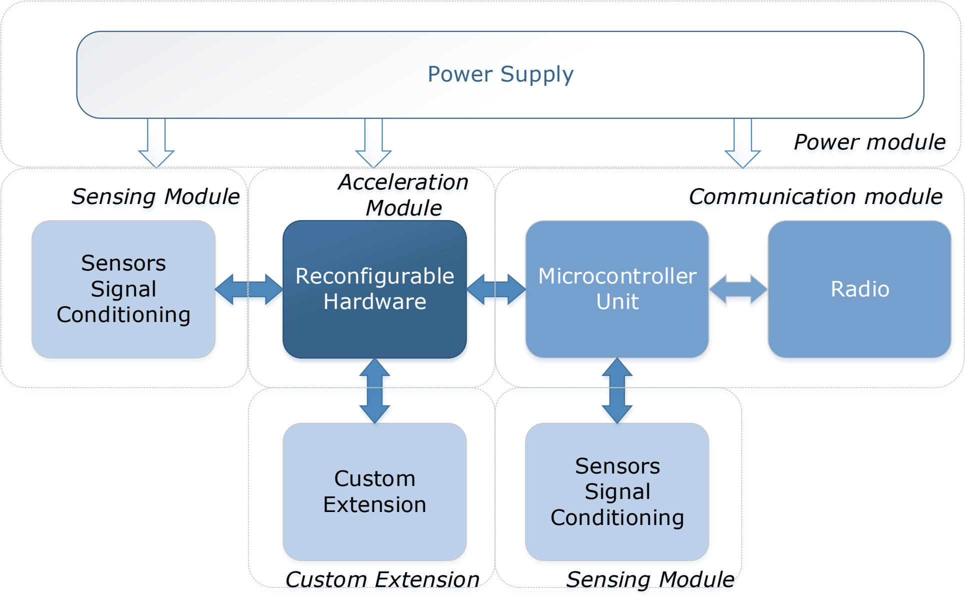


Figure 4.4.: Generic architecture of the FPGA-based sensor node

potential to be extended with additional modules. The core architecture of traditional motes is intrinsically suitable for modular design when distinguishing processing, sensing, radio and power supply (see Figure 2.2). Many of the related works are designed in a modular way as well: the PicoNode [BMR02], the 25 mm Tyndall node [Bel+05], Cookies [Por+06b], Microsoft mPlatform [LPZ07], Hyperion [Hil10], Marmote SDR [Szi+13] or the platform from Hinkelmann et al. [Hin+08].

On the other hand, modularity induces a significant overhead in terms of size, making the platform less suitable for a miniaturized implementation. As the evaluation of different radio modules is outside the scope of this work, no modularity has been planed for the wireless communication interface, thus globally reducing the size of the mote. This allows the utilization of RF SoCs where CPU and radio are already integrated in the same chip. The general concept of the mote is illustrated by Figure 4.4. It can be noticed that no direct interaction between the radio and the reconfigurable hardware module has been planed. This decision is an intentional wish to decouple the FPGA from the wireless communication protocol. Most of WSN microcontrollers and radio chips already include ASIC cores to accelerate MAC tasks, including encryption or error checking. The platform is then mainly targeting applications with data processing requirements at the sensor level, or at higher layer of the communication stack, *i.e.* at application level.

Based on the considerations from the previous chapters, the FPGA type has been fixed to Microsemi Igloo AGL1000V2, which is the largest available device from the Igloo family. A package which is compatible with smaller versions of the chip has been selected in order to downgrade the capability of the device if the target application has less stringent requirements. The chip supports supply voltage down to 1.2 V, which is compatible with a very low-power operation.

4.2.2 The HaLOEWEn platform

4.2.2.1 Main board

The HaLOEWEn platform implements the generic architecture described in the previous section with the only difference that the communication and acceleration module are associated on the same main board.

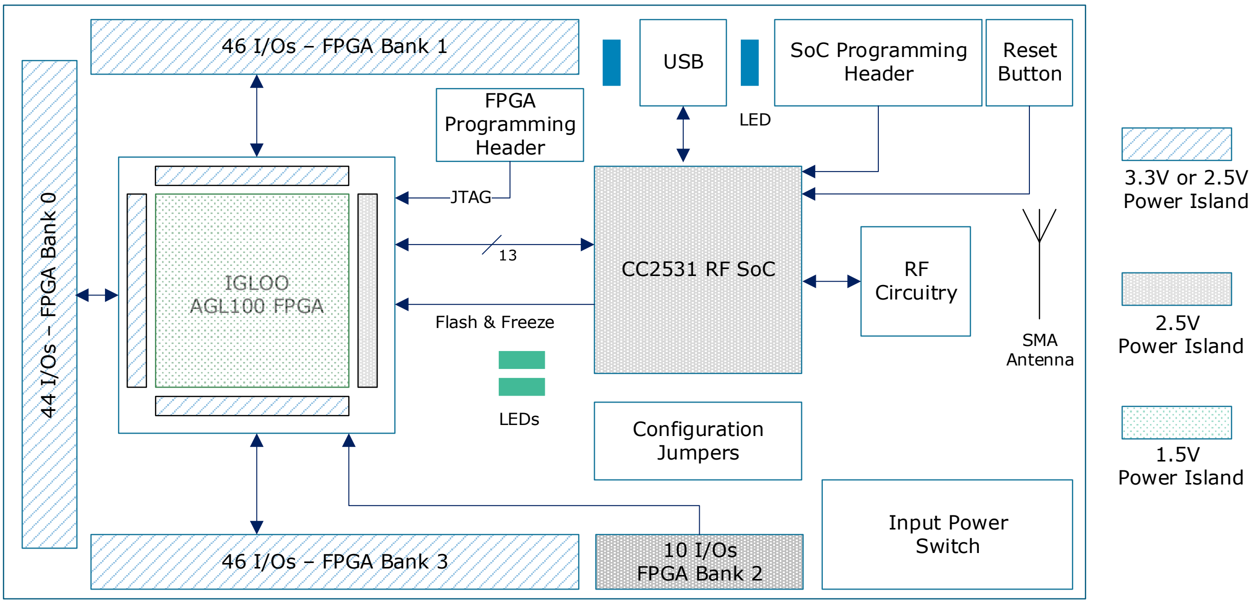


Figure 4.5.: Block diagram of the HaLOEWEn main board

The CC2531 chip was selected as RF SoC for its low power consumption in both active and sleep mode and efficient 2.4 GHz radio with IEEE 802.15.4 MAC support. This SoC is based on a 8051 8-bit CPU core.

A block diagram of the platform is given in Figure 4.5. The main characteristics of the board are reported in Table 4.4. A photograph of the board along with the platform logo are depicted in Figure 4.6. The complete schematics of the platform can be found in Appendix A and in the technical report [PP12].

4.2.2.2 Extensions

FPGA extensions

Thanks to the FPGA, the HaLOEWEn platform has a high I/O connectivity for external modules. Four extension headers are available to plug various application-specific modules. Each extension includes power I/O pins to supply extension boards from the main board with a selectable 3.3 or 2.5 voltage. HaLOEWEn-specific extension modules include an SRAM-FRAM memory module with parallel access for memory intensive applications, a generic ADC-DAC board with different acquisition ranges, sensor modules and an additional RF SoC extension to create multi-radio applications. An adapter has been conceived to support Digilent Pmods [Dig], low-cost digital breakout boards for sensing, data acquisition, communication, memory, I/O adapters and actuators. A VHDL driver is already available for most of these modules, enabling an easy import into the FPGA design.

This high connectivity gives a large design freedom. As modular hardware is getting increasingly popular thanks to open-source hardware projects like Arduino [Ard], the availability of breakout extension boards for a broad spectrum of applications is very high.

Energy harvesting

Even though the HaLOEWEn mote is commonly powered by two AA batteries, a hybrid energy harvesting module couple to a rechargeable Lithium-ion battery was developed [Zha12b][2]. The module simultaneously collects energy from a 12.5 cm x 6.5 cm solar panel (photovoltaics) and from a 30 mm x 90 mm thermoelectric generator (Seebeck effect). The generator outputs are regulated by a Single-Ended

---

[2] Preliminary results have been published in [Phi+12b] and a demonstration has been prepared for the DATE'11 conference [Phi+11]

Feature	Value
RF SoC	TI CC2531
CPU Core	8-bit 8051
Flash size	256 kB
RAM size	8 kB
ADC	8 channels 12-bit
Peripherals	2 USART - AES Coprocessor - USB core
Current draw	3.4 mA (Active)
Current draw	1 μA (Sleep)
Radio	2.4 GHz IEEE 802.15.4
Sensitivity	-97 dBm
TX current	28.7 mA (@1 dBm)
RX current	24.3 mA
FPGA	Igloo AGL1000V5
System gates	1 M
Low-Power mode	53 μW
RAM	144 kbits
On board oscillator	20 MHz
Size	96x60 mm^2

Table 4.4.: Main features of the HaLOEWEn platform

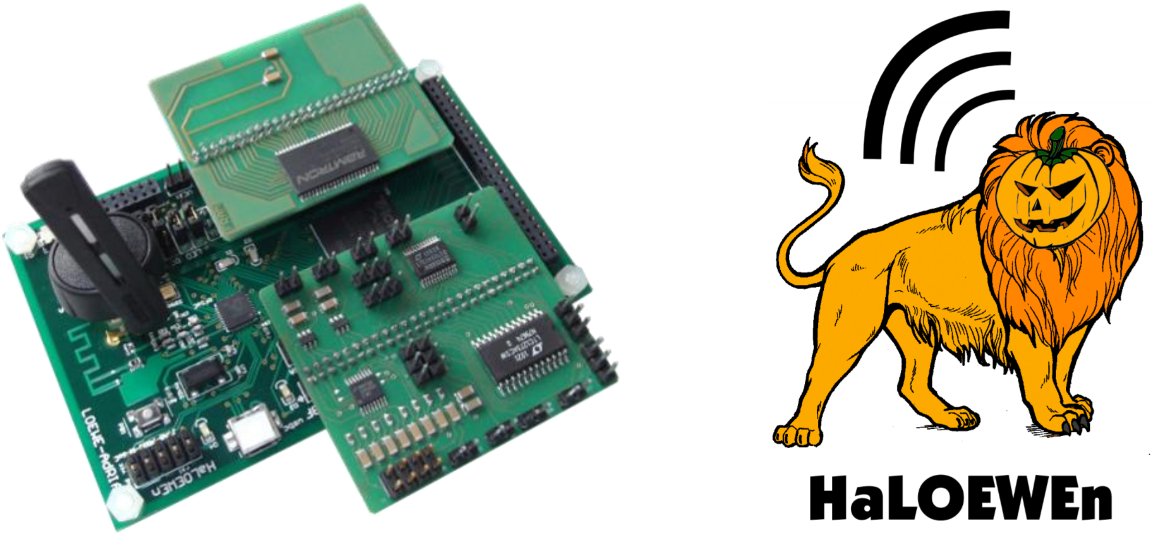


Figure 4.6.: Photo of HaLOEWEn version 3 and HaLOEWEn logo

Primary-Inductance Converter (SEPIC) DC/DC converter. This type of converter is controlled by a pulse wave modulation (PWM) signal, which can be tuned to achieve optimal impedance matching between the generator and the DC/DC converter. Searching for this optimal PWM duty cycle at runtime is a technique known as Maximum Power Point Tracking (MPPT). As the environmental conditions are changing, the operation of the DC/DC converter must be adapted to maximize the power output. In order to simplify the control logic necessary to implement this algorithm, the method known as fractional open-circuit voltage has been applied [Ahm10]. It relies on the quasi linear relationship between the optimal duty cycle and the open-circuit voltage of the energy source. By regularly sensing this voltage, the corresponding optimal PWM duty-cycle can be determined via a look-up table. On the developed energy harvesting module, the sensing, look-up and PWM functionalities for both the solar cell and the thermoelectric generator were implemented on an Igloo low-power FPGA with a very low amount of cells (AGL060 with 60k equivalent gates). Additional power management logic has been implemented to automatically cut the FPGA supply power when the amount of available power is too low for efficient recharging.

Experiments have been performed to measure the performance of the energy harvesting circuit in outdoor conditions [Zha12b]. The temperature gradient for the thermoelectric generator was created by exposing one side to direct sunlight while the other side is attached to a thermal mass used as heat capacitance. On a twelve days experiment with summer weather, it was estimated that the module could approximately generate an average power of 4.7 mW (day and night combined). From this 4.7 mW, the contribution of the thermoelectric generator was estimated to 4 %. This particularly low value is due to the low temperature gradient achieved in this scenario. Only the side of thermoelectric generator exposed to the sun is considered as the *hot* spot, which does not create a temperature variation sufficient to generate a significant amount of power.

This result gives an approximate value for the average power consumption that the wireless sensor node should reach to enable self sufficient operation, *i.e.* batteries do not need to be replaced. In practice, this setup is limited by the aging of the Lithium-ion battery, which does not support well continuous recharging processes. It however gives a significant lifetime extension boost for outdoor deployments.

Self-extension

For computation-intensive and time-critical applications, several HaLOEWEn boards can be combined into a multi-FPGA, multi-transceiver platform[3]. The FPGA I/O headers can be used for high bandwidth inter-FPGA communication while RF SoCs can communicate over wireless links. This setup is a particularly suitable for networked control systems where concurrent RF transmissions on different channels and highly parallel computing can significantly reduce processing delays. Distributed active vibration control is a typical example where motes must intensively coordinate with each other while implementing resource-consuming adaptive control schemes such as Least-Mean-Square (LMS) algorithms [PSG11a; Lav11]. As actuating and control systems are outside the scope of this thesis, no further details will be given on this aspect.

4.2.2.3 Software

The Contiki OS [DGV04; Con] was selected to manage the resources of the mote and implement the wireless communication stack. The OS includes a variety of libraries to combine event-driven operation with multi-threading. In particular, Contiki support *Protothreads* [Dun+06a], which are a lightweight memory-friendly mechanism to control multiple sequential flows running in parallel. The Contiki framework includes a simulator (COOJA) [Ost+06], which facilitates the test and evaluation of network-level tasks.

Contiki for HaLOEWEn was ported from the implementation for the TI CC2530 SoC from Sensinode [Sen], available in the main online repository of the OS. The modifications include board-level I/O

---

[3] This approach has been considered in the work published in [SPG11]

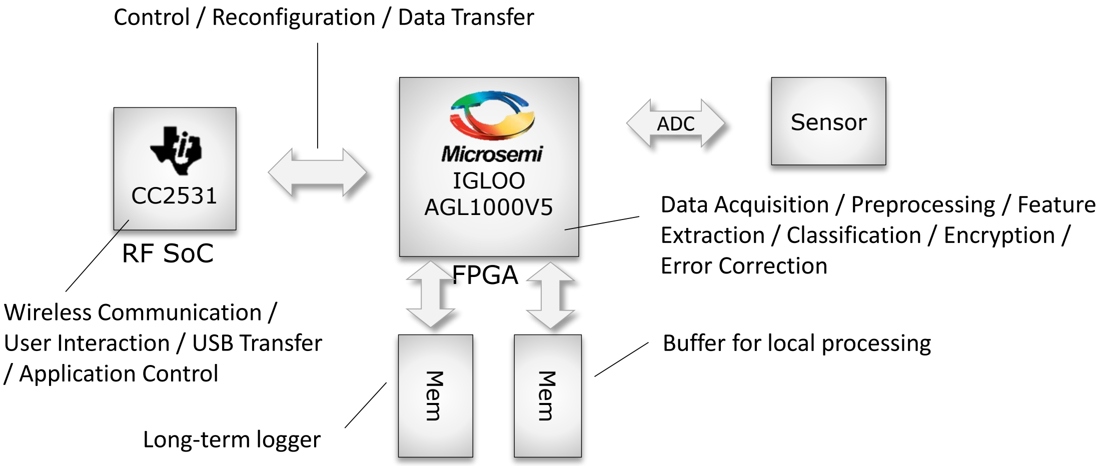

Control / Reconfiguration / Data Transfer
CC2531
RF SoC
Wireless Communication /
User Interaction / USB Transfer
/ Application Control
Microsemi
IGLOO
AGL1000V5
FPGA
ADC
Sensor
Data Acquisition / Preprocessing / Feature
Extraction / Classification / Encryption /
Error Correction
Buffer for local processing
Mem
Mem
Long-term logger

Figure 4.7.: Task distribution on HaLOEWEn

mapping and the integration of low-level drivers for USB and FPGA communication. In addition to the default Contiki Rime networking stack, the Sensinode group was particularly active in deploying the 6LowPan stack, a communication protocol commonly used for Internet of the Things (IoT)s applications. HaLOEWEn supports both communication stacks.

Based on the available resources and already available code fragments, the tasks on the sensor node can be distributed as illustrated by Figure 4.7. The MCU is handling communication and networking tasks while the FPGA implements all functions related to sensor data acquisition and processing, as well as all computationally demanding tasks.

4.2.3 MoDe LPSIP

The LPSIP implements the architecture described by Figure 4.4 with stackable boards. A block diagram of the system is depicted in Figure 4.8. The platform has been designed in such a way that the FPGA module (AB1MODE acceleration module) is optional. When not required, this module can be removed, leaving a traditional mote architecture without reconfigurable hardware extension.

The radio module is derived from the S3TAG mote developed by the WSN division of Fraunhofer IIS [Iis]. The microcontroller implements a proprietary operating system and very low-power communication protocol based on a tree hierarchical topology and Time-Division Multiple Access (TDMA) MAC.

The sensing module integrates four 12-bit ADCs accessible over different SPI busses with a sampling frequency configurable though a separate I^2C bus. The MCU and the FPGA also share the signals from four SPI busses for multi channel data exchange. ADCs and FPGA are clocked by a common oscillator operating at a frequency of 6.78 MHz. If a higher frequency is required to drive the FPGA internal logic, the internal PLL and clock conditioning circuits of the device must be used. The other main features of the platform are summarized in table 4.5. A complete description of the mote functionalities and operation modes can be found in [Els10].

4.3 Performance evaluation

As both platforms are based on a similar architecture, the performance evaluation will be focused on HaLOEWEn. The main target of this section is to highlight specific metrics of the platform, in particular in terms of power consumption and duty-cycled activity. As these metrics principally depend on the selected FPGA, they can serve as proof-of-concept for both platforms.

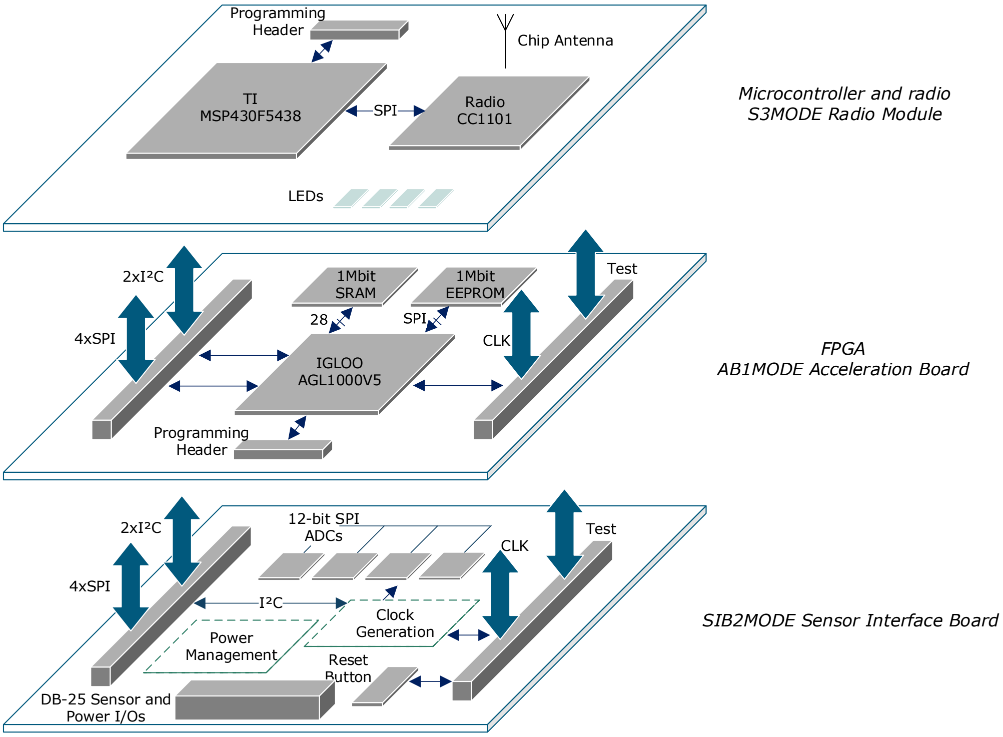
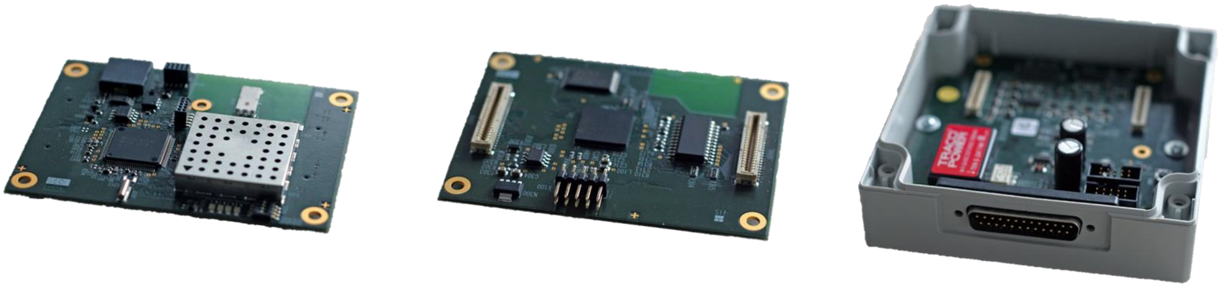


Figure 4.8.: Block diagram of the LPSIP platform

Feature	Value
MCU	TI MSP430F5438
CPU Core	16-bit MSP430
Flash size	256 kB
RAM size	16 kB
ADC	14 channels 12-bit
Peripherals	4 USART- 2 USCI - RTC
Current draw	2.5 mA (Active at 8 Mhz)
Current draw	1.8 μA (Sleep)
Radio	Sub-1-GHz CC1101
Sensitivity	-112 dBm
TX current	34 mA (@12 dBm)
RX current	14.7 mA
FPGA	IGLOO AGL1000V5
System gates	1 M
Low-Power mode	53 μW
RAM	144 kbits + 1 Mbits (external)
EEPROM	1 Mbits (external serial access)
On board oscillator	6.78 MHz
Size (enclosure)	120x120x60 mm^2

Table 4.5.: Main features of the LPSIP platform



(a) Radio module

(b) FPGA module

(c) Sensing module and case

Figure 4.9.: Photos of LPSIP modules

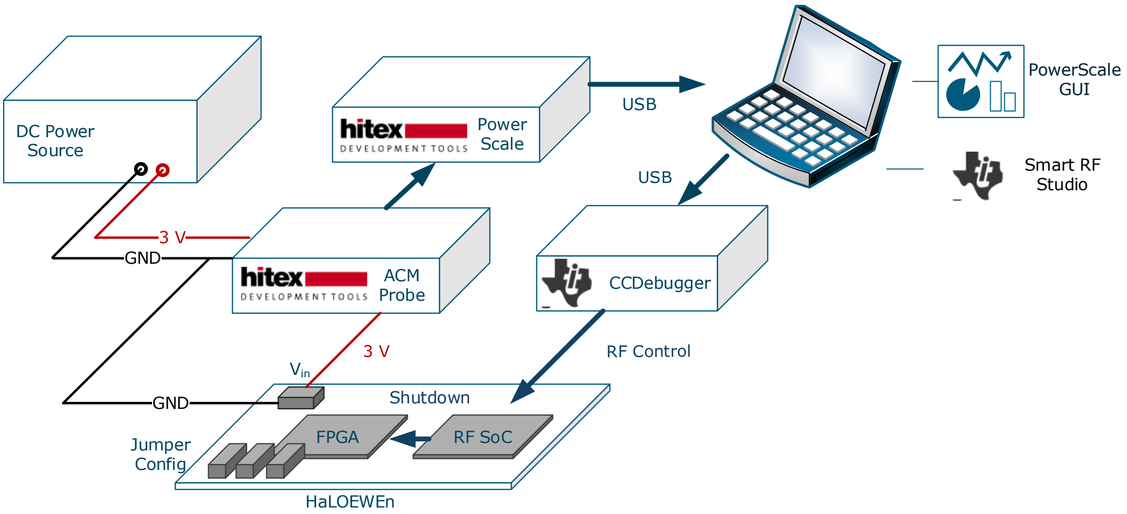


Figure 4.10.: Setup for the measure of power consumption

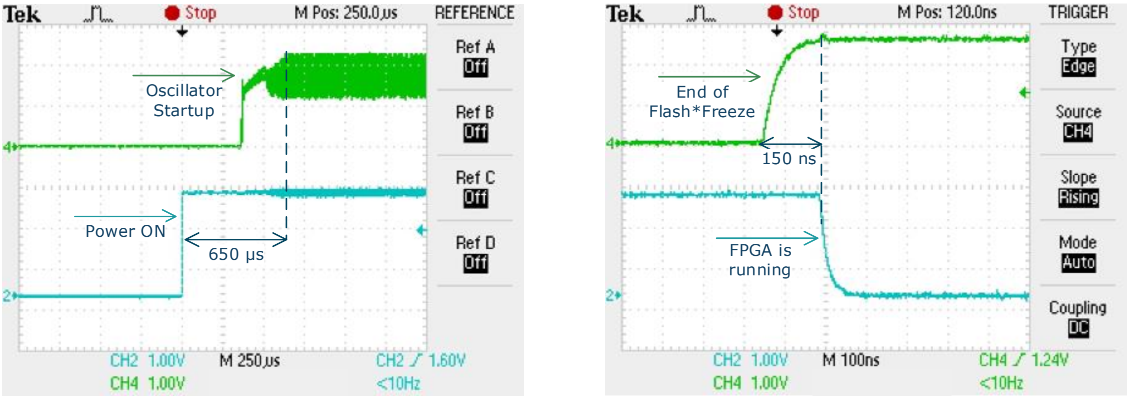
4.3.1 Power consumption

As highlighted in Chapter 3, the dynamic power consumption of the FPGA cannot be precisely estimated without taking a real application into account. Power consumption can however be estimated for operation modes where the FPGA is set in sleeping mode. As highlighted in the introductory chapters, it is also important to estimate the transition times between different operation modes. The platform current draw has been estimated using the Hitex ACM (Active Current Measurement) power probe for the PowerScale device [Hit]. Such a setup was also used to measure board-level average power consumption in the other experiments realized in the frame of this thesis. RF transmission was controlled with the CC2531 control panel integrated in the Texas Instruments Smart RF Studio 7 environment. The FPGA can be completely turned off by shutting down the DC/DC converter delivering I/O (3.3 V) and core supply power (1.5 V). The measure setup is depicted in Figure 4.10. Table 4.6 summarizes the power consumption of HaLOEWEn in the considered modes of operation. The 20 MHz oscillator was shutdown for these measurements.

Switching between different mode of operations might be costly in terms of delay and power consumption. For instance, the on-board 20 MHz oscillator driving the FPGA clock has a significant power consumption when active (> 6 mW) [Ltc]. Therefore, this component should be switched off simultaneously with the FPGA device to save energy. However, the startup time is restrictive as well. This delay

SoC operation mode	FPGA operation mode	Power consumption
Deep Sleep mode (LPM2)	Off	470 μW
Deep Sleep mode (LPM2)	Flash*Freeze	960 μW
Sleep mode (LPM1)	Off	750 μW
Sleep mode (LPM1)	Flash*Freeze	1.2 mW
CPU Idle (no radio)	Off	29.8 mW
CPU Idle (no radio)	Flash*Freeze	30.3 mW
TX mode (1 dBm)	Flash*Freeze	92.1 mW
TX mode (-16 dBm)	Flash*Freeze	78.3 mW
RX mode	Flash*Freeze	95 mW
CPU Idle (no radio)	On (16-bit counter)	38 mW

Table 4.6.: Power consumption of HaLOEWEn in different operation modes



(a) External Oscillator

(b) Flash*Freeze

Figure 4.11.: FPGA startup delay

illustrated by the measurement screenshot in Figure 4.11a is the combined effect of the slow startup time of the DC/DC converter and the oscillator. On the other hand, recovering from the FPGA Flash*Freeze mode is performed within a few nanoseconds (Figure 4.11b). This significantly lower delay makes the FPGA ready to process data almost instantaneously and enables very short duty cycle activity. It however requires that the oscillator stays continuously active. To overcome this issue, a solution where the FPGA can autonomously control its sleep activity without external oscillator is presented.

4.3.2 Autonomous control of sleep mode

In the classical approach, managing the sleep activity of the FPGA is carried out by the MCU by asserting the dedicated Flash*Freeze pin. This implies that this controller must synchronize its activity with the reconfigurable hardware device. This is a main issue if the controller must maintain its own duty-cycled activity. It must first come out of a sleep cycle before being able to activate or deactivate the second device. Section 2.2.5 has shown that this approach is not scalable when the duty cycle period is short, as the MCU might need to stay continuously active.

There is however a method to maintain a wake-up self-awareness by keeping an internal part of the logic activated, even during sleep periods. For Microsemi Flash FPGAs, the sleep Flash*Freeze mode is activated when both the external pin and an internal enable signal are asserted. When this mode is active, external I/Os are deactivated so that no external clock signal can toggle the logic. As a consequence, only

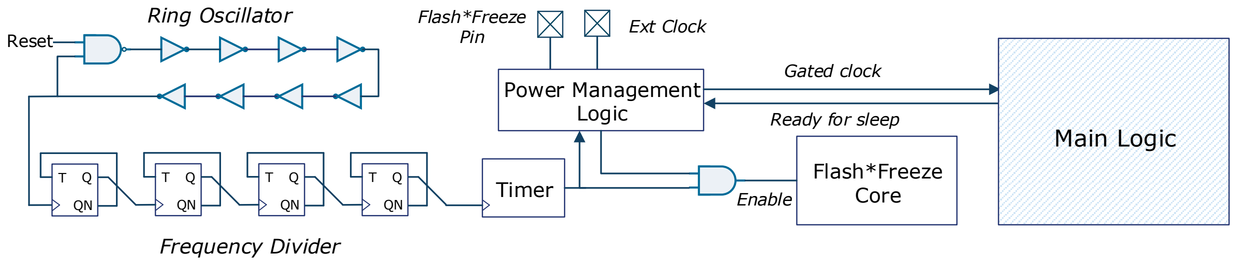


Figure 4.12.: Concept for internal wake-up process on FPGAs with Flash*Freeze technology

the deassertion of the external pin can wake the FPGA up from the Flash*Freeze. This problem can be overcome by implementing an internal ring oscillator [Msa; Gas+11b; GS12]. Based on an odd chain of inverters, this ring will continue running as long as the core internal voltage is supplied. Indeed, as its operation does not depend on external signal, it can continuously run even if the sleep mode is activated. This mechanism can be used to control a timer, which regularly change the status of the internal *sleep enable* signal and allow the device waking up on its own in order to perform a custom operation, typically acquiring sensor data or starting a processing task. A block diagram illustrating this concept is shown in Figure 4.12.

Ring oscillators in FPGA are however particularly sensitive to external factors such as temperature but also to internal unpredictable delays due to routing. In order to obtain a predictable output frequency, FPGA cells implementing the inverters and frequency divider must be placed close to each others. This is achieved by constraining cell placement in the Place & Route tool. The result of such a placement in the Libero FPGA Editor tool is depicted in Figure 4.13. A ring composed of 25 inverters (including a NAND gate connected to the reset signal) triggering a 10 stages frequency divider has been implemented. This circuit generates a clock signal with frequency 25.5 kHz on an Igloo AGL1000 FPGA. This circuit has an estimated power consumption of 351 μW. The clock frequency can be tuned by adjusting the number of T-Flip-Flops and inverters in the chain. Without frequency divider, the ring oscillator generates a clock at 13.05 MHz, which corresponds to a unit cell delay of approximately 3 nanoseconds. The generated clock frequency follows then the relationship

$$f_{intclk} = \frac{1}{3 \cdot 2^{N_{TFF}} N_{Inv}} \tag{4.1}$$

where N_{TFF} is the number of T-Flip-Flops and N_{Inv} the number of inverters. This frequency can be adjusted according to the application requirements. If the accuracy of the clock frequency is not critical, this signal can even be used as a main clock for the design, thus canceling the need for the external oscillator and drastically reducing the power consumption of the platform in sleep mode.

As highlighted by the previous chapters, it is important to precisely estimate the costs from transiting from one state to another for a duty-cycled activity. Table 4.7 reports these costs when taking the different clocking solutions into account (ring oscillator or external oscillator). The indication *(+Osc)* in the table indicates that the external oscillator is running. Clearly, the solution using the internal ring oscillator is the most efficient. The FPGA can resume its activity instantaneously while the overall power consumption stays very low.

4.3.3 Hardware abstraction layer

In order to optimize the interaction of the MCU with the FPGA, a custom interface has been implemented in VHDL to access the functionalities of the FPGA. Based on the considered distribution of tasks, the MCU is operating as master and is always initiating transactions between the two components. Two

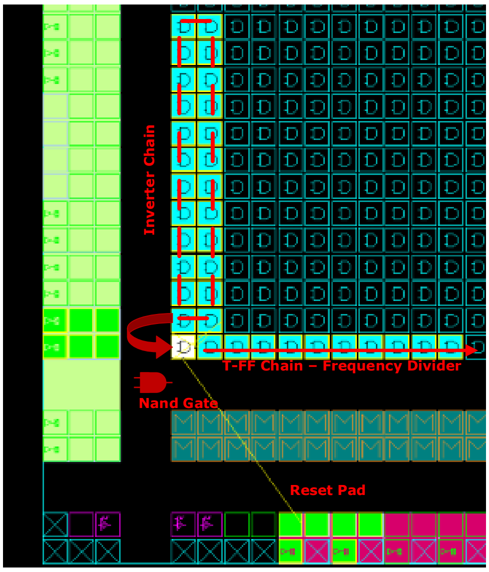


Figure 4.13.: Low-level placement of ring oscillators components on an Igloo FPGA

Transition	Sleep power	Wake-up time
Active (+Osc) - Off (+Osc)	8.1 mW	315 μsec
Active (+Osc) - Off	0.75 mW	650 μsec
Active (+Osc) - Flash*Freeze (+Osc)	9.3 mW	$<$ 1 μsec
Active (+Osc) - Flash*Freeze	1.2 mW	650 μsec
Active - Flash*Freeze	1.4 mW	$<$ 1 μsec

Table 4.7.: Transition metrics for the FPGA on HaLOEWEn

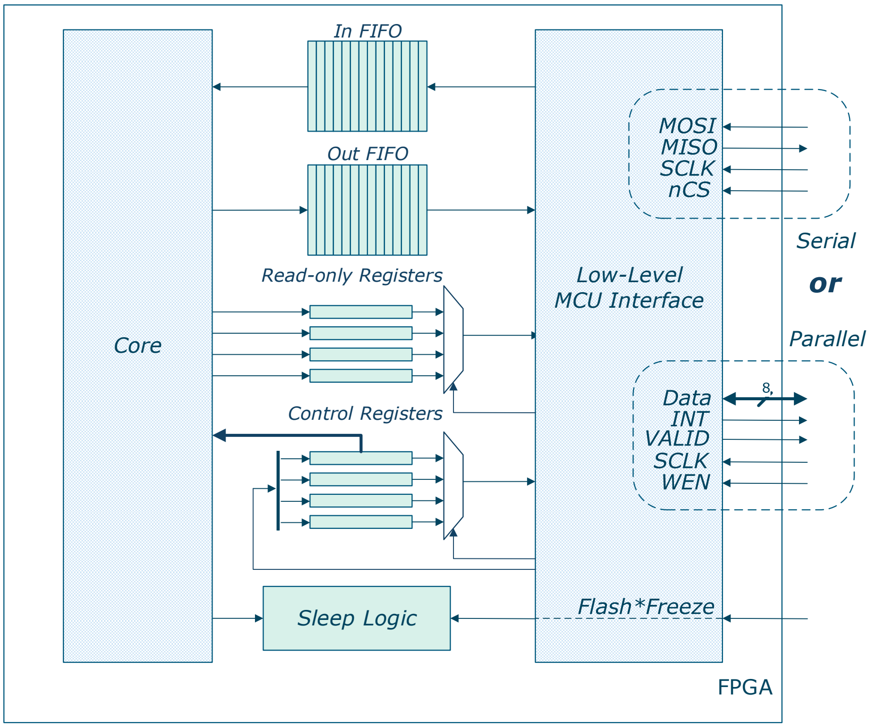


Figure 4.14.: MCU-FPGA Interface

access modes are supported: register access and memory access, both with byte granularity. The interface integrates one or several block memories operating as a first-in first-out (FIFO) module for streamed data exchanges. Depending on the pin connectivity between the FPGA and the MCU, the low level data link can either be serial (SPI based) or parallel. In both cases, a communication sequence between both components starts with a command (one byte) sent by the MCU indicating the access type of the required transaction. A separated pin is reserved for enabling the Flash*Freeze mode of the FPGA.

The corresponding driver for both parallel and serial interfaces has been implemented in the Contiki-OS files specific for HaLOEWEn. A maximum transfer speed of 2 Mbps is reachable with the serial interface while 6.2 Mbps can be achieved with the parallel interface. Similarly, an SPI driver has been integrated in the firmware of the LPSIP for a maximum speed of 847 kbps. Up to four SPI interfaces can run in parallel on this platform so that the transfer rate can be increased but this feature has not been evaluated.

4.3.4 Application examples

In order to illustrate the utilization and the performance of the platform with real-world applications, several examples have been selected. The objective of this section is to show the feasibility of the FPGA-based platform for the selected scenarios.

4.3.4.1 FFT processing

The Fast Fourier Transform (FFT) is a common reference algorithm to compare the performance of motes when considering processing power. State-of-the-art motes usually hardly support FFT algorithms because of a low support for DSP operations and memory limitations. In this scenario, we consider that HaLOEWEn is sensing a signal from a 12-bit ADC with at a sampling rate f_s and computing the average magnitude of the FFT of this signal over eight non-overlapping windows of size 2^N samples. This scenario is typical for

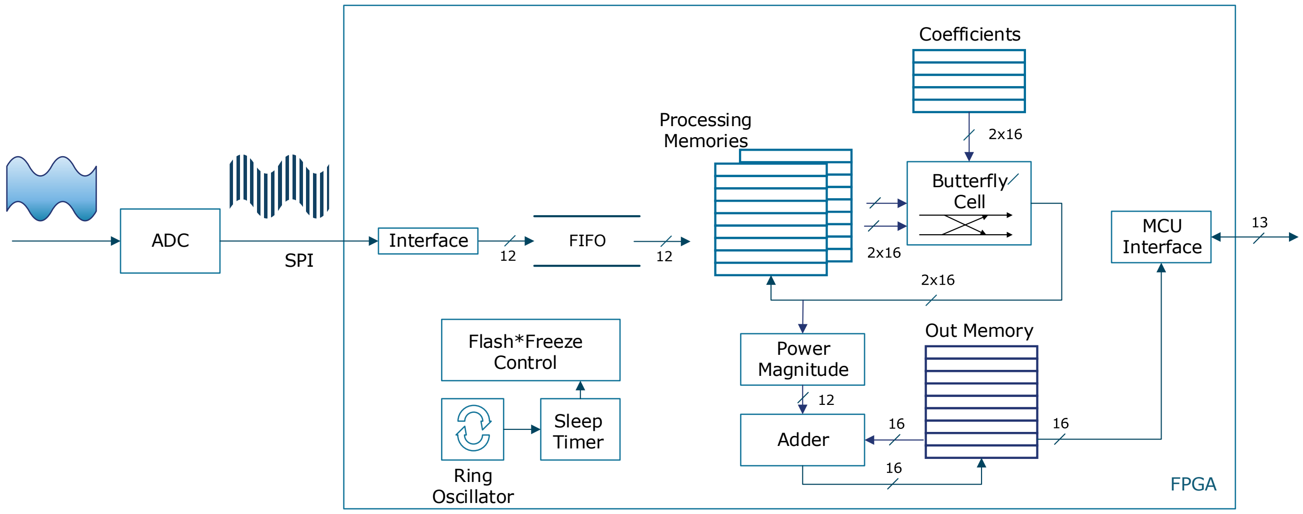
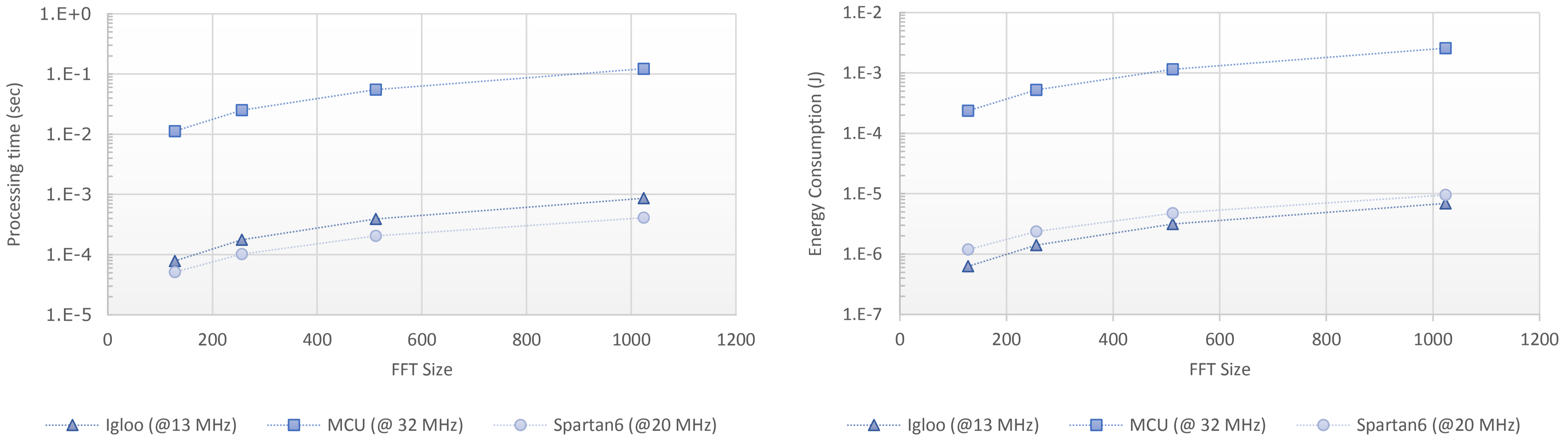


Figure 4.15.: FPGA design for simple FFT processing

condition monitoring applications based on vibration or current signature analysis as it will be described in details in the Part III dedicated to application scenarios. Such algorithms are also commonly used in audio signal processing applications or biomedical smart systems. A similar scenario is also considered in [Nac+08] with the Imote2 platform, and [Tan+08] and [VS+10] with a mote extended with an Igloo FPGA.

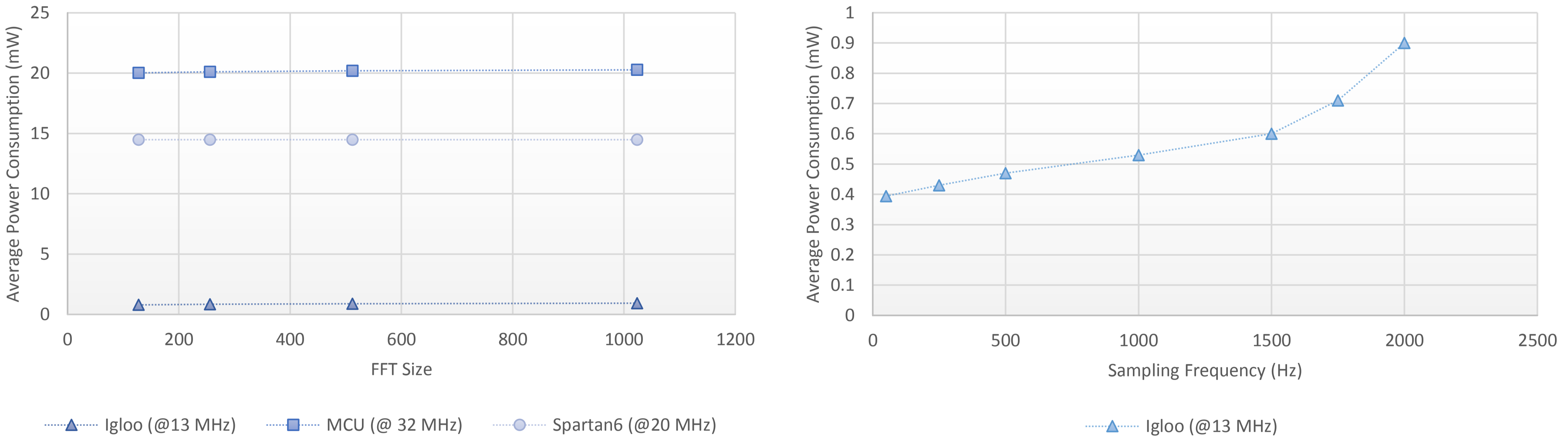
The FFT core considered in this application is based on a single butterfly cell and two processing memories alternately used to store intermediate processing results. Results from butterfly operations are computed at each clock cycle thanks to an internal pipelining of operators. Including further butterfly units to increase the parallelism of the execution will result in a bottleneck at the memory level. Data samples for different butterfly units can not be read or written back simultaneously from the same memory and the target device has not enough memory resources for parallel block memory instantiation. This architecture is typical for resource-constrained FFT implementations and is also used in IP cores from Xilinx or Microsemi [Xilb; Mic13]. It is assumed that the twiddle coefficients are already computed and available in an extra block memory. This architecture is suitable for Cooley-Tukey 2^N-FFT schemes. The FPGA is operating independently from the MCU: the internal wake-up logic as described in section 3.3 is used to regularly turn on the logic to acquire sensor data from the ADC. A higher frequency ring oscillator is used to generate the ADC SPI clock allowing to turn off the the external oscillator for most of the time. When sufficient data have been accumulated in the internal FIFO, the FFT core is activated. Results are accumulated eight times before an interrupt signal is sent to the MCU to retrieve the data. Figure 4.15 shows a block diagram of the design while Figure 4.16 gives results of the performance evaluation. Processing time is evaluated as the delay between the start (all samples are available) and the end (magnitude of the half spectrum is available) of the FFT. As a matter of comparison, the same algorithm is implemented on the CC2531 MCU in pure software and on the same VHDL code has been ported to the Xilinx Spartan6 LX16 FPGA for a comparison with SRAM-based technology. However, the implementation on Spartan6 does not use a ring oscillator and the sleep mode is activated by simply gating the main clock to the core design. No microcontrollers are used in parallel of the Spartan6 FPGA to control sleep mode. The data is directly transfered to a PC over a serial link. The power consumption of the Spartan6 is measured on a standard Xilinx development board (Nexys3).

The curves in Figure 4.16 show the different metrics evaluated for different FFT sizes, target devices and sampling frequencies. For the curves 4.16a, 4.16b and 4.16c a sampling frequency of 1 kHz was used. An FFT size of 1024 points was used for curve 4.16d.



(a) Delay (processing only)

(b) Energy (processing only)



(c) Dependency of average power consumption against FFT size (sampling included)

(d) Dependency of average power consumption against sampling rate (sampling included)

Figure 4.16.: Comparison of different implementations for FFT processing on the wireless sensor node

	Logic occupancy	Blocks memory	Max. frequency
Igloo	32.1 %	66 %	22.3 MHz
Spartan6	14 %	28 %	60.7 MHz

Table 4.8.: Resource consumption of the 1024-FFT design

In terms of processing delay, the FFT is completed in a comparable amount of clock cycles for both FPGAs, resulting in similar computation times. The Xilinx FPGA is faster here since it is clocked by a faster oscillator. The MCU is however taking a significantly longer amount of time (about two orders of magnitude), which may prevent the execution of other tasks related to networking as it has been shown in the example from section 2.2.5.1. The processing time could be improved by implementing the FFT with a radix-4 *dragonfly* cell or with a streaming architecture at the cost of additional memory resources and logic resources. However, streaming FFT is not a viable solution for the Igloo FPGA as the necessary logic is not available to implement all ten butterfly operators that would be required in this case. Similarly, the resources of the Igloo would be pushed to their limits with a *dragonfly* operator, which requires two additional complex multipliers and ten additional adder/subtractors.

In terms of energy consumption (considered for the processing task only), the trend is similar: both FPGAs demonstrate similar performance whereas the MCU has a significantly higher cost. The Igloo FPGA has a slightly better value than the Spartan FPGA because of its lower static power consumption. At this frequency, static power is dominating the overall power consumption on the SRAM FPGA.

When taking the time and energy spent during the sampling period, *i.e.* before enough sensor data is available to start the FFT, a clear distinction can be made between each platform. It can be first noticed that there is only a small dependency with the size of the FFT. Indeed, as the sampling period is relatively small compared to the processing time, the average consumption is highly dominated by the power spent during sampling. For instance, for a 1024 points FFT, the computing time represents less than 0.1 % of the activity for the FPGAs, respectively 10 % for the MCU. As a consequence, the Flash FPGA can reach a much lower power consumption value thanks to the autonomous control of sleep mode introduced in the previous section. On the other hand, the Spartan6 FPGA is suffering from its high static power consumption and consumes up to ten times more power than the Flash-based counterpart. The MCU reaches also a high average value because of the additional time required for processing and its inability to switch faster in a very deep low-power mode.

Finally, the last curve is showing the dependency of the average power consumption against the size of the FFT on the Igloo FPGA. This value starts to increase more rapidly for higher sampling frequencies because of the increasing number of transitions to sleep mode. As some internal parts of the device are shutdown during the Flash*Freeze mode, there is still a certain inertia in power consumption during the transitions. This overhead is more noticeable when the sampling frequency increases.

The resources consumption largely vary between the Spartan6 and the Igloo FPGA. This difference is mainly due to the availability of DSP embedded blocks in the Xilinx part. The maximum frequency is higher accordingly.

A fair comparison with related works is difficult since the implementation details of the algorithm are different. However, the FFT computation on the Imote2 [Nac+08] is taking up to 700 milliseconds for 1600 points, which is significantly longer as what could be expected with an FPGA implementation.

At last, it is worth estimating the energy spent by the whole platform when taking the wireless communication into account. At 1 kHz sampling rate, it is not feasible to transfer the complete sensor data stream with an acceptable reliability. By averaging the FFT computed on eight windows, the amount of data to transmit can be efficiently reduced by a factor 16 (it is sufficient to transfer a half spectrum), which reduces the data arrival rate to the MAC layer of the communication stack to 1 kbps. The pie chart 4.17 shows the measured power consumption when the MCU running the ContikiMAC protocol in a similar fashion as in the example from section 2.2.5.1.

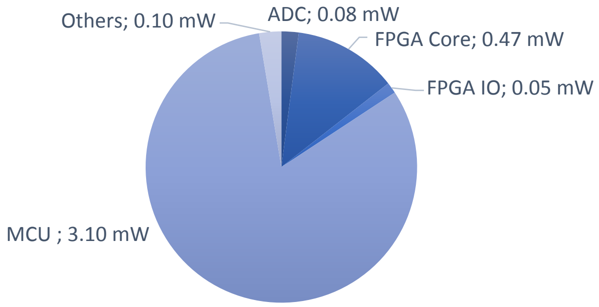


Figure 4.17.: Board-level distribution of power consumption

The power consumption of the board is largely dominated by the MCU because of the wireless communication costs. The share of the FPGA stays very low thanks to the low-power management techniques and the fast execution speed. It can be noticed that the overall power consumption stays below 4 mW, which makes the solution suitable for a power supply with energy harvesting.

4.3.4.2 Localization[5]

This example investigates the implementation of an engine for acoustic source localization. Clustered motes use differences of time of arrival of acoustic waves to precisely estimate the location where they were originally emitted. Indeed, as acoustic waves have a low propagation speed, they will reach spatially distributed sensor nodes at different times. This difference can be easily measured if nodes implement a synchronization mechanism. Such a setup can be applied for sniper localization as in [Sim+04] and [Vol+07], for structural health monitoring purposes as in [GGK10] or for localization of motes between themselves, like it could be the case for the Cricket motes, which are equipped with ultrasonic transceivers [PCB00]. A wireless sensor network combining ultrasonic localization and external acoustic source localization has been realized with the platform from Hinkelmann & al. [Hin+08]. The details of the selected localization algorithm called *spherical intersection* can be found in [Phi09; PSG11a; PSG11b]. It is mainly based on the solving a system of second order equations obtained from matrix of time difference measurements. Similar location engines can be found as hardware IPs in some WSN-specific SoCs such as the TI CC2431 [Cc2a] or in the ASIC developed by Karalar & al. in [Kar+04].

The FPGA is used here as an accelerator for the localization algorithm and not as a sensor interface. From this perspective, the FPGA is used for *on-demand* processing. It is sufficient to power it up only when necessary as the startup delay is negligible. In this application, starting the computation corresponds to the time when difference of time of arrival measurements from neighboring nodes have been collected. This approach is also suitable for an heterogeneous sensor network where only one mote extended with an FPGA is available. Computationally-intensive tasks such as the localization algorithm are centralized on this mote.

Figure 4.18 shows the architecture of the design. Most of the resources are consumed by the two multiply-accumulate units although the second degree solver also requires adders and a square root unit. It has been shown in [Phi09] that the best accuracy is achieved when several estimates from small node clusters are combined whereas using all measurements unnecessarily increases the complexity and duration of the computation without giving better results. In a two-dimensional localization system, four measurements are already giving a good approximation. Therefore, the core considered here implements a localization algorithm for four nodes. Further measurements can be taken into account by computing other estimates and combining them. Table 4.10 reports the processing time and average power consumption measured for the same targets as the ones considered in the previous sections.

---

[5] This section is based on the Diploma thesis [Phi09] and the publications [PSG11a; PSG11b]

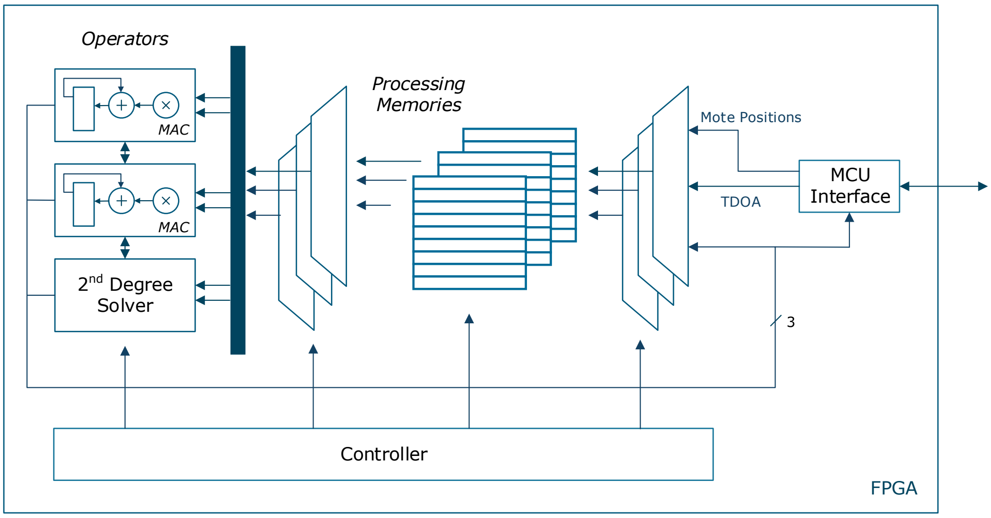


Figure 4.18.: FPGA design for localization accelerator

	Logic occupancy	Blocks memory	Max. frequency
Igloo	24.1 %	9.4 %	25.6 MHz
Spartan6	8.3 %	9.4 %	86.3 MHz

Table 4.9.: Resource consumption of the localization accelerator

Platform	Time	Power	Normalized energy
CC2531 (32 MHz)	7.6 msec	29.8 mW	297
Spartan6 (20 MHz)	114 μsec	17.1 mW	2.5
Igloo (13 MHz)	177 μsec	4.3 mW	1

Table 4.10.: Performance evaluation for the localization process

The benefit of implementing the algorithm on the Igloo FPGA can clearly be identified. When compared to software, up to 300 times less energy is consumed. The Spartan6 FPGA consumes slightly more energy for the processing, but the results in this table do not take the startup delay from the shutdown mode into account. Indeed, if the FPGA is not running when the localization is demanded, the FPGA needs to be preliminarily configured, which is taking up to 50 milliseconds with the best optimizations [Lom+12].

In general, Igloo represents the best alternative to execute a computationally demanding algorithm as long as the data transfer between the MCU and the device is not becoming the bottleneck. For instance, the same localization algorithm implemented on the LEON2 32-bit CPU core used in the prototype of Hinkelmann & al. is taking 764.3 microseconds without reconfigurable extension (software) and 13 microseconds with the reconfigurable function unit (reconfigurable hardware) [Hin11]. In the latter case, the data was directly available in the memory of the CPU and could be fetched directly to the reconfigurable function unit. With the Igloo approach, data must be exchanged externally with the MCU, which is taking a longer amount of time.

4.3.4.3 Combined channel coding and cryptography[7]

Forward error correction (FEC) and encryption are essential features of wireless systems. Even if FEC is only energy-efficient when the coding and decoding costs stay very low [SBW09], the vulnerability of the wireless medium made systematic ciphering mandatory for trustworthy deployments. However, cryptography has a significant cost in terms of energy and processing delay, which is not always affordable for a low-resourced wireless sensor node. As they are often use together, the possibility to combine both algorithms into a single primitive becomes an interesting approach to reduce the overall system costs. Nevertheless, the simultaneous optimization of both features in a single algorithm is not straightforward as maximum security and maximum error correction capability are natural contradictory features. Exploitable synergies between the primitives can be identified: a first simple basis is the common arithmetical background used by selected FEC and encryption algorithms: Galois-field arithmetic. Secondly, both algorithms process data block-wise. But most importantly, the most exploitable similarity of these primitives is the *diffusion* property. Famous encryption (Rijndael AES, SHARK, Twofish) and error correction schemes (Reed-Solomon) are both using Maximum-Distance Separable (MDS) matrices to achieve a maximal diffusion.

Based on this similarity, a combined error correction - cryptography hardware accelerator has been implemented. The core is based on the so-called *High-Diffusion* algorithm parametrized for blocks of 128 bits with a coding rate of 33 %. Implementation details of the algorithm can be found in Appendix B.1.

As in section 4.3.4.2, the core is evaluated without sensor attached to the FPGA. The combined cryptographic-error correction core is called by the MCU when required. If the encryption key is fixed for the duration of the application, it can be saved in the internal read-only memory (ROM) of the Igloo FPGA. In this case, the FPGA can be safely shutdown between several encryption cycles. Otherwise, the FPGA must be kept in *Flash*Freeze* mode since it would be unsafe to transfer the key between the MCU and the FPGA at each cycle.

Figure 4.19 shows the hardware architecture of the core. It is based on a classical AES hardware accelerator with minor modifications for the *Transpose* and *Diffusion* step. Additional logic is included to handle the coding part of the algorithm. This basis architecture is closely operating with a Reed-Solomon decoder, which is used to identify and correct transmission errors. The performance metrics of this core are reported in table 4.12.

Once again, the FPGA is demonstrating a largely better energy-efficiency than the MCU. Even if the absolute gain is not large for a single execution of the algorithm, encryption and error correction are operations that are ran very frequently. Accumulated, this difference can become significant on a long-term basis. A major drawback of this approach is when the FPGA must stay in a sleep mode to keep track of

---

[7] This section is based on the Master thesis work [Kly11] and the publication [Phi+12c]

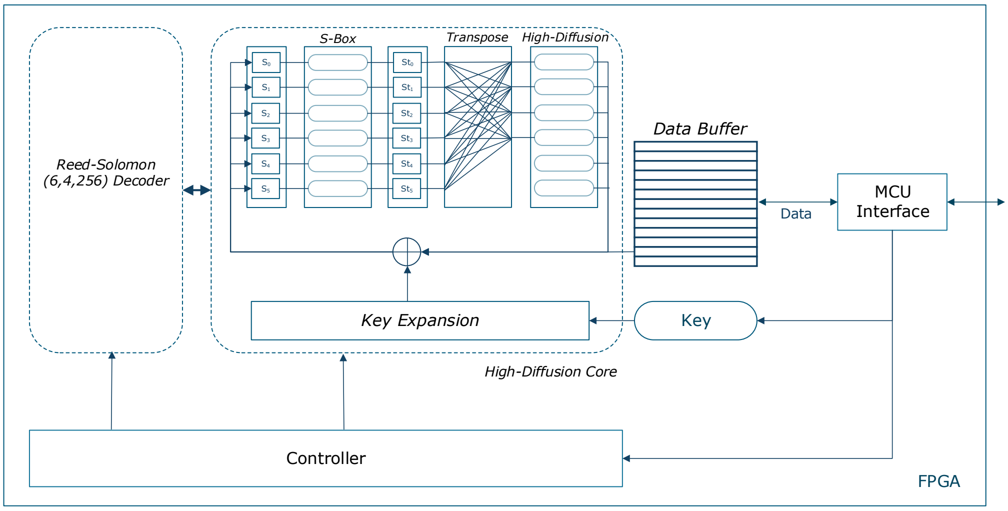


Figure 4.19.: FPGA design for high-diffusion accelerator

	Logic occupancy	Blocks memory	Max. frequency
Igloo	41 %	6.2 %	41 MHz
Spartan6	34.3 %	6.2 %	122 MHz

Table 4.11.: Resource consumption of the High-Diffusion core

	Platform	Throughput	Energy efficiency
Coding	Igloo (20 MHz)	182 Mbps	112.8 pJ/bit
	Spartan6 (20 MHz)	182 Mbps	208.8 pJ/bit
	CC2531 (20 MHz)	21 kbps	51 nJ/bit
Decoding	Igloo (20 MHz)	37 Mbps	310 pJ/bit
	Spartan6 (20 MHz)	37 Mbps	675 pJ/bit
	CC2531 (20 MHz)	12.7 kbps	87 nJ/bit

Table 4.12.: Performance evaluation for combined encryption and forward error correction

the key. The hardware solution is no longer suitable in this case since the energy gain of the hardware implementation will not cover the energy spent by the FPGA during sleeping. As a consequence, this solution is only suitable if it is associated with sensor data acquisition core as in the example with the FFT.

Another scenario where the approach is efficient is when a large number of data blocks has to be processed. In [Val+12a], the authors consider this scenario for various encryption algorithms implemented on an SRAM-FPGA. Here, the reconfiguration time of the FPGA must be taken into account as well. They demonstrated that the hardware solution is more energy-efficient than software when the number of data blocks is high (> 2000). As this situation is unlikely to happen in a typical wireless sensor network scenario, SRAM-FPGAs are not a good target device for this type of processing. On the other hand, computation time can be largely reduced with the hardware implementation: this solution can therefore be adopted for time-critical implementations.

4.4 Conclusion

FPGA-based wireless sensor nodes are popular research platforms because they give designers a large freedom to investigate customized hardware architectures, whether it is new types of CPUs or co-processing units. Three main types of mote architectures were identified were FPGA are used for three main reasons: design space exploration, application-specific acceleration and runtime reconfigurability.

However, many works were restricted to the processing-only tasks and are failing to demonstrate the efficiency of the approach on longer activity periods. A platform architecture and two different implementations have been presented to demonstrate the feasibility of very low-power hardware accelerated wireless sensor nodes. Thanks to the nonvolatile technology and efficient wake-up schemes, the platform can operate with very low current consumption, making it suitable for scenarios with power supply based on energy harvesting. In particular, high bandwidth sensing applications where sampling periods are very short can be efficiently implemented. On-demand hardware acceleration is also possible for applications where computationally demanding tasks need to be implemented.

Compared to the related work, the works based on SRAM-FPGA are clearly ruled out for the considered range of applications when taking the analysis done in this chapter into account. This technology keeps however certain advantages, notably in terms of speed and reconfigurability at runtime. Indeed, SRAM-based FPGAs are usually faster and more suitable to implement digital signal processing or image processing tasks than the Flash counterparts. Motes using Igloo FPGAs appear in the literature around the same time frame (2009 - 2011) [VS+10; Gas+11a; Khu+11; Tan+08; Sch+08; Kos+10]. However, these works fail to introduce a platform with a certain degree of universality as it has been done here. This was later on corrected with one of the layer for the Cookies mote, but in all cases, a detailed analysis of power consumption and duty-cycled activity is missing.

The examples taken in this chapter demonstrated the suitability of the platform for application-specific acceleration[8]. On the roadmap to the unexplored zone depicted in Figure 4.3, more flexibility has to be given to the platform in order to embrace applications with more advanced processing requirements. A suitable solution pulling the system towards more flexibility is dynamic reconfiguration. However, the non-volatility of the FPGA, which was a significant advantage for low-power consumption, becomes in this case a major obstacle that must be overcome.

---

[8] Additional publications and application examples with HaLOEWEn can be found in [ELK11; ELK12a; ELK12b]

5 Design of a Virtually Reconfigurable FPGA Overlay Architecture for Resource-Constrained Devices

> *Generality* always win
>
> G. Bell, “Bell’s Law for the Birth and Death of Computer Classes”. In: *Communications of the ACM*, January 2008, Vol. 51, No. 1

Contents

4.1 Related work **45**
  4.1.1 Wireless sensor nodes using an FPGA for SoC prototyping 46
  4.1.2 Wireless sensor nodes with standalone FPGA 48
  4.1.3 Wireless sensor nodes with a co-processing unit based on programmable logic . . . 49
  4.1.4 General considerations on related work 51
4.2 Design of a modular FPGA-based low-power mote **53**
  4.2.1 Core architecture 53
  4.2.2 The HaLOEWEn platform 54
    4.2.2.1 Main board 54
    4.2.2.2 Extensions 55
    4.2.2.3 Software 57
  4.2.3 MoDe LPSIP 58
4.3 Performance evaluation **58**
  4.3.1 Power consumption 60
  4.3.2 Autonomous control of sleep mode 61
  4.3.3 Hardware abstraction layer 62
  4.3.4 Application examples 64
    4.3.4.1 FFT processing 64
    4.3.4.2 Localization 68
    4.3.4.3 Combined channel coding and cryptography 70
4.4 Conclusion **72**

5.1 Introduction

In [EK09], A. El Kateeb emphasized the need for the development of a third generation of wireless sensor nodes with hardware reconfiguration capability. Beyond the first generation of static motes and the second generation with software update capability, this new class of motes will support a full adaptability to adjust all application-level functionalities after deployment. He however points out that an appropriate infrastructure is missing and that the power consumption overhead of reconfigurable hardware is still a major issue.

The design of such sensor nodes with runtime hardware reconfiguration capability has been also addressed in multiple works in the literature [CTA08; Cal+05; Por+06b; Kra+11; Lom+12; Val+12a; EKD06; Nah+07; Wil+07; Ple+03; Lu+09]. However, all these works were based on SRAM FPGAs, which support this feature inherently, but still suffer from a large static power consumption overhead as it has been highlighted in the previous chapters. With Flash FPGAs, this restriction disappear but the devices do not support hardware reconfiguration natively. A solution where both hardware dynamic reconfiguration and low power consumption are combined is therefore required to fully enable this new generation of advanced sensor nodes.

To achieve this goal, it is important to first identify the aspects motivating and restricting runtime hardware reconfiguration on sensor nodes in general, and with Flash-based FPGAs in particular:

- **Device limitation**: wireless sensor nodes are intrinsically embedded systems which are limited in resources. When using reconfigurable hardware, the size of the device cannot be freely extended without negatively impacting the current draw and the size of the platform. The number of reconfigurable cells available to implement hardware accelerators must be therefore limited. As a consequence, all tasks that could potentially benefit from an implementation on the reconfigurable hardware cannot be programmed at the same time. With dynamic reconfiguration, each task could be loaded on the hardware in a time-multiplexed manner in order to comply with these restrictions. Flash FPGAs are particularly suffering from this limited size since the still large process technology (130 nm) does not allow producing chips with very large capacity. For instance, table 5.1 reports the resources usage of selected IP cores on an Igloo AGL1000 FPGA, the largest available in this family. It can be noticed that the device becomes very quickly overloaded when several cores are combined, notably when dealing with digital signal processing. This weakness is due to the lack of embedded multipliers, which must be implemented with the programmable logic. A single multiplier will already cost 10% of the resources (24-bit), which stringently limits their utilization and speaks in favor of a reutilization across multiple signal processing units. Larger Flash-based devices exist in the Igloo-e family, a subfamily of Igloo FPGAs. In particular, the AGLE3000 can hold three times the capacity of the Igloo AGL1000. This large size would partially solve this resources limitation issue but the associated price (over 400€) is fully incompatible with the low-priced feature of wireless sensor nodes[1].

- **Adaptiveness**: Runtime reconfiguration enables functional update of the motes with hardware accelerators that were not planed at programming time. Thus, the device can be modified with cores which are more appropriate for a given situation (context-awareness) or updated with a bug fix or with a design improving the system performance. Once deployed, motes are not directly accessible and are in some cases even unreachable. It is also likely that there is a important number of nodes being part of the network. Reprogramming each single node using a manual programmer is therefore a time costly task that should be avoided. Enabling runtime and remote update capability would solve this issue. The ability to remotely modify the functionality of a wireless node is known in the literature as *over-the-air* reprogramming.

- **Reconfiguration limit**: Flash cells are sensitive to program-erase cycles. Microsemi reports that Igloo FPGAs can go through an average of 1,000 programming cycles before getting damaged. This limitation makes such FPGAs unsuitable for applications which regularly require a complete reconfiguration of functionality. A solution to implement different tasks without modifying the underlying configuration saved in the Flash cells is therefore needed.

---

[1] As a matter of comparison, the IGLOO AGL1000V5 FPGA used in the frame of this work is available for about 75 €

[2] COordinate Rotation DIgital Computer

Core	Cells utilization	Block RAMs utilization
UART	1,247 (5%)	2 (6%)
3DES	1,316 (5%)	0
AES	4,049 (16%)	8 (25%)
CORDIC[2] (vector serial)	1,560 (6%)	0
CORDIC (vector parallel)	17,563 (71%)	0
DES	1,187 (5%)	0
RS Decoder	9,701 (39%)	1 (3 %)
16-bit FIR 8-tap	2,702 (11 %)	0
16-bit FIR 16-tap	4.926 (20 %)	0
Cortex-M1	7,491 (30 %)	4 (13%)
1024-FFT (16-bit)	6,681 (28%)	18 (56 %)
Multiplier 16-Bit	1,041 (4%)	0
Multiplier 24-Bit	2,103 (9%)	0

Table 5.1.: Resource utilization of selected cores for Igloo AGL1000 FPGAs

- **Reconfiguration time**: Time and energy spent during the hardware reconfiguration are pure losses for the application, since nothing useful can be executed during this process. Minimizing these metrics is therefore critical to improve the time and energy-efficiency of the device. As reprogramming the configuration memory of Flash FPGAs is taking up to several minutes [HWH12], it must preferably stay untouched, which corroborates the previous statement about limiting the amount of core reconfigurations. When possible, the dynamic reconfiguration process must stays as short as possible. The reconfiguration time larger than 1 second reported in [Lom+12] for SRAM-based FPGAs may already be considered as too restrictive, so that beyond the technology, different reconfiguration models are required.

- **Bitstream size**: One of the cause for FPGA long reconfiguration times is large configuration bitstreams. As FPGAs are fine-grained reconfigurable, each cell require a significant amount of configuration data to implement a gate-level functionality, resulting in large programming files (in the range of hundreds of kilobits [Val+12a]). Holding or managing this amount of data on wireless sensor nodes is challenging since they usually have stringent memory restrictions and limited bandwidth for remote transmission. For instance, it is unlikely that a standard mote is able to store a complete FPGA configuration bitstream as the memory capacity is too low.

When considering hardware dynamic reconfiguration, it is also important to identify the purpose and the frequency of the reconfiguration. Figure 5.1 shows a graph where different types of reconfiguration processes are classified according to their level and frequency. For instance, runtime reconfiguration applied because of resources limitations will correspond to a time-multiplexed hardware, where an operator or logic resources are shared across multiple accelerated tasks. Responding to changing environmental conditions will correspond to the model of context-awareness where the functionality of the reconfigurable hardware is modified in a more global manner.

Taking the previous considerations into account, all types of reconfiguration are appropriate. Limited resources and inaccessibility make wireless sensor nodes ideal candidates for supporting all types of approaches, which is a feature rarely available in traditional embedded systems based on reconfigurable hardware.

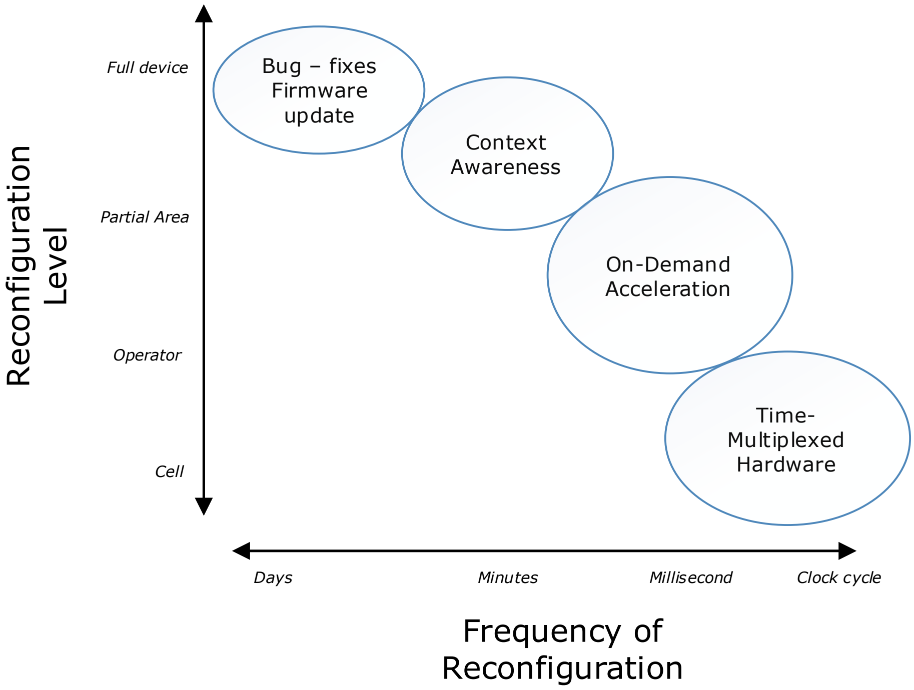


Figure 5.1.: Levels of hardware dynamic reconfiguration

In order to handle all these requirements, the solution proposed in this chapter will take profit from the features of another type of reconfigurable architectures: **coarse-grained** arrays. Introduced in Chapter 3, this type of architecture potentially solves the problems of long reconfiguration times and large configuration bitstreams since significantly less data is needed to describe the functionality, as it has been highlighted by Hinkelmann in a previous work [Hin11]. This property makes CGRAs easily scalable to all types of dynamic reconfiguration as described in Figure 5.1. Profiting from the flexibility of the underlying logic provided by the FPGA, a coarse-grained reconfigurable architecture can be implemented on top of the programmable logic, as a **virtually reconfigurable overlay**. Such an overlay alleviates the restriction of integrated CGRAs, which may suffer from a too high degree of specialization for a certain domain of applications. The design of Hinkelmann [Hin11] was for instance restricted to a few types of algorithms and an extension of functionality required modifications of the core architecture. Thus, the solution introduced here is based on a template CGRA, which can be freely customized for application-specific requirements. As an *overlay*, the architecture does not interfere with the configuration of the underlying Flash cells at runtime. The reprogramming restrictions of the Igloo FPGAs are thus overcome. In addition, such an overlay might be ported to other target technologies and represents therefore a more generic alternative to FPGA dynamic reconfiguration, which is often device specific, *i.e.* the reconfiguration flow does not need to be adapted for each vendor or family of chips.

This chapter describes the design and the features of such a CGRA layer implemented on top of the Flash FPGA embedded in the mote architecture. After a review of the related works, the components of the template architecture are described. Details of selected processing elements and configuration controllers are given. The chapter ends with an evaluation of the resources consumed by the virtually reconfigurable overlay in various configurations.

5.2 Related work

5.2.1 Low-power coarse-grained reconfigurable architectures

Although it has been demonstrated that CGRAs achieve intermediate performance between ASICs and FPGAs, most of the architectures were dedicated to High-Performance Computing (HPC). More recent works also identify the benefit of CGRA for low-power and low-energy systems. This is the case of Hinkelmann [HZG10] who extended the datapath of a WSN CPU with a configurable array of functional units. It was shown that the approach is a good compromise in terms of chip area, performance and energy-efficiency when compared to pure software and pure ASIC approaches. In [KM11], several techniques are introduced to improve the energy-efficiency of CGRAs. The authors demonstrated that bitstream compression and cost-effective array architectures have a large impact on the power consumption of the design.

Several works focused on CGRAs for biomedical signal processing. This range of applications can highly benefit from very low-power implementations while being computationally demanding. The SYSCORE has been developed with the intention to reduce the energy consumption of biomedical signal processing tasks such as EEG or ECG analysis [PMB11]. Up to 62 % savings compared to traditional architectures could be reached by selecting appropriate function units and reducing the density of the interconnect. The Ultra Low-Power - Samsung Reconfigurable Processor also targets biomedical signal processing on battery-powered wearable devices [Kim+12]. The core supports several modes of operation (high performance to low power), which exploit power gating of CGRA cells to reduce the current draw of the device.

In general, these works showed that the efficiency of CGRAs in very low-power applications relies on two main aspects: reduced and optimized interconnect between processing elements and management of configuration bitstreams with reduced overhead. The power spent in interconnect and memory was already identified as critical for FPGA designs in Section 3.2. These optimizations also go in the sense of the area and memory restrictions of the proposed mote architecture.

5.2.2 Virtually reconfigurable hardware

Several research works investigated the possibility to experiment and evaluate novel FPGA or reconfigurable hardware architectures on top of an existing device. As FPGAs theoretically allow the implementation of a large variety of digital circuits architecture, they intrinsically permit the test of another FPGA architecture, which is a digital circuit by itself. The underlying component is here used for its great prototyping and emulation potential. Designing and experimenting new FPGA architectures requires however to have the corresponding CAD tools for the complete FPGA design steps such as synthesis or place and route.

Virtualization is also a good way to make a design compatible with different technologies. Like the Java Virtual Machine, a virtual architecture will use the same top-level configuration or instructions while the underlying implementation can vary from one platform to the other. This approach makes the design very portable but has a cost in terms of performance and resource consumption, which must be reduced as much as possible.

Hübner et al. proposed in [Hüb+11] a virtual FPGA architecture to enable dynamic reconfiguration on devices which are not natively suitable for this feature. The architecture of custom cells and routing blocks were directly mapped on top of the underlying FPGA. As routing resources consume a significant amount of logic in FPGA architectures, only a small virtual architecture (10x10 CLBs) could fit into a Microsemi ProAsic3 A3P1000 device (equivalent number of system gates as the Igloo used for HaLOEWEn and LPSIP).

Similarly, the ZUMA overlay is an open FPGA architecture [BL12] which can be emulated and tested on another FPGA. Different types of logic elements and interconnect components can be evaluated and compared on real hardware.

In [CA12], Capalija & al. implemented a coarse-grain FPGA overlay to accelerate critical code fragments running on a softcore processor. During the execution, the overlay is dynamically reconfigured to improve the execution times of changing data-flow graphs. The coarse-grained overlay is preferred to fine-grain logic since a specific task can be mapped faster on the array, enabling just-in-time compilation of hardware accelerators.

The QUKU [SBB06] is another example of such a CGRA overlay. The authors used the reduced configuration bitstream to reconfigure an array of processing elements for digital signal processing faster. The authors show with the example of an FIR filter that this approach reduces the size of the design by encouraging the sharing of resources while achieving intermediate performance between custom FPGA implementation and software.

A CGRA-based overlay for high performance digital signal processing has been proposed by McGettrick & al. in [MPB11]. With this approach, the authors intend to reduce the long reconfiguration times inherent to FPGAs. They demonstrated that the overlay architecture outperforms traditional CPU and DSP implementations but also custom FPGA implementations for certain benchmarks. The performance loss of the CGRA overlay is then compensated by the lower time necessary to reconfigure it.

5.2.3 Dynamic reconfiguration for Flash-based devices

Although Microsemi Flash devices support in-system-programming via a MCU controlling the JTAG (Joint Test Action Group) ports of the FPGA [Igla], this process is too resource and time consuming for a limited device such as a wireless sensor node. However, some related works have explored this possibility.

The DANCE framework [HWH13] covers the need for runtime reconfiguration on Flash-based reconfigurable SoCs (Microsemi SmartFusion). The authors point out that the amount of programmable logic available on such devices is not sufficient to hold several accelerators simultaneously (FFT with 65.78% cell occupancy and AES with 45.18%). Therefore, they propose a task distribution scheme optimizing quality of service metrics by assigning hardware accelerators based on spatiality. The same authors evaluated in [HWH12] the costs of the dynamic reconfiguration on the reconfigurable hardware fabric of the SmartFusion device (CES prototype mote). Reconfiguration delays are going up to 360 seconds, including erase, and verification steps. A power consumption of 28 mW was measured during this process. This delay is clearly critical when it comes to frequent reconfiguration and can only be applied for firmware update or complete change of functionality.

5.3 Template architecture

5.3.1 Overview

The proposed architecture is based on a combination of *domain-specific* processing clusters. Each cluster is a one-dimensional array of function units comprising pipelined configurable operators, data sources and memory elements. The structure is suitable for streamed data processing schemes where each step of a data flow graph is either time-multiplexed on the same cluster or mapped on a subsequent cluster. Each cluster is managed by its own configuration controller, which controls the configuration context of each element independently. At a higher level, each cluster controller may be reconfigured by a general control unit dispatching configuration data in each cluster according to the global application flow. Each cluster can be built with different types of processing elements so that they can be individually tuned for a specific range of algorithms. This method allows allocating resources that cannot be intrinsically shared within a task in a parallel execution pattern. For example, operators used for cryptography can be used in parallel to digital signal processing operators as they are rarely used simultaneously. A block diagram of this template architecture with two clusters is shown in Figure 5.2.

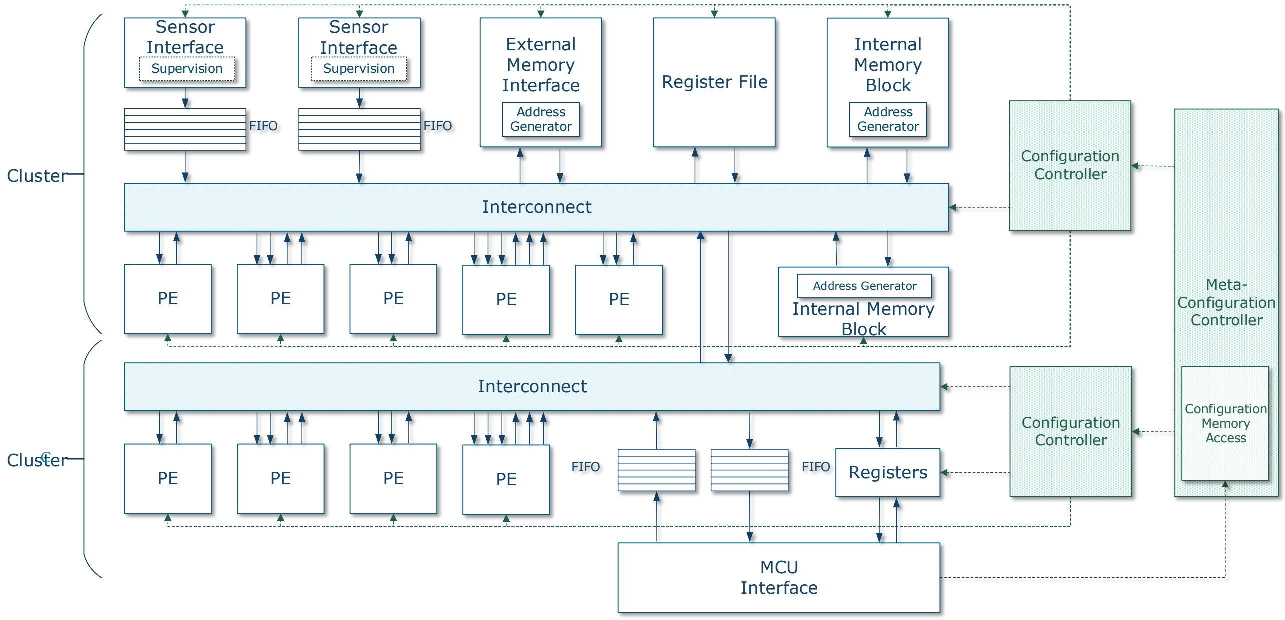


Figure 5.2.: Architecture of the coarse-grained overlay based on two clusters

Based on this approach, several levels of runtime reconfiguration can be achieved. Figure 5.3 shows the complete stack of reconfigurability levels enabled by the proposed infrastructure. At a low level, the functionality of each operator implemented on the FPGA can be modified by the changing the configuration bits driving them. When considering the configuration of all available operators as a whole, it describes a whole function, *i.e.* a succession of individual operations. This flow of of operations can be modified at a higher level by changing the content of the configuration vector. With a succession of reconfiguration, a complete algorithm flow can be implemented.

5.3.2 Producer-consumer transactions and interconnect

Within each cluster, elements can be connected to each others by following a *producer-consumer* transaction. *Producers* are delivering data to a shared interconnect line while *consumers* are reading and processing it. Each producer-consumer relationship is based on a single data word. Thus, a producer can deliver data to several consumers simultaneously, *i.e.* consumers use the same data, but a consumer cannot read data from several producers since it would result in a conflict in the data line access. A complete chain of producers-consumers can be configured in order to implement a virtual pipeline within the cluster.

Each configuration context describes a set of producer-consumer transactions between the different elements of a cluster. The number of transactions l_{context} determines the duration of the context, *i.e.* when the next configuration context will be loaded. A global counter keeps track of the current amount of transactions and will send a signal to the configuration controller when a threshold has been reached. A single configuration word is thus sufficient to describe a simple task if no modifications of the interconnect or memory access are necessary.

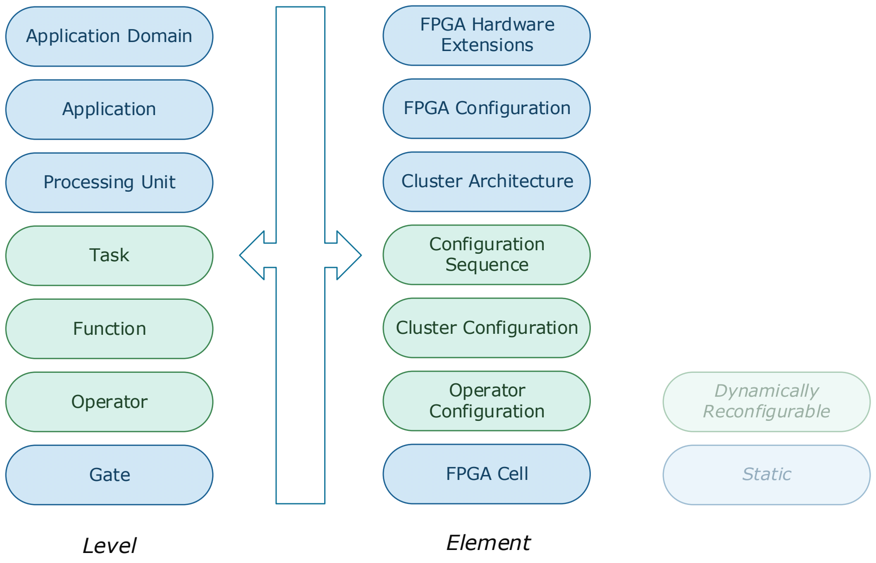


Figure 5.3.: Levels of reconfiguration enabled by the virtually reconfigurable architecture

	Parameter	Value - # bits	Function
static	$n^{i}_{\text{interconnect}}$	$\mathbb{Z} \cap [1,8]$	Number of lines, cluster i
	$n_P^{i,j}$	$\mathbb{Z} \cap [1,16]$	Number of producers, cluster i, line j
	$P^{i,j}$	-	Type of producers, cluster i, line j
	$n_C^{i,j}$	$\mathbb{Z} \cap [1,16]$	Number of consumers, cluster i, line j
	$P^{i,j}$	-	Type of consumers, cluster i, line j
	b_{bus}	$\mathbb{Z} \cap [4,32]$	Line wordlength
dynamic	$c_{Psel}^{i,j}$	$\log_2(n_P^{i,j})$	Active producer, cluster i, line j
	$c_{Csel}^{i,j}$	$P^{i,j}$	Active consumers, cluster i, line j

Table 5.2.: Configuration of the interconnect

Table 5.2 gives the detailed configuration parameters of the interconnect within a cluster of the overlay architecture[3]. Table 5.3 gives an estimate of the resource usage and maximum frequency supported by the interconnect for selected parameters[4].

5.3.3 Sensor interfaces

Sensor interfaces are pure producer elements. Data is acquired by accessing external components (ADCs) or by internal generation processes (Random Number Generators). Data is sent to connected consumers at a configurable delivery rate. Other building blocks of sensor interfaces include FIFOs to buffer data

---

[3] Similar tables are used thereafter to describe the different types of elements that can be instantiated in the architecture. Each table is divided between a static and dynamic part, the latter being reconfigurable at runtime. The second column describes the set of valid parameters in the case of static configuration while it gives the configuration size in bits in the case of the dynamic configuration. Static or dynamic parameters marked by a dagger $\{\cdot\}^\dagger$ are optional and correspond to features that might not be instantiated. In general, the notation of the parameters follows: n for an amount, b for a wordlength, v for a value and c for a selection of functionality.

[4] These values were obtained by synthesis for the Microsemi Igloo AGL1000V5 FBGA256 (chip used on both HaLOEWEn and LPSIP) with Synopsys Sinplify Pro H-2013.03M-1. The following tables giving synthesis results for other units are based on the same setup

Configuration $\{n_{\text{interconnect}}, n_P, n_C, b_{bus}\}$	Core cells (out of 24,576)	Max. frequency (MHz)
$\{3,4,4,8\}$	115 (<1 %)	81.3
$\{3,8,8,16\}$	460 (1.8 %)	65.1
$\{4,8,8,16\}$	614 (2.5 %)	65.1
$\{4,8,8,32\}$	1187 (4.8 %)	63.7

Table 5.3.: Resource consumption for one cluster interconnect with identical amount of producers and consumers on each line

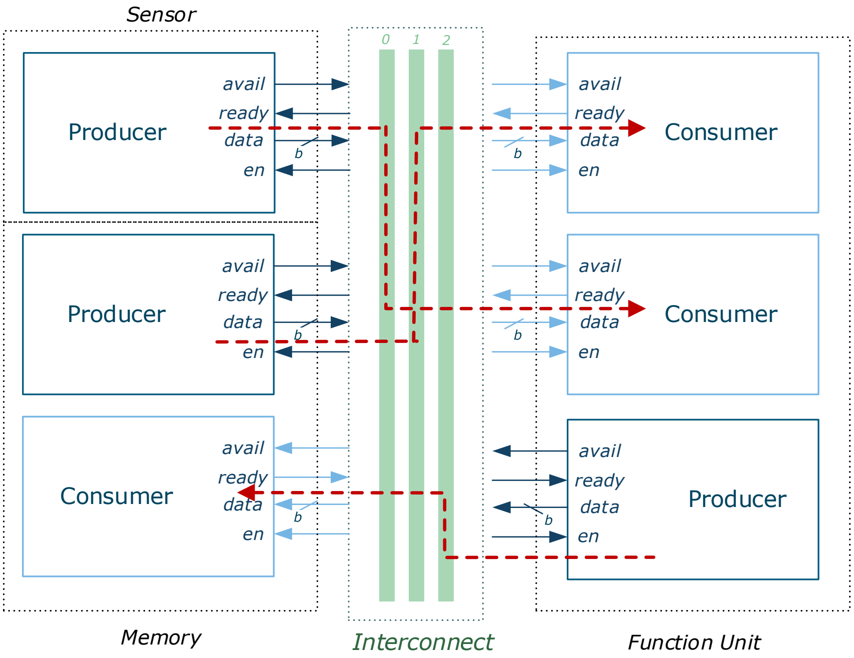


Figure 5.4.: Example of producer-consumer interconnect

	Parameter	Value - # bits	Function
static	$\text{FIFO}_{\text{depth}}$	$\{2^k, 0 \leq k \leq 8\, k \in \mathbb{Z}\}$	FIFO Depth
	b_{sensor}	$\mathbb{Z} \cap [4, 32]$	Sensor data wordlength
dynamic	$v_{thr}\dagger$	b_{sensor}	Supervision threshold
	$c_{\text{supervision}}\dagger$	2	Supervision mode
		sensor specific	

Table 5.4.: Configuration of sensor interfaces

samples while consumers are busy and sensor supervision units connected to an interrupt signal verifying if the data stay within a configurable range. This last feature was introduced by recommendations of the VTT institute for online sensor supervision techniques [Jan+10]. Each sensor unit may integrate custom configuration bits if special functionalities are supported.

5.3.4 Memory elements

Memory elements are simultaneously producers and consumers. In the first mode, data is read from the memory and delivered to connected consumers. In the latter mode, data is written in the memory. Both internal block memories and external memory chips are belonging to this category. In all cases, read and write addresses are generated by independent configurable sequencers. For $0 \leq i < l_{\text{context}}$, sequencers follow the equations :

$$a_{i+1} = \begin{cases} a_{base} + a_i + k_{step} & \text{if } a_{sub,i} \neq 0 \\ a_{base} + a_i + k_{skip} & \text{otherwise} \end{cases} \tag{5.1}$$

$$a_{sub,i+1} = a_{sub,i} + 1 \pmod{l_{subset}} \tag{5.2}$$

where a_i and $a_{sub,i}$ are indexes. The rest of the parameters is explained in Table 5.5. In general, this type of address generators gives a relatively large freedom for patterned memory access and array manipulations. When required, a bit reversal block can be inserted to generate addresses compliant with Cooley-Tukey FFT algorithms. The chosen sequencer architecture is suitable for typical DSP tasks where almost exclusively linear address patterns are used. Matrix manipulations typical from cryptographic algorithms can also be supported by the module. The access mode of the memory describes the format of the data: a data word can be split into two half words and reverse. This enables for example parallel processing of two smaller data items, parallel storage or description of complex numbers.

Address sequencers are commonly used in stream processing. An overview of system based on address generators can be found in [Her+02]. They could already be found in founding CGRAs such as RaPiD [EGF96] and PipeRench [Gol+99]. More recent works still include address generation units for DSP applications [IS11]. These components are important in CGRAs since they relieve the core architecture or the associated CPU from address computation. Using a single configuration word, addresses for a complete stream of data can be generated. This intrinsically reduces the need for reconfiguration.

Special types of memory elements are the FIFOs from the MCU interface. This type of memory only works in FIFO mode and does not include address generators. In order to avoid deadlocks, FIFOs keep producing the last element when empty and consumed data is discarded when full.

ROMs are another type of memory unit working as producer only. Igloo FPGAs include a 128x8 bits Flash ROM that can be used for this purpose. Alternatively, the content of block RAMs can be initialized via JTAG and used as a ROM, as it is also feasible on Xilinx FPGAs.

	Parameter	Value - # bits	Function
static	$\mathrm{mem}_{\mathrm{depth}}\dagger$	$[2^4, 2^{24}]$	Memory Depth
	$\mathrm{mem}_{\mathrm{width}}\dagger$	$\mathbb{Z} \cap [4, 32]$	Memory Width
dynamic	a_{base}	$\log_2(\mathrm{mem}_{\mathrm{depth}})$	Reference address
	k_{step}	$\log_2(\mathrm{mem}_{\mathrm{depth}})$	Primary incremental factor
	k_{skip}	$\log_2(\mathrm{mem}_{\mathrm{depth}})$	Secondary incremental factor
	l_{subset}	$\log_2(\mathrm{mem}_{\mathrm{depth}})$	Length of subset loop
	$c_{am}\dagger$	2	Access mode

Table 5.5.: Configuration of address sequencers

Configuration $\mathrm{mem}_{\mathrm{depth}}$	**Core cells** (out of 24,576)	**Max. frequency** (MHz)
2^8	77	53.1
2^{10}	104 (<1%)	45.1
2^{12}	124 (<1%)	42
2^{16}	168 (<1%)	34.1
2^{24}	260 (1%)	27.1

Table 5.6.: Resource consumption for one address sequencer

5.3.5 Processing elements

Processing elements (PEs) are mixed producers and consumers. Each input of a PE is a consumer while each output is a producer. A register is placed before each PE output to queue data going out. Thereafter, three of the main PEs acting as reconfigurable function units are detailed, *i.e.* the Multiply-Accumulate unit (section 5.3.5.1), the CORDIC unit (section 5.3.5.2) and the ALU tree (section 5.3.5.3). As each PE must be defined with a standard producer-consumer interface, custom processing elements can extend the architecture to implement functions that are not supported by the aforementioned cores, to reduce the costs of a function unit by selecting a dedicated core having less reconfigurability or to enhance the performance by selecting a specialized dedicated core. Each of these PEs is based on fixed-point arithmetic.

5.3.5.1 Reconfigurable multiply-accumulate unit

Multiply-Accumulate is the most typical operation in DSP applications. When no multiplier is available in the datapath of a CPU, implementing the multiplication in software requires a significant amount of time, which may be critical as this operation must often be repeated when processing long data streams. A hardware implementation is therefore of great interest as it can speedup the execution of this operation to a few clock cycles. As Igloo FPGAs do not embed integrated multipliers like Xilinx DSP48 slices, multipliers must be implemented using the programmable gate logic. This consumes a large amount of the available resources and requires particular care to design a multiply unit which combines both flexibility, performance and low area. The architecture of the function unit supports then a subdivision of the input data into two half parts. Each part can be considered independently in order to implement two identical operations in parallel or it can be reconstructed as a bit vector utilizing the full resolution or representing a complex number (real part as least significant bits (LSBs) and imaginary part as most significant bits (MSBs)). The unit has three inputs (consumers) and two outputs (producers). One output is linked to the internal multiplier while the second output is linked to the accumulator and main adder. The result of the final adder-subtracter can be linked back to the input of the multiplication so that this

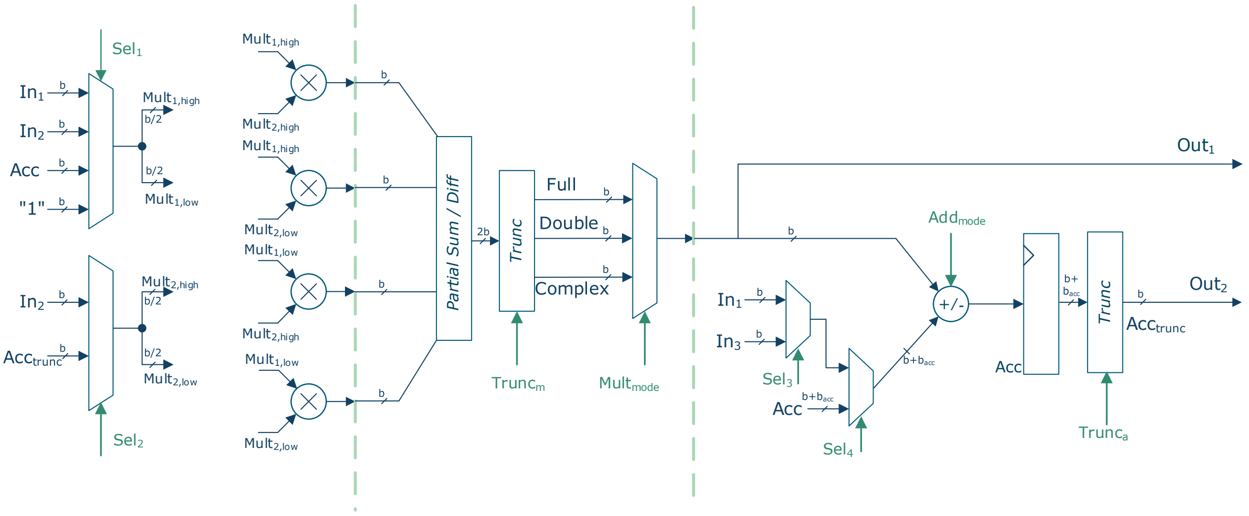

Sel_1
In_1
In_2
Acc
"1"
b
$\text{Mult}_{1,\text{high}}$
b/2
$\text{Mult}_{1,\text{low}}$
In_2
$\text{Acc}_{\text{trunc}}$
$\text{Mult}_{2,\text{high}}$
$\text{Mult}_{2,\text{low}}$
Sel_2
Partial Sum / Diff
2b
Trunc
Full
Double
Complex
Trunc_{m}
$\text{Mult}_{\text{mode}}$
Out_1
Add_{mode}
+/-
In_1
In_3
Sel_3
Acc
b+b_{acc}
Sel_4
Acc
Trunc
Out_2
$\text{Acc}_{\text{trunc}}$
Trunc_{a}

Figure 5.5.: Architecture of reconfigurable Multiply-Accumulate unit

	Parameter	Value - # bits	Function
static	b_{rMAC}	$\{8, 12, 16, 20, 24, 32\}$	Wordlength
	b_{acc}	$\mathbb{Z} \cap [0, 12]$	Accumulator bit extension
dynamic	c_{sel1}	2	Input 1 Select
	c_{sel2}	2	Input 2 Select
	c_{mmode}	2	Multiplication Mode
	c_{mtrunc}	$\log_2(b_{\text{rMAC}})$	Multiplication Truncation
	c_{sel3}	2	Adder Select 1
	c_{sel4}	1	Adder Select 2
	c_{amode}	2	Adder Mode
	c_{atrunc}	$\log_2(b_{\text{rMAC}} + b_{\text{acc}})$	Accumulator Truncation

Table 5.7.: Configuration of reconfigurable Multiply-Accumulate unit

operation can be executed at first. As all data inputs and outputs must have the same size, an internal logic for truncating is available. Figure 5.5 depicts the main architectural components of the unit.

The most common operations supported by the function unit include product, square, sum, difference, accumulate, sum-of-products, product-of-sums or multiply-accumulate, all in either full resolution, parallel half resolution or in complex form. The large majority of multiplication-based operations in digital signal processing is based on one of these operations, giving the unit full support for this range of applications.

5.3.5.2 CORDIC unit

CORDIC is a lightweight iterative algorithm used to implement standard arithmetic operations which are usually costly in terms of hardware resources if implemented as standalone cores. In addition to trigonometric operations, CORDIC is also useful to compute magnitude of complex numbers or less standard functions such as logarithm or square root when applying appropriate pre- or post-processing [KC11]. The number of iterations necessary to reach the final value with the best accuracy usually depends on the data wordlength. It is generally accepted that one iteration per bit is required when using the normal CORDIC algorithm. A major restriction of CORDIC is the limited convergence range due to the magnitude restriction that must be imposed on the inputs. In addition, accuracy issues may emerge

Configuration $\{b_{\text{rMAC}}, b_{\text{acc}}\}$	**Core cells** (out of 24,576)	**Max. frequency** (MHz)
$\{8, 0\}$	887 (3.6 %)	37.7
$\{16, 0\}$	2,487 (10.1%)	24.1
$\{16, 8\}$	2,598 (10.5%)	24.1
$\{32, 8\}$	7,449 (30.3%)	15.5

Table 5.8.: Resource consumption for the reconfigurable Multiply-Accumulate unit

	Parameter	**Value - # bits**	**Function**
static	b_{CORDIC}	$\{16, 20, 24\}$	I/O wordlength
	b_{eCORDIC}	$\{0, 2, 4, 6\}$	Internal wordlength extension
	n_{ext}	$\mathbb{Z} \cap [0, 10]$	Extra convergence iterations
	b_{frac}	$\mathbb{Z} \cap [0, b_{\text{CORDIC}} - 1]$	Internal fraction width
dynamic	c_m	2	Coordinates Mode
	c_r	1	Operation Mode
	$n_{iter}\dagger$	$\log_2(b_{\text{CORDIC}} + b_{\text{eCORDIC}})$	Number of iterations

Table 5.9.: Configuration of CORDIC

because of the limited dynamic range of the fixed-point representation. There are numerous methods to alleviate these restrictions with optimizations on speed, area or accuracy [Pon12]. The following solutions have been adopted based on the recommendations from [HHB91] and [KC11]:

- Internal wordlength extension to avoid overflows and underflows. Input can be scaled to fit to the convergence range using other units available within the architecture.
- Custom additional iterations to selectively speedup the convergence or increase the input range. Each iteration slightly improves the convergence range.

The extra convergence iterations come at the cost of an additional adder-subtracter and memory to store the values of θ_i corresponding to this extension. All algorithmic details corresponding to the operation of CORDIC and its range extension can be found in appendix B.2.

The resulting hardware implementation of the CORDIC unit is depicted in Figure 5.6. The execution is iterative, *i.e.* the same hardware resources are reused for each iteration. The unit owns a consumer and producer interface for each input x,y and z. The internal memory used to store the values of the iteration angle in each coordinates mode is implemented with a preinitialized memory block. Different coordinates correspond to different address ranges. No pre- or post-processing stage is included as it is usually the case for CORDIC units since these operations can be mapped on other function units of the architecture like the reconfigurable multiply-accumulate unit or the ALU unit. The resulting resource usage reported in Table 5.10 is given for a number of extra iterations n_{ext} equals to 8.

5.3.5.3 ALU unit

The most common PE used in CGRAs is the Arithmetical Logical Unit (ALU) since it is a lightweight but yet powerful and flexible function unit. When arranged in a regular structure, a set of ALUs can implement a simple sequence of operations with a single configuration vector. ALUs are arranged in successive stages whose outputs are interconnect with the inputs of a next level. The best structure and configuration for such an array of ALUs has been studied in the frame of two Master theses [Bor13; Zha12a]. While evaluating the mapping of different algorithms on different topologies, the tree structure turned out to

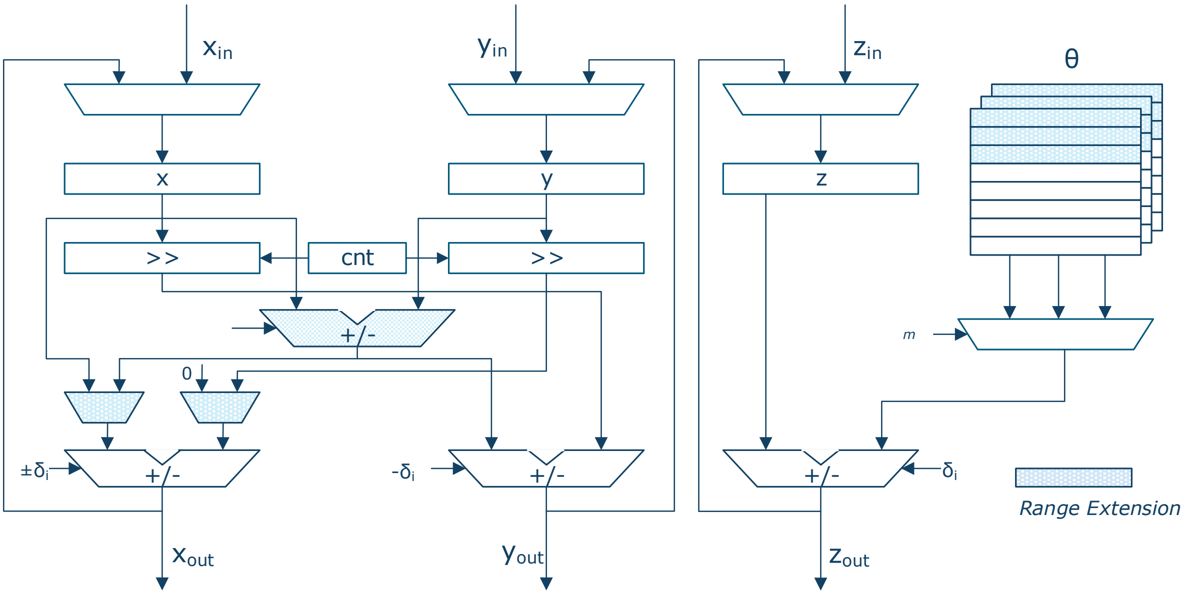


Figure 5.6.: Architecture of the CORDIC unit

Configuration $\{b_{\text{CORDIC}}, b_{\text{eCORDIC}}\}$	**Core cells** (out of 24,576)	**Max. frequency** (MHz)
$\{16, 0\}$	572 (1.7%)	35.0
$\{16, 4\}$	756 (3.1%)	31.1
$\{20, 4\}$	956 (3.9%)	27.9
$\{24, 6\}$	1,237 (5.0%)	23.5

Table 5.10.: Resource consumption for the CORDIC unit

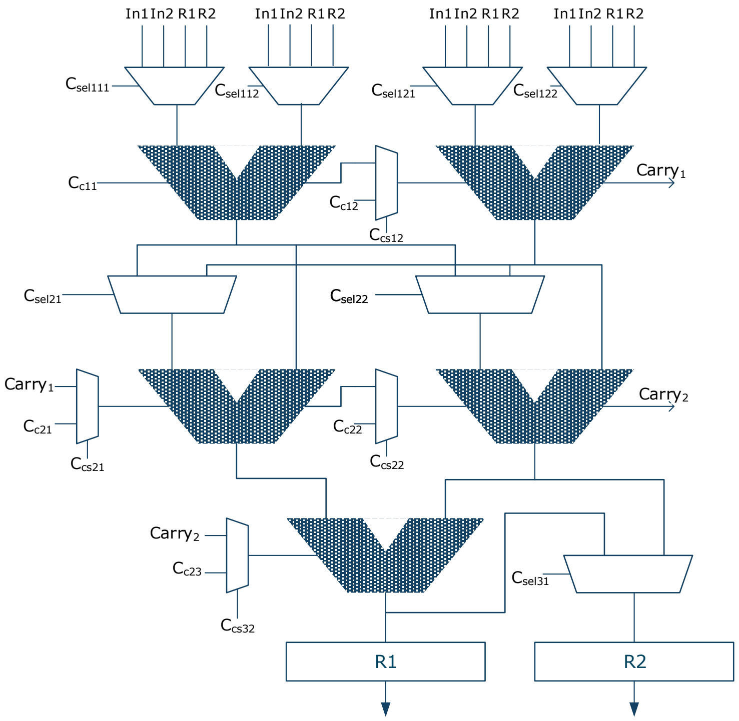


Figure 5.7.: Architecture of a $\{2,2,1\}$ ALU unit

	Parameter	Value - # bits	Function
static	b_{ALU}	$\{8,16\}$	I/O wordlength
	n_{ALU}	$\{1\}$, $\{2,1\}$,$\{2,2\}$,$\{2,2,1\}$,$\{4,4,2\}$	ALU topology
dynamic	$c^{i,j}_{\text{ALU}}$	5	ALU(i,j) operation code
	$c^{i,j}_{sel}$	$[1,3]$	ALU(i,j) input selection
	$c^{i,j}_{cs}$	1	Carry select
	$c^{i,j}_{c}$	1	Forced carry

Table 5.11.: Configuration of the ALU array

be the more efficient. Indeed, as the computation is progressing, intermediate results are combined into single results. This affects the bottom units of the array, which are used less frequently than the top ones. This also minimizes the amount of required multiplexers, which reduces the area required by the circuit. Five topologies can be selected depending on the application requirements. Each topology is described by the n-tuple $\{a_1, a_2, \ldots, a_n\}$ where each a_i represents the number of ALUs at stage i and n is the number of stages. The following topologies can be instantiated: $\{1\}$, $\{2,1\}$,$\{2,2\}$,$\{2,2,1\}$,$\{4,4,2\}$. The number of producer and consumer interfaces depends on the selected topology.

Each ALU has four configuration bits and a carry select bit. The operations implemented by the ALU include the logical functions AND, OR, XOR, NOT, shift and the arithmetical functions increment, decrement, addition, subtraction and shift. Two internal registers are available to locally store intermediate results.

Configuration $\{b_{\text{ALU}}, n_{\text{ALU}}\}$	**Core cells** (out of 24,576)	**Max. frequency** (MHz)
$\{16, \{1\}\}$	328 (1.7%)	71.0
$\{8, \{2, 1\}\}$	513 (2.0%)	55.0
$\{16, \{2, 1\}\}$	1080 (4.4%)	39.7
$\{16, \{2, 2, 1\}\}$	1598 (6.5%)	33.1

Table 5.12.: Resource consumption for the ALU unit

5.3.5.4 Other units

The architecture is not only limited to the aforementioned components. As long as the producer-consumer interface is implemented, any type of operator can be instantiated. Other types of units with a simpler functionality or less reconfigurability include:

- The *inter-cluster connector*: an input and an output data line are available for the exchange of data between two clusters.
- The *register file* is a set of registers used to store single intermediate results. It can be used in addition to the MCU interface registers if they are not sufficient (8x8 bits) or if they are connected to this cluster. The number of registers available within the register file is statically configurable, as the number of producer and consumer interfaces. Registers selected for write or read operations are dynamically configured with separated configuration vectors. The content of the registers can also be set through configuration data. This is useful to set single constants required for the execution of a task.
- Arithmetic units specialized for different application-domains such as cryptography or forward-error-correction were developed for the reconfigurable datapath of Hinkelmann's architecture [Hin11]. This notably includes a Galois-field multiply-accumulate array and a Galois-field inverter. Simplified versions of these units were integrated into the array. However, only a 32-bit wide interconnect is supported.
- *Division* or *square roots* are operations which are commonly required in DSP applications, but used at low rates. Implementing a full divider or square root unit is therefore very costly in terms of resources and should only be done if mandatory. Besides, the extended CORDIC unit already supports these operations over a large convergence range. However, if the CORDIC unit is oversized for the target application, specialized square root and division units based on iterative computation can be instantiated instead.
- As the ALU does not implement any comparison operation, a *comparator unit* has been implemented. This unit is useful to find extrema within a data stream of for sorting purposes.

One can imagine the integration of further components such as actuator and communication interfaces but these features were not considered in the frame of this thesis.

5.3.6 Reconfiguration layer

At runtime, a single configuration vector is usually not sufficient to describe a complex task. When referring to Figure 5.3, a context would correspond to the operation level. A complete function or task is thus defined as the sequence of configuration contexts, *i.e* operations, necessary to complete it. However, the utilization of address sequencers and patterned connections between units usually prevent the need for

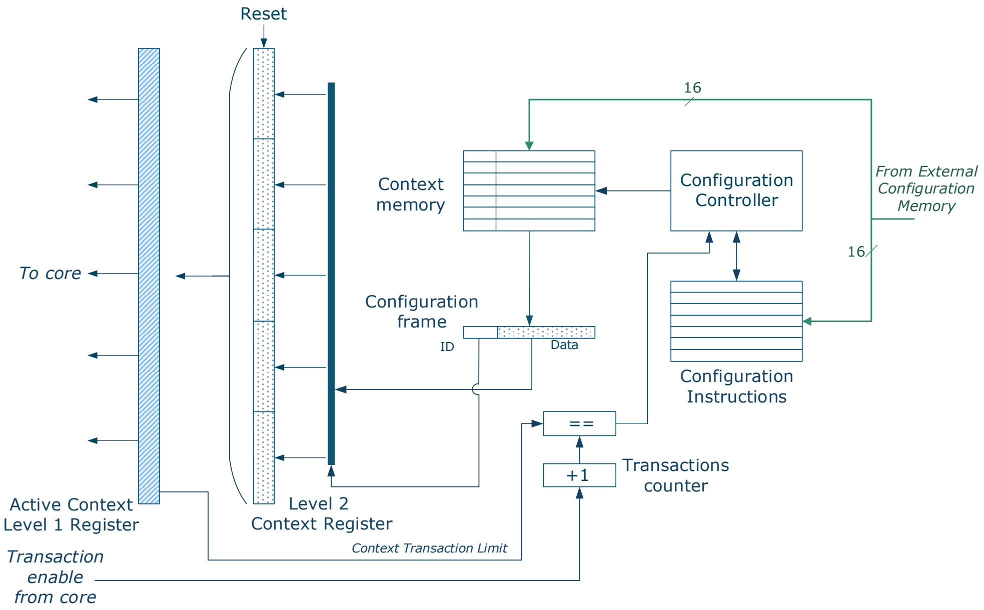

Reset
16
Context memory
Configuration Controller
From External Configuration Memory
16
To core
Configuration frame
ID
Data
Configuration Instructions
==
+1
Transactions counter
Active Context Level 1 Register
Level 2 Context Register
Context Transaction Limit
Transaction enable from core

Figure 5.8.: Block diagram of the configuration controller

a reconfiguration at each clock cycle. Indeed, stream processing is commonly based on vector operations, so that a configuration generally stays active for a number of clock cycles greater or equal to the number of vector elements. As the target platform has stringent resource constraints, the reconfiguration mechanism should not consume a large amount of logic and memory. The traditional multi-context approach of CGRAs consists in multiple configuration contexts saved in parallel to the active one [KM11]. A new context can be switched through a multiplexer within one clock cycle. However, this method has a large cost in terms of logic gates and memory cells that cannot be afforded here. As a reconfiguration at each clock cycle is not fully exploitable here, a solution where reconfiguration time is slightly longer but logic requirements are reduced is adopted.

The active configuration is stored into a register denoted *level 1* context register. This register controls all dynamic configuration bits required by the architecture. The next active configuration is prepared in a second register denoted *level 2* context register. The rest of the non-active configuration data is stored in a block memory. The level 1 context stays active as long as the number of corresponding producer-consumer transactions has not been executed. When this occurs and the level 2 configuration register is ready, the content of the level 1 register is automatically replaced and the transaction counter is reset. Otherwise, the processing elements stay idle until the reconfiguration process of the level 2 register has been finished, *i.e.* the corresponding content of the context memory has been read. The architecture of the configuration control is depicted by the block diagram of Figure 5.8. As each cluster of the overlay architecture is associated with a reconfiguration controller, each cluster can be controlled individually.

5.3.6.1 Difference-based reconfiguration

As all units are not necessarily modified during each context switch, reloading the totality of the context register would cost a significant amount of time and memory at each step. Indeed, PEs are rarely all active at the same time as the template architecture can only support a limited number of simultaneous producer-consumer transactions. This number is related to the amount of lines available in the cluster interconnect. For example, if three data lines are available, a maximum of three processing elements can be used simultaneously. As it is usually the case, the number of PEs is high compared to the number of interconnect lines. As a consequence, only a small part of the configuration word needs to be changed between successive producer-consumer transactions.

Size of configuration context (bits)	**Core cells** (out of 24,576)	**Max. frequency** (MHz)
100	1,547 (6.2 %)	68.3 MHz
200	1,881 (7,6 %)	67.8 MHz
350	2,151 (8.7 %)	67.3 MHz

Table 5.13.: Resource consumption for the configuration controller

Different methods have been proposed to reduced the unnecessary overhead of full reconfiguration such as the compression of configuration data [KM10] or the tag-matching mechanism from Hinkelmann [Hin11]. However, the latter method still relies on large configuration tables and multiplexers that are not area efficient. The method applied here is based on differences between two successive contexts. The level 2 context register is subdivided into an array of configuration frames that can be individually addressed. Therefore, the configuration data contains only the information that needs to be modified for the next configuration. A global default configuration is also available as a reference for evaluating the difference between two configurations. Thus, if the next context requires less modifications when compared with the default context as with the previous one, the second level register can be reset before the reconfiguration starts. The first context of the sequence is always based on the difference with the default context.

In order to save memory resources, the context memory is always tuned to use a single block RAM (256x18 used as 256x16 in order to facilitate data alignment). Items of the memory are configuration frames corresponding to the ID and the data of one subdivision of the level 2 context register. In order to maximize the memory utilization, configuration data should be designed to have a size which is a multiple of 16. Large configuration frames will be more memory efficient since they will require less IDs but they are less selective, so that they will be required more often, even if a small change in the configuration is necessary. On the other hand, smaller configuration frames will require more IDs but can focus on smaller parts of the context register. A trade-off can be found by selecting a length fitting with the average size of the configuration data for a single block.

Based on these considerations, the size of the configuration frame has been set to 32 including 4 ID bits. This corresponds to two memory items and a maximum of 384 bits for the context register. This also implies that a minimum of two clock cycles are required to switch between contexts, as loading one configuration frame takes two cycles. The reconfiguration process takes at most 32 cycles when all frames need to be reloaded. The context memory can store between 7 and 128 contexts, which is in general already sufficient to describe moderately complex tasks. Figure 5.9 shows the threshold amount of frames requiring reconfiguration between successive contexts so that difference-based reconfiguration is faster and less memory consuming than full reconfiguration. Although 32 bits frames seem more efficient than 16 bits, it is also more likely that the share of frames to reconfigure is higher when using 32 bits than with 16 bits. In general, the threshold is relatively high (> 70 %), so that difference-based reconfiguration is almost always advantageous. Moreover, the default reconfiguration context offers a second chance to go below this threshold if two successive configuration contexts have too many differences.

Table 5.13 reports the amount of resources consumed by the configuration controller for different sizes of context, including the two levels of configuration registers. In each case, two blocks RAM are used to store the operators configuration data and the configuration instructions respectively. The amount of logic cells used is already non negligible. Including additional registers for caching reconfiguration data would unnecessarily increase the complexity and the resource usage of the controller and would make the reconfiguration overhead non bearable anymore for an implementation on a resource-constrained FPGA.

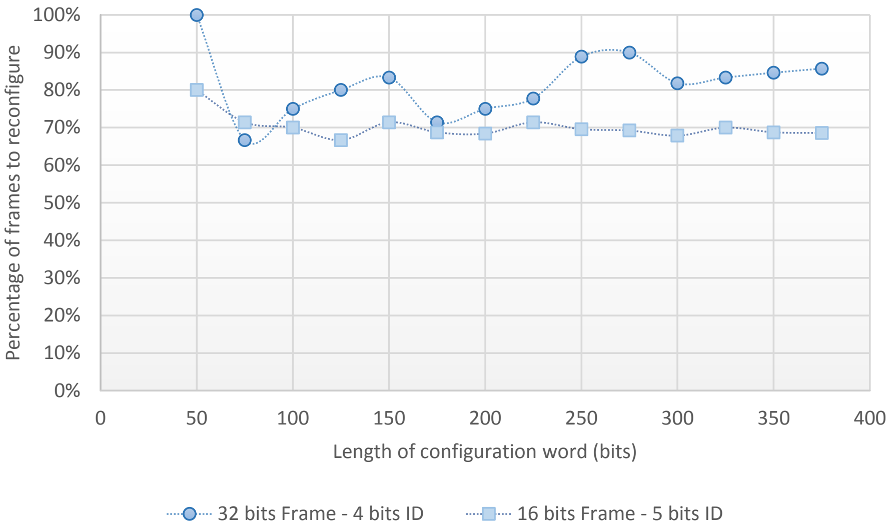


Figure 5.9.: Threshold percentage of the number frames to reconfigure to make difference-based reconfiguration more efficient than full reconfiguration

5.3.6.2 Context flow

During the execution of a task, it is likely that contexts or sequences of contexts must be repeated. DSP or cryptographic algorithms are often based on inner loops or on the repetition of the same operation pattern. In order to eliminate the unnecessary repetitive portion of this flow of contexts, a hierarchical approach has been adopted. The execution sequence of the contexts stored in the context memory is determined by a a set of instructions associated with the task. As the execution of contexts does not always follow a linear pattern, a programmable configuration controller has been implemented to control this sequence. It has been shown with a similar approach in [Wan+13b] that reconfiguration time can be reduced. The set of available instructions is reported in Table 5.14. Each instruction is 16 bits wide and include a parameter whose functionality depends on the instruction type. Like the context memory, the instructions are stored in a block RAM (256x16). In addition to flow control, additional instructions controlling the global status of the processing unit have been included.

The EXE instruction defines how many context frames should be read in the context memory for reconfiguration. The SENSE instruction is an EXE instruction where the core is allowed to switch in a low-power sleep mode is no sensor units are currently active, *i.e.* acquiring data from an external component. The LOOP instruction is automatically followed by a context based on the difference with the default context since the content of the previous context can differ. The RESET instruction reset components attached to the cluster. Interrupt flags can be set by the INT instruction while the WAIT instruction is maintaining the processing idle as long as the operation has not been resumed by the processor interface.

An example of the reconfiguration flow is illustrated by Figure 5.10. The instructions describes a filtering task. At first the coefficients are loaded in a local memory. Then a sequence where data is sampled, filtered and the units are reset is repeated 16 times. The EXIT instruction indicates the end of the task. The current instruction is SENSE, which requires the modification of two configuration frames (0 and 4).

This set of instructions has been developed in order to access all the control mechanisms available in the template architecture. This includes the difference-based reconfiguration (EXED), low-power and sensor management (SENSE), interface with the MCU (INT), and basic control operations (RESET, LOOP, WAIT, EXIT).

Instruction	Parameter	Function
EXE	size of context	Execute the current context once (difference with previous)
EXED	size of context	Execute the current context once (difference with default)
SENSE	size of context	Execute one context with sleep enable
LOOP	$\{n_1, n_2\}$	Loop the n_1 next instructions n_2 times
INT	type	Set an interrupt flag
RESET	type	Reset selected internal registers
WAIT	none	Wait until execution is resumed
EXIT	none	End of the context sequence

Table 5.14.: Instructions of the configuration controller

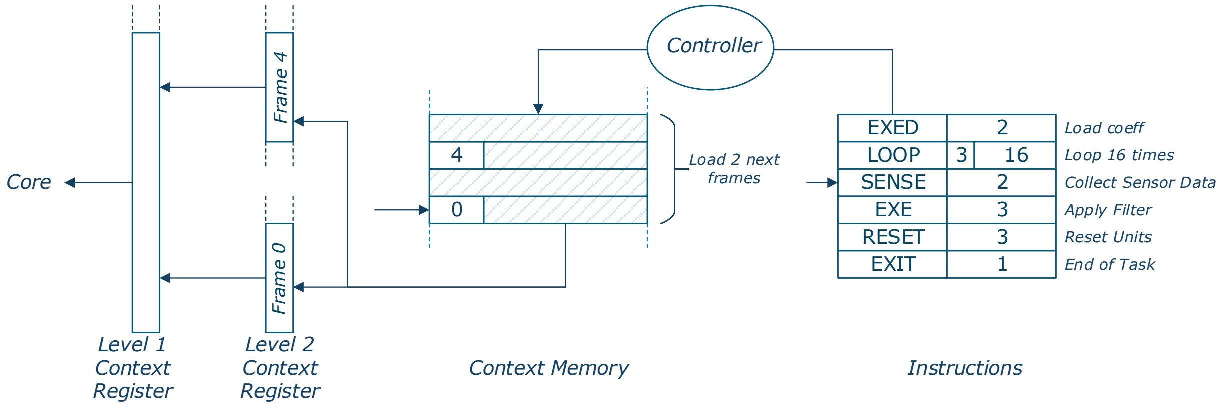


Figure 5.10.: Example of reconfiguration process for a filtering task

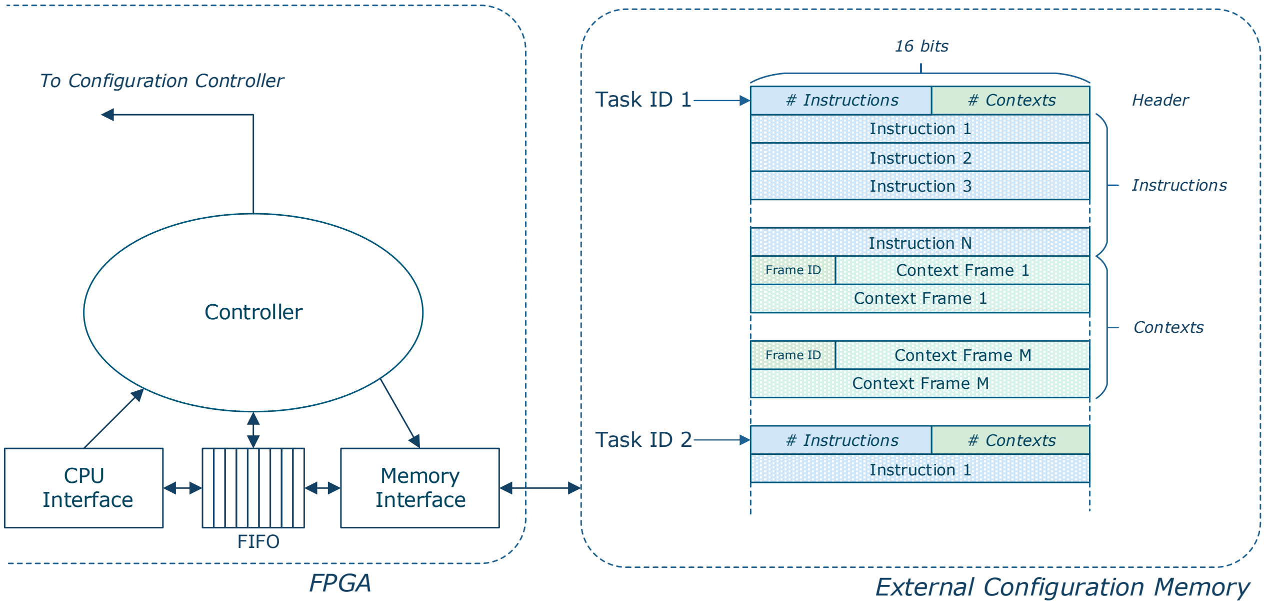


Figure 5.11.: Task-level reconfiguration

5.3.6.3 Meta-reconfiguration

For each cluster, the complete reconfiguration bitstream comprises the content the configuration memory plus the configuration instructions. This set of configuration data corresponds to the task level in Figure 5.3. In order to implement a different task, the content of the corresponding block memories must be replaced. A top-level configuration controller is available for this purpose. As this unit is reconfiguring a lower level reconfiguration unit, it has been named *meta*-configuration controller. A single top-level controller is available to reconfigure the configuration memories of each cluster.

When a task has been completed, an interrupt is sent to inform the main processor. If further tasks need to be accelerated, a command is sent to the meta-configuration controller to load the corresponding configuration data from an external non-volatile memory where it is stored. Each task configuration is identified by an ID which corresponds to its address in the configuration memory as depicted by Figure 5.11.

If no external configuration memory is available, the configuration data has to be loaded every time by the main processor. On the HaLOEWEn platform, configuration data can be loaded from an external parallel nonvolatile FRAM memory chip. On the LPSIP, configuration data is stored in a serial EEPROM. As the serial transfer of the data might cost a long time for reconfiguration between tasks, it can also be loaded into the parallel SRAM during an initialization sequence.

The content of the configuration memory is modified by using a special access mode of the MCU interface. Data is transfered from the MCU to the memory by transiting in the FPGA. A bidirectional FIFO is used for this purpose.

5.3.7 Clock and power management

As highlighted in chapter 4, the FPGA can be clocked by an external oscillator at the cost of a large power consumption overhead. Alternatively, the internal ring oscillator used to control the sleep mode can be used. Each of the clocks is connected to the internal clock tree of the FPGA and can be used by any component. A global timer unit accessible by all processing elements has not been implemented. As a consequence, each sensor interface must implement its own timer in order to regulate the sampling rate.

A special controller has been developed to manage the sleep cycles of the platform. In general, the system will switch automatically in Flash*Freeze mode if the following conditions are fulfilled :

1. No timer of an active sensor interface is running with the external clock
2. The Flash*Freeze pin is asserted
3. All sensor interfaces are idle
4. All clusters are inactive or running the SENSE or WAIT instruction

In Flash*Freeze mode, the clock generated by the internal ring oscillator is gated for all processing elements, memories and other external interfaces.

5.4 Evaluation

This section gives examples of overlay architectures that can be instantiated using the proposed template. Three architectures depicted in Figure 5.12 were selected:

- **(a)**: The first architecture integrates three 12-bit ADCs and three main processing units within one single cluster. This cluster is specialized for signal processing tasks where the input signals must be combined. An example of such an application is detailed in Chapter 8 where a sensor node for the condition monitoring of an electrical motor is implemented.
- **(b)**: The second architecture has two separated clusters, each with a 12-bit ADC interface. In this case, the two sensor data streams can be processed in parallel. This is useful if sensors requiring different types of data processing are used.
- **(c)**: The third architecture has two clusters customized for different application domains. While the first cluster is specialized for simple signal processing, the second cluster integrates units for cryptography and general-purpose processing.

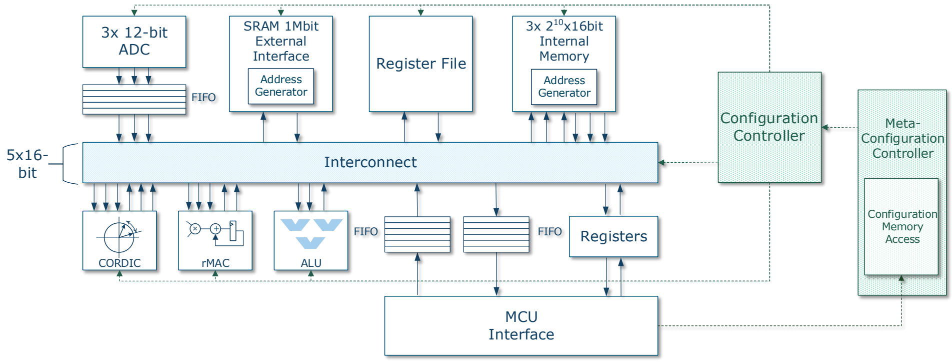
5.4.1 Resources

The resources usage for the three architectures is summarized in Table 5.15. As a matter of comparison, the overlay has also been synthesized for the Spartan6 LX16 FPGA. This shows that the architecture is not only implementable on the Igloo FPGA, but also on other families of devices. It can be noticed that two clusters are already filling the Igloo FPGA. An architecture based on three clusters is therefore not worth considering since it would require the reduction of the amount of PEs in each cluster and by extension their potential to implement complex tasks. However, one cluster is already sufficient for a large range of applications, so that this restriction is not critical.

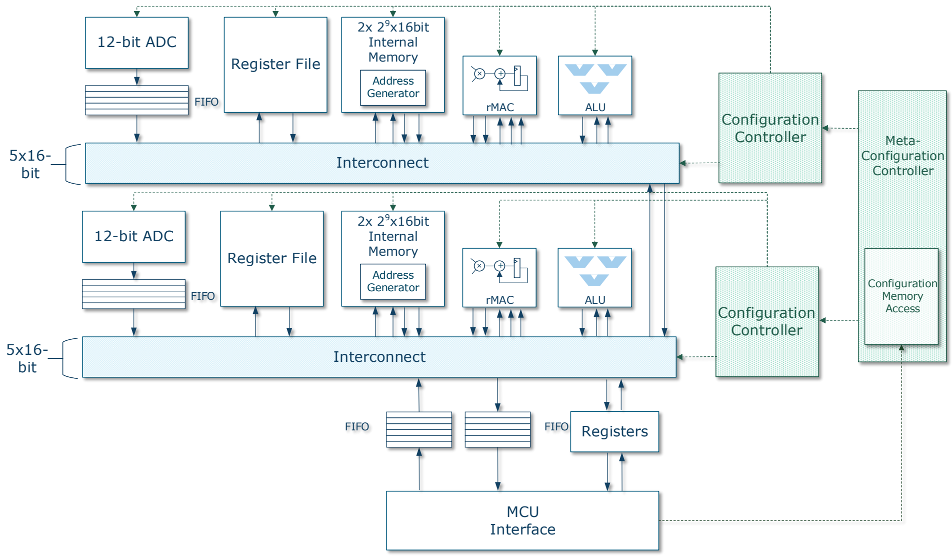
Secondly, although the amount of internal memory used by architecture **(a)** is not high, most of the block memories are already used. This configuration is for instance not sufficient to implement a 1024-FFT for which the external memory should be used. As a consequence, the size of the configuration vector increases because a large part is dedicated to the address generators of the external SRAM.

In terms of maximum frequency or critical path, all architectures are limited by the reconfigurable multiply-accumulate unit. The value stays nevertheless over the frequencies which are commonly used for applications running on wireless sensor nodes, *i.e.* under 20 MHz.

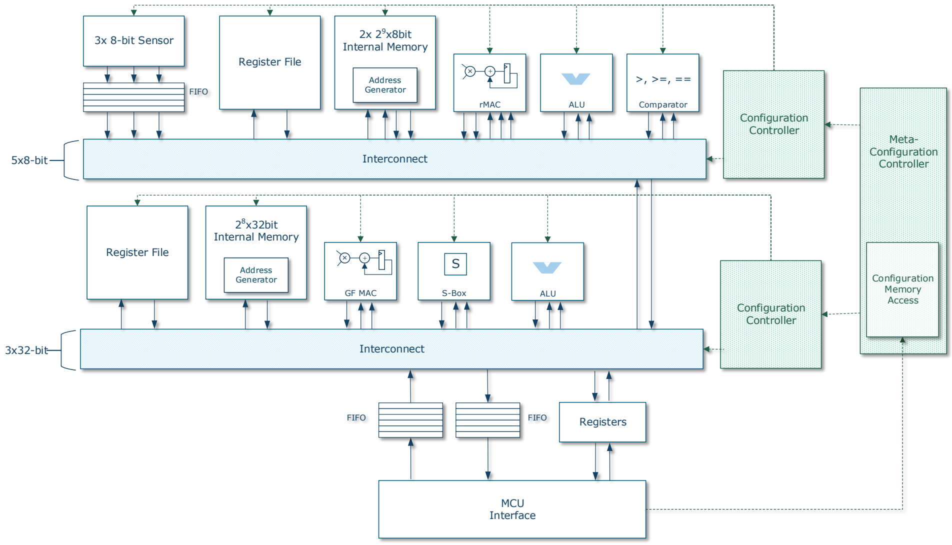
For the Spartan6 FPGA, the general occupancy stays relatively low, leaving more space for subsequent clusters. Again, the availability of DSP slices makes the design smaller and faster.



(a) Architecture customized for signal processing



(b) Architecture customized for a dual sensor interface



(c) Architecture customized for multi-domain processing

Figure 5.12.: Customization of the architecture template for three types of applications

		Arch. (a)	Arch. (b)	Arch. (c)
Igloo	**Cells**	14,664 (59.7 %)	19,906 (80.1 %)	19,192 (78.1 %)
	Memory blocks	20 (62.5 %)	17 (53.1 %)	14 (43.7 %)
	Max. frequency	22.1 MHz	22.2 MHz	30.2 MHz
Spartan6	**Slices**	870 (38.2 %)	1,020 (44.7 %)	1,168 (51.3 %)
	Memory blocks	11 (34 %)	13 (40.6 %)	16 (50 %)
	Max. frequency	63.4 MHz	65.1 MHz	97.44 MHz
	Size of context	292 bits	{172,182} bits	{135, 88} bits

Table 5.15.: Resource utilization of the example architectures

5.4.2 Performance

In order to estimate the performance and the overhead introduced by the reconfigurability of the design, the example scenarios from Chapter 4 have been ported on the architecture **(a)**. The bar charts in Figure 5.13 shows the differences in terms of processing time and energy consumption between an implementation on the developed overlay and a direct implementation where only the considered algorithm can be run.

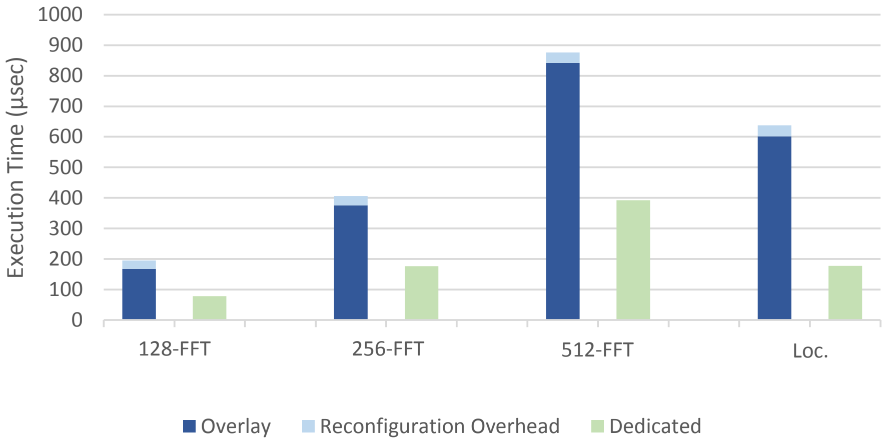
On the overlay, the execution time is in average 2.6 times longer than on the dedicated architecture. During this time, the reconfiguration process, *i.e.* the time required to load the configuration data in the internal memory, is taking only a short amount of time (5% in average). This share decreases when the complexity of the algorithm increases since the overlay stays in a longer time in a processing mode where no reconfiguration is required. Similarly, when several configurations are executed sequentially, the time overhead of the reconfiguration stays low since the next reconfiguration can be preloaded from the external configuration memory while the current one is executed.

In general, the reconfiguration time (in the range of microseconds) stays much shorter than the ones of classical FPGA reconfiguration schemes (in the range of several milliseconds up to several seconds).

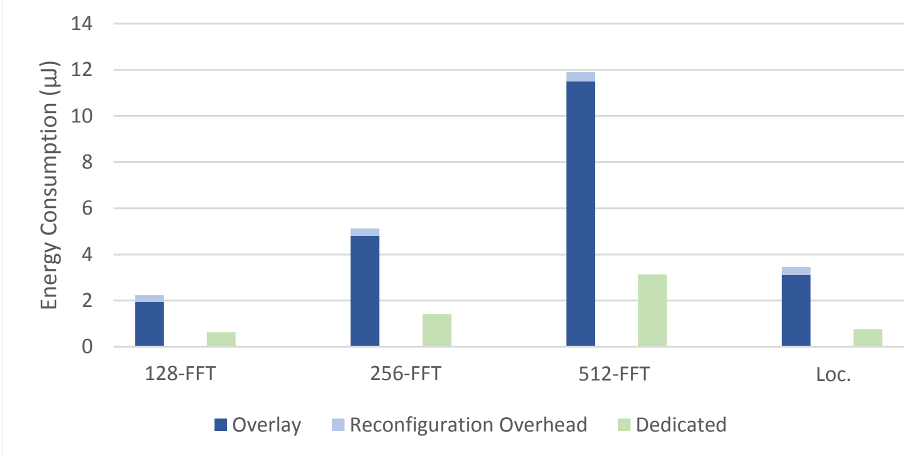
In terms of energy, the consumption is about 3.8 times higher using the reconfigurable overlay when considering the processing time only. The absolute value stays still in a range which is much lower than equivalent software implementations. The overhead is low for the considered algorithms because the dedicated implementation cannot fully exploit operation level parallelism. Indeed, because of the FPGA resources restrictions, the dedicated implementations are centralized around one or two operators used sequentially. As the overlay architecture is based on a similar scheme, the differences in time and energy consumption are not substantial.

Again, this value must be balanced with an evaluation on a longer term, *i.e.* by taking the sampling period into account. The graph 5.13c shows the average power consumed by both overlay and dedicated architecture when sampling a sensor signal at 1 kHz and processing it with an FFT. As it can be observed, the overall power consumption stays almost the same as the processing time is negligible in comparison to the time necessary to sample all the data. Thanks to the low-power sleep mechanisms of the Igloo FPGA, an average relative increase of 3.1 % of the power consumption is observed with the overlay. The absolute increase is negligible when compared to equivalent MCU or SRAM-based FPGA implementations. For such a scenario, the overall overhead of the reconfiguration process stays very low as it is very short and needs to be executed only once (the content of the internal block RAM and register is not lost when the FPGA is switched in Flash*Freeze mode).

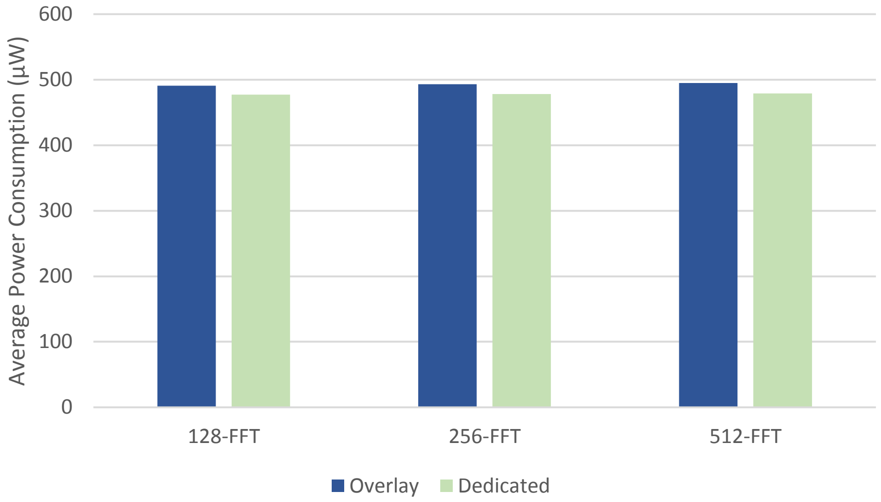
This evaluation is focusing on the execution of single tasks on the overlay architecture. A more complete analysis where multiple tasks are run in a sequential fashion would demonstrate the full potential of the reconfigurability in a better way. However, a relevant scenario can only be considered in the frame of real-word applications where a precise set of tasks needs to be implemented. Such examples and the



(a) Processing time



(b) Processing energy



(c) Average power consumption (sampling included)

Figure 5.13.: Performance evaluation of the overlay architecture

corresponding evaluation are given in the part of this thesis dedicated to applications (Chapter 7 and Chapter 8).

5.5 Conclusion

WSNs need a high level of adaptability to comply with their low accessibility and always changing application requirements. By making the hardware accelerators available on each sensor node reconfigurable at runtime, a whole new range of possibilities is open. Even though Flash-based FPGAs do not support well a core reconfiguration, one can exploit the gain provided by their lower power consumption to implement more generic hardware architectures which can be reconfigured at a higher level of abstraction. However, the overhead introduced by this genericity must stay in a reasonable range, so that the architecture still fit on a resource-constrained device.

By reusing the architectural concepts of CGRAs, the proposed overlay architecture for the Igloo FPGA combines both low reconfiguration costs with general-purpose flexibility, allowing a reusing of arithmetic operators across multiple algorithms. Depending on the final application, the overlay can be parameterized in many different ways, so that a specialization for a specific sensor interface or a specific type of arithmetic domain is still possible. The lightweight reconfiguration mechanisms reduce both the amount of configuration data and the size of the necessary control logic. In the end, the complete architecture is almost filling the considered device, so that the maximum of the available logic is used and resources are not wasted. Such scenario often occurs in FPGA-based designs where the selected chips are often oversized with respect to the target application.

In conclusion, runtime reconfiguration at task level has been virtually enabled on a Flash-based FPGA. The scheme go beyond the state-of-the-art where virtual reconfiguration on Flash-based devices was only evaluated for virtual FPGA architectures [Hüb+11], or coarse-grained overlay architectures were used for high-performance computing [SBB06; MPB11; BL12]. The design combines the best of both approaches in a lightweight template tailored for WSN applications.

6 Tools for Generation and Online Dissemination of Dynamically Reconfigurable Hardware Accelerators

Convenient programmability across several orders of magnitude of energy consumption and data processing requirements is a worthy research goal for pervasive computing

G. J. Pottie and W. J. Kaiser, "Wireless Integrated Network Sensors". In: *Communications of the ACM*, May 2000, Vol. 43, No. 5

Contents

5.1 Introduction 73

5.2 Related work 76

5.2.1 Low-power coarse-grained reconfigurable architectures 77

5.2.2 Virtually reconfigurable hardware 77

5.2.3 Dynamic reconfiguration for Flash-based devices 78

5.3 Template architecture 78

5.3.1 Overview 78

5.3.2 Producer-consumer transactions and interconnect 79

5.3.3 Sensor interfaces 80

5.3.4 Memory elements 82

5.3.5 Processing elements 83

5.3.5.1 Reconfigurable multiply-accumulate unit 83

5.3.5.2 CORDIC unit 84

5.3.5.3 ALU unit 85

5.3.5.4 Other units 88

5.3.6 Reconfiguration layer 88

5.3.6.1 Difference-based reconfiguration 89

5.3.6.2 Context flow 91

5.3.6.3 Meta-reconfiguration 93

5.3.7 Clock and power management 93

5.4 Evaluation 94

5.4.1 Resources 94

5.4.2 Performance 96

5.5 Conclusion 98

The FPGA overlay template introduced in the previous chapter fixes the basis infrastructure for hardware reconfiguration on the wireless sensor node at runtime. Despite the restrictions introduced by the unalterable communication infrastructure or the format of the operators, the design space of the template is sufficiently large to customize architectures for specific applications. However, parameterizing the template demands a significant effort as each component has a set of settings that must be individually adapted. As the nature and the organization of the operators is heterogeneous, each component must be meticulously tuned to ensure that each unit fits to each other. More importantly, the content of the configuration memory describing the operation of the hardware accelerator cannot be defined manually without a deep knowledge of the reconfiguration process and the cluster architecture. As the format of the configuration data varies according to the implemented architecture, tools abstracting and automating these tasks are therefore required to make the static and dynamic configuration of the architecture more accessible, even to users with no knowledge of hardware description languages (HDL). For this purpose, a graphical interface can be considered as the most intuitive solution, which gives in addition a visualization frame of the developed architecture and associated configurations. Developing such a software configuration tool is also a good way to centralize and standardize the code for all architectural elements implementable in the design. A graphical configuration tool for the FPGA overlay architecture is therefore introduced in the section 6.2 of this chapter[1].

Once configuration data has been generated, another important aspect requiring a generic approach to enhance the accessibility of the hardware accelerator is the remote management of the dynamic reconfiguration. Indeed, the task-level reconfigurability of the hardware accelerator can only be fully exploited if the software service controlling it is not static. Once deployed, wireless sensor nodes are usually hardly accessible and manual reloading of configuration data is not desirable. Furthermore, the small size of the task description on the reconfigurable hardware is compatible with the low bandwidth of WSNs so that a dissemination of the configuration data on the wireless link is worth considering. The section 6.3 addresses these aspects by describing the operation of a middleware service handling the management of configuration data at runtime[2].

6.1 General overview

The set of configuration tools has been unified in a software program named **(GECO)**[2], standing for Graphical Environment for COnfiguration and GEneration of bitstreams for a COarse-grained dynamically reconfigurable architecture. It has be developed in the Java programming language with the standard widget tool (SWT) library for graphical user interface (GUI) components [Swt]. A general overview of the proposed tool chain is depicted in Figure 6.1. At first, the static configuration of the architecture is elaborated based on the specifications of the hardware platform and the application profile. This allows selecting the appropriate sensor and memory interfaces or processing elements. This results in a set of HDL files that can be used to generate a FPGA configuration bitstream through the corresponding synthesis tool. Then, the architecture can be programmed on the platform. The details of this tool are given in section 6.2.2.

In a second step, the architecture must be configured with the appropriate tasks implementing the application functionalities. The second purpose of the GUI is to parametrize all units for each configuration context. This can be realized by loading the configuration data corresponding to pre-existing task descriptions or by creating a custom task. Section 6.2.3 reports the working principle of this tool.

At last, the task must be loaded and activated into the sensor node. Even if this can be done statically in software, the dynamic reconfiguration feature is not fully exploited. A generic framework is then described to remotely load and schedule hardware accelerated tasks on the sensor node at runtime. This functionality is implemented as a software service running closely to the operating system of the platform.

---

[1] The work presented in this section is related to the publication [PG12]

[2] The work presented in this section is related to the publication [PG13b]

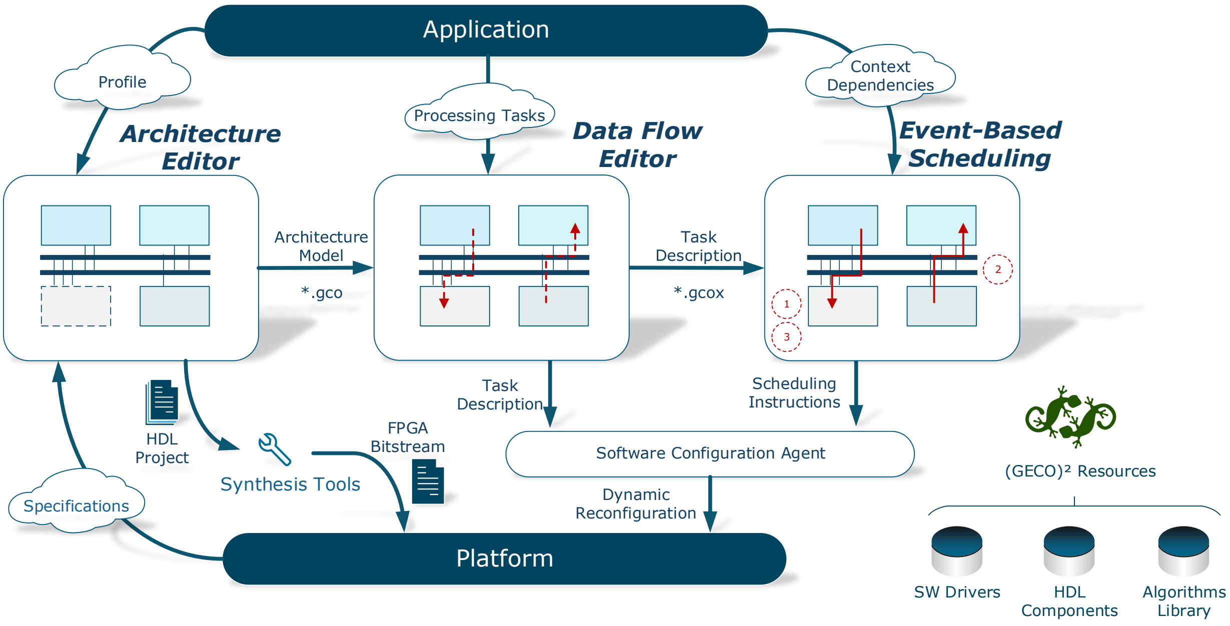


Figure 6.1.: General overview of the configuration tools

It joins the networking layer with the low-level access to the configuration memory of the hardware accelerator. This aspect is detailed in Section 6.3.

6.2 Graphical configuration interface

6.2.1 Related work

Design software of programmable hardware vendors all include graphical user interfaces for fast and user-friendly parameter settings. From the low-level FPGA editor to the customization of IP cores or SoC architectures, *e.g.* with the Xilinx Platform Studio [Xila] or Microsemi Libero SoC [Libb], these interfaces abstract the underlying implementation and prevent an error-prone parameterization of components in the source code. Even with no HDL knowledge, optimized source files describing complex hardware architectures can be generated in a short amount of time, thus reducing the total design time. Similarly, the graphical design interface introduced in this chapter intends to make the design of the reconfigurable hardware accelerator more accessible, even for traditional WSN developers with no particular skills for FPGA or digital logic design.

Parameterizable processor architectures are often accompanied by a configuration software, *e.g.* the LEON3 processor [Aer], or even special semantics, *e.g.* The Incredible Machine (TIM) from Silicon Hive [Okm11] or the CGRA description language from [Cha+08], to describe the desired system settings.

Some academic projects are also supported by graphical design interfaces. A set of programing and configuration tools has been created for the MorphoSys coarse-grained reconfigurable architecture [Sin+00], the Butter coarse-grained architecture for multimedia [BGN08] or the AMDREL low-energy FPGA architecture [Sou+07]. The virtual FPGA overlay proposed by Hübner et al. [Hüb+11] is supported by a graphical configuration editor and bitstream generator. Individual configurable logic blocks (CLBs), switching matrices and I/O cells can be manually configured to implement a custom functionality. Bitstreams for the virtual FPGA are generated accordingly.

Most of advanced CGRAs are extended with a compiler to map the application on the processing elements [TSV07]. The description language can be specific to the CGRA or derived from a standard embedded programming language such as C. As the development of such as tool requires a significant effort because of the heterogeneity and flexibility of the template, it was excluded from the scope of this thesis. However, as the developed architecture still requires a comfortable and automated way to generate configuration data, the GUI presented thereafter was introduced to set both static and dynamic parameters of each individual components used in the design. This low-level configurability, comparable to the one of an FPGA cell editor, is rare to find in the related work, although it allows a custom optimization of a CGRA configuration.

The challenge of rapid and facilitated design of digital logic on FPGA-based wireless sensor nodes has been also emphasized by Völgyesi & al. in [Vol+10]. The authors extended a graphical development environment for the component-oriented operating system for WSN TinyOS with the possibility to map components directly on an FPGA. As component-based design fits also well to hardware design, suitable modules can be arbitrarily implemented in hardware or in software if the appropriate interfaces are available. After a graphical entry, the tool is able to generate software and hardware code corresponding to the described application. The authors also emphasized the need for a unified hardware-software framework where components can be arbitrarily interchanged. Unfortunately, the tool stayed at a low development advancement, in particular with regards to the integration of VHDL components.

6.2.2 Architecture editor

The architecture editor is the first step towards the deployment of the hardware accelerator on the target platform. The template architecture defined is Chapter 5 is customized for the desired application, *i.e.* all *static* parameters of the architecture and processing elements are set. In Chapter 5, these parameters were listed in tables corresponding to selected components. As a result, a compilation of HDL files that can be used for FPGA synthesis, simulation and validation is created.

The tool has been designed in a generic way so that it is not only usable with the HaLOEWEn and the LPSIP platforms presented in the frame of this thesis. In particular, the tool is also able to generate HDL files that can be used with other types of FPGAs. As a consequence, the specifications of the target platform are first required. This includes the nature of the target device, but also the nature all external components that are going to be used for the application, *e.g.* sensors, memories or communication interfaces. Secondly the main processing requirements of the application must be identified in order to select the appropriate processing and memory elements. An application *profile* summarizing the desired functionalities or determining the specialization domain must be established. For this purpose, most of the approaches in related work are relying on the analysis of existing code written in C/C++ or assimilated. Critical loops or functions are automatically identified and the best-suited processing elements are selected. As the number of available processing elements is relatively low for the developed architecture, the choice of the processing elements is left here to the decision of the designer. Furthermore, the tool targets the acceleration of algorithms which were not previously implemented because of the processor restrictions, so that a preliminary version written in C language is not always available.

At last, the interconnect between the components is manually defined by setting the desired connections between each busses and the corresponding components. Once elaborated, the corresponding HDL project can be generated. For (GECO)^2, this implies two steps: the creation of a top-level configuration file and the modification of the top-level HDL file from the template architecture. The configuration file *config.vhd* compiles all the static parameters required by the different components of the architecture. In particular, it describes the format of the configuration register so that the overlap of single component configurations on several frames is minimized.

Each sub-component is associated with an HDL file and IP cores available in the tool library. New components can be added to this library as long as the mandatory producer-consumer interfaces are implemented. Parameterizable, technology-independent implementations are preferred for enhanced genericity. A mix of VHDL *generate* and *configuration* statements is used to differentiate technology-specific architectures and selectively instantiates the appropriate entities (see Listings 6.1 and 6.2). A diagram

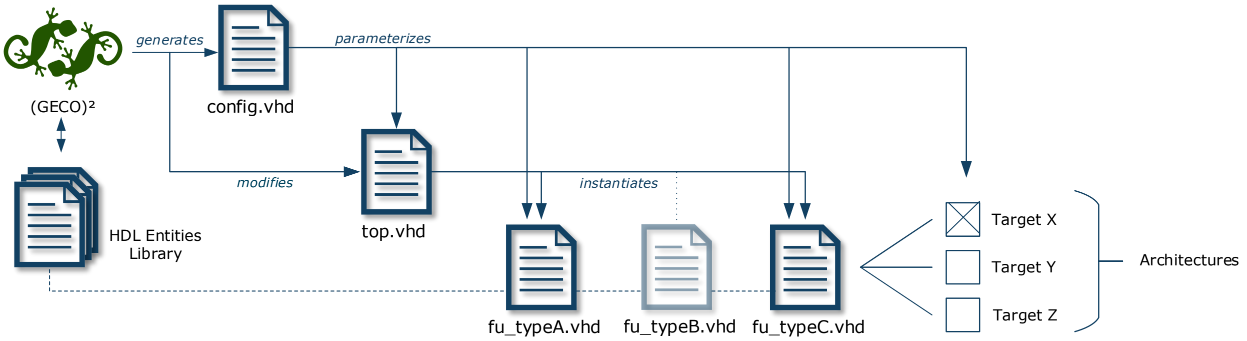


Figure 6.2.: (GECO)^2 concept for HDL parameterization

illustrating the (GECO)^2 concept for HDL parameterization is included in Figure 6.2. In [Por+06a], a similar library of sensor-actuators interfaces written in HDL was developed for the Cookie FPGA-based wireless sensor node. The library was however limited to low-level communication cores such as I^{2}C or SPI and does not include operators or an elaborate processing architecture as it is the case here.

```
-- IGLOO Specific instance
multiplier_igloo : if TARGET_IGLOO generate
   if dw = 16 generate --Data width of inputs
     mul0 : mul_16 port map( ... );   --16 bits Multiplier IP
   end generate;

   if dw = 24 generate --Data width of inputs
     mul0 : mul_24 port map( ... );   --24 bits Multiplier IP
   end generate;

         ...
end generate;

-- Target independent instance
multiplier_bhv : if not(TARGET_IGLOO) generate
   mul0 : mul generic map(dw => dw) --Generic Behavioral Multiplier
                  port map( ... );
end generate;
```

Listing 6.1: Target-aware instantiation of a multiplier

```
configuration IGLOO of fifo_top is
for fifo_top_rtl -- architecture body of fifo_top
  for fifo0 : fifo_ram -- for instance fifo0
    use entity work.fifo(fifo_igloo_arch) --Use IGLOO specific architecture
    port map ( ... );
  end for;
end for ;
end IGLOO;
```

Listing 6.2: Target-aware instantiation of a FIFO using VHDL configuration

The generated HDL files can be imported in a project of the target FPGA design software or serve as input for a synthesis script. When associated with the platform constraint files, the design can be synthesized and implemented in order to generate the FPGA configuration bitstream. It must be noticed that the (GECO)^2 tool does not guarantee the fulfillment of any area or timing requirements, so that design check rules might be broken during the implementation. The FPGA on the target platform is finally programmed with the generated bitstream with classical programming tools and devices, *e.g.* FlashPro programmers for Microsemi devices.

The architecture editor GUI is composed of two main panels: one for global parameterization of the architecture components and the second for visualization. A screenshot of the (GECO)^2 main window is included in Figure 6.3. Each individual component is associated with its own configuration window in order to customize its parameters. A Java interface is available to develop configuration GUIs for newly created components. The visualization frame makes the architecture more readable and gives a better overview on the currently designed architecture. The task mapping is facilitated by visualizing the interaction between the different components.

Once an architecture has been elaborated, the specifications can be saved in an *architecture description file* (extension *.gco in Figure 6.1). The design can be thus reloaded as basis for further modifications or for the design of new task flows.

6.2.3 Data flow editor

Once the static architecture has been elaborated and loaded on the target platform, compatible task configurations corresponding to the desired application functionalities must be generated so that they can be dynamically loaded into the FPGA. This second phase of configuration can only be started once the static architecture has been locked. Indeed, a modification of the static architecture is likely to impact the format of the configuration register, so that configuration data is no longer aligned with the corresponding units. A **.gco* file is therefore mandatory to start this phase. All *dynamic* parameters of the processing elements described in Chapter 5 can be configured. A different value can be assigned for each instruction executed by the configuration controller.

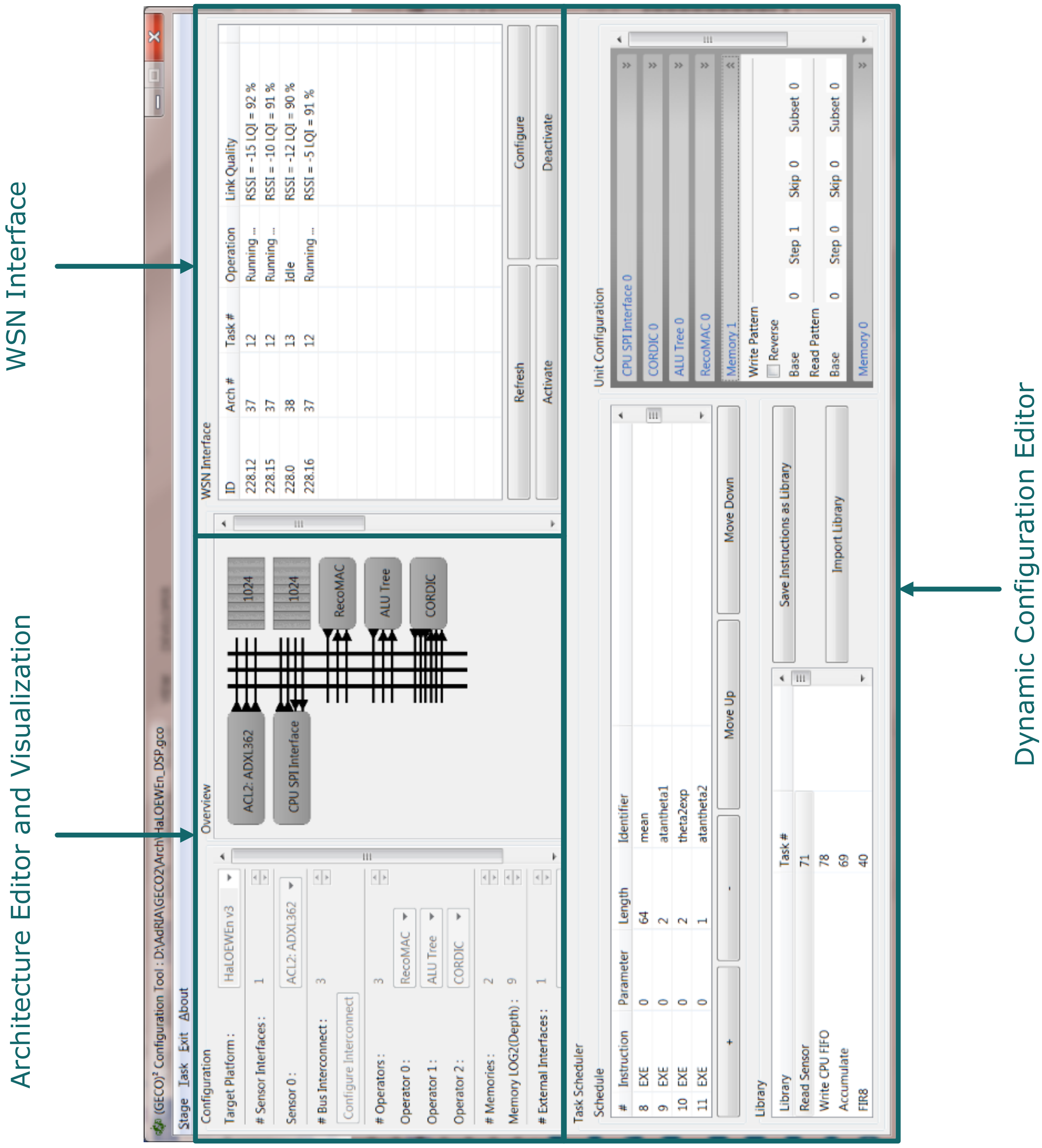
The data flow editor GUI is subdivided into three parts (Figure 6.3):

- With the *instruction editor*, the instructions describing the sequence of contexts as explained in section 5.3.6.2 can be entered. Each instruction is associated with its corresponding parameter and transaction count. Additionally, a comment can be associated to the instruction in order to facilitate the identification of the implemented functionality.

- For each execution instruction (EXE, EXED or SENSE), the configuration of each dynamically reconfigurable component can be defined in the *unit configuration* panel. This includes processing elements, address sequencers, sensors and the interconnect.

- An existing instruction or set of instructions with the corresponding unit configurations can be imported via the *library* panel. The imported instructions can be placed at any position in the current sequence. Reversely, a set of configured instructions can be exported to the library for use in other tasks or applications based on the same architecture. Each library item is saved in a **.gcox* file for reuse among different designs. These files are associated with the architecture for which they were developed and cannot be used with different architectures. Hence, an architecture with a rich library content can easily be configured with an already preprogrammed task by importing the corresponding sequence of instructions.

Once the set of instructions and unit configurations have been fixed, the corresponding bitstream can be generated. For each execution instruction, the configuration vector (context) corresponding to all elements is generated. Then, the tool automatically applies the difference-based reconfiguration process described in Section 5.3.6.1 to remove redundant frames. The resulting context data is packed into the 32 bits format that can be saved in the cluster's configuration memory. The binary data corresponding to the instructions is generated as well. At last, the two sets of configuration data describing the task are



(GECO)² Configuration Tool : D:\AdRIA\GECO2\Arch\HaLOEWEn_DSP.gco

Configuration

Target Platform : HaLOEWEn v3

Sensor Interfaces : 1

Sensor 0 : ACL2: ADXL362

Bus Interconnect : 3

Operators : 3

Operator 0 : RecoMAC

Operator 1 : ALU Tree

Operator 2 : CORDIC

Memories : 2

Memory LOG2(Depth) : 9

External Interfaces : 1

WSN Interface

ID	Arch #	Task #	Operation	Link Quality
228.12	37	12	Running ...	RSSI = -15 LQI = 92 %
228.15	37	12	Running ...	RSSI = -10 LQI = 91 %
228.0	38	13	Idle	RSSI = -12 LQI = 90 %
228.16	37	12	Running ...	RSSI = -5 LQI = 91 %

Task Scheduler

Schedule

#	Instruction	Parameter	Length	Identifier
8	EXE	0	64	mean
9	EXE	0	2	atantheta1
10	EXE	0	2	theta2exp
11	EXE	0	1	atantheta2

Library

Library	Task #
Read Sensor	71
Write CPU FIFO	78
Accumulate	69
FIR8	40

Unit Configuration

CPU SPI Interface 0

CORDIC 0

ALU Tree 0

RecoMAC 0

Memory 1

Write Pattern

☐ Reverse

Base 0 Step 1 Skip 0 Subset 0

Read Pattern

Base 0 Step 0 Skip 0 Subset 0

Memory 0

Figure 6.3.: Screenshot of the (GECO)^2 graphical user interface

written into a single binary file (see format in Figure 5.11), which can be tested and simulated with the HDL model of the architecture or directly loaded into the external configuration memory of the platform.

6.2.4 Evaluation

The $(GECO)^2$ framework was used to generate architectures and task configurations for multiple platforms with different target technologies. The platforms for which the projects were successfully synthesized and programmed include:

- HaLOEWEn - Microsemi IGLOO AGL1000V5 FPGA
- LPSIP - Microsemi IGLOO AGL1000V5 FPGA
- IGLOO development kit AGL-DEV-KIT - Microsemi IGLOO AGL600V2 FPGA
- Digilent Nexys 3 - Xilinx Spartan 6 XC6LX16 FPGA
- Digilent Atlys 2 - Xilinx Spartan 6 XC6LX45 FPGA

It must be noticed that the three latest platforms do not embed an MCU. In this case, the software is running on a laptop connected to the board over a UART serial link. A special framework implemented in Java has been implemented for this purpose. Although the design cannot run in an autonomous fashion in these cases, the platform can be used for test and validation purposes, *e.g.* testing an interface to a new sensor board. In addition, Xilinx FPGAs do not have sleep modes so that the device stays idle when no task is processed. The implementation on these boards serve also as proof of concept that the design can be ported to different FPGAs.

The list of HDL components currently available for $(GECO)^2$ is summarized in Table 6.1. This table does not include internal memory components such as blocks RAM and ROM. Most of the ADC interfaces are based on the SPI protocol. The existing HDL files are then easily reusable for other chips based on the same interface. Analog sensors such as accelerometers or microphones were used in combination with these ADCs for the implementation of real-world applications [ERA13; Abd13; Man12b; Rie11; Gre14]. When no sensor is available, sensor data can be emulated using the *virtual* sensor interface. Data streams stored on a computer are sent to the device over a serial interface in a Hardware-in-the-loop fashion.

A task library based on the architecture for DSP described in Figure 5.12a was intensively populated using configuration data generated with the $(GECO)^2$ application. A non-exhaustive list of functions available in this library is given in Table 6.2. Most of the functions need to be adapted with application-specific parameters such as length or constant values, but each elementary configuration gives a good basis that only need a slight modification. The bitstream size is given for a configuration where the task is used as standalone function. The size is given here as an indicator and might vary according to application-specific customization of the task, *e.g.* constant address offsets in the memory. The given number of instructions do not include EXIT instructions. If enough instructions are available, tasks can be combined to form more complex functions within one configuration bitstream. Such a library allows an easy composition of elementary tasks to build a more complex algorithm. For instance, a spectral analysis of an input signal could be built by combining the tasks *Collect* for the acquisition of data, *Cross* for the windowing operation, a set of *FFTStep* functions and the final *Magnitude*. Other transforms ore signal processing tasks can be built by following a similar scheme. More advanced examples of such compositions are given in the Chapter 7 related to applications.

6.2.5 Methodology for application-specific customization and programming of the architecture

In this section, a standard methodology is proposed to design an architecture and implement a given algorithm with the proposed template. By following these successive design steps, all features of the

Type	Name	Comment
Sensing	ADC121S051	12-bit ADC (HaLOEWEn)
	ADC128S022	8 channels 12-bit ADC (HaLOEWEn)
	TI-ADS7866	12-bit ADC (LPSIP)
	AD-7476	Two channels 12-bit ADC (PmodAD1)
	ADXL362	3-axis 12-bit accelerometer (PmodACL2)
	LIS3LV02DL	3-axis 12-bit accelerometer
	ADC122S021	2 channels 12-bit ADC
	IMU-3000	3-axis I^2C gyroscope
	Virtual sensor	UART interface to PC
Memory	CY62126EV30	1 Mbit parallel SRAM (HaLOEWEn)
	FM21L1660TG	2 Mbit parallel FRAM (HaLOEWEn)
	ISSI IS62WV1288	1 Mbit parallel SRAM (LPSIP)
	ST M95M01	1 Mbit serial EEPROM (LPSIP)
	CY7C1041	4 Mbit parallel SRAM (AGL-DEV-KIT)
	JS28F650	16 Mbit parallel Flash (AGL-DEV-KIT)
	N25Q12	256 Mbit serial Flash (Atlys)
Communication	SPI slave	SPI access (HaLOEWEn - LPSIP)
	MCU parallel access	8-bit data line (HaLOEWEn)
	UART	Serial access
Operators	rMAC	Integer multiply-and-accumulate
	CORDIC	Iterative extended CORDIC
	ALU	ALU tree
	Reg	Register file
	S-Box	Combinatorial S-Box
	GF MAC	Galois-Field MAC
	DIV	Iterative Divider
	SQRT	Iterative square root
	COMP	Comparator unit
	MUL	Iterative multiplier

Table 6.1.: Components available in the $(GECO)^2$ HDL library

Name	Function	# Instr.	Bitstream size (bits)
Collect	Save sensor data in memory	1	112
Store	Save data in external memory	1	112
LoadMCU	Load data from the MCU	1	112
Add	Add two vectors	1	176
Cross	Cross product of two vectors	1	176
Dot	Dot product of two vectors	1	176
MatrixMul	Matrix Multiplication	3	336
ArithMean	Arithmetic Mean of a vector	3	272
Var	Variance of a vector	4	384
MS	Mean Square of a vector	4	384
FIR	FIR filter of a vector	2	240
IIR	IIR filter of a vector	4	448
Wavelet	Step of a Wavelet Transform	3	304
FFTStep	Step of a Fourier Transform	3	400
DFT	Discrete Fourier Transform	4	384
Twiddle	Compute DFT coefficients	3	368
SQRT	CORDIC square root	3	368
Magnitude	CORDIC magnitude	1	176

Table 6.2.: List of elementary functions available in the $(GECO)^2$ DSP library

architecture can be exploited. This methodology is referring at Figure 6.1 and Figure 6.4 to emphasize the relationships between each phase.

1. **Application profiling**: the preliminary design task before the configuration of the architecture is the identification of operation patterns in the algorithms being part of the application. Using a validation model of the algorithm written in a high-level modeling language, *e.g.* Matlab, Java or C, critical loops and repeated operation patterns must be identified. Independently from the amount of sensor units, which directly depends on the application, these patterns will fix the nature of the PEs available in each cluster. The size and the amount of memory blocks must be evaluated in a similar fashion. As memory is very limited in Igloo FPGAs, external memory chips are usually required to hold large data streams. Implementing several clusters is justified only if the computations in each cluster can be well parallelized, *i.e.* the computations are independent from each other. It is not always necessary and efficient to implement a full interconnect between PEs. It is generally sufficient to have interconnect patterns following a typical flow from sensor to output, *i.e.* (a) sensors to memory and PEs (b) memory to PEs (c) (optional) PEs between themselves (d) PEs to memory and output (e) memory to output.

2. **Algorithm implementation**: Once an architecture is fixed, tasks can be mapped on it. A good basis is the task library available for predefine architecture. If a new architecture has been elaborated, basis tasks can easily be derived from the pre-existing ones. The algorithm implemented on the core should be rewritten in such a way that a maximum number of linear patterns are executed. The FFT algorithm can for example be rewritten with a *perfect shuffle* address pattern, which can be mapped more easily on the address generators [Sto71]. This pattern is described in Appendix B. In general the number of instructions within a task description should be minimized in order to avoid a reconfiguration overhead. Loops and repetitions should be exploited to reduce the amount of instructions. The difference-based reconfiguration can also be exploited by leaving the configuration of unused PEs unchanged. A PE can thus be dynamically configured by anticipation or left configured after its operation is terminated in order to save configuration time and memory.

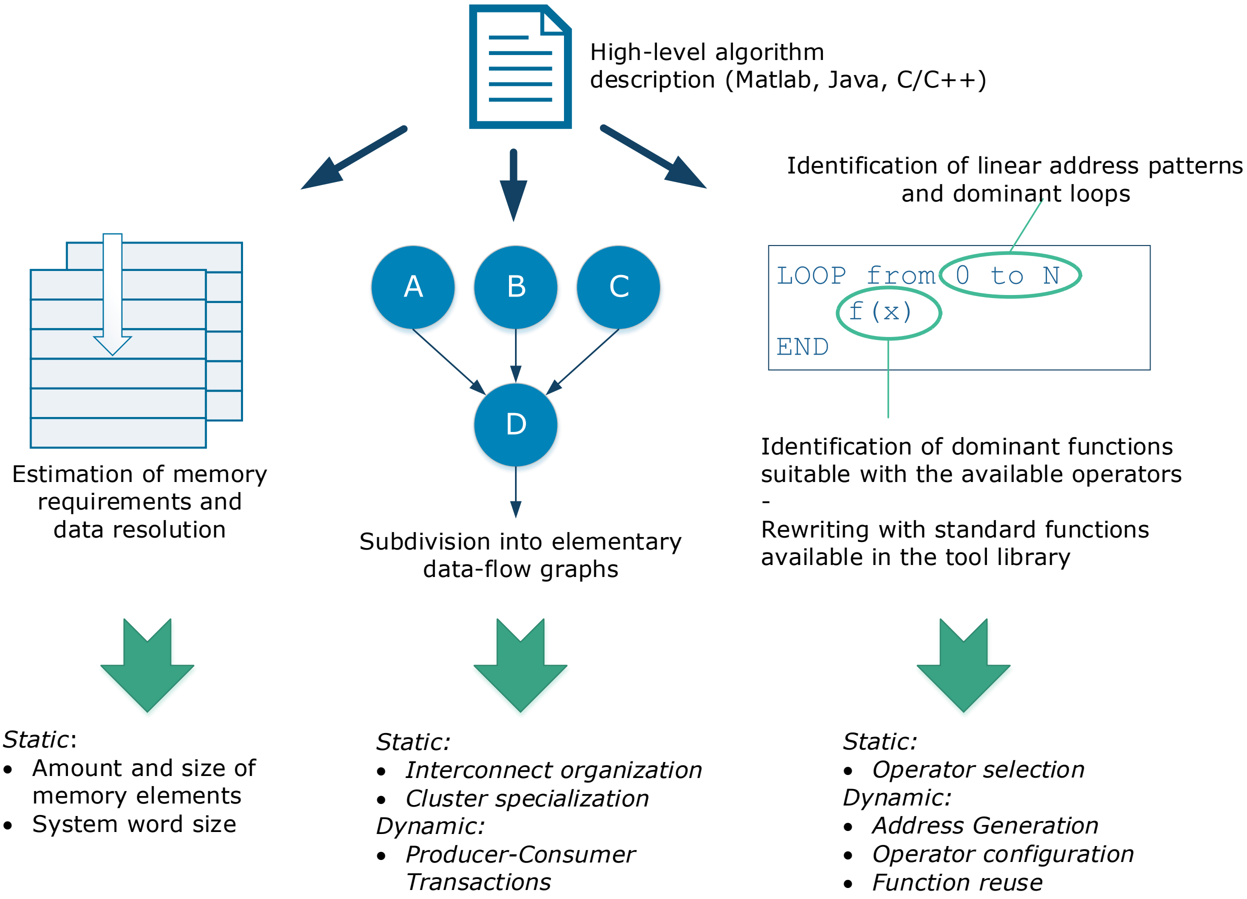

High-level algorithm
description (Matlab, Java, C/C++)
Identification of linear address patterns
and dominant loops
A
B
C
D
LOOP from 0 to N
f(x)
END
Estimation of memory
requirements and
data resolution
Subdivision into elementary
data-flow graphs
Identification of dominant functions
suitable with the available operators
-
Rewriting with standard functions
available in the tool library
Static:
Amount and size of memory elements
System word size
Static:
Interconnect organization
Cluster specialization
Dynamic:
Producer-Consumer Transactions
Static:
Operator selection
Dynamic:
Address Generation
Operator configuration
Function reuse

Figure 6.4.: Methodology for architecture and algorithm design

6.3 Middleware for configuration management

In order to load and activate the configuration data saved in the external configuration memory of the platform, the MCU has to sent a specific sequence of commands over the dedicated interface (see Section 4.3.3). If the flow of tasks implemented by the reconfigurable hardware accelerator is fixed, static calls to the function sending these commands are sufficient to manage the dynamic reconfiguration of the hardware accelerator. However, WSN applications often require a level of adaptability where such a static approach is not scalable [EK09]. A higher level of control can be achieved for the hardware accelerator if it can be called *on-demand*, according to user or applications requests.

6.3.1 Related work

Dynamic software reconfiguration in wireless sensor networks is a popular topic that has been often addressed by extending the underlying operating system. The **Maté** framework was for example enabling portable program migration by defining a virtual machine running on top of TinyOS [LC02]. Small code fragments for the virtual machine named *capsules* can be dynamically transfered and activated on any sensor node at runtime. The **Agilla** middleware for TinyOS proposes another programming model where mobile software agents can be transfered between neighboring nodes [FRL09]. In Contiki, the software can be modified at runtime by using the dynamic linking functionality integrated in the core of the operating system [Dun+06b].

For dynamic hardware reconfiguration, the management of bitstreams is in general managed by the operation system as well, *e.g.* the reconfiguration agent in the **MORPHEUS** framework [TB09]. In [Ull+04], FPGA slots of a partially reconfigurable system are dynamically reconfigured according to commands sent over a CAN bus. Issues such as resource allocation and context saving are addressed. However, the nature and the amount of partially reconfigurable bitstreams available in the local configuration memory cannot be modified at runtime. The remote access to configuration data is investigated in [Ind+03] where an API is develop to load bitstreams over an Internet protocol link.

When considering hardware reconfiguration in WSNs, the Cookie sensor node integrates a layer for reconfiguration control within its software stack [Kra+11]. However, the work focuses only on the reconfiguration process itself and not on the dynamic management of multiple bitstreams within the network. In particular, the authors highlighted the particularly high costs of transmitting FPGA configuration bitstreams over a low-power wireless link, even when compression algorithms are used. Hinkelmann handled the top-level reconfiguration of its coarse-grained reconfigurable function unit by introducing a dedicated processor instruction [Hin11]. Calling this instruction was triggering the reconfiguration process, which could run in the background of the normal processor operation. The loading and activation of configuration bitstreams that were not available at compile time was not considered. In this case, this feature would require an additional software reconfiguration, so that the global application stays always static. In [HHP13], the authors integrate a so-called *Flexible Radio Kernel* within the application stack of wireless platform in order to facilitate runtime reconfiguration of the protocol functionalities. Tasks are described using *Waveforms*, which are a combination of predefined functions available in a library. The software kernel is then able to select and configure on the hardware at runtime the different functions required by the waveform.

The need for a unified component integrator which could arbitrarily load and activate any type of hardware configuration on a sensor node during runtime has been emphasized in [EK09] but no solution was introduced.

6.3.2 General overview

Using the presented reconfigurable architecture, the amount of configuration data necessary to describe a task is considerably reduced when compared to an FPGA bitstream. This advantage can be exploited to handle the transfer of configuration data within the network in a lightweight fashion. Particularities of WSN operating systems and software dynamic reconfiguration schemes can be reused for abstracting hardware dynamic reconfiguration on the sensor node. However, the presented framework stays as generic as possible so that it can be applied to other types of dynamically reconfigurable systems, *e.g.* for partially reconfigurable FPGA designs.

For dynamically reconfigurable systems, two scenarios are distinguished to decide when and what to reconfigure:

- The reconfiguration process follows a static schedule. It is either fixed by the functionality, *e.g.* a hardware task is started after a data packet has been received, or it is delimited in time, *e.g.* the hardware task is started at regular time intervals.
- Reconfiguration is requested *on-demand* by another layer of the application stack. The hardware might be busy running another task while the request is incoming and must react accordingly.

In order to cover these situations in a uniform environment, an event-based programming model is adopted to control the reconfiguration process. Each event can be generated by an external component of the system or it can be generated internally, in response to an imposed scheduling strategy. This approach gives an additional level of flexibility to the system where both external and internal reconfiguration requests can occur. It also offers a global framework supporting multiple runtime reconfiguration policies which can be dynamically and remotely tuned.

An event-based middleware abstracting the reconfiguration process has been then implemented as an intermediate software agent between the application and the reconfigurable hardware as illustrated by Figure 6.5. Unlike compiler-based approaches, the control of dynamic reconfiguration is not made entirely transparent to the user. Reconfiguration must be explicitly requested by using the middleware API and specific scheduling rules.

The rest of this section is organized as follows: in section 6.3.3, the different elements of the middleware framework are shortly introduced. The strategy to dynamically handle different types of events is described in section 6.3.4 while the command-based API is explained in section 6.3.5. Usage examples are finally given in section 6.3.7.

6.3.3 Middleware components

The middleware is divided in several sub-components handling separate parts of the complete reconfiguration flow as follows:

- The **remote interface** is running on a remote device that can be directly accessed by the user. It generates the commands describing the reconfiguration rules for a specific task. The configuration database indexes and stores all configuration files available to the reconfigurable device.
- The **gateway** interprets the reconfiguration instructions sent over the network and dispatch them to the middleware kernel or the event handler.
- The **middleware kernel** manages individual reconfiguration requests and a global table indexing all configuration bitstreams currently available on the platform. It applies the instructions associated with each reconfiguration event.
- The **event handler** monitors external events which trigger reconfigurations as specified by the commands sent by the user. It also handles internal events such as timing or control events. This component is further detailed in section 6.3.4.
- The **resource management** unit select the most appropriate target to implement a task when multiple options are available. For example, a software implementation can be preferred to a hardware reconfiguration if the hardware device is busy.

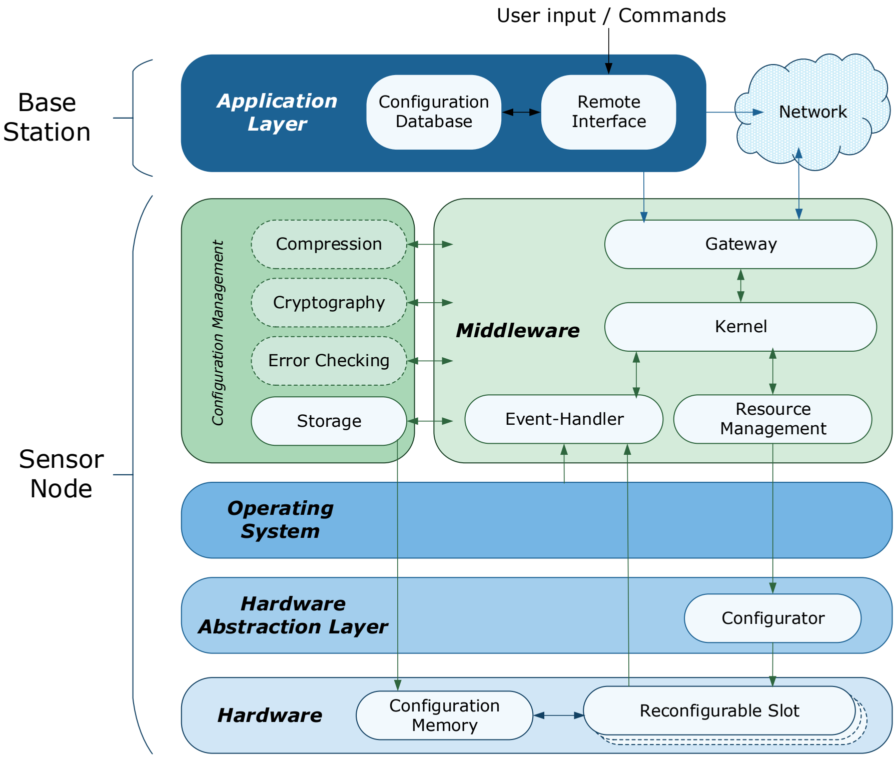


Figure 6.5.: Block diagram of the middleware components

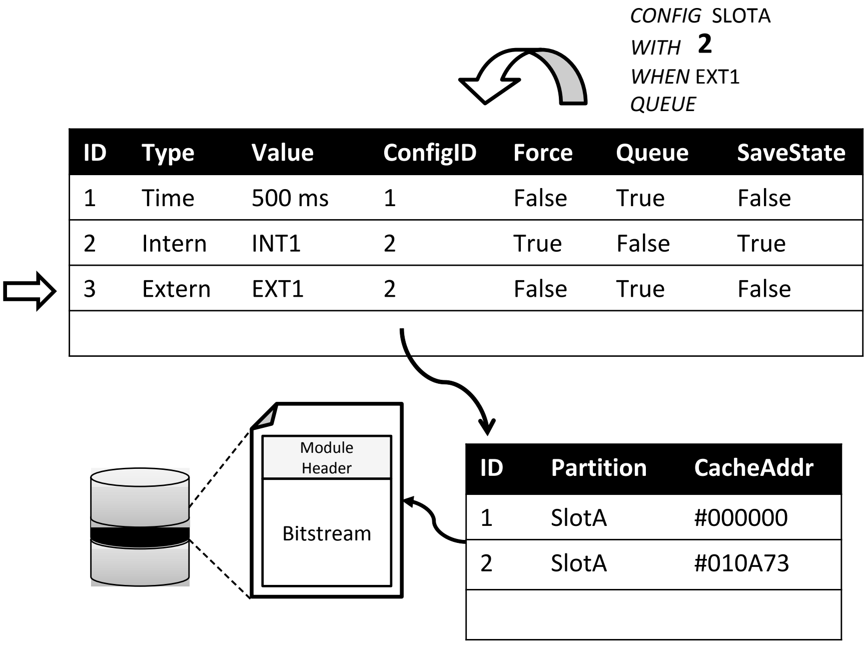
- The **bitstream management** unit combines a set of services guaranteeing a secure, error-free and resource-friendly handling of configuration data. These services are particularly important when considering remote transfer of configuration data as the data is more likely to be damaged or altered during the transfer. The software managing the organization of the configuration memory is included here. Some of these components are not mandatory for the operation of the middleware (marked with a dotted line in Figure 6.5). Similar services are included in the existing FPGA programming tools, *e.g.* AES decryption and CRC check in Igloo FPGAs [Igla] or Lempel-Ziv-Welch (LZW) compression in [Ull+04].

- The **configurator** module implements the actual dynamic reconfiguration. It controls the loading and activation of the configuration data (bitstream) via one of the available reconfiguration methods, *e.g.* access to the internal configuration access port (ICAP) or JTAG port, depending on the target platform. In the case of the overlay architecture introduced in Chapter 5, this function refers to the sequence of commands sent by the MCU to activate the loading of configuration data from the external configuration memory.

6.3.4 Event-based reconfiguration

Except in the case of system updates, the reconfiguration process is not explicitly initiated by the user but it is started only if a specific set of conditions is fulfilled. Meeting these conditions will then trigger an event, which will be processed by the event-handler module accordingly. Three classes of events have been distinguished:

- **External**: Reconfiguration is triggered by an event reported by higher application layers. In this case, the evaluation of the system conditions triggering the event is not performed by the platform itself. New types of external events can be integrated into the system while running.



ID	Type	Value	ConfigID	Force	Queue	SaveState
1	Time	500 ms	1	False	True	False
2	Intern	INT1	2	True	False	True
3	Extern	EXT1	2	False	True	False

ID	Partition	CacheAddr
1	SlotA	#000000
2	SlotA	#010A73

Figure 6.6.: Event and configuration tables

- **Internal**: Signal from lower layers (hardware, operating system) are indicating a need for reconfiguration. This class includes time events, which schedule reconfiguration according to an underlying timer. The different types of internal events is fixed at compile time and cannot be extended at runtime.

- **Direct**: Reconfiguration is requested independently from the state of the system for an immediate schedule (system update).

In order to index the events and associate them with reconfiguration instructions, the event-handler is using a table as illustrated by Figure 6.6. Each event is associated with an ID, the identifier of the corresponding configuration data and a set of flags defining reconfiguration rules. When the event occurs, its ID is looked up in the table and the middleware kernel is requested to configure the hardware with the corresponding rules. In order to avoid conflicts, an event cannot appear several times in the table but the same configuration data can be associated with different events. Only **external** and **internal** events are stored in the table. **Direct** events occur only once and are directly processed by the middleware kernel. The reconfiguration instructions or *rules* allow defining an individual reconfiguration policy for each event and module. Current parameters allow *forcing* reconfiguration, even if a previous task has not been completed. This implies that the operation of the hardware is stopped and that the new configuration data is directly loaded and activated. The state of the previous configuration can be previously *saved* at the cost of additional configuration delay. At last, the configuration can be *queued* if the task can not be loaded directly, *e.g.* because the hardware is busy or the configuration data is not available. A task which is neither queued nor forced and for which the hardware can not be reconfigured immediately will be dropped.

The event table is closely coupled to another table indexing the available configuration data, which is managed by the middleware kernel. This table lists all currently registered configuration bitstreams and their addresses in the configuration memory.

The content of both tables can be modified at runtime through the middleware API. For instance, new events can be registered or out-of-date associations can be removed.

Command	Function
`ADD`	Register a bitstream in the configuration table
`REM`	Remove a bitstream from the configuration table
`MOD`	Modifiy an entry in the configuration table
`CONFIG`	Register a reconfiguration process in the event table
`CANCEL`	Remove a reconfiguration process in the event table
`EVENT`	Trigger and external event
`SET`	Transfer data to the hardware
`GET`	Read data from the hardware

Table 6.3.: Summary of the middleware commands

6.3.5 Middleware commands

The middleware offers an interface based on commands to interact with the kernel and the event handler. A syntax has been defined to describe all types of interactions with the middleware. This approach is a simple and readable way to remotely modify the behavior of the system at runtime without need for software reconfiguration, which is adopted in most of the currently existing approaches.

Commands written by the user are coded and encapsulated in messages passed to the middleware API. Commands are built as a succession of keywords and parameters. The main commands functionalities are summarized in table 6.3. The `ADD`, `REM` and `MOD` commands are used for basic interaction with the configuration table managed by the middleware kernel. The `CONFIG` and CANCEL are used for the event table management. Commands can also be directly coded by the application internally so that nodes within a network can send commands between themselves.

The syntax of an `ADD` command is given in the listings 6.3. A similar syntax can be used for `REM` and `MOD`. Each configuration bitstream must be associated with a unique identifier and a location in the configuration memory. For systems with multiple reconfigurable partitions, *e.g.* a slotted partially reconfigurable system or the multi-clustered architecture as described in Chapter 5, the partition compatible with the bitstream must be specified.

```
ADD <module name>
    LOC <address>
    ID <value>
    FOR <reconfigurable partition>
    [OPTION <value> , ... ]
```

Listing 6.3: Syntax of the ADD command

The command can be optionally extended with the option `LOCAL` to indicate that the bitstream must be saved in the platform configuration memory. If this option is not specified, the middleware kernel will retrieve the configuration information from the remote server over the network interface every time the task must be configured. This option is useful if a bitstream is rarely used and the memory available for configuration data is limited.

For the management of the event table, the syntax of the `CONFIG` command is given in the listings 6.4. It is associated with a single task or a sequence of tasks which should be executed on the hardware. If no further options are entered, the command is interpreted as a *direct* event and the corresponding bitstream is immediately loaded. Otherwise, the event triggering the reconfiguration of the sequence of tasks is specified after the `WHEN` keyword. Timed events can be specified as well using this keyword. The reconfiguration rules (force, queue, save state) are given as options of the command. When this command is received by the middleware, it will create a novel entry in the event table. If the event was already registered, the entry will be replaced.

```
CONFIG <module_0> [,..., <module_i>, ...]
    [WHEN <event>]
    [OPTION <value> , ... ]
```

Listing 6.4: Syntax of the CONFIG command

An entry of the event table can be deleted by using the `CANCEL` command followed by the value of the event. The `EVENT` command is used to trigger an external event within the system. If the event is associated with configuration data in the event table, the reconfiguration will be started according to the specified rules. Otherwise, the event is ignored.

At last, the `GET` and `SET` command are used for data transfer between a remote location and the target hardware. This commands are useful to transfer custom data to the hardware accelerator before a task is implemented and the results accordingly. The commands take the size and the location of the data in the target device memory as argument.

```
SET [data]
    FORMAT <format>
    LOC <address>
```

Listing 6.5: Syntax of the SET command

6.3.6 Implementation

The middleware framework was implemented for the HaLOEWEn platform and the coarse-grained overlay architecture using the event management features already available in the Contiki operating system [DGV04]. Thus, the event handler process can easily be called by posting dedicated events as illustrated by the listings 6.6. Support for timed events is enabled by the Contiki internal timers. This feature largely simplifies the implementation of the rest of the middleware, which mainly consists in the management of the tables and the external interfaces.

```
#include "contiki.h"

/* Event Handler Process */
PROCESS(ev_hdlr_process , "Event␣Handler␣Process");
AUTOSTART_PROCESSES(&evt_hdlr_process);

PROCESS_THREAD(evt_hdlr_process , ev , data){
PROCESS_BEGIN();

\* Event table and internal events initialization *\
...

while (true) {
        PROCESS_WAIT_EVENT();
        task = lookup_event(ev);
        if(task != null){
                hw_reconfigure(task);
        }
        ...
}
PROCESS_END();
}
```

Listing 6.6: Event management in Contiki

The *Configurator* component is implementing the transfer of the reconfiguration commands and reconfiguration data to the FPGA interface. If the external configuration memory is available, *i.e.* the FRAM memory module is plugged, the data is stored in this chip.

Wireless communication is implemented with the Rime networking stack available in the Contiki operating system. The *mesh* networking primitive is used to exchange configuration data or commands between any nodes of the network. All routing and network maintenance routines are handled by the Rime stack autonomously.

Without taking the optional bitstream management services into account, the middleware has a code footprint of 7.4 kBytes, which represents about 30 % of the Flash memory available for a Contiki application running on HaLOEWEn.

The remote server has been implemented as a part of the $(GECO)^2$ application. For instance, when a wireless sensor network is deployed and connected to the $(GECO)^2$ application, a list of available nodes is displayed in the GUI (see Figure 6.3). The interface allows sending configuration commands and events to individual nodes. In particular the loading and activation of a configuration bitstream on the sensor node can be fully controlled from this interface.

The transfer costs in terms of time and wireless communication energy were estimated for a single-hop scenario. A configuration bitstream with variable size is sent from a central node connected to the $(GECO)^2$ tool to the target platform by issuing an `ADD` command. The energy and transfer time is estimated until the bitstream transfer is completed, not including the transfer in the configuration memory. The Contiki-MAC protocol with a channel check rate of 8 Hz and a maximum useful payload of 100 bytes per packet has been used for this experiment. The resulting curves are shown in Figure 6.7.

When compared to [Kra+11] or [HWH12] where remote reconfiguration of a complete FPGA is implemented, the low size of the configuration bitstream makes the wireless transfer a significantly less costly operation. A configuration bitstream can be sent within one second with the presented framework, whereas transfer duration up to several minutes are reported in the related works. With this low size, it becomes possible to store and manage multiple configuration bitstreams on the node itself.

The middleware presents however some limitations in terms of latency. Indeed, it has been estimated that a delay going up to 15 milliseconds is introduced between the occurrence of an event and the actual start of the reconfiguration process. This delay is due to the processing delays of the event handler and the middleware kernel as well as the internal mechanisms of the operating system, which does not give immediate priority to the event handler process. The middleware is thus not suitable for reconfiguration with real-time constraints. Nevertheless, this latency is specific to the target MCU and operating system and may be reduced on different platforms. In addition, the framework is targeting applications for continuous monitoring so that real-time constraints are rarely required.

6.3.7 Application examples

A few command examples are given in this section to illustrate the potential of the middleware to configure hardware tasks on the sensor node.

- **On-demand computation**: in wireless sensor networks with heterogeneous platforms, it is often beneficial to reallocate the computation of a function on the most powerful nodes. This approach is for instance adopted in [Rei11] where a protocol based on lightweight Remote Procedure Calls (RPC) is implemented. Here, a custom hardware accelerator can be called on the sensor node by sending direct reconfiguration events. Data must be preliminary loaded with `SET`, before the `CONFIG` is sent. Results are eventually read by issuing a `GET` command. For example, the localization application introduced in part 4.3.4.2 can be called using the commands

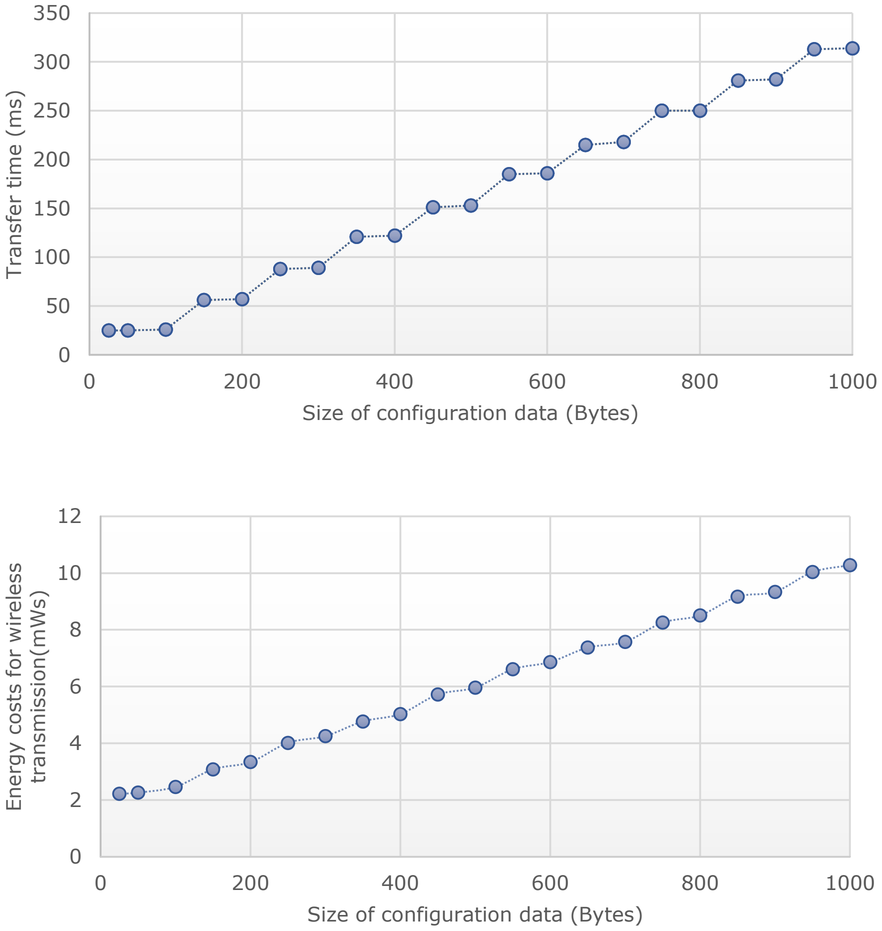


Figure 6.7.: Duration and energy costs of configuration data transmission

from the listings 6.7. Real data has been replaced by the label *dtoa_data* for more readability.

```
SET dtoa_data FORMAT int16_t LOC fifo
CONFIG localization
GET 3 FORMAT int16_t LOC fifo
```

Listing 6.7: Commands for remote function call

- **Computation scheduling**: the middleware allows scheduling the execution of a task at regular time intervals. For example, a continuously running machine whose condition is monitored by vibration analysis do not need to be constantly examined as its degradation is a rather slow process. The analysis, for example using wavelet analysis, can be scheduled every five minutes with the command from listing 6.8. Later on, the algorithm can be rescheduled with different time intervals or with different algorithms if the operation of the machine is more critical. A similar example is studied in more details in the application Chapter 8.

```
CONFIG wavelet_analysis WHEN @300s QUEUE
```

Listing 6.8: Command for task scheduling

- **Context-aware computation**: it is likely that different sensor data processing algorithms are required when environmental conditions are changing. A wearable activity recognition body area network may for example be programmed differently according to the location of the user. When location changes are coded to generate events, the feature extraction algorithms necessary to classify the user activity can be executed according to the commands given in the listings 6.9.

```
CONFIG featureSetA WHEN evtLocA QUEUE
CONFIG featureSetB WHEN evtLocB QUEUE
CONFIG featureSetC WHEN evtLocC QUEUE
```

Listing 6.9: Commands for context-aware computation

6.4 Conclusion

Taking profit of the hardware accelerator embedded on the wireless sensor node implies a significant programming effort. Designing custom FPGA projects and the corresponding software drivers may consume precious time. Including the capability for dynamic reconfiguration introduces a further level of complexity preventing rapid deployments. The tools proposed in this chapter bypass these issues by providing an universal framework to handle the hardware accelerator at several level of abstractions. The template FPGA overlay introduced in Chapter 5 can be easily parameterized through an accessible graphical user interface. Generated HDL files can be used for implementation on devices from multiple FPGA vendors. At the next level, configuration data compatible with the generated architecture can be created using the same tool. A library of preprogrammed tasks encourages the composition of elementary tasks to built complex algorithms. Finally, this configuration data can be freely loaded and scheduled on the sensor node hardware at runtime thanks to a lightweight middleware service implemented on top of the mote operating system. Dynamic reconfiguration can be thus exploited at all levels using programming and scheduling within a single tool.

In addition to the genericity required by the complete design stack, the tools introduced in this chapter contribute to improve the accessibility of the hardware accelerator. This aspect is often seen as a major obstacle when dealing with FPGAs. Simplified programming models and complete toolchains are the essence of popular smart sensing and actuating platforms such as Arduino [Ard]. Ranging from the hardware PCB design to the integrated development environment, internal implementation details of the

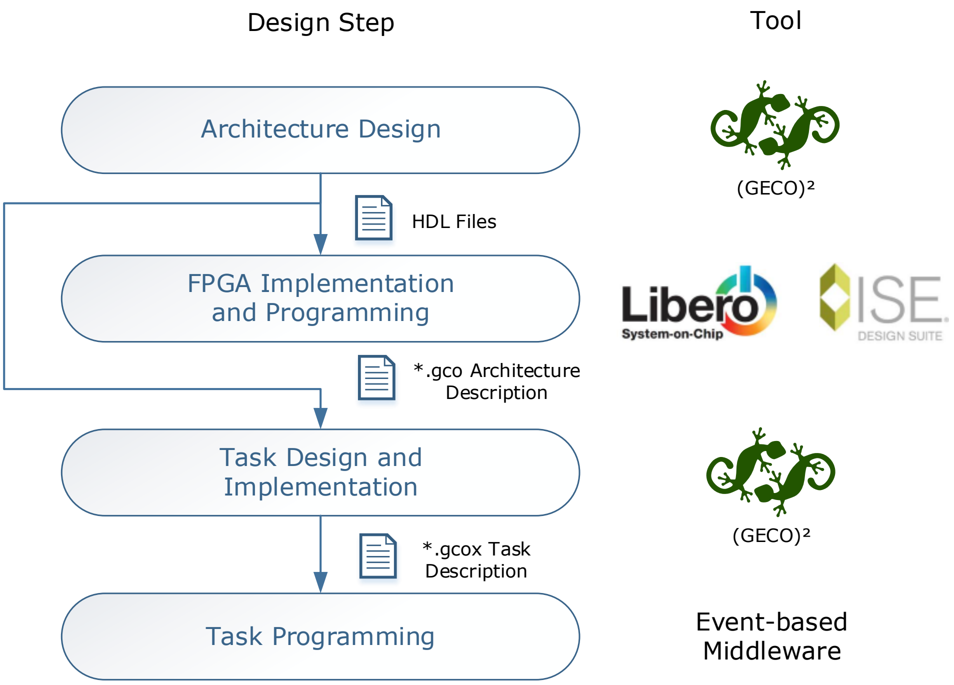


Figure 6.8.: Tool-flow for generating and running hardware reconfigurable tasks on the sensor node

microcontroller-centric platform are abstracted to allow the user focusing on the application functionality. A similar approach was applied here to a programmable hardware architecture. Even if the different abstraction levels have an impact on the global performance, energy-efficiency and flexibility of the platform, the improved programmability makes the approach more usable in practice. The variety of applications applying this framework and developed on the basis of student projects [Abd13; Gre14; Man12b] show that the design environment has also a potential for educational purposes[3]. More importantly, the complete set of hardware components, template architecture and programming tools can be applied to wireless sensor network deployments with industrial relevance. The next part of this thesis will give an overview of two of these applications.

---

[3] The publication [PG14b] discusses this aspect in details

Part III.

Application of Wireless Sensor Networks Strengthened with Reconfigurable Hardware to Condition Monitoring Systems

7 Condition Monitoring of a Shock Absorber for Predictive Maintenance

> Maintenance on Demand: Designed in a very modular and flexible way
>
> B. Favre, “Maintenance on Demand”. In: *Transport*, 2012, Vol. 29

Contents

6.1 General overview . . . 100

6.2 Graphical configuration interface . . . 101

6.2.1 Related work . . . 101

6.2.2 Architecture editor . . . 102

6.2.3 Data flow editor . . . 104

6.2.4 Evaluation . . . 106

6.2.5 Methodology for application-specific customization and programming of the architecture . . . 106

6.3 Middleware for configuration management . . . 110

6.3.1 Related work . . . 110

6.3.2 General overview . . . 110

6.3.3 Middleware components . . . 111

6.3.4 Event-based reconfiguration . . . 112

6.3.5 Middleware commands . . . 114

6.3.6 Implementation . . . 115

6.3.7 Application examples . . . 116

6.4 Conclusion . . . 118

Among the vast range of applications enabled by WSNs, a large number could be found where the complete set of concepts and tools implementing reconfigurable hardware acceleration could have a big potential. **Condition monitoring** is probably one of the most relevant one because of its intrinsic needs for reliable, accurate and cost-effective processing solutions [Kos+10; Ram+07; SSS10; LL06; Kri+05]. The next two chapters describe the development of two condition monitoring applications using the framework introduced in the previous part of this thesis. These applications are used to demonstrate the flexibility of the approach and its suitability for applications with industrial relevance.

Currently, many concepts for condition monitoring are being developed and validated for the detection and diagnosis of premature damage in mechanical systems and for the prevention of hazardous failures in the case of damage. These methods, indicated as *health* monitoring methods, aim at controlling and reducing the life-cycle costs in safety-critical components of vehicles (such as wheels, brakes, power trains) [Mic12; MD08], civil structures (such as cable-stayed bridges) [LL06; SSS10] or machinery (such as

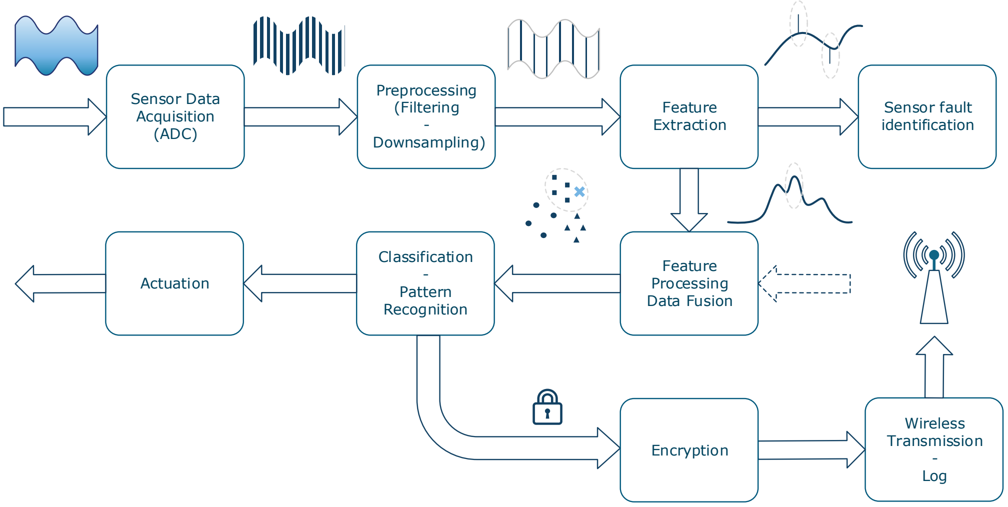

Sensor Data Acquisition (ADC)
Preprocessing (Filtering - Downsampling)
Feature Extraction
Sensor fault identification
Actuation
Classification - Pattern Recognition
Feature Processing Data Fusion
Encryption
Wireless Transmission - Log

Figure 7.1.: Typical flowchart of a distributed condition monitoring application running on a sensor node (inspired from [Kul10])

electrical motors, industrial robot arm) [Kri+05; HB12]. Improvement of the life-cycle costs is achieved by reducing product maintenance costs and improving product availability and reliability.

Even if most of the critical equipment is already equipped with wired monitoring systems, there is still a large potential for WSNs. Indeed, in a large-scale context, WSNs present many inherent advantages such as reduced cabling costs and ease of installation. Moreover, WSNs enable continuous monitoring, which is not supported by portable monitoring equipment often used as an alternative to wired systems [Mic12]. In an industrial environment, a high level of reliability is expected for the WSN. Node failures or erroneous measurements cannot be afforded, so that dependable monitoring systems able to implement complex functions in a flexible manner must be introduced. Figure 7.1 shows the data processing flow of a typical distributed condition monitoring application running a sensor node [Kul10]. The complexity and the computational burden of the complete flow makes traditional microcontroller-based wireless sensor nodes no longer suitable. The following results aim to show that a sensor with runtime hardware reconfigurability fits better to these requirements[1].

7.1 Concept

7.1.1 General overview

In the *Maintenance on Demand* project (MoDe) [Mod], condition monitoring concepts are combined with wireless communication and advanced data management to enable a complete chain of services optimizing the maintenance of a truck fleet. Sensor nodes deployed within the vehicle are collecting data from on-board sensors, performing a distributed analysis and delivering health indicators to the truck central control unit. Information is further forwarded to the truck company back office for prediction of the remaining useful lifetime of damaged or aged structures and optimal scheduling of maintenance operations. The route and the maintenance schedule of the vehicle is thus dynamically adapted to reduce risks of accidents, downtime of the vehicle and environmental pollution.

Three parts of the truck were selected for condition monitoring, *i.e* the shock absorbers, the fuel injector system and the oil system. Each part is equipped with appropriate sensors and processing units, which

---

[1] The results of this section are based on the articles [Bei+14] and [Fav12] and on the project internal reports [Els10; Jan+10; Els12; Pee12; Pee11]

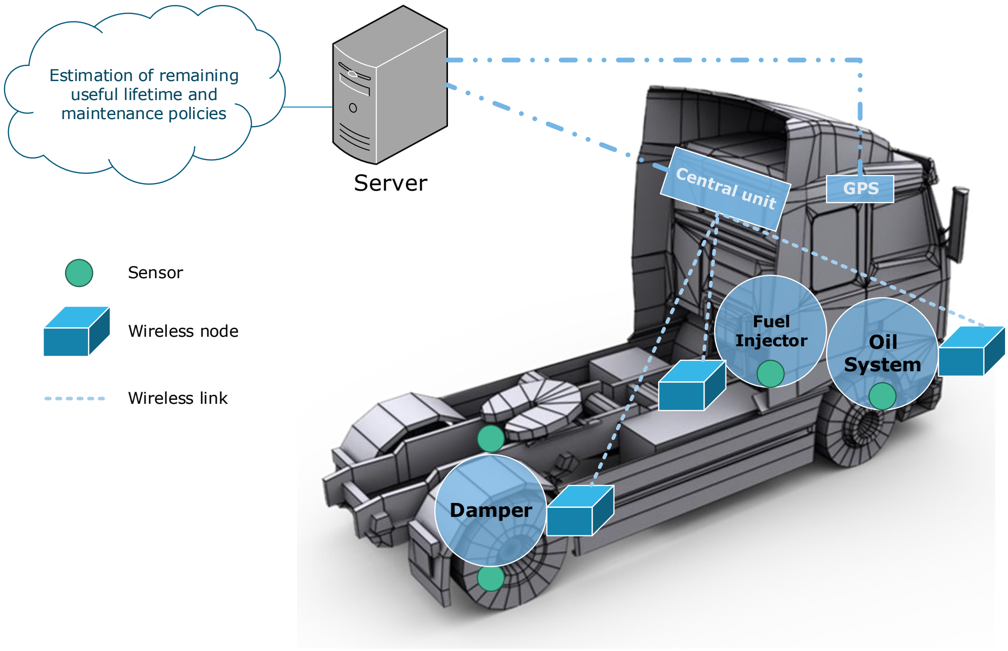


Figure 7.2.: Overview of the MoDe condition monitoring infrastructure [Bei12]

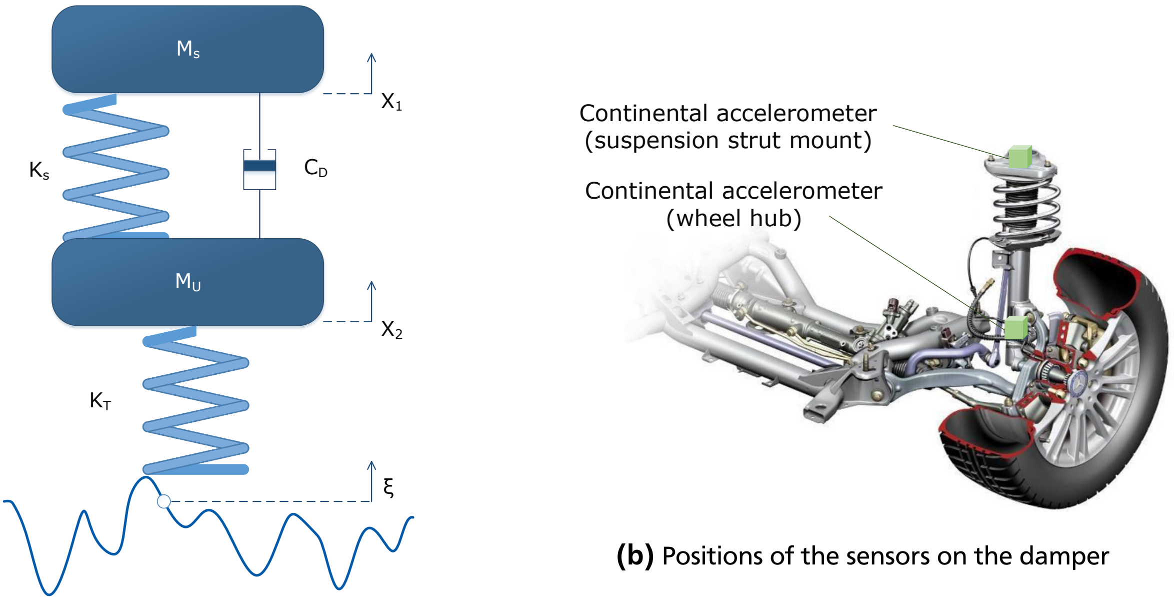
are forming a WSN within the vehicle. A general overview of the WSN concept in the MoDe project is shown in Figure 7.2.

7.1.2 Condition monitoring of the damping system

With regard to computationally intensive data processing, the most relevant test case is the damping system. As it requires the simultaneous and continuous analysis of several data streams from accelerometer sensors, the distributed computing approach enabled by the sensor node architecture proposed in this thesis is very appropriate. Furthermore, the wireless communication protocol selected for the in-vehicle sensor network has a very low throughput [Iis]. Packets with a limited payload can only be sent at a rate below 10 Hz. Edge-processing is therefore mandatory in order to transfer the condition indicators to the truck central control unit. The rest of this section will then focus on the infrastructure deployed for this part.

Three different health monitoring methods have been considered [Pee12]. The first method uses the *transmissibility* as an indicator of the damper condition. Transmissibility is a quantity widely used in vibration engineering and is an indicator of the relative vibration levels between two points. Figure 7.3b shows a model of a vehicle suspension system with two degrees of freedom (X_1 represents the vertical displacement of the vehicle body and X_2 represents the displacement at the wheel hub). The symbols in Figure 7.3a refer to the sprung mass [M_S] (portion of the vehicle mass supported above the suspension), the unsprung mass [M_U] (includes the masses associated to the braking system, tire, wheel rim, etc.), the spring stiffness [K_S], the radial stiffness of the tire [K_T], the damping ratio of the shock absorber [C_D] and the road excitation [ξ].

The transmissibility T_{12} between the points 1 and 2 depends on the spring stiffness K_S, the vehicle mass M_S (which depends on the vehicle loading) and the shock absorber damping ratio C_D [Pee11]. As K_S stays constant during the vehicle lifetime and M_S can be estimated before the start of each vehicle, the transmissibility becomes a function of the damping ratio. As this value is changing with aging or malfunction, the transmissibility can be used as a damage indicator of the damping system.



(b) Positions of the sensors on the damper

(a) Damper model

Figure 7.3.: Concept for transmissibility measurement

The transmissibility values are computed in the frequency domain. The transmissibility function can be seen as a frequency response function between the input (acceleration at the wheel hub) and output (acceleration at the strut mount) signals acquired during the normal operation of the vehicle. Computing this frequency response function requires an accurate and reliable processing of the time series. First, the accelerometer data needs to be filtered to remove any DC component. This is typically done using an IIR filter. The data can be then re-sampled at a lower frequency in order to reduce the amount of data and reduce the signal noise. The power spectra of the signals are then required. A popular non-parametric method to estimate a power spectrum is known as the Welch's modified periodogram [Pee+07]. It first relies on the DFT of the input sequences, which can be written for a sequence y of length N as:

$$Y(\omega) = \sum_{k=0}^{N-1} y_k w_k e^{\frac{-2\pi i \omega k}{N}} \tag{7.1}$$

where w_k is a windowing function, *e.g.* $w_k = 0.5\left(1 - \cos\left(\frac{2\pi k}{N-1}\right)\right)$ for the Hanning window. The DFTs of P small overlapping frames of the complete time series are computed. Then, the power spectrum, also denoted auto power spectrum, is evaluated over each time frame with the formula:

$$S_{yy}^{(j)}(\omega) = \frac{1}{\sum_{k=0}^{N-1} |w_k|^2} Y(\omega) Y^*(\omega). \tag{7.2}$$

When the power spectra of all time blocks are available, the global power spectrum of the time series can be computed as their average:

$$S_{yy}(\omega) = \frac{1}{P} \sum_{j=0}^{P-1} S_{yy}^{(j)}(\omega). \tag{7.3}$$

This averaging method tends to reduce the noise in the spectrum, which is a desirable effect. Similarly, Welch's method is used to compute the cross-spectrum of the two input sequences. The cross-spectrum of the sequences x and y over a time frame j is defined as:

$$S_{xy}^{(j)}(\omega) = \frac{1}{\sum_{k=0}^{N-1} |w_k|^2} X(\omega) Y^*(\omega). \tag{7.4}$$

The frequency response of the system can finally be computed using an H_1 estimator as:

$$H(\omega) = \frac{S_{xy}(\omega)}{S_{xx}(\omega)} \tag{7.5}$$

The transmissibility T_{12} of the damping system is then estimated with:

$$T_{12}(\omega) = \frac{S_{12}(\omega)}{S_{11}(\omega)}. \tag{7.6}$$

A test drive on a passenger car with and without a damaged damper has been performed in order to test the reliability of the target sensors and algorithms. Continental BSZ04D low-cost automotive accelerometers were compared with laboratory sensors in order to validate their ability to sense damaged dampers. A picture of the LPSIP hardware with the Continental accelerometers is shown in Figure 7.7. Results of the transmissibility computation during this test are depicted in Figure 7.4. It can be clearly observed that a damaged suspension will result in a higher transmissibility. This difference was observed with both Continental and laboratory accelerometers. The transmissibility function can thus be used as a reliable estimator for the condition monitoring of the vehicle damping system. However, it can only be used in conjunction with other indicators such as the vehicle mass and the road profile. These aspects were investigated in other work packages of the project related to the estimation of the remaining useful lifetime and maintenance strategies [Mod].

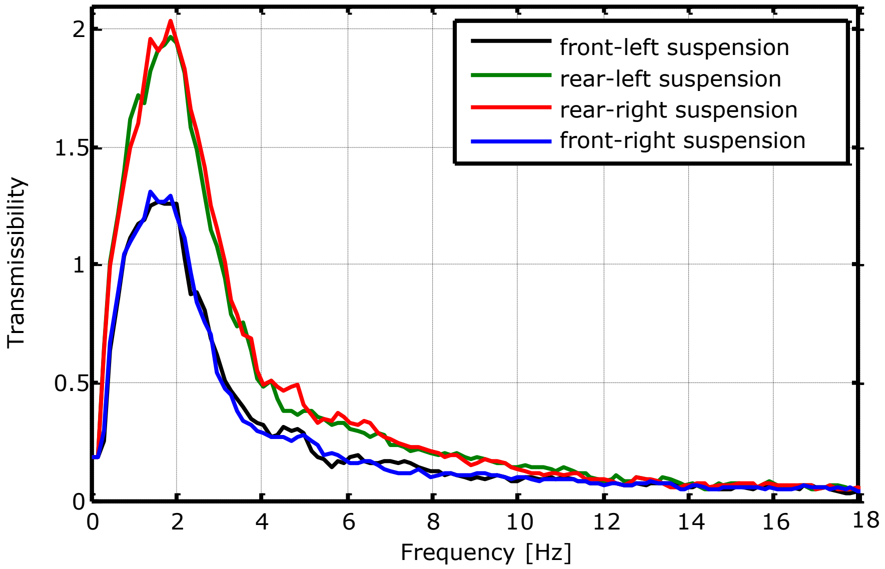
A second method investigated to monitor the condition of the shock absorber is the random-decrement technique [Bei+14]. Here, the acquired sensor data time series are averaged when a given trigger condition is fulfilled, *e.g.* a threshold value is crossed. The result of this averaging process is denoted as the random decrement signature. After a large number of averaging processes, the random part of the signature will tend to disappear, leaving only data which can be interpreted as the response of the system to the conditions defined by the trigger. The information contained in this signature can thus be used to analyze the system's behavior. Although differences could be observed between the random-decrement signature of a damaged and undamaged shock absorber, they were difficult to assert. In particular, tests validating the approach were performed with laboratory sensors but COTS sensors will tend to make the difference very difficult to detect. For this reason, this technique has not been retained for implementation on the platform [Pee12].

The third technique is based on the computation of correlation factors between the acceleration at the wheels and the acceleration at the center of gravity of the vehicle. These factors are defined as [MD08]:

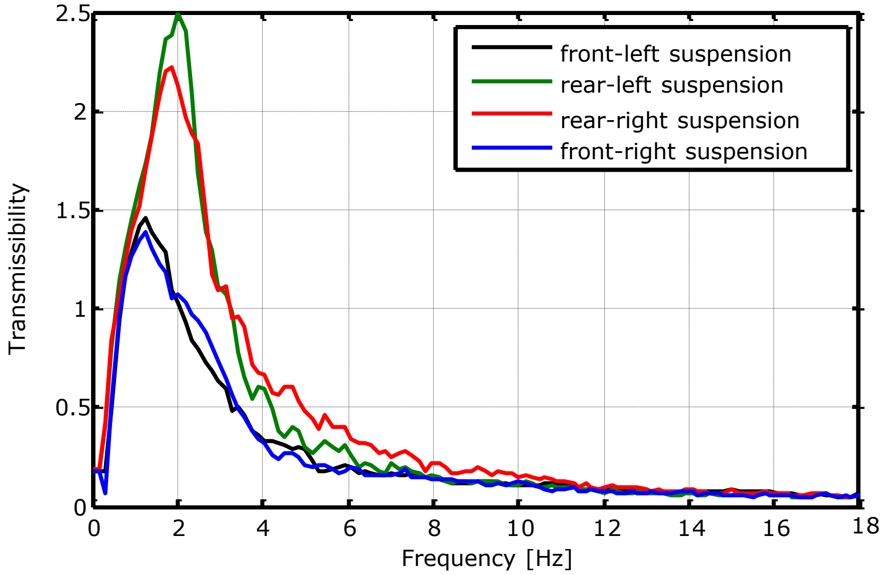
$$K_{xy} = \frac{\sum_{i=1}^{N} x(i)y(i)}{\sum_{i=1}^{N} x(i)x(i)} \tag{7.7}$$

where x is the vertical acceleration at the center of gravity of the vehicle and y is the vertical acceleration at the wheels level. Simulation and tests have shown that this method is reliable to detect damaged shock absorbers. Over time, the correlation coefficient of a damaged damping system will tend towards a value close to zero at a slower rate than undamaged systems. Figure 7.5 shows how the coefficient tends rapidly towards a constant value and stays constant. These graphs were generated using real measurements on a vehicle with undamaged dampers.

This method has however the disadvantage that the computation of the correlation coefficient requires the acceleration time series at two different locations of the vehicle (wheel and center of gravity). As the sensor network protocol does not support the transfer of large data streams at high rate, this method could practically be applied only if a wired connection is available between the sensor located at the center of gravity and the wheel sensors. This setup would however cancel the inherent advantages of a wireless network. In addition, even if the data transfer would be possible, a high level of synchronization is required between nodes, so that the correlation coefficient can be estimated in a reliable way. As the



(a) Undamaged shock absorbers



(b) Rear-left timeworn shock absorber

Figure 7.4.: Transmissibility function of the damping system during the test drive [Bei+14]

implementation is straightforward, the algorithm has been ported on the hardware accelerator for the sake of feasibility analysis.

In order to validate this method, a test drive with a passenger car equipped with accelerometers has been realized on November 28th 2012 at the Fraunhofer LBF institute in Darmstadt. Analog Devices ADW22035Z MEMS sensors were positioned at each damper in order to measure vertical acceleration. An Analog Devices ADXL325 3-axis accelerometer was positioned at the center of gravity of the car. Three LPSIP nodes were deployed to respectively measure acceleration at the front dampers (two sensors), rear dampers (two sensors) and center of gravity (three sensors). The route of the car for the test was a typical inner city curse of five kilometers. As a damper could not be easily replaced by timeworn equipment, two test drives were performed with different loading and tire pressure. Figure 7.5 shows the times series and the running correlation coefficients for a test drive realized with three passengers and a tire pressure of 1.8 bars. A second test was performed with five passengers and a tire pressure of 2.4 bars, but the results showed that these modifications had no or a very little impact on the correlation coefficient [Pee12].

During the tests, the correlation coefficients could not be directly computed on the sensor nodes as the accelerometer at the vehicle's center of gravity was separated from other sensors. The computation of the correlation coefficients on the FPGA was nevertheless tested offline by virtually injecting the sensor data to the device.

7.2 Motivation for on-demand reconfiguration

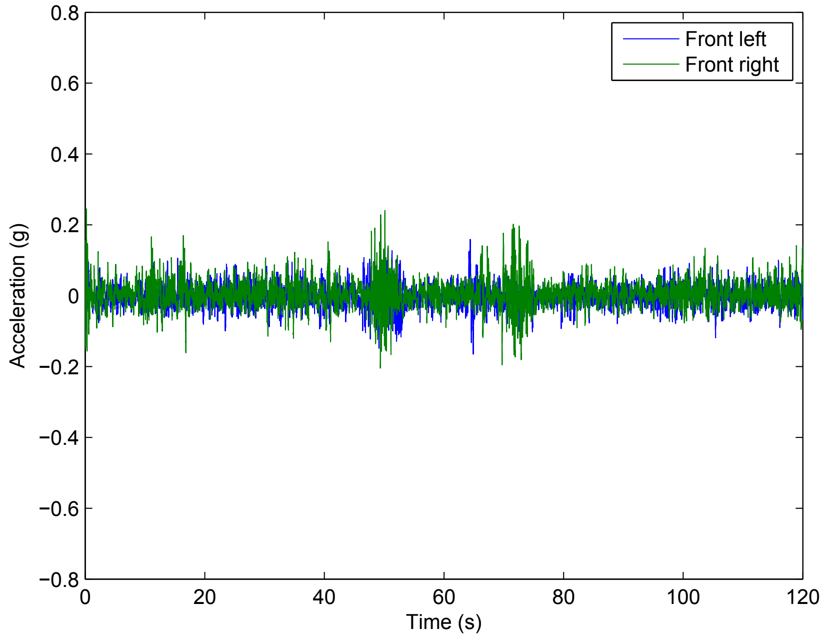
In the MoDe project, dynamic reconfiguration was mainly motivated by the possibility to remotely update the firmware. Indeed, after deployment of the sensor nodes in a fleet of vehicles, a manual reprogramming of each board would cost a significant amount of time. A framework to update both software and hardware has been then developed to enable a seamless remote modification of the core functionalities of the node [Els12].

In addition, the hardware accelerator can also be dynamically reconfigured according to environmental conditions. As some of the algorithms depend on the mass of the vehicle or on the road profile, other algorithms can be used if these parameters are not available or if they perform better under specific circumstances.

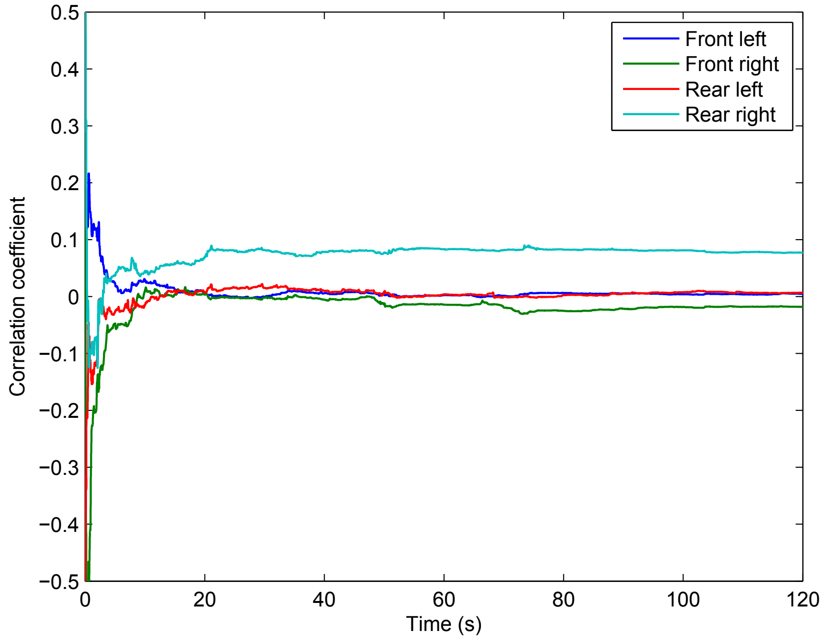
7.3 Implementation and results

The condition monitoring algorithms were implemented on the LPSIP platform, which was specifically developed in the frame of this project. The radio module was implementing the operating system and communication protocol developed by the WSN research group from the Fraunhofer Institute for Integrated Circuits [Iis]. The profile of the algorithms described in the previous section suggest a high demand for DSP operations. The reconfigurable FPGA overlay implemented with the help of the (GECO)^2 tool on top of the IGLOO FPGA is then very close to the one presented in Figure 5.12a. A notable change is the replacement of the CORDIC unit by an iterative divider. Indeed, division is required for both the computation of the transmissibility function and the correlation coefficient, while CORDIC would only be required for the initialization of the FFT algorithm (computation of twiddle factors). It has been assumed that these factors could be computed by the microcontroller and transfered to the FPGA during an initialization phase. Four sensor interfaces are available as the LPSIP sensor board is equipped with four ADCs. A 6.78 MHz clock signal generated on the sensor module of the LPSIP is used to clock both the FPGA and the ADCs. No internal ring oscillator is used in this design. A summary of the resource consumption of this architecture were included in Table 7.1.

Two main tasks have been programmed for the architecture: the computation of the transmissibility function as described in the previous section and the computation of the correlation factor. Both tasks are based on running computation, *i.e.* the FPGA is constantly sampling and updating the result. For the computation of the transmissibility, the data from the two accelerometers is first sampled at 200 Hz, the



(a) Time series



(b) Running correlation coefficient

Figure 7.5.: Test measurements on a passenger vehicle

Cells	Blocks memory	Max. Frequency	Size of context
19,784 (80.5 %)	27 (81.2 %)	14.9 MHz	272 bits

Table 7.1.: Resource consumption of the FPGA overlay used for damper condition monitoring

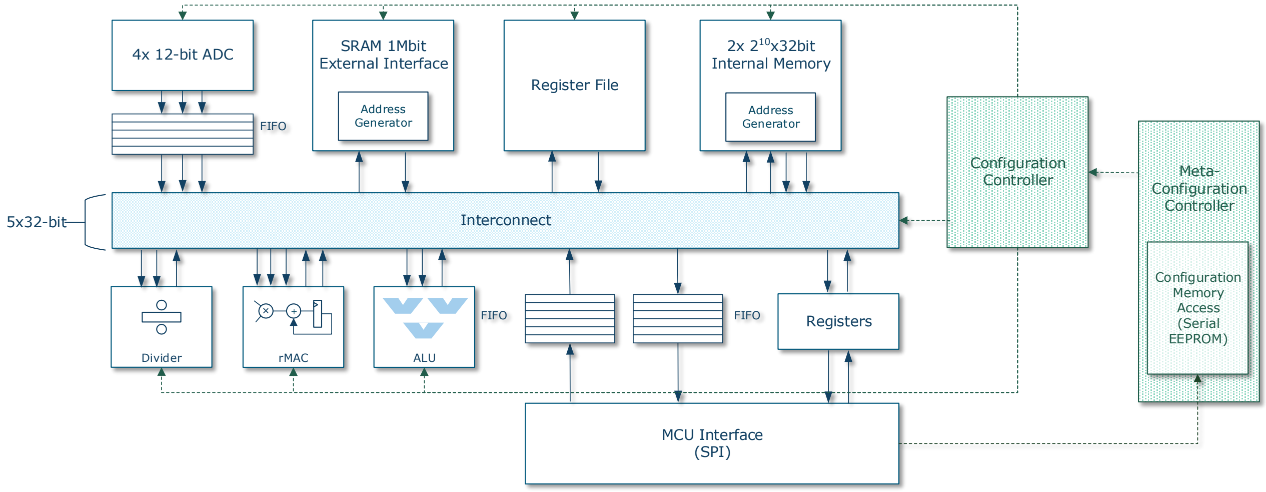


Figure 7.6.: FPGA overlay for the condition monitoring of the shock absorber with LPSIP

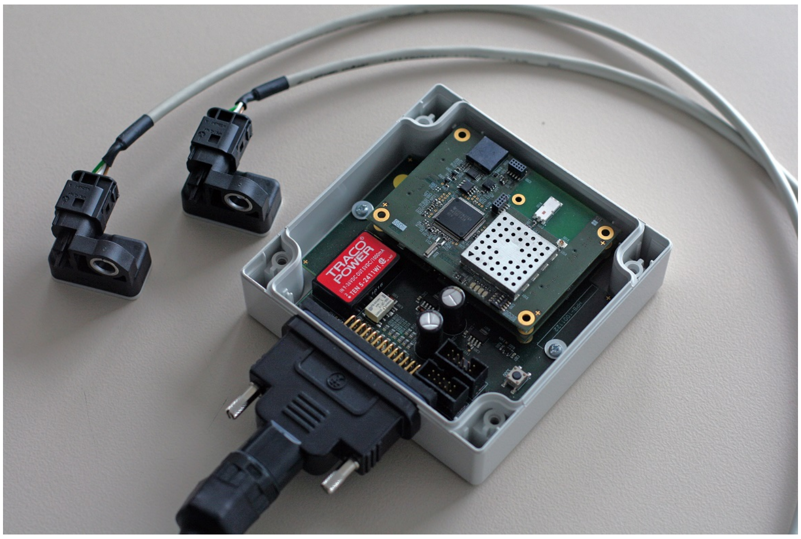


Figure 7.7.: LPSIP hardware with Continental BSZ04D wheel acceleration sensors

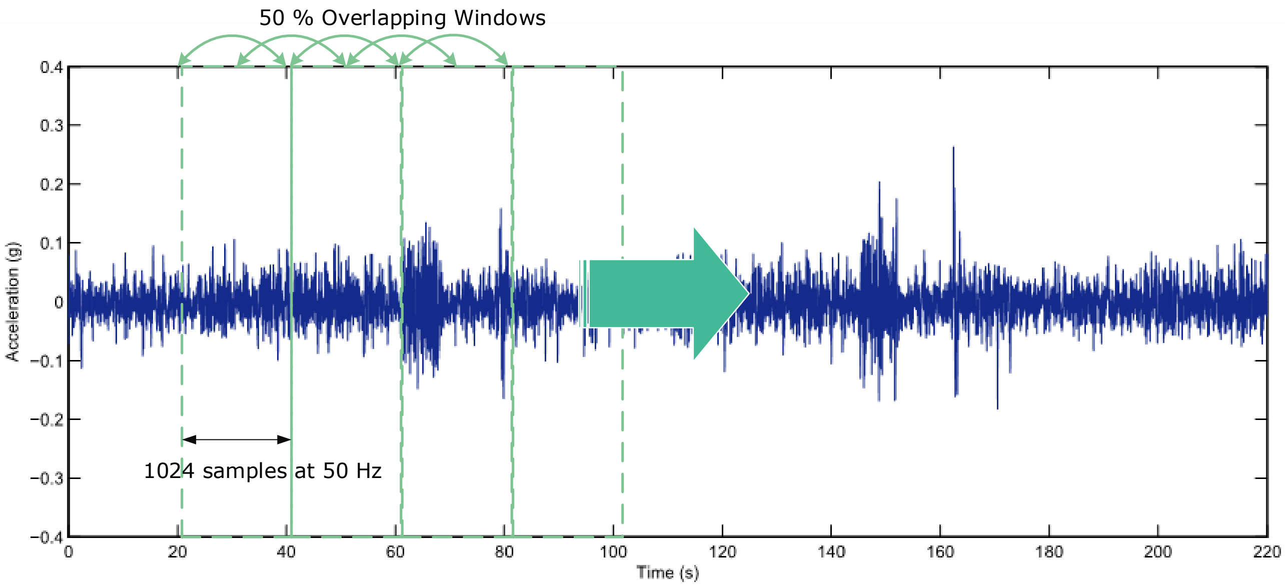


Figure 7.8.: Computation of Welch's modified periodogram on the vertical acceleration time series

DC part is removed and the data is down-sampled at 50 Hz before being stored in the SRAM memory. Using an overlapping window of 50 %, the auto-power spectrum and the cross-power spectrum of the two streams are regularly computed using a 1024 points FFT. This approach is illustrated by Figure 7.8. Half-spectra are saved back in the memory. After five minutes of sampling, the computed spectra are averaged and the transmissibility function is computed as their ratio. For the correlation factor, a new value is computed every time a new sample arrives. The accumulated values of the autocorrelation and the cross-correlation are summed with the product of the new samples. The ratio of the accumulated values is then computed. The complete flow of the tasks is represented by Figure 7.9.

Table 7.2 reports the execution costs of the considered tasks. The computation of the correlation coefficient takes a very small amount of time since it is reduced in accumulating values and computing their ratio. The global average power consumption of the device when applying this technique is therefore almost equal to the power consumption of a scenario with sampling only, *i.e* 201 μW at 50 Hz.

For the transmissibility approach, computing both auto- and cross-power spectra costs for each window 8 milliseconds (a total of 30 windows are considered). The final averaging and ratio of the spectra has then a negligible duration in comparison to the total time spent for the spectral transformation. Figure 7.10 shows the detailed distribution of the time spent for each of the sub-tasks. The average power consumed during the execution has been estimated using the power estimation tools from the FPGA vendor. This value presents only the power spent by the FPGA and does not include external components such as the microcontroller, memory or the clock generation circuit. When no computation is required, the FPGA is automatically set in Flash*Freeze mode between the acquisition two samples. During sampling, the average power consumption of the device is efficiently lowered down to 475.2 μW. The overall duty cycle of the FPGA when considering both acquisition and processing time is lower than 0.1 %. This drastically reduces the contribution of the FPGA to the global power consumption of the sensor node, which stays below 500 μW.

The computation of the correlation coefficient is potentially more efficient than the transmissibility. In addition to the short processing overhead, the external SRAM memory is not required to store intermediate results.

For all tasks, the reconfiguration overhead stays low, even if a serial configuration memory is used on this platform. Indeed as the loading of the configuration data as to occur only once at the initialization of the task, it does not impact significantly the overall processing time.

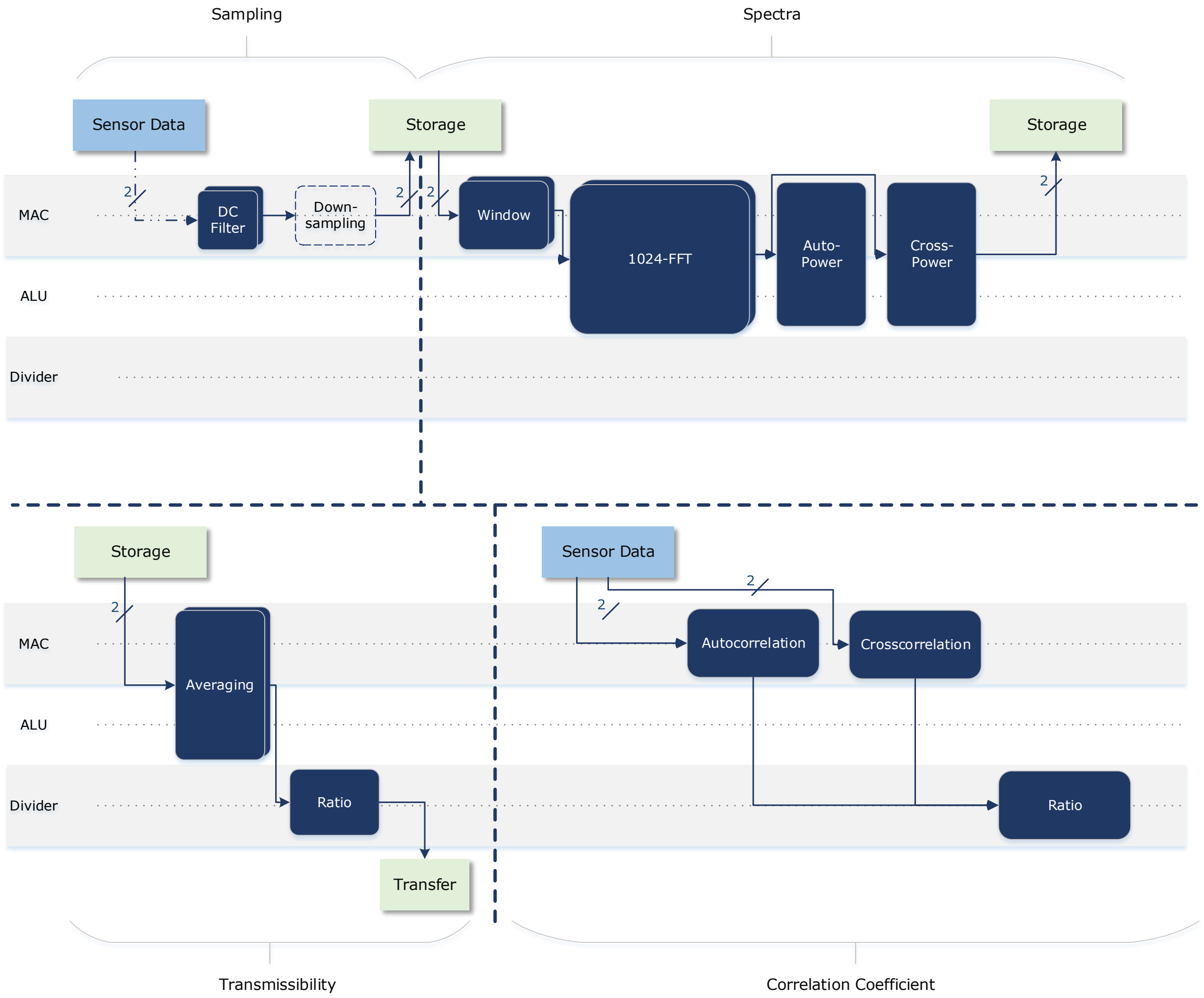
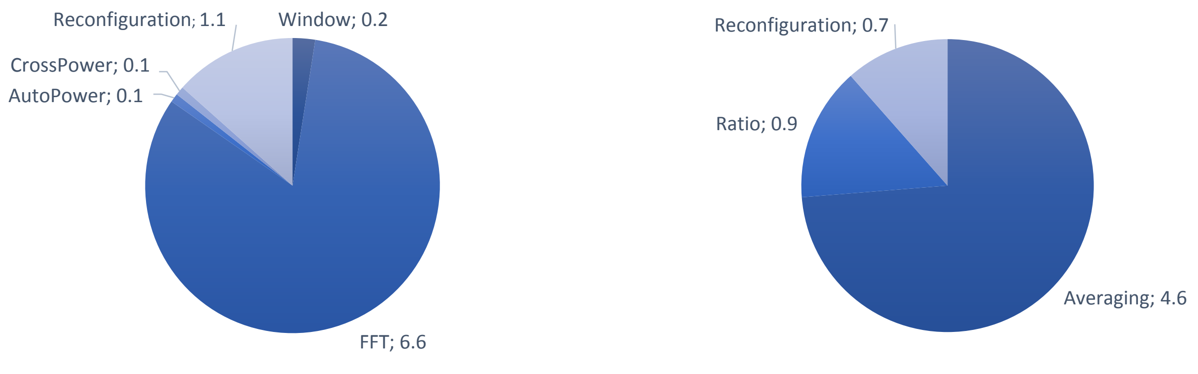


Figure 7.9.: Time-space partitioning of the damper condition monitoring tasks on the overlay architecture

Task	Duration (ms)	Power consumption (mW)	# Instructions	Bitstream size (bits)
Spectra	15.9	5.4	28	1,792
Transmissibility	11.2	4.2	9	480
Correlation coefficient	0.012	1.7	5	320

Table 7.2.: Performance metrics for the damper condition monitoring tasks



(a) Spectra

(b) Transmissibility

Figure 7.10.: Breakdown of time spent for the damper condition monitoring tasks (time is given in milliseconds)

In terms of accuracy, the fixed-point implementation was particularly problematic for the tasks accumulating values on long data streams, *i.e.* the final step of the transmissibility and the correlation coefficient.

7.4 Conclusion

Implementing condition monitoring tasks on distributed sensor nodes require particular care in the choice of algorithms and placement of sensors. Even if three different methods were foreseen to detect damages in the shock absorber, only the one based on the transmissibility function could be meaningfully ported on the target sensor network. The restrictions in term of bandwidth and synchronization capability of the wireless protocol prevented the deployment of the other techniques although they potentially generate a lower computational overhead. On the other hand, the FPGA-based accelerator was providing all the necessary computational resources to implement the computation of the transmissibility function online. The execution scheme of the FPGA overlay was suitable for a smooth and autonomous implementation of the algorithm without intermediate intervention from the microcontroller. The computation time and the average power consumption of the hardware accelerator stay very low thanks to the implementation on the Flash FPGA.

In the related work, the authors of [Ven+08] suggested to use a smart wireless sensor to implement the transmissibility function for the condition monitoring of a shock absorber. This work can be seen as the realization of this vision. Other approaches for in-vehicle wireless sensors are mostly based on centralized processing, such as the sensor nodes from MicroStrain [Mic12]. This implies that all measurements must be wirelessly transmitted to the central unit, which is not a viable solution for a long-term deployment with a battery-powered node. The proposed approach reduces both the risks of data losses or corruption during the wireless transfer and the processing energy costs, which are minimized by the hardware acceleration.

In conclusion, it was demonstrated that deploying hardware-accelerated sensor nodes for the condition monitoring of a vehicle shock absorber is a feasible and reliable approach. By combining a low-power wireless communication protocol and a low-power processing engine, energy costs are reduced at the two critical levels of a high-bandwidth sensing application, so that long-term deployments can be foreseen. The functionality of the node is dynamically reconfigurable over a wireless link, which minimizes the maintenance costs of the network and enables customization of condition monitoring algorithms according to load or environmental conditions. With this set of features, the platform is a potential candidate to implement vibration-based condition monitoring in other types of vehicles, such as aircraft or trains, or different types of mechanical systems, such as bridges or buildings, where similar sensor setups and data processing schemes can be applied.

8 Diagnosis of Induction Motors

Contents

7.1 Concept **124**

7.1.1 General overview 124

7.1.2 Condition monitoring of the damping system 125

7.2 Motivation for on-demand reconfiguration **129**

7.3 Implementation and results **129**

7.4 Conclusion **134**

Electrical machines like induction motors are widely used in industry, *e.g.* for traction purposes. A faulty operation of the motor would potentially damage it and induce a down time of a plant. In critical industrial applications like a production chain, these inactive periods must be avoided as much as possible in order to maximize the productivity. The condition of electric motors can be monitored under different aspects: a review of standard diagnosis techniques can be found in [NTL05]. This article notably states that the most common methods are based on currents analysis, vibration patterns or analysis of thermal effects. The techniques based on the analysis of the stator currents are referred in the literature as Motor Current Signature Analysis (MCSA). Their main advantage is the use of non-intrusive sensors at only one point of the machine. The following section describes the implementation of MCSA techniques on the hardware-accelerated wireless sensor node. At first, the considered faults and the corresponding features in the motor current signal are shortly explained in Section 8.1. The signal processing techniques enabling their accurate detection are presented. This section is followed by a short overview on the related work. The implementation details and the results are then given in section 8.3[1].

8.1 Concept

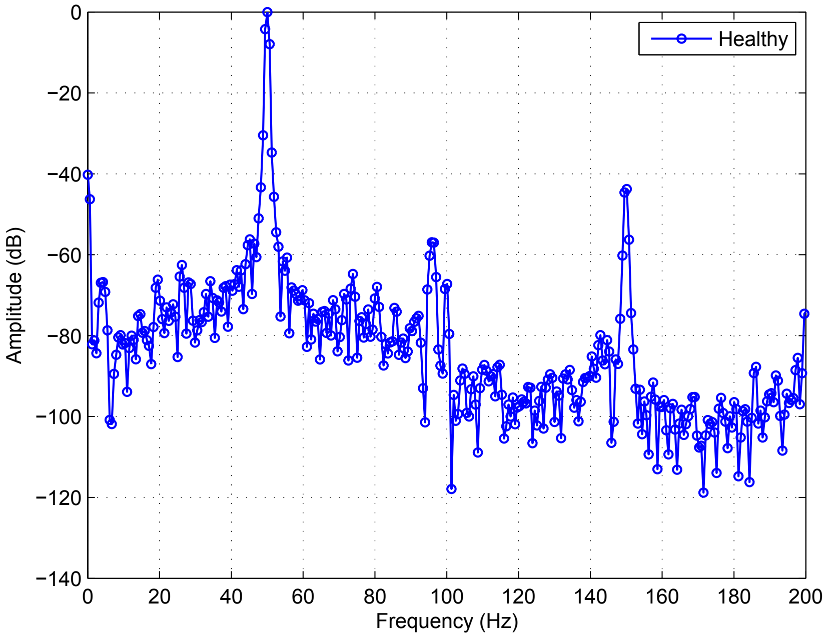
This section briefly reviews the recent literature for stator current indicators of the considered faults, *i.e.* dynamic eccentricity, broken bars and inter-turn short circuit. These faults were selected since they are common references in motor fault diagnosis. They have in common to induce the emergence of specific components in the current spectrum, so that spectral analysis techniques can be used to identify them. Beyond the theory, the presence of these components has been verified using a finite element model of a motor [Mar+14] and experiments on real machines. In particular, the curves shown in Figure 8.1 and 8.2 were extracted from real measurements where the different faults were artificially recreated. On all curves, the amplitude has been normalized with respect to the component at the supply frequency (50 Hz). The characteristics of the motor under test are summarized in Table 8.1. For all measurements, the motor was operating at a speed of 1350 revolutions per minute.

8.1.1 Detection of broken bars

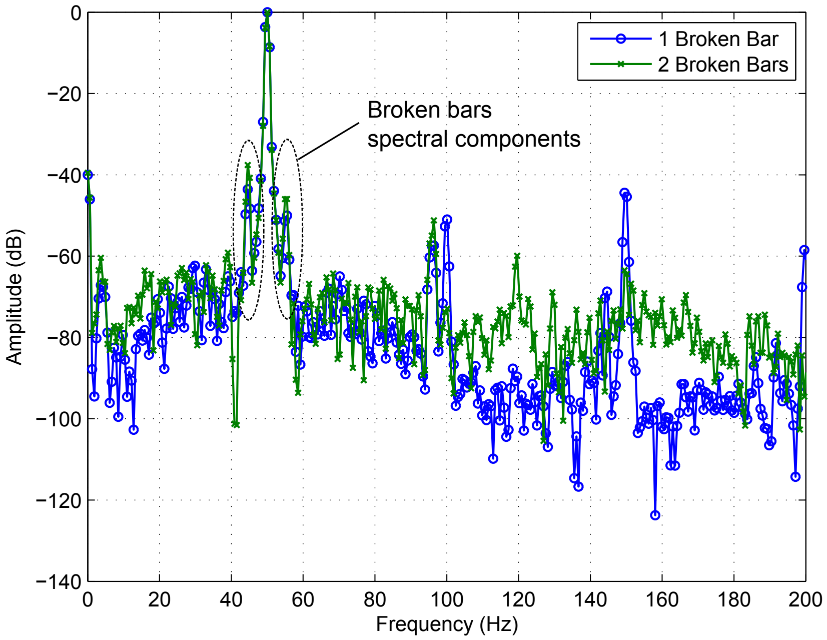
Broken bars are one of the most rare faults. Whenever a bar breaks in the rotor, the current distribution of the squirrel cage loses its symmetry. This asymmetry produces an inverse field that induces new spectral

---

[1] The results of this section are based on the conference papers [Phi+12a; Mar+13]. This work is the fruit of a collaboration with Javier Martinez Garcia-Teronio and Antero Arkkio from the School of Electrical Engineering, Aalto University, Finland

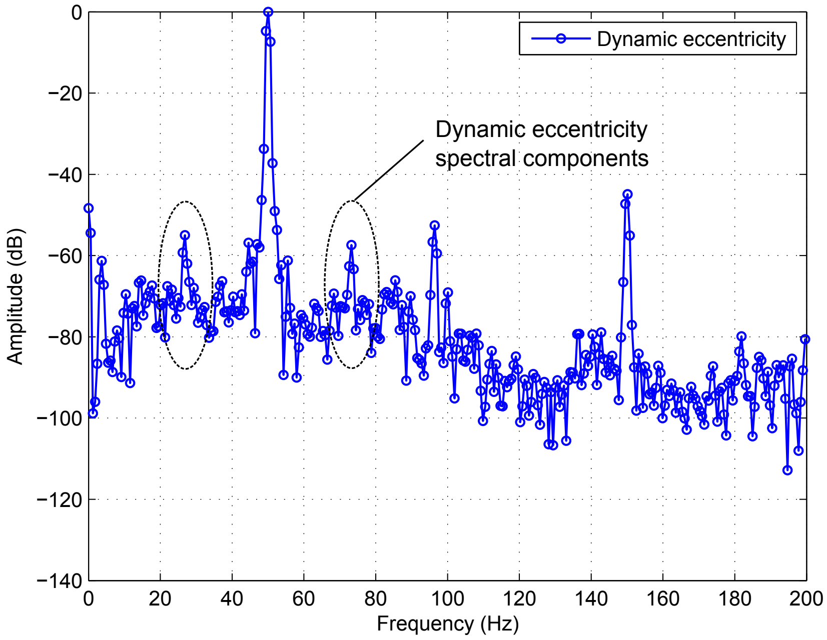


(a) Healthy

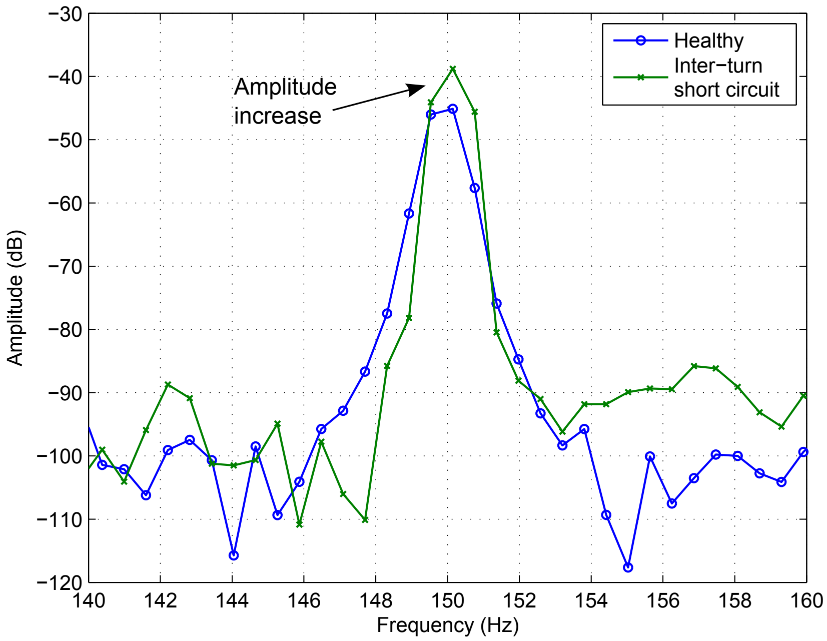


(b) Broken bars

Figure 8.1.: Spectral analysis of the induction motor currents (Part 1)



(a) Dynamic eccentricity



(b) Inter-turn short circuit

Figure 8.2.: Spectral analysis of the induction motor currents (Part 2)

Pole pairs, p	2
Parallel branches	2
Stator slots, Z_1	48
Rotor slots, Z_2	40
Rated slip	5.5 %
Rated voltage	380 V
Rated current	41 A
Supply frequency	50 Hz
Rated Power	22 kW
Skew factor, c_{sk}	0.98
Electric connection	Star

Table 8.1.: Nameplate and constructive parameters of the studied motor

components in the stator windings. In addition to these new stator current components, the inverse magnetic field interacts with the existing one and produce magnetic torque ripple. These ripples will eventually modulate the pulsating frequency of the magnetic waves inducing a second set of spectral components. This process is repeated with the other set of magnetic waves rotating in the air-gap such as the ones produced by the slotting effect. Induced components can be expressed as [Khe+09]

$$f_{bb}(n_{b1}, n_{b2}) = (n_{b1}(1-s) \pm s \pm 2n_{b2}s) f_s \tag{8.1}$$

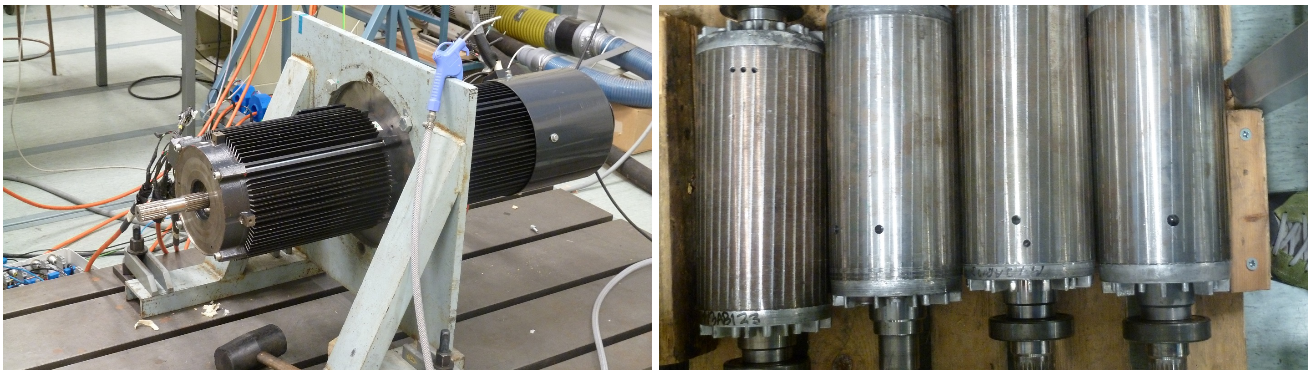
where n_{b1} and n_{b2} are the broken bar harmonic indexes. In general, the analysis is reduced to the harmonics define as:

$$f_{bb} = (1 \pm 2s) f_s. \tag{8.2}$$

The photo 8.3b shows how this fault was artificially recreated. In Figure 8.1b, one can notice that two components appear in the spectrum, respectively at 44 Hz and 56 Hz, close to the main supply frequency.

A problem arising from this proximity to the main frequency component f_s is the need for an increasing spectral resolution when the slip of the motor decreases, *i.e.* when f_{bb} is getting very close to f_s. Indeed, because of the effect of spectral leakage in the discrete Fourier transformation, one might not be able to identify the faulty spectral components which are hidden by the leakage of the main peak, as shown in Figure 8.4. Increasing the number of points of the DFT solves this issue but considerably increases the processing and memory requirements of the sensor node hardware (the operational complexity of the FFT is $O(N\log(N))$, which is not a viable solution when considering a resource-constrained device. An alternative solution consists in applying a frequency shift of the signal towards the region of interest followed by a decimation filter. This technique, known as Zoom-FFT, reduces the computational complexity for the analysis of a specific area of the spectrum with increased resolution [Phi+12a]. Full details on the utilization of this algorithm in spectral analysis for fault detection of induction machines can be found in [Bel+08].

However, this approach is still not fully reliable in all operating conditions. When the machine is very lightly loaded or unloaded, the faulty sideband components are practically overlapping the supply frequency. There are also other frequency components like ball bearing defects, voltage oscillations or load fluctuations frequencies which can appear in the spectrum and overlap with the ones associated with rotor bar rupture. Another monitoring technique overcoming these issues is the analysis of the discrete wavelet transform of the current signal during the startup of the motor. During this period, f_{bb} is not stationary and can be tracked moving between different frequency bands. Figure 8.5b shows this trend in the wavelet coefficients of a motor with two broken bars. The terms A_i and D_i correspond respectively to the approximation and details wavelet coefficients at level i. A Daubechies-40 mother wavelet was used



(a) Motor used to perform the experiments **(b)** Holes drilled to create artificial broken bars

Figure 8.3.: Pictures of the experimental setup

in this experiment. A good indicator to measure the severity of the fault with this wavelet coefficients is to compute the variability, *i.e* the statistical dispersion, of the coefficients as shown by the bar diagram in Figure 8.4b[2].

8.1.2 Detection of dynamic eccentricity

This fault is produced when there exists a shift between the geometric centers of the stator and rotor. The degree of eccentricity is defined as the amount of this shift over the total air-gap length. This shift will then rotate at the same speed as the mechanical speed. When this situation occurs, the spectrum of the stator current is modified due to the existing asymmetric air-gap length. More precisely, the set of additional frequencies, f_{dyn}, added to the stator current spectrum is expressed as [Nan+11]

$$f_{dyn}(n_r, n_d, n_{st}) = \left[(n_r R \pm n_d) \frac{1-s}{p} \pm n_{st} \right] f_s \tag{8.3}$$

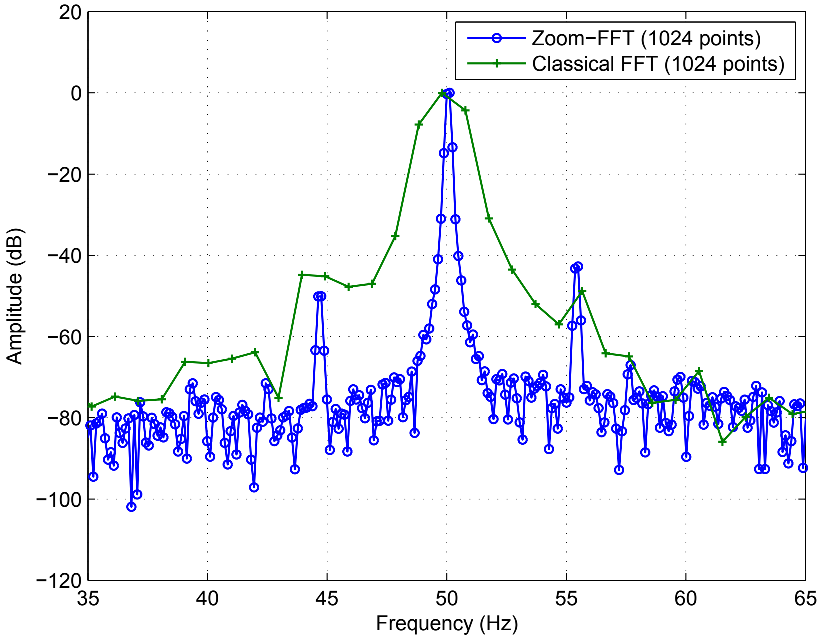
where f_s id the supply frequency, s is the slip parameter, p the number of pole pairs, R the number of rotor bars, and n_r, n_d and n_{st} are the harmonic index for respectively the rotor slot permeance, dynamic eccentricity and stator time. In Figure 8.2a, two faulty components appear in the spectrum at 26.8 Hz and 73.1 Hz respectively. These components can be identify by simple spectral analysis but their amplitude is relatively small. This requires a particularly high level of accuracy for reliable detection. As the implementation is realized in fixed-point on the target platform, particular care should be taken to reduce the rounding errors during the computation. A correction algorithm detailed in appendix B.3 has been introduced for this purpose.

8.1.3 Detection of inter-turn short circuit

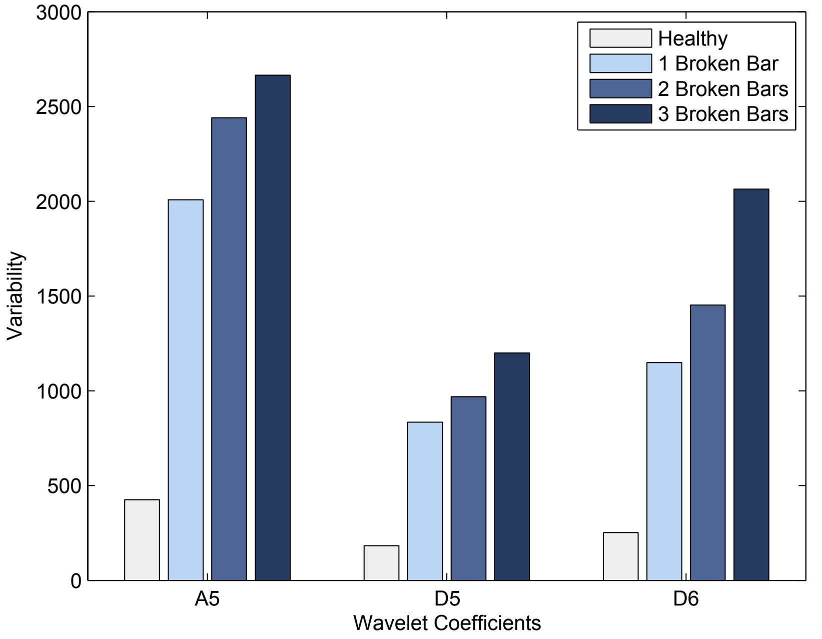
Inter-turn short circuits are the second cause of fault in electric motors [WN10]. Inter-turn arises from over-voltages or hot spots that can damage the insulation of the coil. When the insulation is damaged, a short-circuit is appearing between different turns going through the same slot. This short-circuit is characterized by a short-circuit resistance. Short-circuited turns create a new coil where a circulating current is induced. The target of this circulating current is to oppose the main rotating air-gap flux wave in the motor. In [WN10], the authors monitored the spectrum of the three-phase currents Fortescue's

---

[2] These aspects were studied in details in the Master thesis [Abd13]

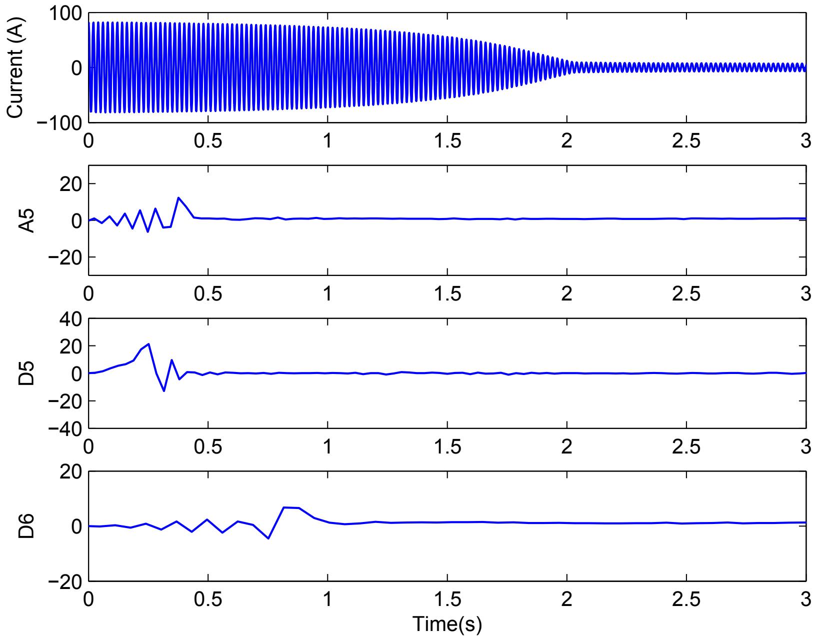


(a) Zoom-FFT of the current spectrum for one broken bar (decimation factor 8)

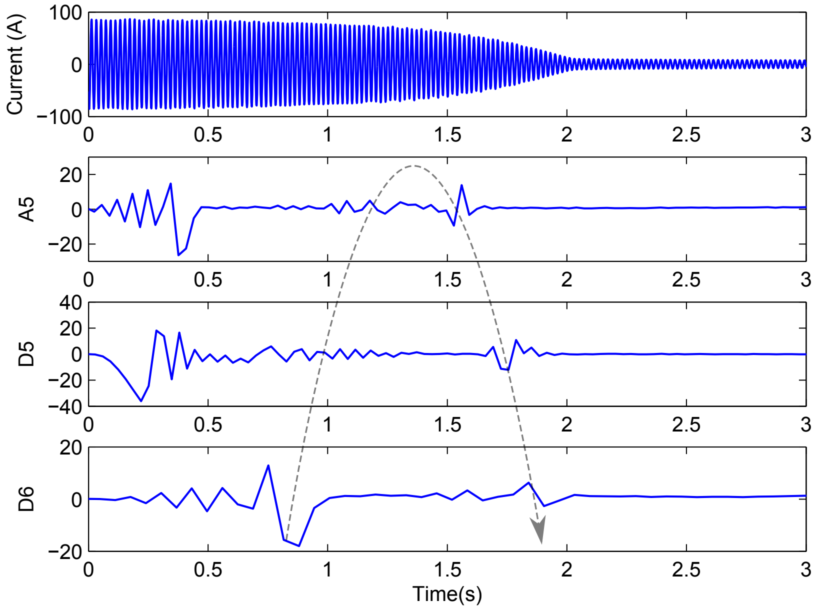


(b) Variability of the wavelet coefficients

Figure 8.4.: Alternative techniques for the detection of broken bars (Part 1)



(a) Wavelet transform of the transient current (healthy)



(b) Wavelet transform of the transient current (two broken bars)

Figure 8.5.: Alternative techniques for the detection of broken bars (Part 2)

transformation and focused on the locations at $\pm 3f_s$. The Fortescue's transformation converts the three-phase I_a,I_b and I_c of the electrical motor into the set of symmetrical components I_0,I_1 and I_2 (respectively zero, positive and negative sequences) such as

$$\begin{pmatrix} I_0 \\ I_1 \\ I_2 \end{pmatrix} = \frac{1}{3} \begin{pmatrix} 1 & 1 & 1 \\ 1 & a & a^2 \\ 1 & a^2 & a \end{pmatrix} \begin{pmatrix} I_a \\ I_b \\ I_c \end{pmatrix}, \tag{8.4}$$

where $a = e^{\frac{2\pi i}{3}}$. Figure 8.2b shows the slight increase of the component located at $3f_s$ in the negative current sequence of a motor suffering from an inter-turn short circuit. Here again, highly reliable amplitude estimation is required to identify this fault as differences are very low. For this purpose, a lightweight signal processing technique correcting spectral leakage has been introduced [Mar+13]. Based on considerations on the used spectral window and neighboring points in the discrete spectrum, the amplitude of a spectral peak can be estimated with a relative error lower than 1 % when applying this method.

8.2 Motivation for on-demand reconfiguration

The complete processing chain required by the studied MCSA flow is based on many subsets that could not fit as individual cores on the FPGA (filtering, Fortescue's transformation, Fourier transform, wavelet transform, magnitude computation). Intuitively, saving resources to implement these algorithms would imply sharing a multiply-accumulate unit, which is precisely what the FPGA overlay presented in Chapter 5 is enabling. In addition, the architecture and the related tools combine all the necessary elements for an easy and efficient control of this operator.

At runtime, there is a need to select whether the Fourier transform or the wavelet analysis is used. Indeed, as the effectiveness of the algorithm depends on the state of the motor (for example steady state for detection of broken bars with FFT, transient state with wavelet analysis), the functionality of the hardware accelerator must be adapted accordingly. For instance, this can be controlled by commands sent to the event-based middleware (listings 8.1).

```
CONFIG Inter—turn WHEN @300s
CONFIG bb—fft WHEN @300s QUEUE
CONFIG wavelet WHEN startup FORCE
```

Listing 8.1: Commands for context-aware selection of algorithms for sensor data analysis

Another parameter affecting the selection of the signal processing algorithm is the load of the motor. For instance, the frequency of the components related to the broken bars fault is getting closer to the supply frequency when the load is decreasing. In this case, the traditional FFT approach must be replaced by the Zoom-FFT algorithm, as mentioned in the section dedicated to the broken bars fault. These algorithms can then be dynamically programmed on the sensor node according to operation of the motor.

```
CANCEL bb—fft
CONFIG zoom—fft WHEN @300s QUEUE
```

Listing 8.2: Commands for modifying the scheduling of a task

At last, it was shown in [Med+10] that the vibrations of the motor could also be analyzed by mean of the Fourier transform to detect and monitor the considered faults. Thus, if accelerometers are available, the processing unit can be reconfigured to analyze these signals as well.

8.3 Implementation and results

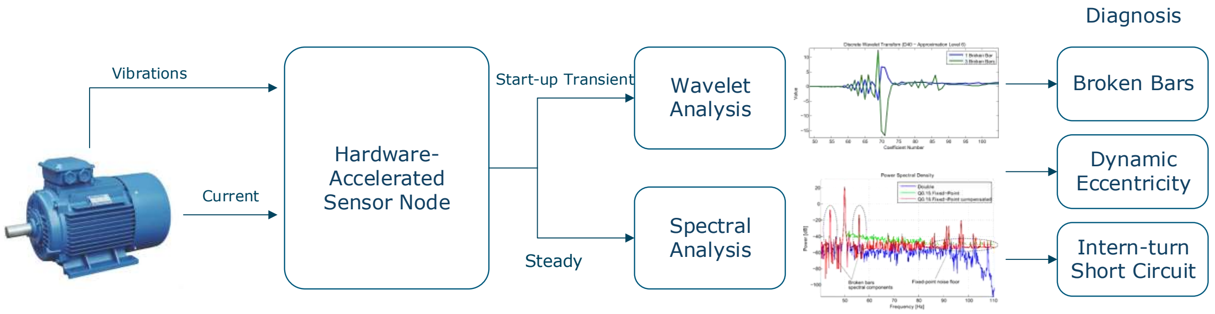


Figure 8.6.: Concept for motor Condition Monitoring

Cells	Blocks memory	Max. frequency	Size of context
20,718 (84.3 %)	26 (81.2 %)	14.9 MHz	260 bits

Table 8.2.: Resource consumption of the FPGA overlay for the motor condition monitoring

8.3.1 Implementation on HaLOEWEn

HaLOEWEn has been used as target platform to implement the different signal processing algorithms. The memory extension is required to handle FFT with large number of points and signal acquisition from three different sources. Indeed, each of the three phase currents are monitored using LEM LT 1005-S/SP19 Hall sensors. If only one phase is sufficient for diagnosing broken bars and dynamic eccentricity, the detection of an inter-turn short circuit requires the computation of the negative current sequence, which is a composition of the three phases.

As the architecture for the FPGA overlay is very close to the one from Figure 5.12a and 7.6, it has not been reproduced here. The notable differences include a wordlength of 32 bits for the bus, the operators and the internal memories. The number of internal memories has been reduced to two. The resulting resource consumption has been reported in Table 8.2. The extension to 32 bits is particularly costly in terms of resources, since the whole design is consuming 35 % more resources compared to the 16-bits version. The FPGA core is running with a 13.25 MHz clock generated internally with the ring oscillator.

The task configurations as depicted by Figure 8.7 have been generated for the architecture using the (GECO)^2 tool. The three phase currents are simultaneously sampled with a frequency of 1 kHz. An FFT size of 2048 points has been selected to guarantee a sufficient accuracy in the spectrum while meeting the memory restrictions of the platform. A Hanning window is used to prior to the FFT in order to reduce the leakage effect. The wavelet transform uses the Daubechies-40 mother wavelet on six levels of decomposition. The middleware has been programmed in such a way that the Wavelet task is automatically started during the startup of the motor. During the steady state, a data sampling task followed by the inter-turn fault diagnosis and the zoom-FFT (dynamic eccentricity and broken bars diagnosis) is started every three minutes.

The mapping of the different subtasks on the arithmetic operators is depicted in Figure 8.7. The size of each block is not proportional to the duration of the task in this diagram. A breakdown of the time spent for each sub-task has been included in the pie charts of Figure 8.8. The reconfiguration time combines the time required for loading the configuration data from the configuration memory and the internal reconfiguration time between successive instructions. In general, the time spent for reconfiguration is very low compared to the overall task duration, so that the reconfiguration overhead stays very low.

Figure 3.4 shows the average power consumed during the execution of each task. The FPGA switches automatically in Flash*Freeze mode between the acquisition of two samples. Once the tasks are completed, the FPGA can be shutdown until the next spectral analysis is started (every three minutes). As a matter of

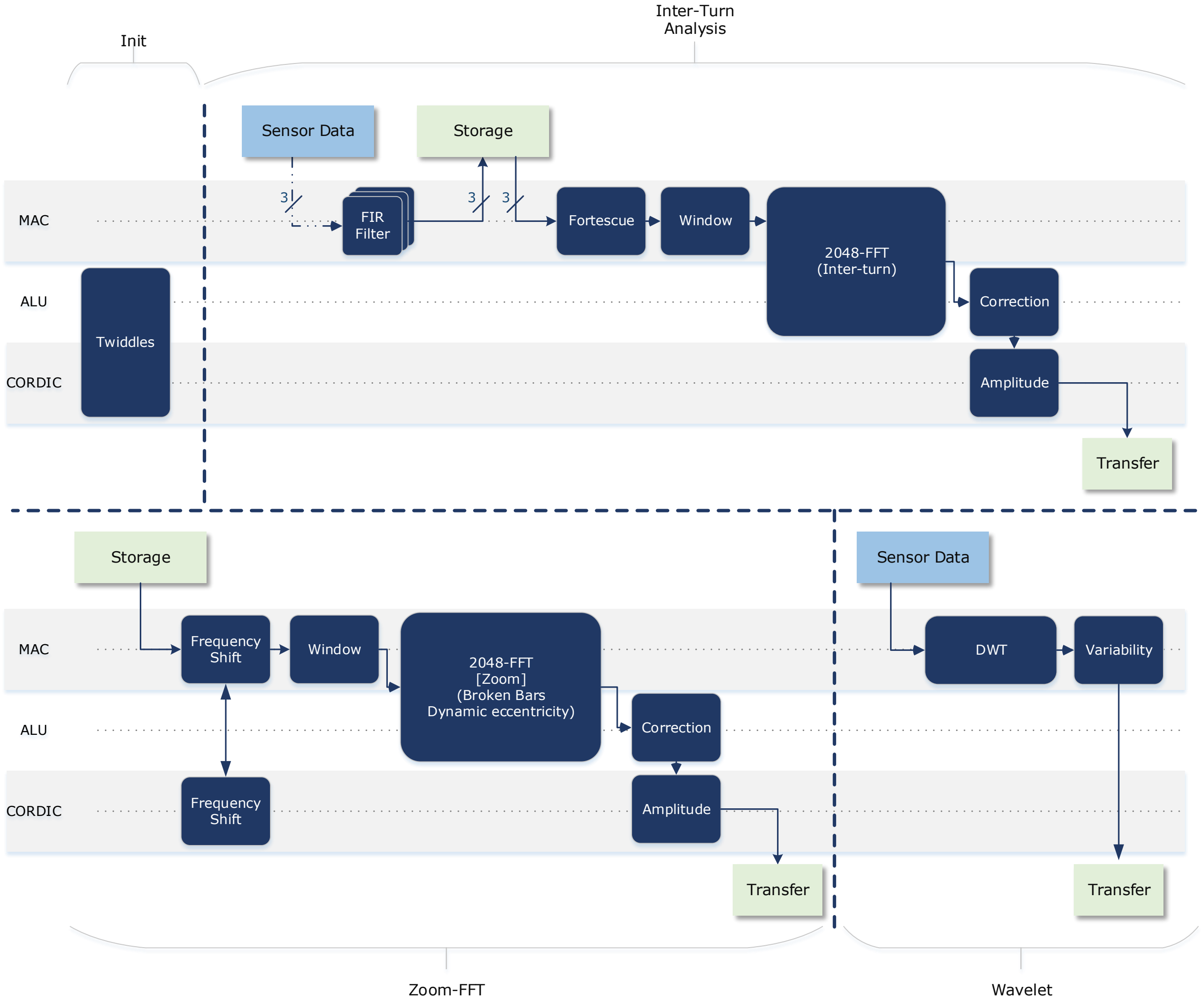
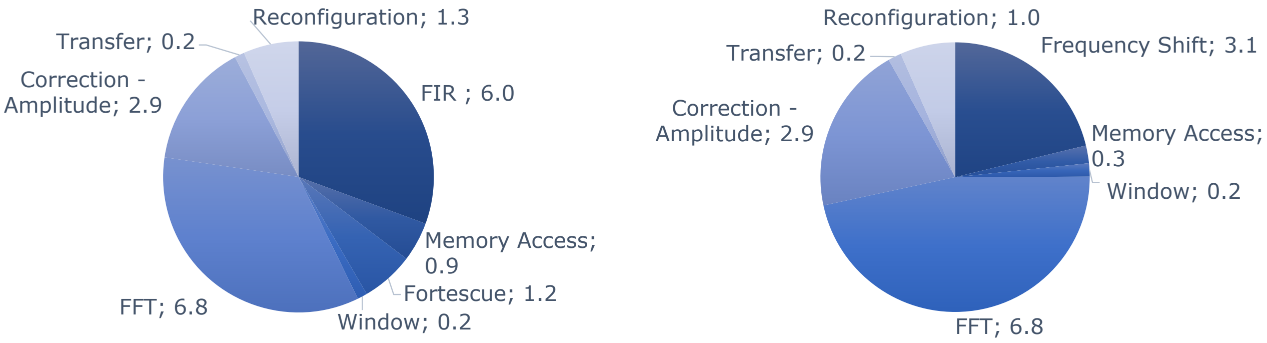


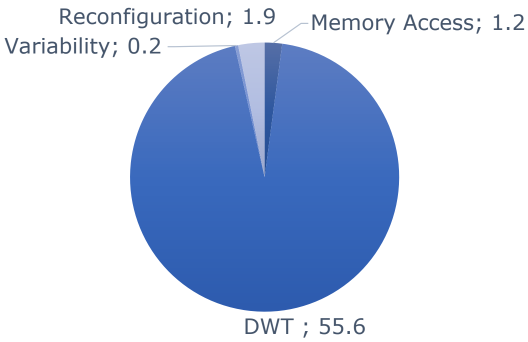
Figure 8.7.: Time-space partitioning of the motor condition monitoring tasks on the overlay architecture

Task	Duration (ms)	Power consumption (mW)	# Instructions	Bitstream size (bits)
Twiddles	2.9	2.3	3	330
Inter-Turn	19.7	8.3	43	4,272
Zoom-FFT	14.5	7.9	31	2,928
Wavelet	58.9	6.8	29	2,352

Table 8.3.: Performance metrics of the motor condition monitoring tasks



(a) Inter-Turn analysis **(b)** Zoom-FFT



(c) Wavelet

Figure 8.8.: Breakdown of time spent for the motor condition monitoring tasks (time is given in milliseconds)

Dynamic eccentricity	Healthy	28 %	42 %
Amplitude at 26.81 Hz (mA)	29.7	32.1	89.7
Amplitude at 73.17 Hz (mA)	32.3	47.1	60.7

Table 8.4.: Estimation results for a motor suffering from dynamic eccentricity

comparison, the static power consumption of a SRAM-based Xilinx Spartan 6 XC6SLX16 FPGA is 14 mW. During the acquisition time, the FPGA must stay active as reconfiguration times are much larger than the sampling period. Thus, the only acquisition of 8,192 samples cost already 114 mWs, just because of the static power consumption. On the other hand, the average power consumption of the Flash-FPGA taking sampling and processing into account (inter-turn plus FFT) was estimated to 498 μW, resulting in an energy consumption of 4.1 mWs, which is already a reduction of almost two orders of magnitude.

8.3.2 Diagnosis results

Once the spectral analysis is completed, a selected part of the magnitude spectrum is transfered to the MCU for the end of the monitoring process. The region of interest is chosen in accordance with the value of the motor slip, which can be monitored externally using a tachometer. Peak frequency and amplitude are then precisely estimated using the approach described in [Mar+13]. Various tests have been performed with different degrees of failure for each of the fault. The average value of the faulty spectral components that were measured using the FPGA implementation described previously are reported in the tables 8.4, 8.5 and 8.6. For each of the considered scenarios, the amplitude of the side band components is increasing with the degree of the fault. This result demonstrates that the processing results delivered by the FPGA can be reliably used for diagnosis purposes.

# Broken bars	Healthy	1	2	3
Amplitude at 44.71 Hz (mA)	58	369.8	634.9	1,940
Amplitude at 55.36 Hz (mA)	43.5	106.3	262.8	688

Table 8.5.: Estimation results for a motor with broken bars

Inter-turn short circuit	Healthy	1.5Ω	0.75Ω
Amplitude at -50 Hz (A)	2.26	2.64	3.22
Amplitude at 150 Hz (A)	0.1	0.18	0.58

Table 8.6.: Estimation results for a motor suffering from inter-turn short circuit

8.4 Conclusion

Different MCSA techniques have been successfully implemented on the HaLOEWEn platform. All tests showed that the implementation of the spectral or wavelet analysis on the FPGA overlay architecture was sufficiently accurate to diagnose a faulty operation of the motor. This demonstrates that the architecture and programming tools are suitable to implement advanced signal processing techniques. All tasks can be executed in an autonomous way by the hardware accelerator without the support of the MCU to compute intermediate results. During the acquisition and computing time, the MCU can stay in sleep mode, which further reduces the average power consumption of the node.

In the literature, several research works addressed the utilization of FPGAs for the online diagnosis of induction motors. The used techniques include the analysis of stator current signatures with a discrete wavelet transform (DWT) [OM+08], vibration analysis [Med+10] or mixed approaches suitable for both vibration and current signature analysis using DWT and STFT according to target device [CY+13]. In all cases, it has been showed that FPGAs demonstrate a significant improvement compared to offline diagnosis methods where the analysis of the data by an expert is usually required. However, all these works were relying on SRAM-based FPGA platforms requiring a supply from a power line. Here, a similar level of accuracy and performance is achieved with an architecture which is dynamically reconfigurable, *i.e.* new or adapted techniques can be easily loaded after deployments. The average power consumption is also sufficiently low for battery-powered operation, which enables an easy and straightforward installation in an industrial environment.

Condition monitoring of electrical machines with wireless sensor networks has also been the subject of the work carried in [HB12]. The authors are measuring the stator current and the vibrations of a motor to identify potential faults with a neural network classifier. The feature extraction tasks are based on a 512 points FFT for the vibrations and variance coupled to peak-to-peak amplitude for the current. A Jennic JN5139 32-bit microcontroller is used for the implementation of the algorithms. The authors measured that the total running time of their feature extraction algorithm is larger than one second with an average power consumption of 20 mW. This results in an energy consumption value which is almost equivalent to the energy necessary for a wireless transfer of the raw data. The proposed solution based on the low-power FPGA was shown to achieve a much higher level of energy-efficiency.

Thanks to the genericity of the FPGA overlay architecture, the current design can be extended to support the diagnosis of further faults, notably using vibration analysis, such as unbalance or looseness [Med+10]. Such a device would result in an universal motor diagnosis device, which could attract much attention in industrial applications.

9 Conclusion

A complete framework for accelerating computationally intensive tasks with programmable hardware on wireless sensor nodes has been introduced in this thesis. By combining an FPGA based on non-volatile memory with a lightweight overlay architecture and its associated programming tools, a new level of energy-efficiency has been reached for low power high-bandwidth sensing applications.

9.1 Contributions of the work

This work addressed issues covering multiple domains, from digital electronics design to fault diagnosis of electromechanical systems via cryptography and signal processing. The main contributions emerging from this multi-disciplinary work can be summarized as follows:

- *Exploitation of low-power technologies and design techniques to reduce the power consumption of an FPGA-based wireless sensor node*: the low static consumption of an FPGA based on non-volatile memory has been fully exploited to reduce the large overhead that the SRAM-based counterpart would induce. Mechanisms for fast and autonomous wake-up control based on an internal ring oscillator maintain the duty cycle of the device very low, even with high sampling rates. The main processor can go in deeper sleep mode by reallocating all sensor data acquisition and processing tasks to the reconfigurable hardware device. Fast wake-up times make the FPGA also suitable for on-demand acceleration. In general, the superiority of reconfigurable hardware based on Flash technology against other types of COTS processing elements for intensive distributed computation in wireless sensor networks has been demonstrated.
- *Introduction of an architecture for virtual runtime reconfiguration at low cost on resource-constrained FPGAs*: Although Flash-based FPGAs achieve a lower power consumption, they are limited by a low amount of available resources preventing the implementation of complex data processing designs. As a fast dynamic reconfiguration of the FPGA core at runtime is inconceivable, a template for an overlay architecture based on coarse-grained reconfigurable operators has been introduced. Data flows between operators is regulated by a lightweight reconfiguration mechanism, which compromises the size of the configuration data with the amount of extra resources and the reconfiguration delay. Multiple tasks can thus be mapped on the FPGA by dynamically loading different bitstreams in the overlay configuration memory in a minimal amount of time. The overall overhead of the virtual reconfiguration layer has a low impact on the energy efficiency of the design. The concept is not only limited to Flash FPGAs and it has been already successfully ported to different platforms and target technologies.
- *Introduction of tools and methods for the runtime deployment of hardware accelerators*: Development and deployment of hardware accelerated tasks on the sensor node hardware has been facilitated by a set of graphical tools and software drivers. Customizing the template overlay architecture. Custom applications can be created by reusing preprogrammed tasks as building blocks of new data processing functions. Generated configuration data can be remotely loaded and scheduled on the sensor node using a custom middleware infrastructure, making the reconfigurable hardware an intrinsic part of the wireless sensor node operating system. This set of tools make the utilization of the hardware accelerator accessible to developers not familiar with the reconfiguration mechanisms of the FPGA overlay or with reliable transfer of data blocks within wireless sensor networks.

- *Improvement of condition monitoring systems with hardware accelerated wireless sensor nodes*: The enhanced computational capability and energy-efficiency of the hardware-accelerated sensor node has been successfully exploited to develop innovative systems for condition monitoring applications. Signal processing algorithms to track damages in a vehicle shock absorber could be ported to a wireless sensor node, efficiently compensating the sensor node low throughput with enhanced processing power. Following the same principle, algorithms for diagnosis of induction motors using motor current signature analysis have been implemented. Multiple techniques for detecting faults in different modes of operation were programmed in such a way that the node select the most suitable signal processing tasks for the current context.

9.2 Outlook

While new solutions have been brought to enhance wireless sensor networks applications and low-power reconfigurable hardware, new problems and new investigation opportunities have arisen from this work. The following items lists suggestions for future work and newly open research directions:

- At the technological level, the lack of embedded multipliers on Igloo FPGAs was causing a significant overhead for the performance and the resource consumption of the designs. DSP blocks were included in the Igloo2, the newest generation of Microsemi's Flash FPGAs. Unfortunately, the static power consumption of these devices increased to more than one of order of magnitude compared to similar counterparts from the previous generation. A large number of IP cores for interfaces have been integrated into the device architecture, which potentially improves its performance but also induces a non-negligible overhead in sleep mode. This limitation makes the chip less efficient in applications with very low duty cycles such as the ones considered in the frame of this thesis. Therefore, there is still a potential to design devices supporting ultra-low static power consumption with an intermediate amount of features which could fit with WSN applications. Eventually, combining the programmable logic with a microcontroller, programmable analog and RF circuitry on the same chip would implement the mote-on-a-chip vision of the PicoRadio [Rab+00]. The PSoC family of devices is already getting towards this direction but still lacks flexibility for the custom digital hardware functionalities and support for RF communication. Recent RF SoCs such as the Atmel SAM R21 family are integrating new types of very low-power processor cores such as the 32-bit ARM Cortex M0. The delay of software tasks related to networking or to the background operations of the operating system could be significantly reduced with this type of core, so that the chip could switch earlier in low-power mode. In addition, delays to recover from deep-sleep mode are getting lower with the new generation of SoCs, so that high sampling rates can be better supported. This new generation of chips should be considered with more attention in the future work.

- At a higher level, the FPGA overlay can be extended with new types of operators fitting to other application domains, such as image processing, or operators with support for floating-point arithmetic if the available resources are sufficient. The configuration controller requires extensions to gain support for data-dependent flow control, notably by enabling instruction branching. Tasks such as data compression could then be mapped as well on the overlay. By further extending the capability of the reconfigurable architecture and keeping the genericity level of the template high, the development tools can be configured for automated design space exploration.

- Acceleration of signal processing tasks is not only limited to condition monitoring applications. The sensor node has been already successfully used for the implementation of online inertial sensor data classification[1]. This type of tasks is widely used in electronic devices for video games

---

[1] Results concerning this aspect have been already published in the conference paper [PG13a] and the article [PG13c]. The student works [Gre14; Mou13; Man12b; KA12; Rad13] were also addressing this topic

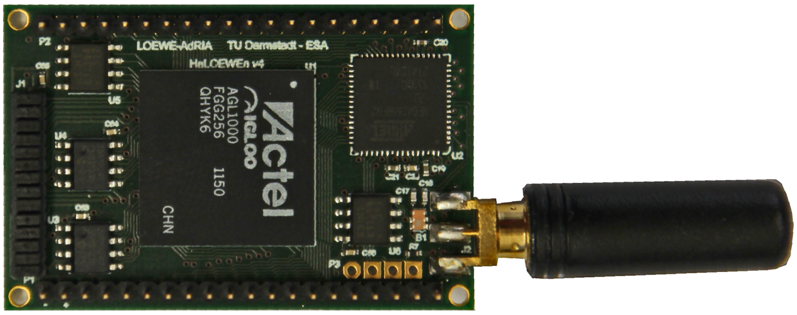


Figure 9.1.: Miniaturized implementation of the HaLOEWEn platform

controllers, smartphones, human-machine interaction, fitness tracking or biomedical application such as actigraphy. Based on the continuous extraction and comparison of large feature sets from accelerometer and gyroscope data, classification algorithms are usually considered as too heavy for online execution. The proposed framework overcomes this limitation with energy-efficient hardware acceleration. In general, biomedical signal processing is another important potential application domain for the designed system. Wireless body area networks used for wearable computing are based on EKG or EEG sensors requiring high sampling rate and complex signal processing, making the low-power FPGA an ideal fit.

- A non-invasive deployment of the FPGA-based wireless sensor nodes in wearable computing applications requires however a drastic size reduction of the platform. The next generation of the HaLOEWEn motes[2] has been miniaturized (Figure 9.1) for a seamless integration in most of real world applications. As a new type of RF SoC with lower power consumption has been used, future work include porting the software and hardware drivers enabling the design programming of the overlay architecture with the (GECO)[2] tool and the middleware services.
- The freedom to execute hardware acceleration on-demand on the sensor node opens new research opportunities for developing adaptivity at the network level. The low reconfiguration overhead makes a fast and frequent modification of the node functionality feasible. Based on environmental or application parameters such as the amount of harvested energy or the topology of the network, energy can be balanced between nodes by selectively executing tasks with lower or higher energy trace, *e.g.* using with lower sampling rates or lower resolution[3].

9.3 Final conclusion

Single processors are not sufficient to efficiently handle the processing of high-bandwidth data streams on wireless sensor nodes, not only because of performance issues, but also because of a poor support for high frequency duty cycling. Relieving the processor from this task by using a Flash-based FPGA is a viable and efficient solution when the suitable infrastructure is available. More flexible and more efficient than application-specific processors, reconfigurable hardware is not only restricted to high-performance computing. With the right technology and design methods, programmable logic has also a bright future in very low-power applications.

---

[2] This mote has been developed in the last phase of the LOEWE project AdRIA by the Embedded Systems and Applications group of TU Darmstadt

[3] Initial results on this aspect were published in the paper [Phi+12b] and the Master thesis [Jun13]

Part IV.

Appendix

A HaLOEWEn Design Files

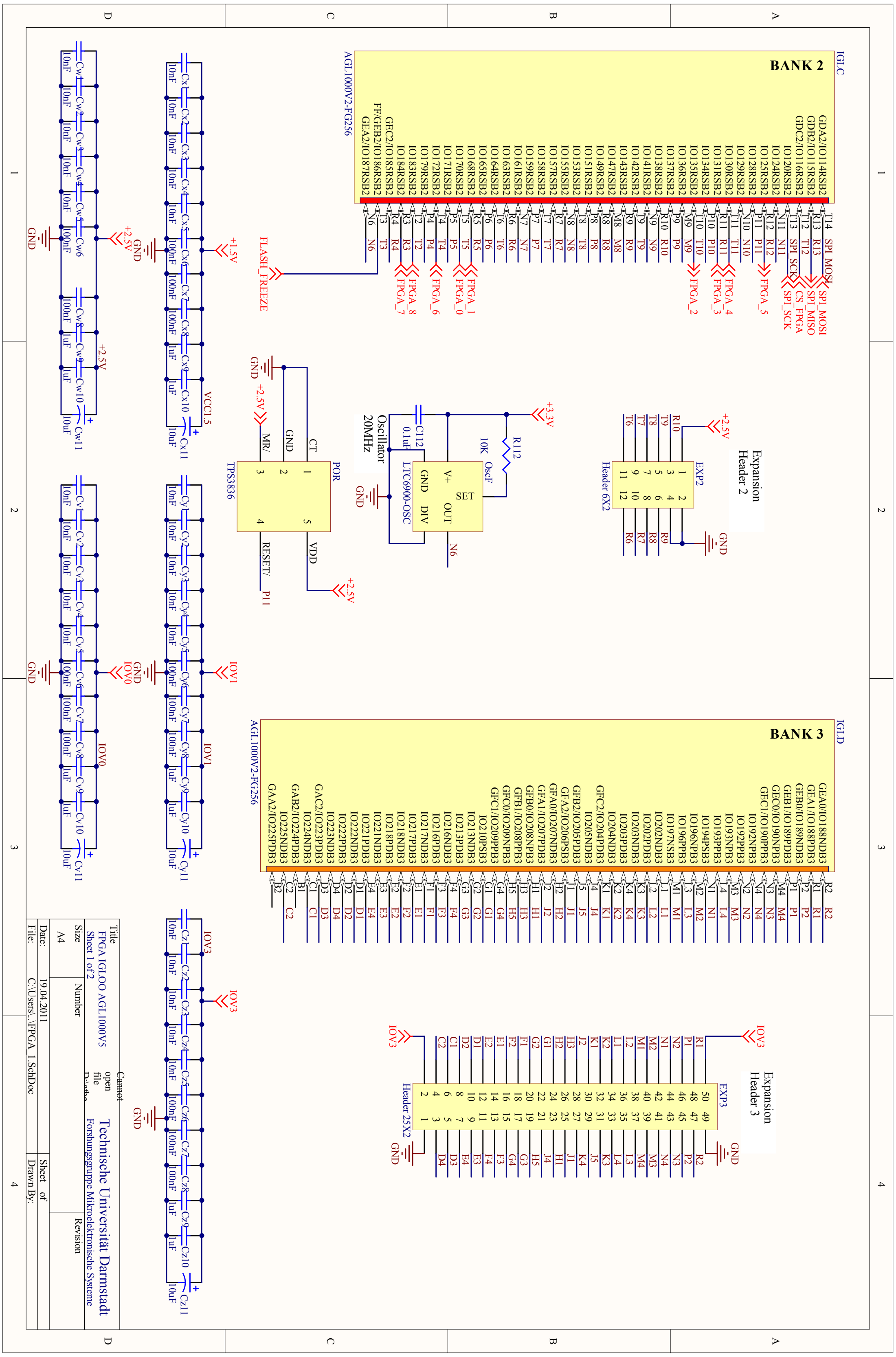


Figure A.1.: HaLOEWEn schematics - FPGA page 1/2

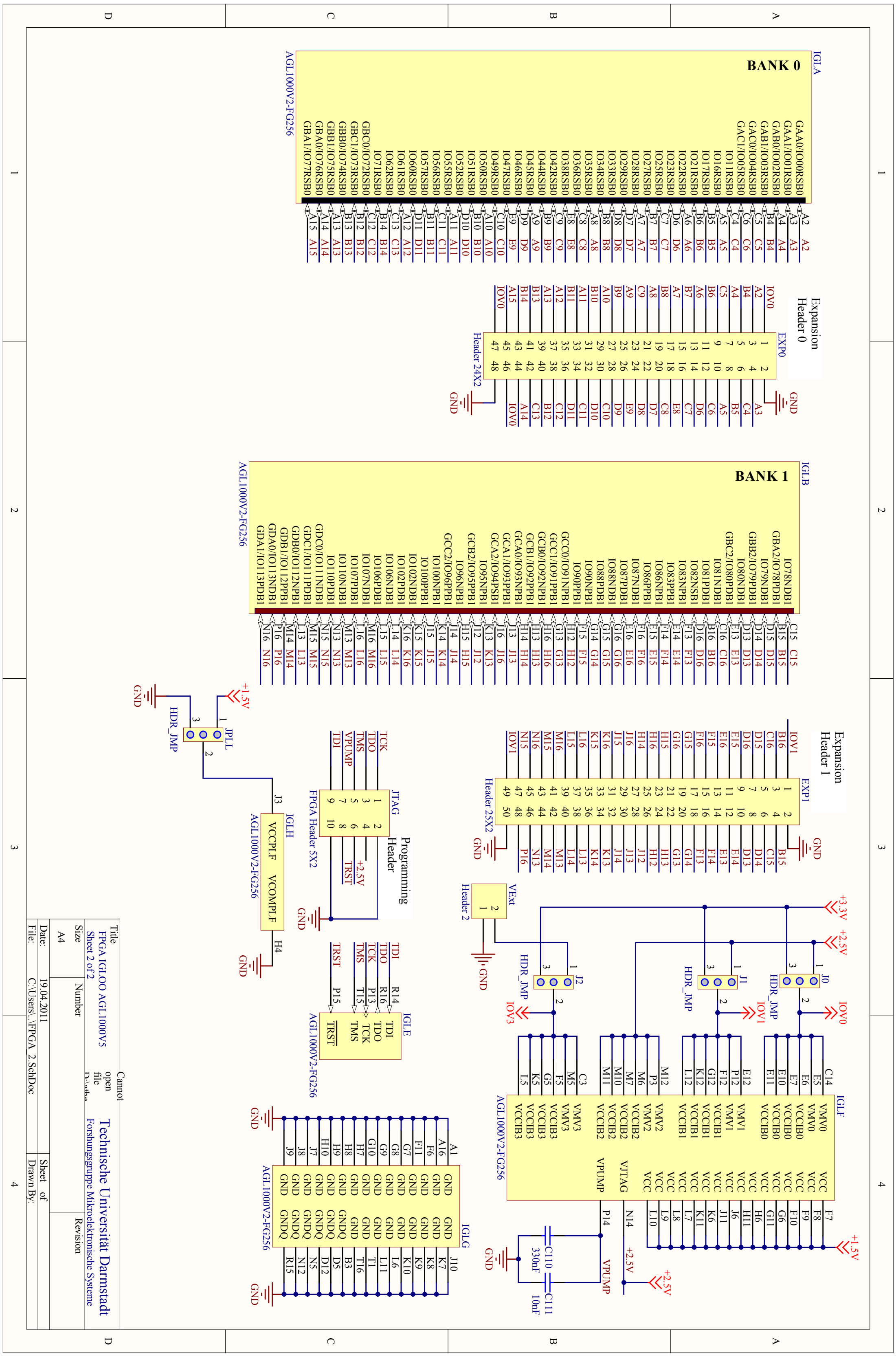


Figure A.2.: HaLOEWEn schematics - FPGA page 2/2

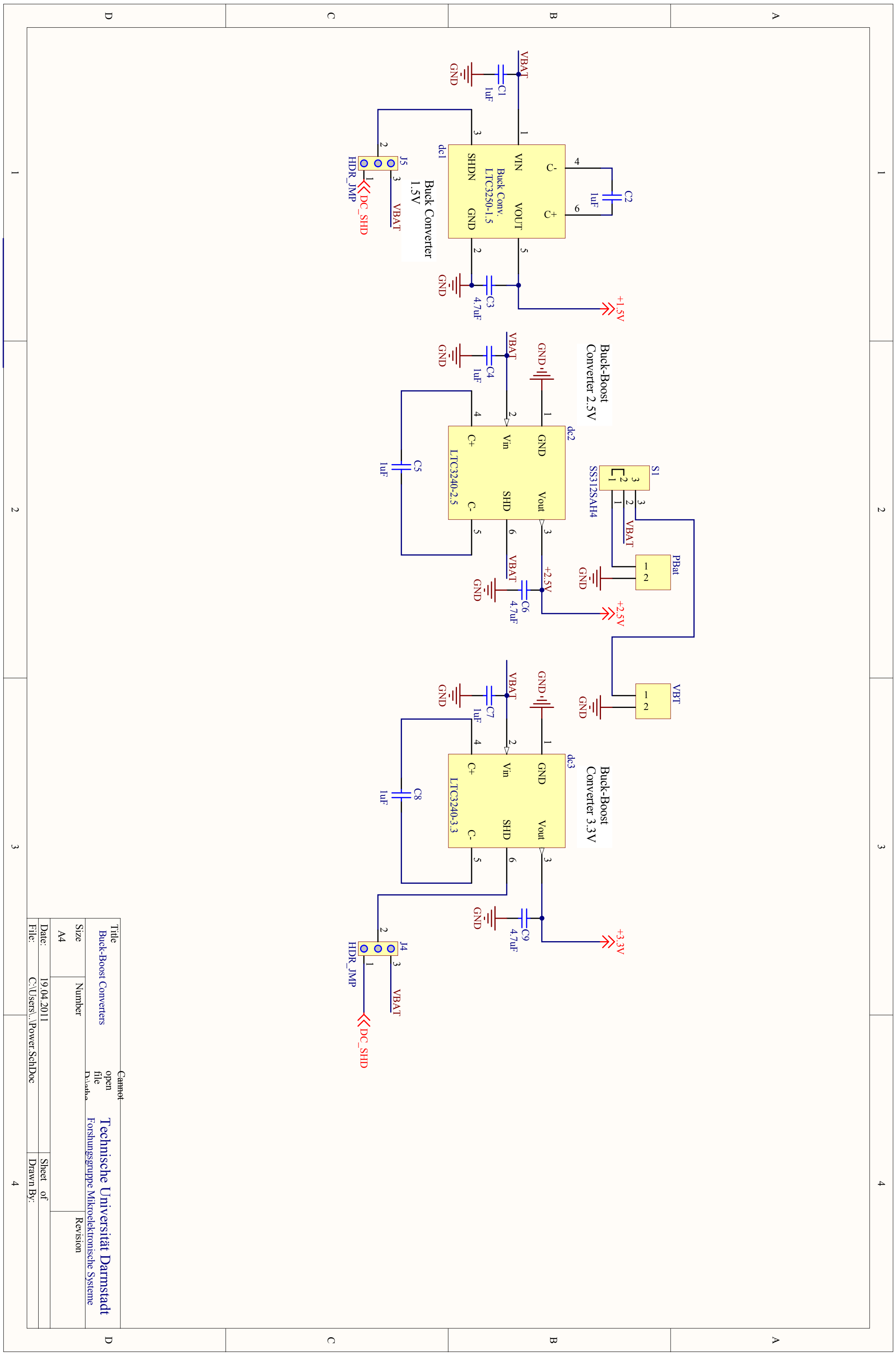


Figure A.3.: HaLOEWEn schematics - Power

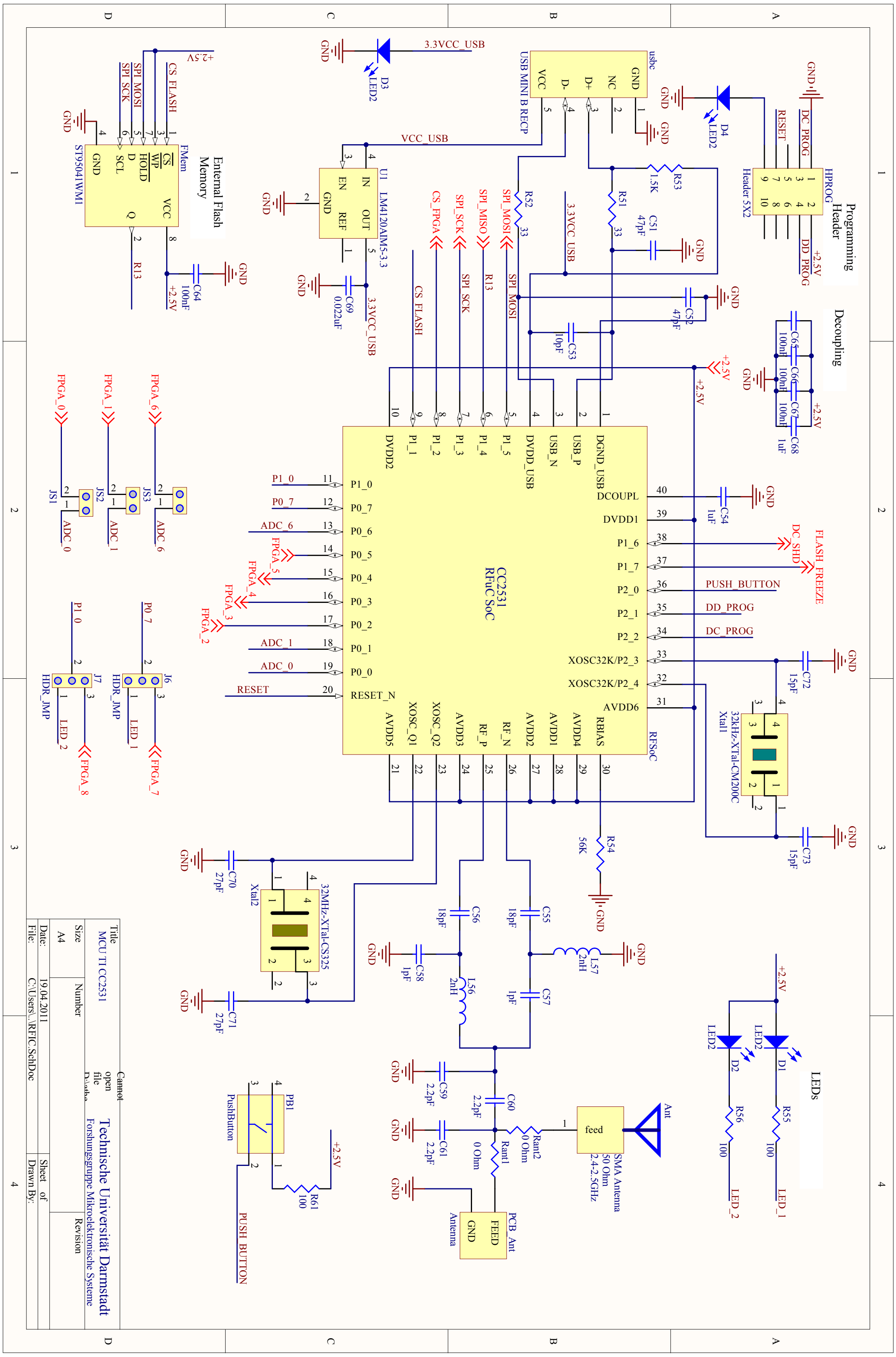


Figure A.4.: HaLOEWEn schematics - RF SoC

B Details of implemented algorithms

B.1 High-Diffusion

The general flow-chart of the High-Diffusion algorithm [MNS06] is depicted in Figure B.1. Only a specific case of the High-Diffusion algorithm with specifications suitable for the utilization in a wireless sensor network is described here. The algorithm takes a 128 bits plaintext data as input, which is encrypted and coded to a 192 bits ciphertext with a 128 bits key. The structure of the algorithm is close to the Rijndael AES [Nat11].

All operations take place in the $GF(256)$ Galois Field with Rijndael generator polynomial $g(x) = x^8 + x^4 + x^3 + x + 1$. A block data is arranged in a $u \times v$ matrix of bytes. The algorithm is based on ten successive rounds composed of the following operations:

- *Key mixing* operation σ: the key mixing operation at a round r is a bitwise XOR operation of the state x_σ^r with the round key k^r resulting in x_γ^{r+1} :

$$x_\gamma^{r+1} = \left(x_\sigma^r \oplus k^r\right) \tag{B.1}$$

- *Substitution* operation γ: this operation is non-linear reversible transformation identical to the Rijndael *S-Box* denoted S_γ. The *extended* S-Box operation applies the substitution on a 192 bits array. The output x_π^r of the substitution operation at round r follows:

$$x_\pi^r = S_\gamma\left(x_\gamma^r\right) \tag{B.2}$$

- *Transpose* operation π: this operation intends to *diffuse* non-linear effect to all elements of the state matrix. The output x_θ^r of the transpose operation at round r follows:

$$x_\theta^r = \left(x_\pi^r\right)^T \tag{B.3}$$

- *High-Diffusion encoding* operation θ: each column of the input matrix x_θ^r is coded with an HD code $[4, 4, 256]$ and the generator matrix G:

$$G = \begin{pmatrix} 1 & 1 & 3 & 2 \\ 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \end{pmatrix} \tag{B.4}$$

  During the ninth round, the High-Diffusion code [4,6,256] with the generator matrix G_{ext} is used:

$$G_{ext} = \begin{pmatrix} 1 & 1 & 3 & 2 & 203 & 127 \\ 2 & 1 & 1 & 3 & 86 & 141 \\ 3 & 2 & 1 & 1 & 64 & 189 \\ 1 & 3 & 2 & 1 & 42 & 101 \end{pmatrix} \tag{B.5}$$

  Both G and G_{ext} are MDS matrices. If $c_i(x)$ denotes the ith column of the matrix x, the *High-Diffusion encoding* operation is such that:

$$c_i(x_\sigma^r) = \begin{cases} \left(c_i(x_\theta^r)^T \cdot G\right)^T & r \in [0, 8], i = 0 \ldots 4 \\ \left(c_i(x_\theta^r)^T \cdot G_{ext}\right)^T & r = 9, i = 0 \ldots 4 \end{cases} \tag{B.6}$$

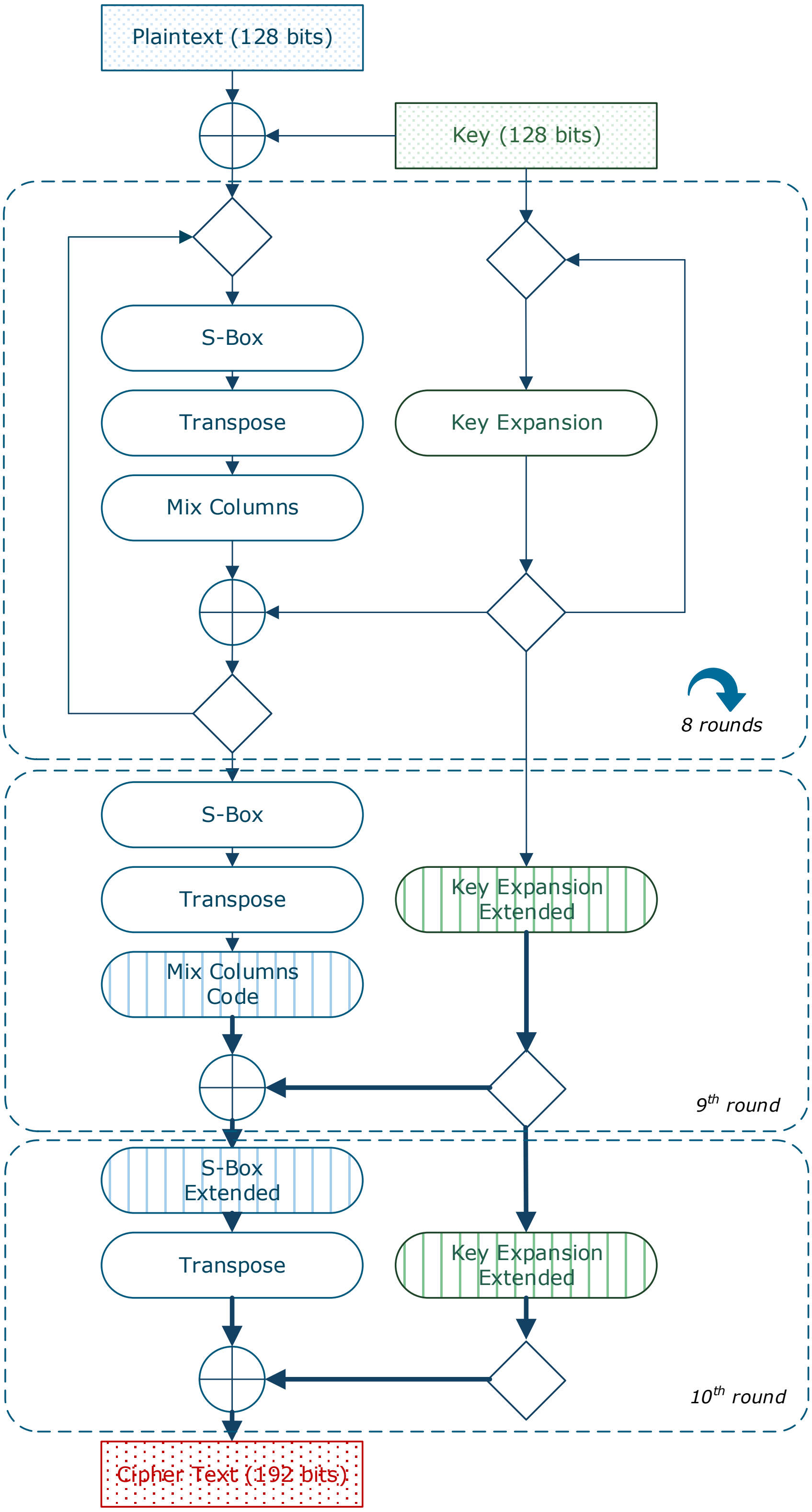


Figure B.1.: Flow chart of the *High Diffusion* algorithm

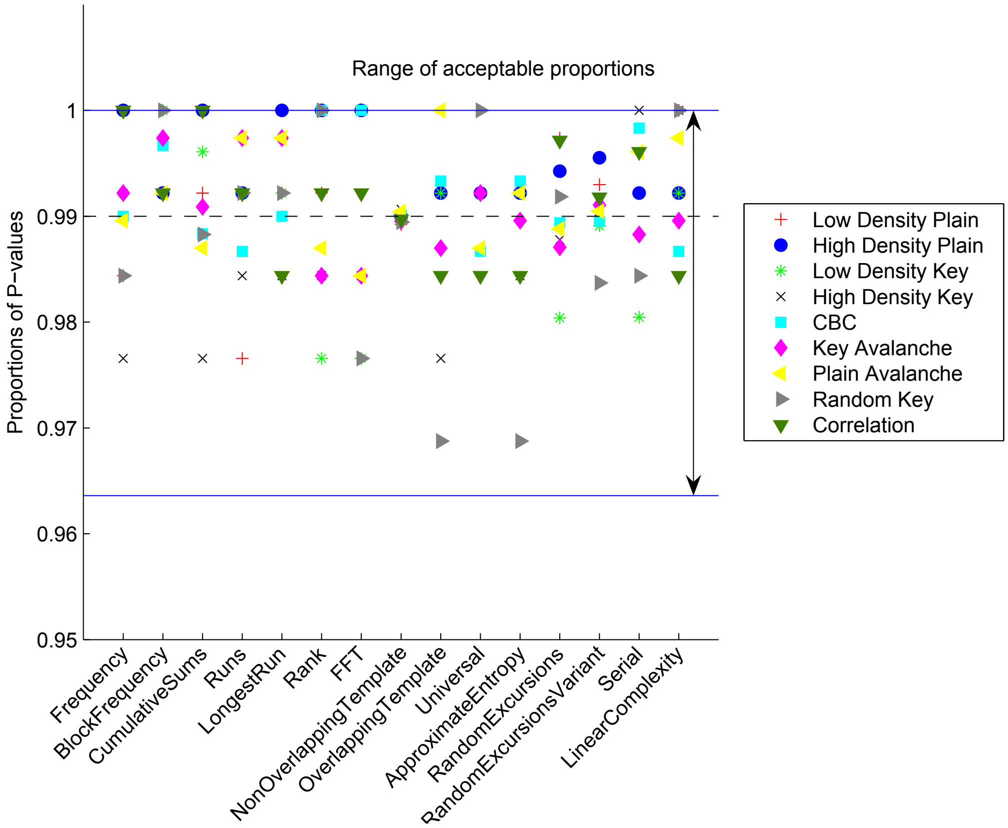


Figure B.2.: Results of NIST pseudo-randomness tests for the High-Diffusion algorithm

- *Key Extension* operation: this operation is identical to the AES-128 key scheduling algorithm [Nat11]. During round 9 and round 10, the extended key scheduling algorithm as used for the AES-192 block cipher is used. If the same key is used for consecutive block encryption, the extended key needs to be computed only once if sufficient memory is available to hold the complete vector.

Deciphering and decoding is performed in the reverse way by using the equivalent inverse function at each step. In particular, this implies the utilization of the inverse S-Box S^{-1} for the *substitution* and the inverse generator matrix G_{inv} for the *High-Diffusion Decoding*:

$$G_{inv} = \begin{pmatrix} 11 & 14 & 9 & 13 \\ 13 & 11 & 14 & 9 \\ 9 & 13 & 11 & 14 \\ 14 & 9 & 13 & 11 \end{pmatrix}. \tag{B.7}$$

Error detection and error correction must be performed by a regular decoder core similar to the one used for Reed-Solomon codes.

A mandatory requirement to validate the security of an encryption algorithm is to demonstrate the pseudo-randomness of the ciphertext. For this purpose, the US National Institute of Standards and Technology (NIST) made a series of benchmark tests available. In each test scenario, a set of randomness metrics are evaluated for a statistically significant amount of input vectors. These metrics must stay in a predefined range in order to demonstrate pseudo-randomness. The results of this test performed on the presented HD code are visible in Figure B.2.

	Hyperbolic	**Linear**	**Circular**
$y \to 0$	$\delta_i = \mathrm{sgn}(x_i y_i)$ $x \to K_h \sqrt{x_0^2 - y_0^2}$ $z \to z_0 + \tanh^{-1}(\frac{y_0}{x_0})$ $\left\lvert \tanh^{-1}(\frac{y_0}{x_0}) \right\rvert \leq 1.1182$	$\delta_i = \mathrm{sgn}(x_i y_i)$ $x \to x_0$ $z \to z_0 + \frac{y_0}{x_0}$ $\left\lvert \frac{y_0}{x_0} \right\rvert \leq 1$	$\delta_i = \mathrm{sgn}(y_i)$ $x \to K_c \sqrt{x_0^2 + y_0^2}$ $z \to z_0 + \mathrm{atan2}(y_0, x_0)$ $\lvert \mathrm{atan2}(y_0, x_0) \rvert \leq 1.7433$
$z \to 0$	$\delta_i = -\mathrm{sgn}(z_i)$ $x \to K_h [x_0 \cosh(z_0) - y_0 \sinh(z_0)]$ $y \to K_h [x_0 \cosh(z_0) + y_0 \sinh(z_0)]$ $\left\lvert \tanh^{-1}(z_0) \right\rvert \leq 1.1182$	$\delta_i = -\mathrm{sgn}(z_i)$ $x \to x_0$ $y \to y_0 + x_0 z_0$ $\lvert z_0 \rvert \leq 1$	$\delta_i = -\mathrm{sgn}(z_i)$ $x \to K_c [x_0 \cos(z_0) - y_0 \sin(z_0)]$ $y \to K_c [x_0 \cos(z_0) + y_0 \sin(z_0)]$ $\lvert z_0 \rvert \leq 1.7433$

Table B.1.: CORDIC modes of operation

B.2 Range-extended CORDIC

The generalized CORDIC algorithm is based on the set of three iterative equations [HHB91]:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & m\delta_i 2^{-i} \\ -\delta_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{B.8}$$

$$z_{i+1} = z_i + \delta_i \theta_i \tag{B.9}$$

where $m \in \{-1, 0, 1\}$ determines the class of functions being evaluated (respectively hyperbolic, linear and circular), $\delta_i \in \{0, 1\}$ is an internal parameter selected such that y (vectoring mode) or z (rotation mode) is converging towards zero and

$$\theta_i = \begin{cases} \tanh^{-1}(2^{-i}) & if\, m = -1 \\ 2^{-i} & if\, m = 0 \\ \tan^{-1}(2^{-i}) & if\, m = 1. \end{cases} \tag{B.10}$$

The six different operation modes of CORDIC result in the functionalities summarized in Table B.1 where

$$K_c = \prod_{i=0}^{n} \sqrt{1 + 2^{-2i}}, \tag{B.11}$$

$$K_h = \prod_{i=0}^{n} \sqrt{1 - 2^{-2i}}. \tag{B.12}$$

According to Hu & al. [HHB91], the convergence range of the CORDIC by adding iterations for negative indexes such that for $i \leq 0$:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & m\delta_i (1 - 2^{i-2}) \\ -\delta_i (1 - 2^{i-2}) & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{B.13}$$

and

$$\theta_i = \begin{cases} \tanh^{-1}(1 - 2^{i-2}) & if\, m = -1 \\ 2^{-i} & if\, m = 0 \\ \tan^{-1}(1 - 2^{i-2}) & if\, m = 1. \end{cases} \tag{B.14}$$

Using six additional iterations [HHB91], the convergence range can be increased such as $\left\lvert \tanh^{-1}(\frac{y_0}{x_0}) \right\rvert$ or $\left\lvert \tanh^{-1}(z_0) \right\rvert \leq 12.4264$ in hyperbolic mode, $\left\lvert \frac{y_0}{x_0} \right\rvert$ or $\lvert z_0 \rvert \leq 32$ in linear mode. Three additional iterations are sufficient to cover the full angle range in circular mode [HHB91]. It must be noted that the value of K_c and K_h must be adapted to the new iterations.

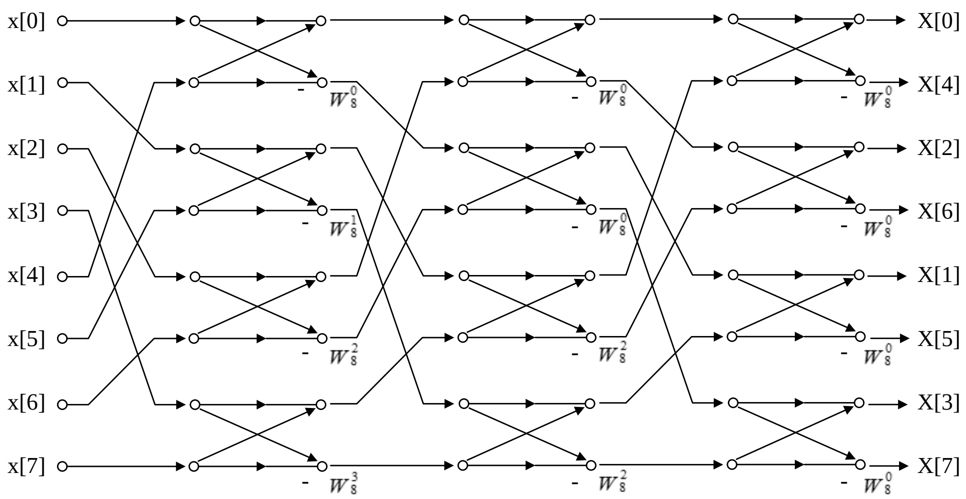


Figure B.3.: 8-FFT decimation in frequency with perfect shuffle address pattern

B.3 Corrected fixed-point Fast Fourier Transform

The discrete Fourier Transform of a complex sequence $a = [a_0, a_1, \ldots, a_{N-1}]$ of length N is the complex sequence $c = [c_0, c_1, \ldots, c_{N-1}]$ defined by

$$c = \mathscr{F}[a] \tag{B.15}$$

$$c_j = \sum_{k=0}^{N-1} a_k W^{jk} \quad j = 0, 1, \ldots, N-1 \tag{B.16}$$

with $W = e^{-2\pi i/N}$.

When $N = 2^M$, one can rewrite the expression of $\mathscr{F}$ by decomposing the indexes j and k. This results in so-called radix-2 FFTs where one can distinguish between Decimation-In-Time (DIT) and Decimation-In-Frequency (DIF) formats. A radix-2 DIF FFT can be written as [Arn10]

$$\mathscr{F}[a]^{(\text{even})} \overset{N/2}{=} \mathscr{F}\left[a^{(\text{left})} + a^{(\text{right})}\right] \tag{B.17}$$

$$\mathscr{F}[a]^{(\text{odd})} \overset{N/2}{=} \mathscr{F}\left[S\left(a^{(\text{left})} - a^{(\text{right})}\right)\right] \tag{B.18}$$

where $\mathscr{F}[a]^{(\text{even})}$ and $\mathscr{F}[a]^{(\text{odd})}$ denote respectively the length $N/2$ subsequences of $\mathscr{F}[a]$ with even and odd indexes, $\mathscr{F}[a]^{(\text{left})}$ and $\mathscr{F}[a]^{(\text{right})}$ denote the first and second half of $\mathscr{F}[a]$ and $S \cdot a$ denotes the sequence with elements $a_k e^{-\pi i k/N}$. The basis operation of equations B.18 is known as the radix-2 DIF *butterfly*. By applying indexes substitution such as in [Sto71], butterfly operations can be arranged in a *perfect shuffle* network as depicted by Figure B.3 for an FFT of size $N = 8$. The FFT is then computed upon M sequences $A_m, m = 1, \ldots, M$ of length N such as

$$A_{m+1}^{\text{even}} \overset{N/2}{=} A_m^{\text{left}} + A_m^{\text{right}} \tag{B.19}$$

$$A_{m+1}^{\text{odd}} \overset{N/2}{=} W_m \left(A_m^{\text{left}} - A_m^{\text{right}}\right) \tag{B.20}$$

The implementation of the butterfly with finite word length implies that the results must be rounded or truncated after each operation. When considering the complex multiplication $z_1 z_2 = (a_1 + ib_1)(a_2 + ib_2)$, the rounded or truncated result denoted by $[.]_T$ is usually such as $[z_1 z_2]_T = [a_1 a_2 - b_1 b_2]_T + i[a_1 b_2 + b_1 a_2]_T$. Only the case of truncation is considered here since rounding operations cost a significant amount of logic that is not affordable within the function units considered in the frame of this thesis. The data is assumed to be in the fixed-point format $Q0.(b-1)$ where b is the word length. If the LSBs of the input data are uniformly distributed, the variance of the error caused by the truncation is such as $\sigma_T = 2^{-2b}/12$. Besides, a common method to avoid overflow in additions is to shift the array A_m by one bit before A_{m+1} is computed. This operation causes an additional loss of accuracy. In the end, the sequence $\hat{c}[k], k = 0, \ldots, N-1$ computed by the fixed-point FFT has an non-zero mean error $\epsilon[k] = \hat{c}[k] - c[k], k = 0, \ldots, N-1$ such as [TTL76]:

$$\epsilon[k] = \frac{-2^{-(b-1)}}{2} \left\{ (1+i)(N(M-t(k))) + \sum_{m=1}^{M-2} k_m \left[N\delta(m, t(k)) - (1+(-1)^{k_{m+1}})2^m \right] \right\} \tag{B.21}$$

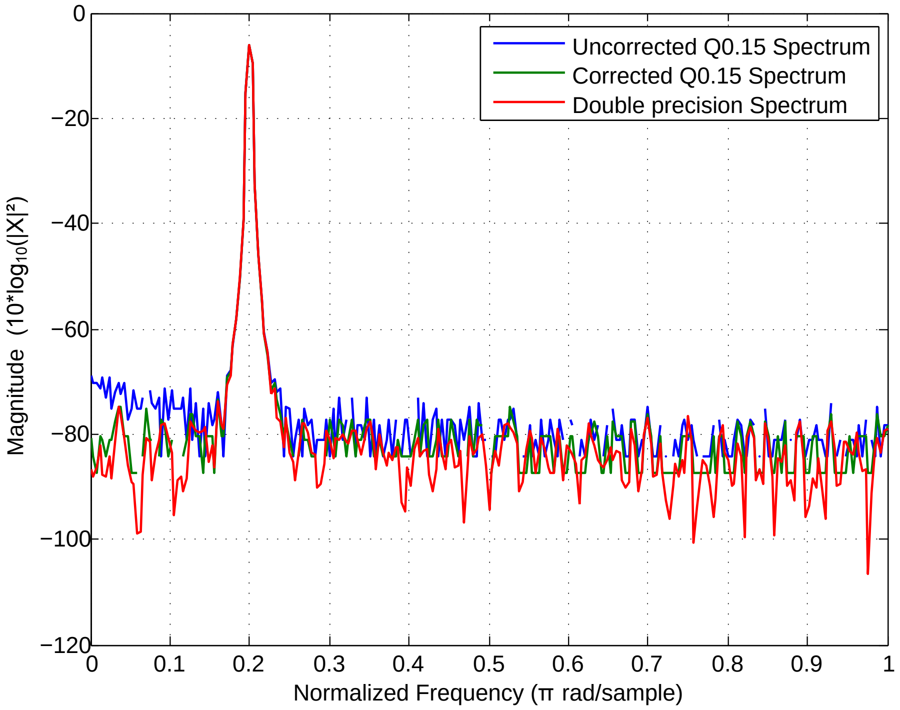
with

$$k = \sum_{j=1}^{M} k_j 2^{j-1},$$

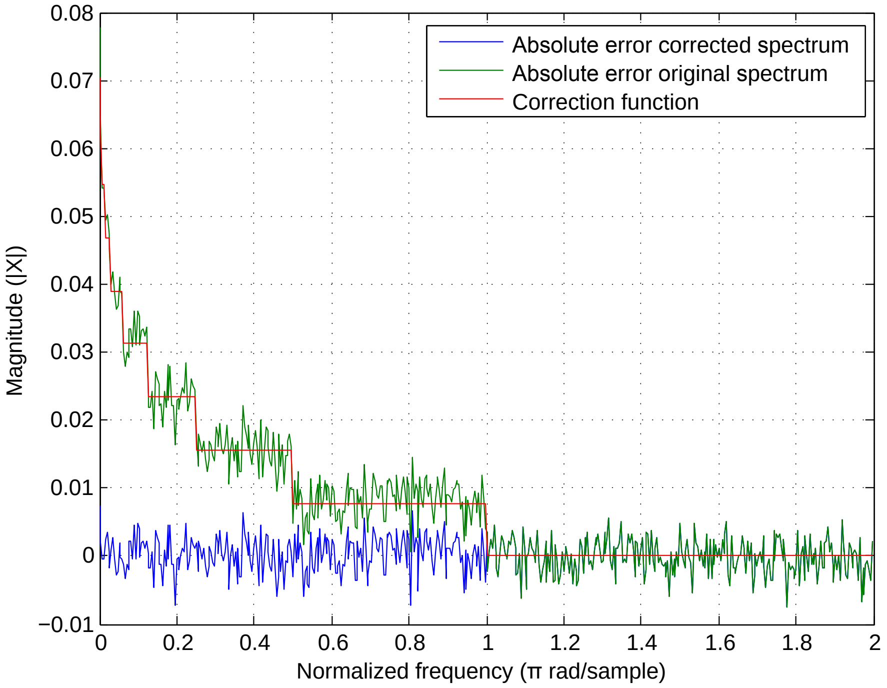
$$t(k) = \begin{cases} 0 & k = 0 \\ \max\{i : k_i = 1\} & \text{otherwise}, \end{cases}$$

$$\delta(x, y) = \begin{cases} 0 & x \neq y \\ 1 & x = y. \end{cases}$$

Figure B.4 shows the power spectrum of a 512-FFT from a sinus signal at the normalized frequency 0.2π radians per sample processed by the radix-2 fixed-point DIF FFT. The non-zero mean error introduced by the implementation is compensated.



(a) Magnitude spectrum of the signal



(b) Magnitude spectrum of the absolute error

Figure B.4.: Error correction for fixed-point FFT

References

[Adi] *ADIS16000/ADIS16229 Digital MEMS Vibration Sensor with Embedded RF Transceiver*. Rev. 0. Analog Devices. Norwood, MA, USA, 2013. URL: `http://www.analog.com/static/imported-files/data_sheets/ADIS16000_16229.pdf`.

[Adr] *LOEWE-Zentrum AdRIA - Home page*. URL: `http://www.loewe-adria.de/`.

[AGP09] R. S. Allgayer, M. Götz, and C. E. Pereira. "FemtoNode: Reconfigurable and Customizable Architecture for Wireless Sensor Networks". In: *Anal., Architectures and Modelling of Embedded Syst., Proc. 3rd IFIP TC 10 Int. Embedded Syst. Symp. (IESS)*. Ed. by A. Rettberg et al. Vol. 310. Langenargen, Germany: Springer, Sept. 2009, pp. 302–309. DOI: `10.1007/978-3-642-04284-3_28`.

[Ahm10] J. Ahmad. "A fractional open circuit voltage based maximum power point tracker for photovoltaic arrays". In: *Proc. 2nd Int. Conf. on Software Technology and Eng. (ICSTE)*. Vol. 1. San Juan, PR, USA: IEEE, Oct. 2010, pp. V1–247–V1–250. DOI: `10.1109/ICSTE.2010.5608868`.

[Aho+07] T. Ahola et al. "Wearable FPGA Based Wireless Sensor Platform". In: *Proc. 29th Annu. Int. Conf. of the IEEE Eng. in Medicine and Biology Soc. (EMBS)*. Lyon, France: IEEE, Aug. 2007, pp. 2288–2291. DOI: `10.1109/IEMBS.2007.4352782`.

[Aky+02] I. F. Akyildiz et al. "A Survey on Sensor Networks". In: *IEEE Commun. Mag.* 40.8 (2002), pp. 102–114. ISSN: 0163-6804. DOI: `10.1109/MCOM.2002.1024422`.

[Alt] *Embedded Design Handbook*. ver 2.9. Altera. San Jose, CA, USA, July 2011. URL: `http://www.altera.com/literature/hb/nios2/edh_ed_handbook.pdf`.

[AM12] H. Ahmed and M. Masmoudi. "Compute intensive in advanced wireless sensor node: Potential solution". In: *Proc. Int. Conf. Commun., Comput. and Applicat. (MIC-CCA)*. Istanbul, Turkey: IEEE, Oct. 2012, pp. 77–83. URL: `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6516787`.

[ANT13] ANT™. *ANT Message Protocol and Usage*. D00000652 Rev 5.0. Jan. 2013. URL: `http://www.thisisant.com/resources/ant-message-protocol-and-usage`.

[Ard] *Arduino - Home page*. URL: `www.arduino.cc`.

[Ard+10] W. Arden et al. *More-than-Moore*. White Paper. International Technology Roadmap for Semiconductors, 2010. URL: `http://www.itrs.net/Links/2010ITRS/IRC-ITRS-MtM-v2%203.pdf`.

[Arn10] J. Arndt. "Matters Computational; Ideas, Algorithms, Source Code". Chap. The Fourier Transfom, pp. 410-439. [online]. 2010. URL: `http://www.jjj.de/fxt/fxtbook.pdf`.

[Atm] *ATMEGA128RFA1, 8-bit Microcontroller with Low Power 2.4GHz Transceiver for ZigBee and IEEE 802.15.4*. Revision D. Preliminary. Atmel. San Jose, CA, USA, July 2012. URL: `http://www.atmel.com/Images/doc8266.pdf`.

[Avr] *Atmel AVR2016: RZRAVEN Hardware User's Guide [Application Note]*. 8117E-AVR-07/12. Atmel. San Jose, CA, USA, 2012. URL: `http://www.atmel.com/Images/doc8117.pdf`.

[Bac+10] A. Bachir et al. "MAC Essentials for Wireless Sensor Networks". In: *IEEE Commun. Surveys and Tutorials* 12.2 (2010), pp. 222–248. ISSN: 1553-877X. DOI: `10.1109/SURV.2010.020510.00058`.

[Bau+03] V. Baumgarte et al. “PACT XPP - A Self-Reconfigurable Data Processing Architecture”. In: *J. Supercomput.* 26.2, Kluwer Academic Publishers (Sept. 2003), pp. 167–184. ISSN: 0920-8542. DOI: `10.1023/A:1024499601571`.

[Bei12] T. Bein. *Maintenance on Demand - MoDe - Advanced Maintenance Concepts for Commercial Vehicles*. presented at the Automotive Future Innovations Conf. Galway, Ireland, July 2012.

[Bel+05] S. J. Bellis et al. “Development of Field Programmable Modular Wireless Sensor Network Nodes for Ambient Systems”. In: *Comput. Commun.* 28.13, Elsevier (Aug. 2005), pp. 1531–1544. ISSN: 0140-3664. DOI: `10.1016/j.comcom.2004.12.045`.

[Bel+08] A. Bellini et al. “High Frequency Resolution Techniques for Rotor Fault Detection of Induction Machines”. In: *IEEE Trans. Ind. Electron.* 55.12 (Dec. 2008), pp. 4200–4209. ISSN: 0278-0046. DOI: `10.1109/TIE.2008.2007004`.

[Bel08] G. Bell. “Bell’s Law for the Birth and Death of Computer Classes”. In: *Commun. ACM* 51.1 (Jan. 2008), pp. 86–94. ISSN: 0001-0782. DOI: `10.1145/1327452.1327453`.

[BGN08] C. Brunelli, F. Garzia, and J. Nurmi. “A coarse-grain reconfigurable architecture for multimedia applications featuring subword computation capabilities”. In: *J. Real-Time Image Processing* 3.1-2, Springer Verlag (2008), pp. 21–32. ISSN: 1861-8200. DOI: `10.1007/s11554-008-0071-3`.

[Bha+05] S. Bhatti et al. “MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms”. In: *Mob. Netw. Appl.* 10.4, Kluwer Academic Publishers (Aug. 2005), pp. 563–579. ISSN: 1383-469X. DOI: `10.1007/s11036-005-1567-8`.

[BL12] A. Brant and G.G.F. Lemieux. “ZUMA: An Open FPGA Overlay Architecture”. In: *Proc. IEEE 20th Ann. Int. Symp. Field-Programmable Custom Computing Mach. (FCCM)*. Toronto, Canada: IEEE, Apr. 2012, pp. 93–96. DOI: `10.1109/FCCM.2012.25`.

[Blu10] Bluetooth®. *Specification of the Bluetooth System*. Version 4.0. June 2010. URL: `https://www.bluetooth.org/en-us/specification/adopted-specifications`.

[BMR02] F. Burghardt, S. Mellers, and J. M. Rabaey. *The PicoRadio Test Bed*. Tech. Rep. Berkeley, CA, USA: Berkeley Wireless Research Center, Dec. 2002. URL: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.4795`.

[Bob07] C. Bobda. “Introduction to Reconfigurable Computing”. In: ed. by C. Bobda. Architectures, Algorithms, and Applications. Springer Netherlands, 2007. Chap. Introduction, pp. 1–14. ISBN: 978-1-4020-6100-4. URL: `http://www.springer.com/engineering/circuits+%26+systems/book/978-1-4020-6088-5`.

[Bou+07] F. Bouwens et al. “Architectural Exploration of the ADRES Coarse-Grained Reconfigurable Array”. In: *Reconfigurable Computing: Architectures, Tools and Applicat., Proc. 3rd Int. Workshop (ARC)*. Ed. by Pedro C. Diniz et al. Vol. 4419. Lecture Notes in Computer Science. Mangaratiba, Brazil: Springer Berlin Heidelberg, Mar. 2007, pp. 1–13. ISBN: 978-3-540-71430-9. DOI: `10.1007/978-3-540-71431-6_1`.

[Bro+11] A. Brokalakis et al. “RESENSE: An Innovative, Reconfigurable, Powerful and Energy Efficient WSN Node”. In: *Proc. IEEE Int. Conf. on Commun. (ICC)*. Kyoto, Japan: IEEE, June 2011, 5 pages. DOI: `10.1109/icc.2011.5963499`.

[BS10] O. Berder and O. Sentieys. “PowWow: Energy-efficient HW/SW techniques for Wireless Sensor Networks”. In: *Proc. Workshop on Ultra-Low Power Sensor Networks (WUPS)*. Hannover, Germany, Feb. 2010. URL: `http://geodes.ict.tuwien.ac.at/PowerSavingHandbook/WUPS/WUPS%202010/05_WUPS2010_Berder.pdf`.

[CA12] Davor Capalija and Tarek S. Abdelrahman. “A Coarse-Grain FPGA Overlay for Executing Data Flow Graphs”. In: *Proc. Workshop on the Intersections of Comput. Architecture and Reconfigurable Logic (CARL)*. Category 2. Portland, OR, USA, June 2012, 5 pages. URL: `http://www.eecg.toronto.edu/~davor/papers/capalija_carl2012.pdf`.

[Cal+05] R. B. Caldas et al. “Low power/high performance self-adapting sensor node architecture”. In: *Proc. 10th IEEE Conf. Emerging Technologies and Factory Automation (ETFA)*. Vol. 2. Catania, Italy: IEEE, Sept. 2005, pp. 973–976. DOI: `10.1109/ETFA.2005.1612777`.

[Car+09] A. Caracas et al. “Mote Runner: A Multi-language Virtual Machine for Small Embedded Devices”. In: *Proc. 3rd Int. Conf. on Sensor Technologies and Applications (SENSORCOMM)*. Athens, Glyfada, Greece: IEEE, June 2009, pp. 117–125. DOI: `10.1109/SENSORCOMM.2009.27`.

[Car+11] A. Caracas et al. “Energy-efficiency through Micro-Managing Communication and Optimizing Sleep”. In: *Proc 8th Annu. IEEE Commun. Soc. Conf. on Sensor, Mesh and Ad Hoc Commun. and Networks (SECON)*. Salt Lake City, UT, USA: IEEE, June 2011, pp. 55–63. DOI: `10.1109/SAHCN.2011.5984943`.

[Cc2a] *CC2431, System-on-Chip for 2.4 GHz ZigBee®/ IEEE 802.15.4 with Location Engine*. Rev. 2.01. Texas Instruments. Dallas, TX, USA, May 2007. URL: `http://www.ti.com/lit/ds/symlink/cc2431.pdf`.

[Cc2b] *CC2530, A True System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee Applications*. Texas Instruments. Dallas, TX, USA, Apr. 2009. URL: `http://www.ti.com/lit/ds/swrs081b/swrs081b.pdf`.

[Cha+08] A. Chattopadhyay et al. “High-level Modelling and Exploration of Coarse-grained Reconfigurable Architectures”. In: *Proc. Design, Automation and Test in Europe Conf. (DATE)*. Munich, Germany: EDAA, Mar. 2008, pp. 1334–1339. DOI: `10.1109/DATE.2008.4484864`.

[CLP06] B. W. Cook, S. Lanzisera, and K. S. J. Pister. “SoC Issues for RF Smart Dust”. In: *Proc. IEEE* 94.6 (June 2006), pp. 1177–1196. ISSN: 0018-9219. DOI: `10.1109/JPROC.2006.873620`.

[Con] *Contiki: The Open Source OS for the Internet of Things - Home page*. URL: `www.contiki-os.org`.

[CSM08] G. Chalivendra, R. Srinivasan, and N.S. Murthy. “FPGA based re-configurable wireless sensor network protocol”. In: *Proc. Int. Conf. Electronic Design (ICED)*. Penang, Malaysia: IEEE, Dec. 2008, 4 pages. DOI: `10.1109/ICED.2008.4786652`.

[CSS11] A. Chefi, A. Soudani, and G. Sicard. “Hardware compression solution based on HWT for low power image transmission in WSN”. In: *Proc. Int. Conf. on Microelectronics (ICM)*. Hammamet, Tunisia: IEEE, Dec. 2011, 5 pages. DOI: `10.1109/ICM.2011.6177387`.

[CTA08] S. Commuri, V. Tadigotla, and M. Atiquzzaman. “Reconfigurable Hardware Based Dynamic Data Aggregation in Wireless Sensor Networks”. In: *Int. J. of Distributed Sensor Networks* 4.2, Hindawi Publishing Corporation (2008), pp. 194 –212. DOI: `10.1080/15501320802001234`.

[CY+13] E. Cabal-Yepez et al. “Reconfigurable Monitoring System for Time-Frequency Analysis on Industrial Equipment Through STFT and DWT”. In: *IEEE Trans. Ind. Informat.* 9.2 (May 2013), pp. 760–771. ISSN: 1551-3203. DOI: `10.1109/TII.2012.2221131`.

[DGV04] A. Dunkels, B. Gronvall, and T. Voigt. “Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors”. In: *Proc. 29th Annu. IEEE Int. Conf. on Local Comput. Networks (LCN)*. Tampa, FL, USA: IEEE, Nov. 2004, pp. 455–462. DOI: `10.1109/LCN.2004.38`.

[Dig] *Digilent Inc. - Digital Design Engineer’s Source - Home page*. URL: `http://www.digilentinc.com/`.

[Dre10] J. Drew. *Energy Harvester Produces Power from Local Environment, Eliminating Batteries in Wireless Sensors*. Design Note 483. Milpitas, CA, USA: Linear Technologies, Oct. 2010. URL: `http://cds.linear.com/docs/en/design-note/DN483.pdf`.

[Dun+06a] A. Dunkels et al. "Protothreads: Simplifying Event-driven Programming of Memory-constrained Embedded Systems". In: *Proc. 4th Int. Conf. on Embedded Networked Sensor Syst. (SenSys)*. Boulder, CO, USA: ACM, Oct. 2006, pp. 29–42. ISBN: 1-59593-343-3. DOI: `10.1145/1182807.1182811`.

[Dun+06b] A. Dunkels et al. "Run-time Dynamic Linking for Reprogramming Wireless Sensor Networks". In: *Proc. 4th Int. Conf. on Embedded Networked Sensor Syst. (SenSys)*. Boulder, CO, USA: ACM, Oct. 2006, pp. 15–28. ISBN: 1-59593-343-3. DOI: `10.1145/1182807.1182810`.

[Dun+07] A. Dunkels et al. "Software-based On-line Energy Estimation for Sensor Nodes". In: *Proc. 4th Workshop on Embedded Networked Sensors (EmNets)*. Cork, Ireland: ACM, June 2007, pp. 28–32. ISBN: 978-1-59593-694-3. DOI: `10.1145/1278972.1278979`.

[Dun11] A. Dunkels. *The ContikiMAC Radio Duty Cycling Protocol*. Tech. Rep. T2011:13. Stockholm, Sweden: SICS, Dec. 2011. URL: `http://dunkels.com/adam/dunkels11contikimac.pdf`.

[EGF96] C. Ebeling, D. C. Green, and P. Franklin. "RaPiD - Reconfigurable Pipelined Datapath". In: *Field-Programmable Logic: Smart Applicat., New Paradigms, and Compilers. 6th Int. Workshop on Field-Programmable Logic and Applicat. (FPL)*. Ed. by R. W. Hartenstein and M. Glesner. Darmstadt, Germany: Springer - Verlag, Sept. 1996, pp. 126–125. URL: `http://dl.acm.org/citation.cfm?id=741212`.

[EK09] A. El Kateeb. "Hardware Reconfiguration Capability for Third-Generation Sensor Nodes". In: *IEEE Computer* 42.5, IEEE Computer Society (May 2009), pp. 95–97. ISSN: 0018-9162. DOI: `10.1109/MC.2009.159`.

[EKD06] D. Efstathiou, K. Kazakos, and A. Dollas. "Parrotfish: Task Distribution in a Low Cost Autonomous ad hoc Sensor Network through Dynamic Runtime Reconfiguration". In: *Proc. 14th Annu. IEEE Symp. on Field-Programmable Custom Computing Mach. (FCCM)*. Napa, CA, USA: IEEE, Apr. 2006, pp. 319–320. DOI: `10.1109/FCCM.2006.56`.

[ELK11] A. Engel, B. Liebig, and A. Koch. "Feasibility Analysis of Reconfigurable Computing in Low-power Wireless Sensor Applications". In: *Proc. 7th Int. Conf. on Reconfigurable Computing: Architectures, Tools and Applicat. (ARC)*. Belfast, UK: Springer-Verlag, Mar. 2011, pp. 261–268. ISBN: 978-3-642-19474-0. URL: `http://dl.acm.org/citation.cfm?id=1987535.1987570`.

[ELK12a] A. Engel, B. Liebig, and A. Koch. "Energy-efficient heterogeneous reconfigurable sensor node for distributed structural health monitoring". In: *Proc. Conf. on Design and Architectures for Signal and Image Processing (DASIP)*. Karlsruhe, Germany: IEEE, Oct. 2012, 8 pages. URL: `http://www.esa.informatik.tu-darmstadt.de/twiki/pub/Staff/AndreasKochPublications/2012_DASIP.pdf`.

[ELK12b] A. Engel, B. Liebig, and A. Koch. "HaLOEWEn: A heterogeneous reconfigurable sensor node for distributed structural health monitoring". In: *Proc. Conf. on Design and Architectures for Signal and Image Processing (DASIP)*. Karlsruhe, Germany: IEEE, Oct. 2012, 2 pages. URL: `http://www.esa.informatik.tu-darmstadt.de/twiki/pub/Staff/AndreasKochPublications/2012_DASIP-D.pdf`.

[Els10] D. Elsberkirch. *MoDe Project - Report on the low power sensor interface platform concept*. Deliverable D3.2. Fraunhofer Institute for Integrated Circuits, 2010.

[Els12] D. Elsberkirch. *MoDe Project - Evaluation prototype for dynamic sensor node reconfiguration*. Deliverable D3.6. Fraunhofer Institute for Integrated Circuits, 2012.

[Eri+09] J. Eriksson et al. “COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks”. In: *Proc. 2nd Int. Conf. on Simulation Tools and Techniques (SIMUTools)*. Rome, Italy: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Mar. 2009, 27:1–27:7. ISBN: 978-963-9799-45-5. DOI: `10.4108/ICST.SIMUTOOLS2009.5637`.

[ERR05] A. Eswaran, A. Rowe, and R. Rajkumar. “Nano-RK: An Energy-Aware Resource-Centric RTOS for Sensor Networks”. In: *Proc. 26th IEEE Int. Real-Time Systems Symp. (RTSS)*. Miami, FL, USA: IEEE, Dec. 2005, 10 pages. DOI: `10.1109/RTSS.2005.30`.

[Fav12] B. Favre. “Public Service Review: Maintenance on Demand”. In: *Transport* 29 (2012), pp. 76–77.

[FGV13] A. Fatecha, J. Guevara, and E. Vargas. “Reconfigurable architecture for smart sensor node based on IEEE 1451 standard”. In: *Proc. IEEE Sensors Conf.* Baltimore, MD, USA: IEEE, Nov. 2013, 4 pages. DOI: `10.1109/ICSENS.2013.6688425`.

[FK11] M. O. Farooq and T. Kunz. “Operating Systems for Wireless Sensor Networks: A Survey”. In: *Sensors* 11.6, MDPI - Open Access Publishing (May 2011), pp. 5900–5930. ISSN: 1424-8220. DOI: `10.3390/s110605900`.

[FRL09] C.-L. Fok, G.-C. Roman, and C. Lu. “Agilla: A Mobile Agent Middleware for Self-adaptive Wireless Sensor Networks”. In: *ACM Trans. Auton. Adapt. Syst.* 4.3 (July 2009), 16:1–16:26. ISSN: 1556-4665. DOI: `10.1145/1552297.1552299`.

[Gas+11a] L. Gasparini et al. “FPGA Implementation of a People Counter for an Ultra-low-power Wireless Camera Network node”. In: *Proc. 7th Conf. on Ph.D. Research in Microelectronics and Electronics (PRIME)*. Trento, Italy: IEEE, July 2011, pp. 169–172. DOI: `10.1109/PRIME.2011.5966244`.

[Gas+11b] L. Gasparini et al. “An Ultralow-Power Wireless Camera Node: Development and Performance Analysis”. In: *IEEE Trans. Instrum. Meas.* 60.12 (Dec. 2011), pp. 3824–3832. ISSN: 0018-9456. DOI: `10.1109/TIM.2011.2147630`.

[Gay+03] D. Gay et al. “The nesC Language: A Holistic Approach to Networked Embedded Systems”. In: *SIGPLAN Not.* 38.5 (May 2003), 11 pages. ISSN: 0362-1340. DOI: `10.1145/780822.781133`.

[GGK10] C. U. Grosse, S. D. Glaser, and M. Krüge. “Initial development of wireless acoustic emission sensor Motes for civil infrastructure state monitoring”. In: *Smart Structures and Systems* 6.3, Techno-Press (2010), pp. 197–209. URL: `http://glaser.berkeley.edu/glaserdrupal/pdf/SS&S%20Cambridge.pdf`.

[GGRG09] R. Garcia, A. Gordon-Ross, and A. D. George. “Exploiting Partially Reconfigurable FPGAs for Situation-Based Reconfiguration in Wireless Sensor Networks”. In: *Proc. 17th IEEE Symp. on Field Programmable Custom Computing Mach. (FCCM)*. Napa, CA, USA: IEEE, Apr. 2009, pp. 243–246. DOI: `10.1109/FCCM.2009.45`.

[GK07] A. Giridhar and P. R. Kumar. “Wireless Sensor Networks: Signal Processing and Communications Perspectives”. In: ed. by A. Swami et al. John Wiley & Sons Ltd, 2007. Chap. In-Network Information Processing in Wireless Sensor Networks, pp. 43–68. ISBN: 978-0-470-03557-3. URL: `http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470035579.html`.

[Gol+99] S. C. Goldstein et al. “PipeRench: a coprocessor for streaming multimedia acceleration”. In: *Proc. 26th Int. Symp. on Computer Architecture (ISCA)*. Atlanta, GA, USA: IEEE, 1999, pp. 28–39. DOI: `10.1109/ISCA.1999.765937`.

[GS12] P. R. Grassi and D. Sciuto. “Energy-Aware FPGA-based Architecture for Wireless Sensor Networks”. In: *Proc. 15th Euromicro Conf. on Digital System Design (DSD)*. Izmir, Turkey: IEEE, Sept. 2012, pp. 866–873. DOI: `10.1109/DSD.2012.50`.

[H.+09] Chao H. et al. “A novel FPGA-based wireless vision sensor node”. In: *Proc. IEEE Int. Conf. on Automation and Logistics (ICAL)*. Shenyang, China: IEEE, Aug. 2009, pp. 841–846. DOI: `10.1109/ICAL.2009.5262805`.

[Han+09] S. Hanson et al. “A Low-Voltage Processor for Sensing Applications With Picowatt Standby Mode”. In: *IEEE J. Solid-State Circuits* 44.4 (Apr. 2009), pp. 1145–1155. ISSN: 0018-9200. DOI: `10.1109/JSSC.2009.2014205`.

[Hay+12] A. Hayek et al. “FPGA-based wireless sensor network platform for safety systems”. In: *Proc. 19th Int. Conf. on Telecommun. (ICT)*. Jounieh, Lebanon: IEEE, Apr. 2012, 6 pages. DOI: `10.1109/ICTEL.2012.6221281`.

[HB12] L. Hou and N. W. Bergmann. “Novel Industrial Wireless Sensor Networks for Machine Condition Monitoring and Fault Diagnosis”. In: *IEEE Trans. Instrum. Meas.* 61.10 (Oct. 2012), pp. 2787–2798. ISSN: 0018-9456. DOI: `10.1109/TIM.2012.2200817`.

[HBW11] M. Hempstead, D. Brooks, and G.-Y. Wei. “An Accelerator-Based Wireless Sensor Network Processor in 130 nm CMOS”. In: *IEEE J. Emerging and Selected Topics in Circuits and Syst.* 1.2 (June 2011), pp. 193–202. ISSN: 2156-3357. DOI: `10.1109/JETCAS.2011.2160751`.

[HC02] J. L. Hill and D. E. Culler. “Mica: A Wireless Platform for Deeply Embedded Networks”. In: *IEEE Micro* 22.6 (Nov. 2002), pp. 12–24. ISSN: 0272-1732. DOI: `10.1109/MM.2002.1134340`.

[Her+02] M. Herz et al. “Memory addressing organization for stream-based reconfigurable computing”. In: *Proc. 9th Int. Conf. on Electronics, Circuits and Syst. (ICECS)*. Vol. 2. Dubrovnik, Croatia: IEEE, Sept. 2002, pp. 813–817. DOI: `10.1109/ICECS.2002.1046298`.

[HHB91] X. Hu, R. G. Harber, and S. C. Bass. “Expanding the range of convergence of the CORDIC algorithm”. In: *IEEE Trans. Comput.* 40.1 (Jan. 1991), pp. 13–21. ISSN: 0018-9340. DOI: `10.1109/12.67316`.

[HHP13] P.-H. Horrein, C. Hennebert, and F. Pétrot. “An environment for (re)configuration and execution management of heterogeneous flexible radio platforms”. In: *Microprocessors and Microsystems* 37.6-7, Elsevier (Aug. 2013), pp. 701–712. ISSN: 0141-9331. DOI: `10.1016/j.micpro.2012.06.002`.

[Hil+04] J. Hill et al. “The Platforms Enabling Wireless Sensor Networks”. In: *Commun. ACM* 47.6 (June 2004), pp. 41–46. ISSN: 0001-0782. DOI: `10.1145/990680.990705`.

[Hil00] J. Hill. *Hardware Profiles - [online]*. JHL Labs. Capistrano Beach, CA, USA, 2000. URL: `http://www.jlhlabs.com/hardware.htm`.

[Hil10] D. Hillenbrand. “A Flexible Design Space Exploration Platform for Wireless Sensor Networks”. PhD thesis. Karlsruhe Institute of Technology (KIT), Jan. 2010. URL: `http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/1200911`.

[Hin+08] H. Hinkelmann et al. “A Reconfigurable Prototyping Platform for Smart Sensor Networks”. In: *Proc. 4th Southern Conf. on Programmable Logic (SPL)*. San Carlos de Bariloche, Argentina: IEEE, Mar. 2008, pp. 125–130. DOI: `10.1109/SPL.2008.4547743`.

[Hin11] H. Hinkelmann. “Entwurf und Energieeffizienzanalyse von dynamisch rekonfigurierbaren Architekturen für drahtlose Sensorknoten”. PhD thesis. TU Darmstadt, 2011. ISBN: 978-3-8439-0331-8.

[Hit] Hitex. *Brochure - PowerScale With ACM Technology - The truly innovative energy profiling tool*. Karlsruhe, Germany. URL: `http://www.hitex.com/fileadmin/pdf/products/hardware_tools/b0-powerscale.pdf`.

[How+06] S. L Howard et al. “Error Control Coding in Low-Power Wireless Sensor Networks: When Is ECC Energy-Efficient?” In: *EURASIP J. Wireless Commu. and Networking* 2006.1 (May 2006). Article ID 74812, pp. 1 –14. ISSN: 1687-1499. DOI: `10.1155/WCN/2006/74812`.

[HRG08] H. Hinkelmann, A. Reinhardt, and M. Glesner. “A Methodology for Wireless Sensor Network Prototyping with Sophisticated Debugging Support”. In: *Proc. 19th IEEE/IFIP Int. Symp. on Rapid Syst. Prototyping (RSP)*. Monterey, CA, USA: IEEE, June 2008, pp. 82–88. DOI: `10.1109/RSP.2008.13`.

[HWH12] C.-M. Hsieh, Z. Wang, and J. Henkel. “A Reconfigurable Hardware Accelerated Platform for Clustered Wireless Sensor Networks”. In: *Proc. IEEE 18th Int. Conf. on Parallel and Distributed Syst. (ICPADS)*. Singapore: IEEE, Dec. 2012, pp. 498–505. DOI: `10.1109/ICPADS.2012.74`.

[HWH13] C.-M. Hsieh, Z. Wang, and J. Henkel. “DANCE: Distributed application-aware node configuration engine in shared reconfigurable sensor networks”. In: *Proc. Design, Automation and Test in Europe Conf. (DATE)*. Grenoble, France: IEEE, Mar. 2013, pp. 839–842. DOI: `10.7873/DATE.2013.177`.

[HZG10] H. Hinkelmann, P. Zipf, and M. Glesner. “Dynamically Reconfigurable Systems. Architectures, Design Methods and Applications.” In: ed. by M. Platzner, J. Teich, and N. Wehn. Springer Netherlands, 2010. Chap. Dynamically Reconfigurable Systems for Wireless Sensor Networks, pp. 315–334. ISBN: 978-90-481-3485-4. DOI: `10.1007/978-90-481-3485-4_15`.

[Hüb+11] M. Hübner et al. “A Heterogeneous Multicore System on Chip with Run-Time Reconfigurable Virtual FPGA Architecture”. In: *Proc. IEEE Int. Symp. on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*. Shanghai, China: IEEE, May 2011, pp. 143–149. DOI: `10.1109/IPDPS.2011.135`.

[IEC12] IEC. *ISO/IEC 14543-3-10:2012 Information technology – Home Electronic Systems (HES) – Part 3-10: Wireless Short-Packet (WSP) protocol optimized for energy harvesting – Architecture and lower layer protocols*. 2012.

[Igla] *IGLOO FPGA Fabric User’s Guide*. Revision 4. Microsemi. Aliso Viejo, CA, USA, 2012.

[Iglb] *IGLOO2 and SmartFusion2 SoC FPGAs Datasheet*. Revision 1. Microsemi. Aliso Viejo, CA, USA. URL: `http://www.microsemi.com/document-portal/doc_download/132042-igloo2-fpga-datasheet`.

[Iis] *Fraunhofer Institute for Integrated Circuits - Wireless Sensor Networks - Home page*. URL: `http://www.iis.fraunhofer.de/en/bf/ec/dk/sn.html`.

[Imo] *IMote2 Datasheet*. Rev. A. Document Part Number: 6020-0117-02. Crossbow Technology, Inc. Milpitas, CA, USA. URL: `http://web.univ-pau.fr/~cpham/ENSEIGNEMENT/PAU-UPPA/RESA-M2/DOC/Imote2_Datasheet.pdf`.

[Ind+03] L. S. Indrusiak et al. “Ubiquitous Access to Reconfigurable Hardware: Application Scenarios and Implementation Issues”. In: *Proc. Design, Automation and Test in Europe Conf. (DATE)*. Munich, Germany: IEEE Computer Society, Mar. 2003, pp. 940–945. ISBN: 0-7695-1870-2. DOI: `10.1109/DATE.2003.1253726`.

[Iri] *IRIS Datasheet*. 6020 - 0124 - 02 Rev. A. MEMSIC Inc. Andover, MA, USA. URL: `http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf`.

[IS11] M. Ilic and M. Stojccev. “Address generation unit as accelerator block in DSP”. In: *Proc. 10th Int. Conf. on Telecommun. in Modern Satellite Cable and Broadcasting Services (TELSIKS)*. Vol. 2. Nis, Serbia: IEEE, Oct. 2011, pp. 563–566. DOI: `10.1109/TELSKS.2011.6143177`.

[Jan+10] J. Jankkari et al. *MoDe Project - Report on the self monitoring concept and platform*. Deliverable D3.3. VTT, 2010.

[Jel+11] V. Jelicic et al. “MasliNET: A Wireless Sensor Network based environmental monitoring system”. In: *Proc. of the 34th Int. Conv. Inform. and Commun. Technology, Electronics and Microelectronics (MIPRO)*. Opatija, Croatia: IEEE, May 2011, pp. 150–155. ISBN: 978-1-4577-0996-8. URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5967041`.

[Kad+10] M. L. Kaddachi et al. “Efficient hardware solution for low power and adaptive image-compression in WSN”. In: *Proc. 17th IEEE Int. Conf. on Electronics, Circuits, and Syst. (ICECS)*. Athens, Greece: IEEE, Dec. 2010, pp. 583–586. DOI: `10.1109/ICECS.2010.5724579`.

[Kar+04] T. C. Karalar et al. “A low power localization architecture and system for wireless sensor networks”. In: *IEEE Workshop on Signal Processing Syst. (SIPS)*. Austin, TX, USA: IEEE, Oct. 2004, pp. 89–94. DOI: `10.1109/SIPS.2004.1363030`.

[KC11] J. Kwong and A. P. Chandrakasan. “An Energy-Efficient Biomedical Signal Processing Platform”. In: *IEEE J. Solid-State Circuits* 46.7 (July 2011), pp. 1742–1753. ISSN: 0018-9200. DOI: `10.1109/JSSC.2011.2144450`.

[Khe+09] A. Khezzar et al. “On the Use of Slot Harmonics as a Potential Indicator of Rotor Bar Breakage in the Induction Machine”. In: *IEEE Trans. Ind. Electron.* 56.11 (Nov. 2009), pp. 4592–4605. ISSN: 0278-0046. DOI: `10.1109/TIE.2009.2030819`.

[Khu+11] K. Khursheed et al. “Exploration of Tasks Partitioning between Hardware Software and Locality for a Wireless Camera Based Vision Sensor Node”. In: *Proc. 6th Int. Symp. on Parallel Computing in Elect. Eng. (PARELEC)*. Luton, UK: IEEE, Apr. 2011, pp. 127–132. DOI: `10.1109/PARELEC.2011.21`.

[Kim+12] C. Kim et al. “ULP-SRP: Ultra low power Samsung Reconfigurable Processor for biomedical applications”. In: *Proc. Int. Conf. on Field-Programmable Technology (FPT)*. Seoul, South Korea: IEEE, Dec. 2012, pp. 329–334. DOI: `10.1109/FPT.2012.6412157`.

[KK06] T. T.-O. Kwok and Y.-K. Kwok. “Computation and Energy Efficient Image Processing in Wireless Sensor Networks Based on Reconfigurable Computing”. In: *Proc. Int. Conference on Parallel Processing (ICPP), Workshop on Parallel and Distributed Multimedia (PDM)*. Columbus, OH, USA: IEEE Computer Society, Aug. 2006, pp. 43–50. ISBN: 0-7695-2637-3. DOI: `10.1109/ICPPW.2006.30`.

[KM10] Y. Kim and R. N. Mahapatra. “Dynamic Context Compression for Low-Power Coarse-Grained Reconfigurable Architecture”. In: *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.* 18.1 (Jan. 2010), pp. 15–28. ISSN: 1063-8210. DOI: `10.1109/TVLSI.2008.2006846`.

[KM11] Y. Kim and R. N. Mahapatra. “Design of Low-Power Coarse-Grained Reconfigurable Architectures”. In: CRC Press, 2011. Chap. Dynamic Context Compression for Low-Power CGRA, pp. 101–122. ISBN: 9781439825105. URL: `http://www.crcpress.com/product/isbn/9781439825105`.

[Kos+10] J. Koskinen et al. “Wireless Sensor Networks for infrastructure and industrial monitoring applications”. In: *Proc. Int. Conf. on Inform. and Commun. Technology Convergence (ICTC)*. Jeju, South Korea: IEEE, Nov. 2010, pp. 250–255. DOI: `10.1109/ICTC.2010.5674672`.

[Kra+11] Y. E. Krasteva et al. “Embedded Runtime Reconfigurable Nodes for Wireless Sensor Networks Applications”. In: *IEEE Sensors J.* 11.9 (Sept. 2011), pp. 1800–1810. ISSN: 1530-437X. DOI: `10.1109/JSEN.2011.2104948`.

[Kri+05] L. Krishnamurthy et al. “Design and Deployment of Industrial Sensor Networks: Experiences from a Semiconductor Plant and the North Sea”. In: *Proc. 3rd ACM Int. Conf. on Embedded Networked Sensor Syst. (SenSys)*. San Diego, CA, USA: ACM, Nov. 2005, pp. 64–75. ISBN: 1-59593-054-X. DOI: `10.1145/1098918.1098926`.

[Kul10] J. Kullaa. “Vibration-Based Structural Health Monitoring Under Variable Environmental Operational Conditions”. In: *New Trends in Vibration Based Structrual Health Monitoring*. Ed. by A. Deraemaeker and K. Worden. Vol. 520. CISM Courses and Lectures. SpringerWienNewYork, 2010, pp. 107–181. DOI: `10.1007/978-3-7091-0399-9_4`.

[KW07a] H. Karl and A. Willig. “Protocols and Architectures for Wireless Sensor Networks”. In: John Wiley & Sons, Ltd, 2007. Chap. Communication Protocols, pp. 83–436. ISBN: 978-0-470-09510-2. URL: `http://www.wiley.com//legacy/wileychi/wsn/`.

[KW07b] H. Karl and A. Willig. “Protocols and Architectures for Wireless Sensor Networks”. In: John Wiley & Sons, Ltd, 2007. Chap. Network Architecture, pp. 59–81. ISBN: 978-0-470-09510-2. URL: `http://www.wiley.com//legacy/wileychi/wsn/`.

[KW07c] H. Karl and A. Willig. “Protocols and Architectures for Wireless Sensor Networks”. In: John Wiley & Sons, Ltd, 2007. Chap. Localization and positioning, pp. 231–250. ISBN: 978-0-470-09510-2. URL: `http://www.wiley.com//legacy/wileychi/wsn/`.

[KW07d] H. Karl and A. Willig. “Protocols and Architectures for Wireless Sensor Networks”. In: John Wiley & Sons, Ltd, 2007. Chap. Single-node architecture, pp. 17–58. ISBN: 978-0-470-09510-2. URL: `http://www.wiley.com//legacy/wileychi/wsn/`.

[Lav11] A. Lavado. “Adaptive Differential Microphone Array and its Implementation on FPGA”. Master Thesis. TU Darmstadt, Oct. 2011.

[LC02] P. Levis and D. Culler. “Maté: A Tiny Virtual Machine for Sensor Networks”. In: *SIGARCH Comput. Archit. News* 30.5, ACM (Oct. 2002), pp. 85–95. ISSN: 0163-5964. DOI: `10.1145/635506.605407`.

[Led+09] A. Ledeczi et al. “Wireless Acoustic Emission Sensor Network for Structural Monitoring”. In: *IEEE Sensors J.* 9.11 (Nov. 2009), pp. 1370–1377. ISSN: 1530-437X. DOI: `10.1109/JSEN.2009.2019315`.

[Lee+13] Y. Lee et al. “A Modular $1mm^3$ Die-Stacked Sensing Platform With Low Power I^2C Inter-Die Communication and Multi-Modal Energy Harvesting”. In: *IEEE J. Solid-State Circuits* 48.1 (2013), pp. 229–243. ISSN: 0018-9200. DOI: `10.1109/JSSC.2012.2221233`.

[Leo07a] M. Leopold. *HogthrobV0 User’s Manual*. Tech. Rep. no. 07/05. Copenhagen, Denmark: Dept. of Computer Science, University of Copenhagen, Sept. 2007. URL: `http://www.diku.dk/~leopold/work/leopold07htv0.pdf`.

[Leo07b] M. Leopold. “Sensor Network Motes: Portability & Performance”. PhD thesis. Department of Computer Science, Faculty of Science, University of Copenhagen, Dec. 2007. URL: `http://www.diku.dk/~leopold/work/leopold08motes_u.pdf`.

[Lev+05] P. Levis et al. “TinyOS: An Operating System for Sensor Networks”. In: *Ambient Intelligence*. Ed. by W. Weber, J. M. Rabaey, and E. Aarts. Part II. Springer Berlin Heidelberg, 2005, pp. 115–148. ISBN: 978-3-540-23867-6. DOI: `10.1007/3-540-27139-2_7`.

[Li+12] Y. Li et al. “Hardware reconfigurable wireless sensor network node with power and area efficiency”. In: *IET Wireless Sensor Syst.* 2.3 (2012), pp. 247–252. ISSN: 2043-6386. DOI: `10.1049/iet-wss.2011.0162`.

[Lia+13] J. Liao et al. “FPGA based wireless sensor node with customizable event-driven architecture”. English. In: *EURASIP J. Embedded Syst.* 2013.1 (Apr. 2013), pp. 1–11. DOI: `10.1186/1687-3963-2013-5`.

[Liba] *Libelium - Home page*. URL: `www.libelium.com`.

[Libb] *Libero SoC User’s Guide*. v.11.3. Microsemi. Aliso Viejo, CA, Mar. 2014.

[LL06] J. P. Lynch and K. J. Loh. “A Summary Review of Wireless Sensors ans Sensor Networks for Structural Health Monitoring”. In: *The Schock and Vibration Digest* 38.2, SAGE Publications (Mar. 2006), pp. 91–128. URL: `http://www-personal.umich.edu/~jerlynch/papers/SVDReview.pdf`.

[Lom+12] M. Lombardo et al. “Power management techniques in an FPGA-based WSN node for high performance applications”. In: *Proc. 7th Int. Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. York, UK: IEEE, July 2012, 8 pages. DOI: `10.1109/ReCoSoC.2012.6322888`.

[Lot] *Lotus Datasheet*. 6020 - 0705 - 01 Rev. A. MEMSIC Inc. Andover, MA, USA. URL: `http://www.memsic.com/userfiles/files/Datasheets/WSN/6020-0705-01_A_LOTUS.pdf`.

[LPZ07] D. Lymberopoulos, N.B. Priyantha, and Feng Zhao. “mPlatform: A Reconfigurable Architecture and Efficient Data Sharing Mechanism for Modular Sensor Nodes”. In: *Proc. 6th Int. Symp. on Inform. Processing in Sensor Networks (IPSN)*. Cambridge, MA, USA: ACM, Apr. 2007, pp. 128–137. DOI: `10.1109/IPSN.2007.4379672`.

[LSR03] S.-F. Li, R. Sutton, and J. M. Rabaey. “Low Power Operating System for Heterogeneous Wireless Communication System”. In: *Compilers and Operating Systems for Low Power*. Ed. by L. Benini, M. Kandemir, and J. Ramanujam. Springer US, 2003, pp. 1–16. ISBN: 1-4020-7573-1. DOI: `10.1007/978-1-4419-9292-5_1`.

[Ltc] *LTC6900: Low Power, 1kHz to 20MHz Resistor Set SOT-23 Oscillator*. Linear Technology. Milpitas, CA, USA. URL: `http://cds.linear.com/docs/en/datasheet/6900fa.pdf`.

[Lu+09] S. Lu et al. “Design and Implementation of an ASIC-based Sensor Device for WSN applications”. In: *IEEE Trans. Consum. Electron.* 55.4 (Nov. 2009), pp. 1959–1967. ISSN: 0098-3063. DOI: `10.1109/TCE.2009.5373756`.

[Mar+06] P. J. Marrón et al. “Challenges of Complex Data Processing in Real World Sensor Network Deployments”. In: *Proc. ACM Workshop on Real-World Wireless Sensor Networks (REALWSN)*. Uppsala, Sweden: ACM, June 2006, pp. 43–48. URL: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.71.6789`.

[Mar+14] J. Martinez et al. “A 2D FEM analysis of electromechanical signatures in induction motors under dynamic eccentricity”. In: *Int. J. Numerical Modelling: Electronic Networks, Devices and Fields* 27.3, John Wiley & Sons, Ltd. (May 2014), pp. 555 –571. DOI: `10.1002/jnm.1942`.

[May+14] D. Mayer et al. “Recent Developments in Adaptronics, Results from the LOEWE Center Adaptronics - Research, Innovation, Application (AdRIA)”. In: ed. by T. Bein. Springer, 2014 (to be published). Chap. Structural Health Monitoring, 16 pages.

[MCP09] G.-G. Mplemenos, P. Christou, and I. Papaefstathiou. “Using reconfigurable hardware devices in WSNs for accelerating and reducing the power consumption of header processing tasks”. In: *Proc. IEEE 3rd Int. Symp. on Advanced Networks and Telecommun. Syst. (ANTS)*. New Dehli, India: IEEE, Dec. 2009, 3 pages. DOI: `10.1109/ANTS.2009.5409874`.

[MD08] T. X. Mei and X. J. Ding. “New condition monitoring techniques for vehicle suspensions”. In: *Proc. 4th IET Int. Conf. on Railway Condition Monitoring*. Derby, UK: IET, June 2008, 6 pages. URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4580869`.

[Med+10] L. M. C. Medina et al. “FPGA-Based Multiple-Channel Vibration Analyzer for Industrial Applications in Induction Motor Failure Detection”. In: *IEEE Trans. Instrum. Meas.* 59.1 (Jan. 2010), pp. 63–72. ISSN: 0018-9456. DOI: `10.1109/TIM.2009.2021642`.

[Mei+03] B. Mei et al. “ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix”. In: *Proc. 13th Int. Conf. on Field Programmable Logic and Applicat. (FPL)*. Ed. by P. Cheung and G. A. Constantinides. Vol. 2778. Lecture Notes in Computer Science. Lisbon, Portugal: Springer Berlin Heidelberg, 2003, pp. 61–70. ISBN: 978-3-540-40822-2. DOI: `10.1007/978-3-540-45234-8_7`. URL: `http://dx.doi.org/10.1007/978-3-540-45234-8_7`.

[Mem] *Memsic - Home page*. URL: `www.memsic.com`.

[Meu+08] G. de Meulenaer et al. “On the Energy Cost of Communication and Cryptography in Wireless Sensor Networks”. In: *Proc. IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Commun. (WIMOB)*. Avignon, France: IEEE Computer Society, Oct. 2008, pp. 580–585. DOI: `10.1109/WiMob.2008.16`.

[Mica] *Mica2 Datasheet*. Document Part Number: 6020-0042-04. Crossbow Technology, Inc. Milpitas, CA, USA. URL: `http://www.eol.ucar.edu/isf/facilities/isa/internal/CrossBow/DataSheets/mica2.pdf`.

[Micb] *MicaZ Datasheet*. 6020 - 0065 - 05 Rev. A. MEMSIC Inc. Andover, MA, USA. URL: `http://www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf`.

[Mic12] MicroStrain. *Heavy-Vehicle Condition-Based Maintenance - Home page*. 2012. URL: `http://www.microstrain.com/applications/vehicle-health-monitoring/heavy-vehicle-condition-based-maintenance`.

[Mic13] Microsemi. *CoreFFT v6.3 Handbook*. 50200267-3/09.13. Aliso Viejo, CA, USA, 2013. URL: `http://www.actel.com/ipdocs/CoreFFT_HB.pdf`.

[Min+00] R. Min et al. “An Architecture for a Power-Aware Distributed Microsensor Node”. In: *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*. Lafayette, LA, USA: IEEE, Oct. 2000, pp. 581–590. DOI: `10.1109/SIPS.2000.886756`.

[Min+01] R. Min et al. “Low-power wireless sensor networks”. In: *Proc. 14th Int. Conf. on VLSI Design*. Bangalore, India: IEEE Computer Society, Jan. 2001, pp. 205–210. DOI: `10.1109/ICVD.2001.902661`.

[MKS12] M. Mahmudimanesh, A. Khelil, and N. Suri. “Intellingent Sensor Networks: The Integration of Sensor Networks, Signal Processing and Machine Learning”. In: ed. by F. Hu and Q. Hao. CRC Press, Taylor & Francis Group, 2012. Chap. Compressive Sensing for Wireless Sensor Networks, pp. 379–395. ISBN: 9781439892817. URL: `http://www.crcpress.com/product/isbn/9781439892817`.

[MNS06] C. N. Mathur, K. Narayan, and K. P. Subbalakshmi. “On the Design of Error-Correcting Ciphers”. In: *EURASIP J. Wireless Commun. and Networking* 2006, Hindawi Publishing Corporation (2006), pp. 1–12. DOI: `10.1155/WCN/2006/42871`.

[Mod] *FP7 Maintenance on Demand project - Home page*. URL: `www.fp7-mode.eu`.

[Moo98] G. E. Moore. “Cramming More Components Onto Integrated Circuits”. In: *Proc. IEEE* 86.1 (Jan. 1998), pp. 82–85. ISSN: 0018-9219. DOI: `10.1109/JPROC.1998.658762`.

[MPB11] S. McGettrick, K. Patel, and C. Bleakley. “High Performance Programmable FPGA Overlay for Digital Signal Processing”. In: *Reconfigurable Computing: Architectures, Tools and Applications; Proc. 7th Int. Symp. (ARC)*. Ed. by A. Koch et al. Vol. 6578. Lecture Notes in Computer Science. Belfast, UK: Springer Berlin Heidelberg, Mar. 2011, pp. 375–384. ISBN: 978-3-642-19474-0. DOI: `10.1007/978-3-642-19475-7_39`.

[MR08] P. Muralidhar and C. B. R. Rao. “Reconfigurable wireless sensor network node based on Nios core”. In: *Proc. 4th Int. Conf. on Wireless Commun. and Sensor Networks (WCSN)*. Allahabad, India: IEEE, Dec. 2008, pp. 67–72. DOI: `10.1109/WCSN.2008.4772684`.

[Msa] *Flash*Freeze Control Using an Internal Oscillator*. Application Note AC332. Actel Corporation. Mountain View, CA, USA, June 2009.

[Nac+05] L. Nachman et al. “The Intel™Mote Platform: a Bluetooth-based Sensor Network for Industrial Monitoring”. In: *Proc. 4th Int. Symp. on Inform. Processing in Sensor Networks (IPSN)*. Los Angeles, CA, USA: IEEE, Apr. 2005, pp. 437–442. DOI: `10.1109/IPSN.2005.1440968`.

[Nac+08] L. Nachman et al. “IMOTE2: Serious Computation at the Edge”. In: *Proc. Int. Wireless Commun. and Mobile Computing Conf. (IWCMC)*. Crete, Greece: IEEE, Aug. 2008, pp. 1118–1123. DOI: `10.1109/IWCMC.2008.194`.

[Nah+07] A. Nahapetian et al. “Dynamic Reconfiguration in Sensor Networks with Regenerative Energy Sources”. In: *Proc. Design, Automation and Test in Europe Conf. (DATE)*. Nice, France: EDAA, Apr. 2007, 6 pages. DOI: `10.1109/DATE.2007.364433`.

[Nan+11] S. Nandi et al. “Detection of Eccentricity Faults in Induction Machines Based on Nameplate Parameters”. In: *IEEE Trans. Ind. Electron.* 58.5 (May 2011), pp. 1673–1683. ISSN: 0278-0046. DOI: `10.1109/TIE.2010.2055772`.

[NTL05] S. Nandi, H. A. Toliyat, and Xiaodong Li. “Condition monitoring and fault diagnosis of electrical motors-a review”. In: *IEEE Trans. Energy Convers.* 20.4 (Dec. 2005), pp. 719–729. ISSN: 0885-8969. DOI: `10.1109/TEC.2005.847955`.

[O’F+05] B. O’Flynn et al. “The Development of a Novel Minaturized Modular Platform for Wireless Sensor Networks”. In: *Proc. 4th Int. Symp. on Inform. Processing in Sensor Networks (IPSN)*. Los Angeles, CA, USA: IEEE, Apr. 2005, pp. 370–375. ISBN: 0-7803-9202-7. DOI: `10.1109/IPSN.2005.1440951`.

[Okm11] Y. Okmen. “SIMD Floating Point Extension for Ray Tracing”. MA thesis. Computer Engineering, Department of Electrical Engineering, Faculty of Electrical Engineering, Mathematics and Computer Science: Delft University of Technology, 2011. URL: `http://www.asam-project.org/information/yokmen2011msc.pdf`.

[OM+08] A. Ordaz-Moreno et al. “Automatic Online Diagnosis Algorithm for Broken-Bar Detection on Induction Motors Based on Discrete Wavelet Transform for FPGA Implementation”. In: *IEEE Trans. Ind. Electron.* 55.5 (May 2008), pp. 2193–2202. ISSN: 0278-0046. DOI: `10.1109/TIE.2008.918613`.

[One] *ONE-NET Low Power Wireless Protocol - Home Page*. 2009. URL: `http://one-net.sourceforge.net/`.

[Ost+06] F. Osterlind et al. “Cross-Level Sensor Network Simulation with COOJA”. In: *Proc. 31st IEEE Conf. on Local Computer Networks (LCN)*. Tampa, FL, USA: IEEE, Nov. 2006, pp. 641–648. DOI: `10.1109/LCN.2006.322172`.

[PA11] D. M. Pham and S. M. Aziz. “FPGA architecture for object extraction in Wireless Multimedia Sensor Network”. In: *Proc. 7th Int. Conf. on Intelligent Sensors, Sensor Networks and Inform. Processing (ISSNIP)*. Adelaide, Australia: IEEE, Dec. 2011, pp. 294–299. DOI: `10.1109/ISSNIP.2011.6146563`.

[PBT12] A. de la Piedra, A. Braeken, and A. Touhafi. “Sensor Systems Based on FPGAs and Their Applications: A Survey”. In: *Sensors* 12.9, MDPI (Sept. 2012), pp. 12235–12264. ISSN: 1424-8220. DOI: `10.3390/s120912235`.

[PCB00] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. “The Cricket Location-support System”. In: *Proc. 6th Annu. Int. Conf. on Mobile Computing and Networking (MobiCom)*. Boston, MA, USA: ACM, Aug. 2000, pp. 32–43. ISBN: 1-58113-197-6. DOI: `10.1145/345910.345917`.

[Pee+07] B. Peeters et al. “Operational Modal Analysis for Estimating the Dynamic Properties of a Stadium Structure during a Football Game”. In: *Shock and Vibration* 14.4, Hindawi Publishing Corporation (2007), pp. 283–303. DOI: `10.1155/2007/531739`.

[Pee11] B. Peeters. *MoDe Project - Report on multi-layer processing*. Deliverable D6.2. LMS Int., 2011.

[Pee12] B. Peeters. *MoDe Project - Report with condition monitoring concept for considered test cases*. Deliverable D6.1. LMS Int., 2012.

[Phi+14] F. Philipp et al. “Embedded Monitoring System for Online Motor Current Signature Analysis”. In: *IEEE Trans. Ind. Electron.* X (2014), 8 pages.

[Phi09] F. Philipp. “Modeling and Implementation of Distributed Algorithms for Acoustic Localization in a Wireless Sensor Network”. Diploma Thesis. TU Darmstadt, Mar. 2009.

[Ple+03] C. Plessl et al. “The Case for Reconfigurable Hardware in Wearable Computing”. In: *Personal Ubiquitous Comput.* 7.5, Springer-Verlag (Oct. 2003), pp. 299–308. ISSN: 1617-4909. DOI: `10.1007/s00779-003-0243-x`.

[PMB11] K. Patel, S. McGettrick, and C. J. Bleakley. “SYSCORE: A Coarse Grained Reconfigurable Array Architecture for Low Energy Biosignal Processing”. In: *Proc. IEEE 19th Annu. Int. Symp. on Field-Programmable Custom Computing Mach. (FCCM)*. Salt Lake City, UT, USA: IEEE Computer Society, May 2011, pp. 109–112. DOI: `10.1109/FCCM.2011.38`.

[Pon12] S. Pongyupinpanich. “Optimal Design of Fixed-Point and Floating-Point Arithmetic Units for Scientific Applications”. PhD thesis. TU Darmstadt, 2012. URL: `http://tuprints.ulb.tu-darmstadt.de/3091/`.

[Por+06a] J. Portilla et al. “A Hardware Library for Sensors/Actuators Interfaces in Sensor Networks”. In: *Proc. 13th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS)*. Nice, France: IEEE, Dec. 2006, pp. 1244–1247. DOI: `10.1109/ICECS.2006.379687`.

[Por+06b] J. Portilla et al. “A Modular Architecture for Nodes in Wireless Sensor Networks”. In: *J. of Universal Comput. Science* 12.3 (Mar. 2006), pp. 328–339. DOI: `10.3217/jucs-012-03-0328`. URL: `http://www.jucs.org/jucs_12_3/a_modular_architecture_for`.

[PP12] S. Padaraju and F. Philipp. *HaLOEWEn User’s Guide*. Tech. Rep. Darmstadt, Germany: Microelectronic Systems Design Group, TU Darmstadt, 2012.

[PSC05] J. Polastre, R. Szewczyk, and D. Culler. “Telos: Enabling Ultra-low Power Wireless Research”. In: *Proc. 4th Int. Symp. on Inform. Processing in Sensor Networks (IPSN)*. Los Angeles, CA, USA: IEEE, Apr. 2005, pp. 364–369. DOI: `10.1109/IPSN.2005.1440950`.

[Pso] *PSoC 5LP: CY8C58LP Family Datasheet*. Document Number: 001-84932 Rev. *F. Cypress. San Jose, CA, USA, 2014. URL: `http://www.cypress.com/?docID=49437`.

[Rab+00] J. M. Rabaey et al. “PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking”. In: *IEEE Computer* 33.7, IEEE Computer Society (July 2000), pp. 42–48. ISSN: 0018-9162. DOI: `10.1109/2.869369`.

[Rab+06] J. M. Rabaey et al. “Ultra-Low-Power Design”. In: *IEEE Circuits Devices Mag.* 22.4 (July 2006), pp. 23–29. ISSN: 8755-3996. DOI: `10.1109/MCD.2006.1708372`.

[Ram+07] H. Ramamurthy et al. “Wireless Industrial Monitoring and Control Using a Smart Sensor Platform”. In: *IEEE Sensors J.* 7.5 (May 2007), pp. 611–618. ISSN: 1530-437X. DOI: `10.1109/JSEN.2007.894135`.

[RCN96] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. “Digital Integrated Circuits: A Design Perspective”. In: Prentice-Hall, 1996. Chap. The Static CMOS Inverter, pp. 176–228.

[Rei11] A. Reinhardt. “Designing Sensor Networks for Smart Spaces”. PhD thesis. TU Darmstadt, 2011. URL: `http://tuprints.ulb.tu-darmstadt.de/2844/`.

[RFB10] 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 (June 2010), 23:1–23:17. ISSN: 1084-4309. DOI: `10.1145/1754405.1754408`.

[RM10] M. Rullmann and R. Merker. “Dynamically Reconfigurable Systems. Architectures, Design Methods and Applications”. In: ed. by M. Platzner, J. Teich, and N. Wehn. Springer Netherlands, 2010. Chap. Design Methods and Tools for Improved Partial Dynamic Reconfiguration, pp. 161–182. ISBN: 978-90-481-3485-4. DOI: `10.1007/978-90-481-3485-4_8`.

[SBB06] S. Shukla, N. W. Bergmann, and J. Becker. “QUKU: a two-level reconfigurable architecture”. In: *Proc. IEEE Comput. Soc. Annu. Symp. on Emerging VLSI Technologies and Architectures (ISVLSI)*. Karlsruhe, Germany: IEEE Computer Society, Mar. 2006, 6 pages. DOI: `10.1109/ISVLSI.2006.76`.

[SBW09] D. Schmidt, M. Berning, and N. Wehn. “Error correction in single-hop wireless sensor networks - A case study”. In: *Proc. Design, Automation and Test in Europe Conf. (DATE)*. Nice, France: EDAA, Apr. 2009, pp. 1296–1301. DOI: `10.1109/DATE.2009.5090865`.

[SC01] A. Sinha and A. Chandrakasan. “Dynamic power management in wireless sensor networks”. In: *IEEE Des. Test of Comput.* 18.2 (Mar. 2001), pp. 62–74. ISSN: 0740-7475. DOI: `10.1109/54.914626`.

[Sch+08] T. Schmid et al. “The True Cost of Accurate Time”. In: *Proc. Workshop on Power Aware Computing and Syst. (HotPower)*. San Diego, CA, USA: USENIX Association, Dec. 2008, 5 pages. URL: `https://www.usenix.org/legacy/event/hotpower08/tech/full_papers/schmid/schmid.pdf`.

[Sen] *Sensinode - Home page*. URL: `www.sensinode.com`.

[SFN00] S. Scalera, M. Falco, and B. Nelson. “A reconfigurable computing architecture for microsensors”. In: *Proc. IEEE Symp. on Field-Programmable Custom Computing Mach. (FCCM)*. Napa Valley, CA, USA: IEEE Computer Society, Apr. 2000, pp. 59–67. DOI: `10.1109/FPGA.2000.903393`.

[She+06] M. Sheets et al. “A Power-Managed Protocol Processor for Wireless Sensor Networks”. In: *Dig. Tech. Papers, Symp. on VLSI Circuits*. Honolulu, HI, USA: IEEE, June 2006, pp. 212–213. DOI: `10.1109/VLSIC.2006.1705385`.

[She+12] Z. Shelby et al. *Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*. Internet Engineering Task Force (IETF), RFC 6775. Nov. 2012. URL: `http://www.rfc-editor.org/rfc/pdfrfc/rfc6775.txt.pdf`.

[Sim+04] G. Simon et al. “Sensor Network-based Countersniper System”. In: *Proc. 2nd Int. Conf. on Embedded Networked Sensor Syst. (SenSys)*. Baltimore, MD, USA: ACM, Nov. 2004, 12 pages. ISBN: 1-58113-879-2. DOI: `10.1145/1031495.1031497`.

[Sin+00] H. Singh et al. “MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications”. In: *IEEE Trans. Comput.* 49.5 (May 2000), pp. 465–481. ISSN: 0018-9340. DOI: `10.1109/12.859540`.

[SLL11] Y. Sun, L. Li, and H. Luo. “Design of FPGA-Based Multimedia Node for WSN”. In: *Proc. 7th Int. Conf. on Wireless Commun., Networking and Mobile Computing (WiCOM)*. Wuhan, China: IEEE, Sept. 2011, pp. 1–5. DOI: `10.1109/wicom.2011.6040365`.

[Sou+07] D. Soudris et al. “Fine- and Coarse-Grain Reconfigurable Computing”. In: ed. by S. Vassiliadis and D. Soudris. Part II. Springer Netherlands, 2007. Chap. A Low-Energy FPGA Architecture and Supporting CAD Tool Design Flow, pp. 153–180. DOI: `10.1007/978-1-4020-6505-7_3`.

[SSS10] M. Sun, W. J. Staszewski, and R. N. Swamy. “Smart Sensing Technologies for Structural Health Monitoring of Civil Engineering Structures”. In: *Advances in Civil Engineering* 2010.724962, Hindawi Publishing Corporation (May 2010), pp. 1 –13. DOI: 10.1155/2010/724962.

[Sta] *Stargate Developer’s Guide*. Rev. B. Document 7430-0317-13. Crossbow Technology, Inc. Milpitas, CA, USA, Jan. 2006. URL: http://www.cse.nd.edu/~cpoellab/teaching/cse40815/Stargate_Manual_7430-0317-13_B.pdf.

[Sto71] H. S. Stone. “Parallel Processing with the Perfect Shuffle”. In: *IEEE Trans. Comput.* C-20.2 (Feb. 1971), pp. 153–161. ISSN: 0018-9340. DOI: 10.1109/T-C.1971.223205.

[Sun] *Sun™SPOT Main Board Technical Datasheet*. Rev 8.0. Sun Labs. Redwood City, CA, USA, Oct. 2010. URL: http://sunspotworld.com/docs/Yellow/eSPOT8ds.pdf.

[Sun10] P. Sundararajan. *High Performance Computing Using FPGAs*. White Paper WP375. San Jose, CA, USA: Xilinx, Sept. 2010. URL: http://www.xilinx.com/support/documentation/white_papers/wp375_HPC_Using_FPGAs.pdf.

[Sut05] H. Sutter. “The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software”. In: *Dr. Dobb’s J.* 30 (Mar. 2005), pp. 1 –7. URL: http://www.gotw.ca/publications/concurrency-ddj.htm.

[Swt] *SWT: The Standard Widget Toolkit - Home page*. URL: http://www.eclipse.org/swt/.

[Szi+13] S. Szilvasi et al. “Marmote SDR: Experimental Platform for Low-Power Wireless Protocol Stack Research”. In: *J. Sensor and Actuator Networks* 2.3, MDPI (Sept. 2013), pp. 631–652. ISSN: 2224-2708. DOI: 10.3390/jsan2030631.

[Tan+08] S. Tanaka et al. “Reconfigurable Hardware Architecture for Saving Power Consumption on a Sensor Node”. In: *Proc. Int. Conf. on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. Sydney, Australia: IEEE, Dec. 2008, pp. 405–410. DOI: 10.1109/ISSNIP.2008.4762022.

[TB09] F. Thoma and J. Becker. “Dynamic System Reconfiguration in Heterogeneous Platforms: The MORPHEUS Approach”. In: ed. by N. Voros, A. Rosti, and M. Hübner. Vol. 40. Lecture Notes in Electrical Engineering. Springer Netherlands, 2009. Chap. Control of Dynamic Reconfiguration, pp. 129–137. URL: http://www.springer.com/engineering/circuits+%26+systems/book/978-90-481-2426-8.

[TD11] P. Turcza and M. Duplaga. “Low power FPGA-based image processing core for wireless capsule endoscopy”. In: *Sensors and Actuators A: Physical* 172.2, Elsevier (Dec. 2011), pp. 552 –560. ISSN: 0924-4247. DOI: 10.1016/j.sna.2011.09.026.

[Tel] *TelosB Datasheet*. 6020 - 0094 - 03 Rev. A. MEMSIC Inc. Andover, MA, USA. URL: http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf.

[Ton+09] J. Tong et al. “Design of Wireless Sensor Network Node with Hyperchaos Encryption Based on FPGA”. In: *Proc. Int. Workshop on Chaos-Fractals Theories and Applications (IWCFTA)*. Shenyang, China: IEEE Computer Society, Nov. 2009, pp. 190–194. DOI: 10.1109/IWCFTA.2009.47.

[TSV07] G. Theodoridis, D. Soudris, and S. Vassiliadis. “Fine- and Coarse-Grain Reconfigurable Computing”. In: ed. by S. Vassiliadis and D. Soudris. Springer Netherlands, 2007. Chap. A Survey of Coarse-Grain Reconfigurable Architectures and CAD Tools, pp. 89 –149. ISBN: 978-1-4020-6504-0. DOI: 10.1007/978-1-4020-6505-7_2.

[TTL76] Tran-Thong and B. Liu. “Fixed-point fast Fourier transform error analysis”. In: *IEEE Trans. on Acoust., Speech, Signal Process.* 24.6 (Dec. 1976), pp. 563–573. ISSN: 0096-3518. DOI: 10.1109/TASSP.1976.1162875.

[Ull+04] M. Ullmann et al. “An FPGA run-time system for dynamical on-demand reconfiguration”. In: *Proc. 8th Int. Parallel and Distributed Processing Symp. (IPDPS)*. Santa Fe, NM, USA: IEEE Computer Society, Apr. 2004, 8 pages. DOI: `10.1109/IPDPS.2004.1303106`.

[Val+12a] J. Valverde et al. “Using SRAM Based FPGAs for Power-Aware High Performance Wireless Sensor Networks”. In: *Sensors* 12.3, MDPI (Feb. 2012), pp. 2667–2692. ISSN: 1424-8220. DOI: `10.3390/s120302667`.

[Val+12b] J. Valverde et al. “Wireless Sensor Network for Environmental Monitoring: Application in a Coffee Factory”. In: *Int. J. Distributed Sensor Networks* 2012.638067, Hindawi Publishing Corporation (Nov. 2012), pp. 1 –18. DOI: `10.1155/2012/638067`.

[Ven+08] P. J. C. Ventura et al. “An embedded system to assess the automotive shock absorber condition under vehicle operation”. In: *Proc. IEEE Sensors Conf.* Lecce, Italy: IEEE, Oct. 2008, pp. 1210–1213. DOI: `10.1109/ICSENS.2008.4716660`.

[Vir+05] K. Virk et al. “Design of a development platform for HW/SW codesign of wireless integrated sensor nodes”. In: *Proc. 8th Euromicro Conf. on Digital System Design (DSD)*. Porto, Portugal: IEEE Computer Society, Aug. 2005, pp. 254–260. DOI: `10.1109/DSD.2005.32`.

[Vol+07] P. Volgyesi et al. “Shooter Localization and Weapon Classification with Soldier-wearable Networked Sensors”. In: *Proc. 5th Int. Conf. on Mobile Syst., Applicat. and Services (MobiSys)*. San Juan, PR, USA: ACM, June 2007, pp. 113–126. ISBN: 978-1-59593-614-1. DOI: `10.1145/1247660.1247676`.

[Vol+10] P. Volgyesi et al. “Software Development for a Novel WSN Platform”. In: *Proc. ICSE Workshop on Software Eng. for Sensor Network Applicat. (SESENA)*. Cape Town, South Africa: ACM, May 2010, pp. 20–25. ISBN: 978-1-60558-969-5. DOI: `10.1145/1809111.1809119`.

[VS+10] L. A. Vera-Salas et al. “Reconfigurable Node Processing Unit for a Low-Power Wireless Sensor Network”. In: *Proc. Int. Conf. on Reconfigurable Computing and FPGAs (ReConFig)*. Quintana Roo, Mexico: IEEE Computer Society, Dec. 2010, pp. 173–178. DOI: `10.1109/ReConFig.2010.48`.

[Wan+01] M. Wan et al. “Design Methodology of a Low-Energy Reconfigurable Single-Chip DSP System”. In: *J. VLSI signal processing systems for signal, image and video technology* 28.1-2, Kluwer Academic Publishers (2001), pp. 47–61. ISSN: 0922-5773. DOI: `10.1023/A:1008159121620`.

[Wan+13a] W. Wang et al. “Thermoelectric Energy Harvesting for Building Energy Management Wireless Sensor Networks”. In: *Int. J. Distributed Sensor Networks* 2013.232438, Hindawi Publishing Corporation (May 2013), pp. 1 –14. DOI: `10.1155/2013/232438`.

[Wan+13b] Y. S. Wang et al. “Hierarchical representation of on-chip context to reduce reconfiguration time and implementation area for coarse-grained reconfigurable architecture”. In: *Science China Information Sciences* 56.11, Springer Berlin Heidelberg (Nov. 2013), pp. 1–20. ISSN: 1674-733X. DOI: `10.1007/s11432-013-4842-5`.

[War+01] B. Warneke et al. “Smart Dust Communicating with a Cubic-Millimeter Computer”. In: *IEEE Computer* 34.1 (Jan. 2001), pp. 44–51. ISSN: 0018-9162. DOI: `10.1109/2.895117`.

[WC02] A. Wang and A. Chandrakasan. “Energy-efficient DSPs for wireless sensor networks”. In: *IEEE Signal Processing Mag.* 19.4 (July 2002), pp. 68–78. ISSN: 1053-5888. DOI: `10.1109/MSP.2002.1012351`.

[Wei+09] J. Wei et al. “Design and Implementation of Wireless Sensor Node based on Open Core”. In: *Proc. IEEE Youth Conf. on Inform., Computing and Telecommun. (YC-ICT)*. Beijing, China: IEEE, Sept. 2009, pp. 102–105. DOI: `10.1109/YCICT.2009.5382416`.

[Wei99] M. Weiser. “The Computer for the 21st Century”. In: *SIGMOBILE Mob. Comput. Commun. Rev.* 3.3, ACM (July 1999), pp. 3–11. ISSN: 1559-1662. DOI: `10.1145/329124.329126`.

[Wil+07] J. Wilder et al. “Wireless Sensor Networks for Updating Reprogrammable Logic Designs in Real-time”. In: *Proc. Reconfigurable Syst., Microsystems, and Nanotechnology Workshop (RSMN)*. Redstone Arsenal, AL, USA, May 2007, 7 pages. URL: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.143.2168`.

[Wir] *IEC 62951 WirelessHART®System Engineering Guide*. HCF_LIT-161, Revision 1.0. HART Communication Foundation. Austin, TX, USA, May 2013. URL: `http://www.hartcomm.org/protocol/training/training_resources_wihart.html`.

[WN10] Q. Wu and S. Nandi. “Fast Single-Turn Sensitive Stator Interturn Fault Detection of Induction Machines Based on Positive- and Negative-Sequence Third Harmonic Components of Line Currents”. In: *IEEE Trans. Ind. Appl.* 46.3 (May 2010), pp. 974–983. ISSN: 0093-9994. DOI: `10.1109/TIA.2010.2045329`.

[Xia08] F. Xia. “QoS Challenges and Opportunities in Wireless Sensor/Actuator Networks”. In: *Sensors* 8.2, MDPI (Feb. 2008), pp. 1099–1110. DOI: `10.3390/s8021099`.

[Xila] *Embedded System Tools Reference Manual*. UG111 (v14.6). Xilinx. San Jose, CA, USA, June 2013. URL: `http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/est_rm.pdf`.

[Xilb] *LogiCORE IP Fast Fourier Transform v8.0*. DS808. Xilinx. San Jose, CA, USA, July 2012. URL: `http://www.xilinx.com/support/documentation/ip_documentation/ds808_xfft.pdf`.

[Xm1] *XM1000 Specifications*. Advanticsys. Madrid, Spain, 2011. URL: `http://www.advanticsys.com/shop/asxm1000-p-24.html`.

[Z1] *Z1 Datasheet*. v.1.1. Zolertia. Barcelona, Spain, Mar. 2010. URL: `http://zolertia.com/sites/default/files/Zolertia-Z1-Datasheet.pdf`.

[Zha+04] P. Zhang et al. “Hardware Design Experiences in ZebraNet”. In: *Proc. 2nd Int. Conference on Embedded Networked Sensor Syst. (SenSys)*. Baltimore, MD, USA: ACM, Nov. 2004, pp. 227–238. ISBN: 1-58113-879-2. DOI: `10.1145/1031495.1031522`.

[Zha11] G. Zhang. “Aviation Manufacturing Equipment Based WSN Security Monitoring System”. In: *Proc. 9th Int. Conf. on Reliability, Maintainability and Safety (ICRMS)*. Guiyang, China: IEEE, June 2011, pp. 499–503. DOI: `10.1109/ICRMS.2011.5979351`.

[Zha12b] P. Zhao. “Energy Harvesting Techniques for Autonomous WSNs/RFID with a Focus on RF Energy Harvesting”. PhD thesis. TU Darmstadt, 2012. URL: `http://tuprints.ulb.tu-darmstadt.de/3102/1/phd_diss_ZhaoPing.pdf`.

[Aer] Aeroflex Gaisler. *LEON3 Processor - Home page*. URL: `http://www.gaisler.com/index.php/products/processors/leon3`.

[Cen] Centro de Electrónica Industrial. *Cookies WSN - Home page*. Universidad Politécnica de Madrid. URL: `http://cookieswsn.wordpress.com/`.

[IEE03] IEEE Computer Society. *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements*. IEEE Std 802.15.3™-2003. Sept. 2003. URL: `http://standards.ieee.org/getieee802/download/802.15.3-2003.pdf`.

[IEE11] IEEE Computer Society. *IEEE Standard for Local and metropolitan area networks. Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Std 802.15.4™-2011. Sept. 2011. URL: `http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf`.

[IEE12] IEEE Computer Society. *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Std 802.11™-2012. Mar. 2012. URL: `http://standards.ieee.org/getieee802/download/802.11-2012.pdf`.

[Int05] International Technology Roadmap for Semiconductors. *Executive Summary*. Tech. rep. 2005 Edition. ITRS, 2005. URL: `http://www.itrs.net/Links/2005itrs/Execsum2005.pdf`.

[Int11] International Society of Automation (ISA). *ANSI ISA-100.11a-2011 Wireless Systems for Industrial Automation: Process Control and Related Applications*. 2011.

[Nat11] National Institute of Standards and Technology - Federal Information Processing Standards Publications. *Advanced Encryption Standard (AES)*. FIPS 197. Nov. 2011. URL: `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[Zig08] ZigBee Alliance. *ZigBee Specification*. Document 053474r17. Jan. 2008.

List of Own Publications

Journal Articles

[PG13c] F. Philipp and M. Glesner. “Low Power Reconfigurable Computing for Biomedical Signal Processing”. In: *Int. J. of Applied Biomedical Eng.* 6.1 (Dec. 2013), pp. 47–55. ISSN: 1906-4063. URL: `http://www.ijabme.org/images/stories/ijabme/2013/ijabme-6-n0-8-2013.pdf`.

[Pon+12] S. Pongyupinpanich et al. “Improvement of Standard and Non-Standard Floating-Point Operators”. In: *ECTI Trans. Comp. and Information Technology (ECTI-CIT)* 6.1, ECTI Association (May 2012), pp. 19–32. URL: `http://www.ecti-thailand.org/assets/papers/1279_pub_44.pdf`.

Book Chapter

[PG14c] F. Philipp and M. Glesner. “The Handbook of Electronic Medicine, Electronic Health, Telemedicine, Telehealth and Mobile Health”. In: ed. by H. Eren and J. G. Webster. CRC Press, 2014. Chap. Context-Aware Biomedical Smart Systems, 15 pages.

Conference Papers

[Bei+14] T. Bein et al. “Maintenance on Demand Concepts for Commercial Vehicles: The MoDe Project”. In: *Proc. Transport Research Arena Conference (TRA)*. Paper ID 17920. Paris, France, Apr. 2014, 10 pages. URL: `http://www.traconference.eu/papers/pdfs/TRA2014_Fpaper_17920.pdf`.

[Erd+12] D. Erdenechimeg et al. “Implementation and outcomes of FPGA-based system design in Mongolian education”. In: *Proc. 22nd Int. Conf. Field Programmable Logic and Applicat. (FPL)*. Oslo, Norway: IEEE, Sept. 2012, pp. 491–494. DOI: `10.1109/FPL.2012.6339181`.

[Mar+13] J. Martinez et al. “An Accurate and Fast Technique for Correcting Spectral Leakage in Motor Diagnosis”. In: *Proc. 9th IEEE Int. Symp. on Diagnostics for Electrical Machines, Power Electronics & Drives (SDEMPED)*. Valencia, Spain: IEEE, Aug. 2013, pp. 215–220. DOI: `10.1109/DEMPED.2013.6645719`.

[PG11a] F. Philipp and M. Glesner. “A Multi-level Reconfigurable Architecture for a Wireless Sensor Node Coprocessing Unit”. In: *Proc. IEEE Int. Symp. Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*. Anchorage, AK, USA: IEEE Computer Society, May 2011, pp. 334–337. DOI: `10.1109/IPDPS.2011.162`.

[PG11b] F. Philipp and M. Glesner. “Mechanisms and Architecture for the Dynamic Reconfiguration of an Advanced Wireless Sensor Node”. In: *Proc. 21st Int. Conf. Field Programmable Logic and Applicat. (FPL)*. Chania, Greece: IEEE, Sept. 2011, pp. 396–398. DOI: `10.1109/FPL.2011.78`.

[PG12] F. Philipp and M. Glesner. “(GECO)^2: A graphical tool for the generation of configuration bitstreams for a smart sensor interface based on a Coarse-Grained Dynamically Reconfigurable Architecture”. In: *Proc. 22nd Int. Conf. Field Programmable Logic and Applicat. (FPL)*. Oslo, Norway: IEEE, Sept. 2012, pp. 679–682. DOI: `10.1109/FPL.2012.6339176`.

[PG13a] F. Philipp and M. Glesner. “A Reconfigurable Wireless Platform for Biomedical Signal Processing”. In: *Proc. 6th Biomedical Eng. Int. Conf. (BMEiCON)*. Amphur Muang, Thailand: IEEE, Oct. 2013, 5 pages. DOI: `10.1109/BMEiCon.2013.6687692`.

[PG13b] F. Philipp and M. Glesner. “An Event-based Middleware for the Remote Management of Runtime Hardware Reconfiguration”. In: *Proc. 23rd Int. Conf. Field Programmable Logic and Applicat. (FPL)*. Porto, Portugal: IEEE, Sept. 2013, 4 pages. DOI: `10.1109/FPL.2013.6645578`.

[PG14b] F. Philipp and M. Glesner. “High-Level Abstraction for Teaching Embedded System Design with Modular Plug-and-Play Hardware”. In: *Proc. 10th Workshop on Microelectronics Education (EWME)*. Tallinn, Estonia: IEEE, 2014, 5 pages.

[Phi+12a] F. Philipp et al. “A Smart Wireless Sensor for the Diagnosis of Broken Bars in Induction Motors”. In: *Proc. 13th Biennial Baltic Electron. Conf. (BEC)*. Tallinn, Estonia: IEEE, Oct. 2012, pp. 119–122. DOI: `10.1109/BEC.2012.6376830`.

[Phi+12b] F. Philipp et al. “Adaptive Wireless Sensor Networks Powered by Hybrid Energy Harvesting for Environmental Monitoring”. In: *Proc. IEEE 6th Int. Conf. on Information and Automation for Sustainability (ICIAfS)*. Beijing, China: IEEE, Sept. 2012, pp. 285–289. DOI: `10.1109/ICIAFS.2012.6419918`.

[Phi+12c] F. Philipp et al. “Hardware acceleration of combined cipher and forward error correction for low-power wireless applications”. In: *Proc. 7th Int. Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. York, UK: IEEE, July 2012, 7 pages. DOI: `10.1109/ReCoSoC.2012.6322886`.

[PSG11a] F. Philipp, F. A. Samman, and M. Glesner. “Design of an Autonomous Platform for Distributed Sensing-Actuating Systems”. In: *Proc. 22nd IEEE Int. Symp. on Rapid System Prototyping (RSP)*. Karlsruhe, Germany: IEEE, May 2011, pp. 85–90. DOI: `10.1109/RSP.2011.5929980`.

[SPG11] F. A. Samman, F. Philipp, and M. Glesner. “Reconfigurable Interconnect Infrastructure for Multi-FPGA-based Adaptive Multiprocessing Systems”. In: *Proc. 1st Int. Workshop on Computing in Heterogeneous, Autonomous ’N’ Goal-Oriented Environments (CHANGE)*. Newport Beach, CA, USA: IEEE Computer Society, Mar. 2011, 8 pages. DOI: `10.1109/CHANGE.2011.6172451`.

[Tov+12] E. Tovar et al. “Networked Embedded Systems for Active Flow Control in Aircraft”. In: *Proc. 11th Int. Workshop on Real Time Networks (RTN)*. Pisa, Italy: Université de Toulouse - IRIT/INPT/ENSEEIHT, France, July 2012, pp. 18–23. URL: `http://irt.enseeiht.fr/scharbarg/rtn2012/ProcRTN12.pdf`.

Extended Abstracts

[GP13] M. Glesner and F. Philipp. “Embedded Systems Design for Smart System Integration”. In: *Proc. 13th Int. Symp. "Topical Problems in the Field of Electrical and Power Engineering" and "Doctoral School of Energy and Geotechnology II"*. Pärnu, Estonia: Tallinn University of Technology, Jan. 2013, p. 7. URL: `http://egdk.ttu.ee/files/parnu2013/Parnu_2013_007-007.pdf`.

[PG14a] F. Philipp and M. Glesner. “Enhancing Industrial Applications with Wireless Sensor Networks”. In: *Proc. 14th Int. Symp. "Topical Problems in the Field of Electrical and Power Engineering" and "Doctoral School of Energy and Geotechnology II"*. Pärnu, Estonia: Tallinn University of Technology, Jan. 2014, 1 page.

[Phi+11] F. Philipp et al. “Demonstration: Monitoring and Control of a Dynamically Reconfigurable Wireless Sensor Node Powered by Hybrid Energy Harvesting”. In: *Proc. Design, Automation & Test in Europe Conf. (DATE), University Booth*. Grenoble, France: EDAA, Mar. 2011, 1 page.

[PSG11b] F. Philipp, F. A. Samman, and M. Glesner. “Real-time Characterization of Noise Sources with Computationally Optimised Wireless Sensor Networks”. In: *Fortschritte der Akustik - DAGA 2011, Proc. 37th German Annual Conf. on Acoustics*. Düsseldorf, Germany: Deutsche Gesellschaft für Akustik e.V, Mar. 2011, pp. 713–714. ISBN: 978-3-939296-02-7. URL: `http://www.dega-akustik.de/publikationen/daga/daga_2011_inhalt.pdf`.

[PSG12] F. Philipp, F. A. Samman, and M. Glesner. “Simulation Environment For FPGA-Based Sensor-Actuators for Active Vibration Control”. In: *Fortschritte der Akustik - DAGA 2012, Proc. 38th German Annual Conf. on Acoustics*. Darmstadt, Germany: Deutsche Gesellschaft für Akustik e.V., Mar. 2012, pp. 735–736. ISBN: 978-3-939296-04-1. URL: `http://www.dega-akustik.de/publikationen/daga/daga_2012_inhalt.pdf`.

Supervised Theses

[Abd13] Haithem Ben Abdelkader. “Design of an FPGA Wavelet Analyzer for the Diagnosis of Broken Bars in Induction Motors”. Master Thesis. TU Darmstadt, Dec. 2013.

[Bah12] Ismail Bahri. “A Library of Signal Processing Algoritms for TmoteSky”. Internship. TU Darmstadt, July 2012.

[Bol11] Tobias Boll. “Modellierung des Energieverbrauchs eines drahtlosen Sensornetzwerkes”. Bachelor Thesis. TU Darmstadt, Apr. 2011.

[Bor11] Alexander Bornschein. “Hardwarearchitektur zur energieeffizienten Datenkompression”. Student Research Project. TU Darmstadt, Feb. 2011.

[Bor13] Alexander Bornschein. “An Evolvable, Time-Multiplexed, Functional Unit for DSP Applications”. Diploma Thesis. TU Darmstadt, June 2013.

[Ehr12] Daniel Ehrhard. “Performance Analysis of Simultaneous Data Transmission on Multiple Wireless Channels”. Bachelor Thesis (co-supervised by Faizal A. Samman). TU Darmstadt, Sept. 2012.

[ERA13] Tarek Yasser El-Rifai and Yara Mahmound Mohamed Abdelsayed. “A Smart Sensor for the Condition Monitoring of an Induction Motor”. Bachelor Thesis. TU Darmstadt, German University in Cairo, Aug. 2013.

[Fat12] Mohammed Reda Fathia. “Design and Implementation of Floating-Point Logarithmic and Exponent Functions”. Bachelor Thesis (co-supervised by Faizal A. Samman and Surapong Pongyupinpanich). TU Darmstadt, Sept. 2012.

[Gre14] Aude Gressier. “Handbewegungserkennung in Echtzeit mittels Implementierung von Klassifikatoren auf einem FPGA”. Master Thesis. TU Darmstadt, Mar. 2014.

[Isl13] Musfiqul Islam. “AMS Models for Application of Piezoelectric Patch Transducers”. Master Thesis. TU Darmstadt, Aug. 2013.

[Jun13] Lukas Johannes Jung. “Management of Orphaned Nodes in Wireless Sensor Networks Powered by Energy Harvesting”. Master Thesis (co-supervised by Saman Halgamuge (The University of Melbourne)). TU Darmstadt, The University of Melbourne, Apr. 2013.

[KA12] Fidel Sam Louis Kaldas and Ahmed Osama Hamed Aboudonia. “Online Gesture Classification on an EZ-430 Watch”. Internship. TU Darmstadt, German University in Cairo, Sept. 2012.

[Kha13] Salman Khalid. “Communication versus Computation Tradeoffs in Wireless Sensor Networks”. Master Thesis. TU Darmstadt, Apr. 2013.

[Kly11] Conrad Klytta. “Eine symmetrische Chiffre mit integrierter Kanalkodierung”. Master Thesis. TU Darmstadt, June 2011.

[Kur11] Anne Kurasiak. “Evaluation of a Multi-Source Energy Harvesting Circuit for a Wireless Sensor Node”. Internship. TU Darmstadt, INSA Strasbourg, July 2011.

[Man12a] Roman Mandryka. “Timerbasierte Schätzung des Energieverbrauchs eines System-on-Chip in Echtzeit”. Bachelor Thesis. TU Darmstadt, July 2012.

[Man12b] Antonio Jesús Moreno Mantas. “A Smart Wireless Sensor for Gestures to MIDI Conversion”. Master Thesis. TU Darmstadt, Sept. 2012.

[Mou13] Jude Ngepi Mouafo. “Energieeffiziente online Merkmalsextraction für einen tragbaren intelligenten drahtlosen Sensor”. Bachelor Thesis. TU Darmstadt, Feb. 2013.

[Mäd12] Sascha Mäder. “Modellierung und Simulation von heterogenen Systemen für Energy Harvesting Anwendungen”. Bachelor Thesis (co-supervised by Faizal A. Samman). TU Darmstadt, Apr. 2012.

[Mün13] Mathieu Münch. “Kanalzugriffsprotokolle mobiler drahtloser Kommunikationsnetze für den Einsatz in Notfallszenarien”. Diploma Thesis (co-supervised by Franck Scheidemann (MSA-Auer GmbH)). TU Darmstadt, July 2013.

[Naj11] Houdhiel Najjar. “Design and Evaluation of Power and Area Efficient FIR Filters based on Fixed-Point Arithmetic”. Diploma Thesis. TU Darmstadt, July 2011.

[Pér12] José Seguí Pérez. “A Resource-constrained Wireless MIDI Digital Synthesizer based on FPGA”. Master Thesis. TU Darmstadt, Nov. 2012.

[Rad13] Hauke Radtki. “Entwicklung eines inertialen, drahtlosen 9DOF Sensorknotens zur Erfassung und Auswertung von Trainingsdaten im Kraftsport”. Bachelor Thesis. TU Darmstadt, Dec. 2013.

[Rie11] Patrick Riedel. “FPGA-Based Configurable Interface for an Acceleration Sensor”. Bachelor Thesis. TU Darmstadt, Jan. 2011.

[Rüc13] Tobias Rückelt. “Development of a Transverse Feedback System for SIS18/100”. Master Thesis (co-supervised by Mouhammad Alhumaidi (Signal Processing Group)). TU Darmstadt, Aug. 2013.

[San12] Jorge Lopez Sanz. “VHDL-AMS Modeling and Simulation of a PMSM Control System for Automotive Applications”. Master Thesis (co-supervised by Faizal A. Samman). TU Darmstadt, Sept. 2012.

[Sei12] Mohamed Osama Ahmed Seif. “A Contiki Driver for USB Communication on HaLOEWEn”. Internship. TU Darmstadt, German University in Cairo, Sept. 2012.

[Yua12] Weichen Yuan. “A Wireless Sensor Network based on Adaptive TDMA for Structural Health Monitoring”. Master Thesis. TU Darmstadt, Jan. 2012.

[Zha12a] Weibin Zhang. “A Reconfigurable Functional Unit based on an ALU Array”. Master Thesis. TU Darmstadt, Jan. 2012.

Curriculum Vitae

Personal Data

Name	François Philipp
Date of birth	4. December 1985
Place of birth	Forbach, France

Education

09/1991–06/1996	Ecole du Creutzberg, Forbach, France
09/1996–06/2000	Collège Jean Moulin, Forbach, France
09/2000–06/2003	Lycée Jean Moulin, Forbach, France
09/2003–06/2005	Classes Préparatoires aux Grandes Ecoles, specialty *Mathematics and Physics*, Lycée Henri Poincaré, Nancy, France
09/2005–06/2007	Electronic Engineer Curriculum, Ecole Nationale Supérieure de l'Electronique et de ses Applications (ENSEA), Cergy, France
09/2007–03/2009	Double Diploma ENSEA – Technische Universität Darmstadt, Department of Electrical Engineering and Information Technology, specialty *Datentechnik*

Work Experience

04/2009–06/2014	Research and Teaching Staff, *Microelectronic Systems Research Group*, Technische Universität Darmstadt