SYSTEM-LEVEL METHODS FOR POWER AND ENERGY EFFICIENCY OF FPGA-BASED EMBEDDED SYSTEMS

PAWEŁ PIOTR CZAPSKI

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2010

To

My Parents

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my supervisor Associate Professor Andrzej Śluzek for his continuous interest, infinite patience, guidance, and constant encouragement that was motivating me during this research work. His vision and broad knowledge play an important role in the realization of the whole work.

I acknowledge gratefully possibility to conduct this research at the Intelligent Systems Centre, the place with excellent working environment.

I would also like to acknowledge the financial support that I received from the Nanyang Technological University and the Intelligent Systems Centre during my studies in Singapore.

Finally, I would like to acknowledge my parents and my best friend Maciej for a constant help in these though moments.

TABLE OF CONTENTS

LIST OF SYMBOLS

(The symbols are defined as below, unless specified in the context)

a	a certain additive value
α	device inactivity coefficient (describing dynamic power used in low switching activity period) of a certain device with a particular design implemented
αd	design inactivity coefficient (describing dynamic power used in low switching activity period) of a design (or a clock domain)
c	processing time (in clock cycles)
cc	amount of clock cycles required to process data
c_x	processing time (in clock cycles) of clock domain x
c_y	processing time (in clock cycles) of clock domain y
C_i	capacitance of resource i
Δf_x	frequency change of clock domain x
Δf_y	frequency change of clock domain y
Δt	overall execution time change due to frequency changes Δf_x and Δf_y
D_x	denotes clock domain x
D_y	denotes clock domain y
E	total energy consumption
E_{bit}	energy required to send 1 data bit
$E_{process}$	processing energy
E_{send}	sending energy
E_{total}	total energy (spent on processing and sending data)
f	clock frequency
f_i	clock frequency of resource i
f_x	clock frequency of clock domain x
f_y	clock frequency of clock domain y

hwa	hardware area (or amount of the equivalent NAND gates)
h_x	hardware resources (system-level equivalent) used by clock domain x
h_y	hardware resources (system-level equivalent) used by clock domain y
H_U	hardware utilization coefficient
$INDATAWIDTH$	input data bitwidth
k	low-level-to-system-level dynamic power consumption coefficient
n	multiply factor
$OUTDATAWIDTH$	output data bitwidth
$OUTVOL$	amount of communicated data
p_i	estimated dynamic power consumption for design with a certain circuit replicated i times
P	total dynamic power consumption
P_{avg}	average dynamic power consumption
P_{total}	total processing power (based on hwa and cc)
P_{DA}	dynamic power consumption of working design (in simplified form)
P_{DU}	dynamic power consumption of inactive design (in simplified form)
$SAMPLELENGTH$	total length of processed data sample
$SAMPLELENGTH\_LCL$	length of data processed locally
S_i	switching activity of resource i
S_{AD}	average switching activity of working design
S_{UD}	switching activity (averaged over the design area) during inactivity period
S_{uncf}	switching activity averaged over the whole area of unused part of FPGA
t	total time of the whole algorithm execution or its part
t_{exec}	data processing execution time
t_x	execution time of algorithm part in clock domain x

t_y	execution time of algorithm part in clock domain *y*
T	a certain execution time
U_i	utilization of resource *i*
V	voltage
V_{dd}	power supply voltage
V_i	voltage swing of resource *i*
V_{th}	threshold voltage
x	denotes clock domain *x* or its part
X	denotes (also in figures) clock domain *x* or its part
y	denotes clock domain *y* or its part
Y	denotes (also in figures) clock domain *y* or its part

LIST OF ABBREVIATIONS

(The abbreviations are defined as below, unless specified in the context)

ADC	Analog-to-Digital Converter
ASIC	Application-Specific Integrated Circuit
ASP	Application Specific Processor
ASSP	Application-Specific Standard Product
BWT	Burrow-Wheeler Transform
BZIP2	Basic Zip with Modifications (data-reduction algorithm)
CAD	Computer-Aided Design
CFAR	Constant False Alarm Rate (e.g. CFAR detector)
CFDF	Clock Frequency Division Factor
CLB	Configurable Logic Block
DAC	Digital-to-Analog Converter
Double	wire that travels two CLB's
DSP	Digital Signal Processor
EDIF	Electronic Design Interchange Format
EEPROM	Electrically Erasable Programmable Read-Only Memory
EWMA	Exponentially Weighted Moving Average (e.g. EWMA filter)
FF	Flip-Flop
FIR	Finite Impulse Response (e.g. FIR filter)
FLASH	Type of EEPROM
FPGA	Field-Programmable Gate Array
GPP	General Purpose Processor
HDL	Hardware Description Language
Hex	wire that travels six CLB's
HLL	High-Level Language
ID	Identification Data
IIR	Infinite Impulse Response (e.g. IIR filter)
IOB	Input-Output Block
IP	Intellectual Property (e.g. IP core)
ISA	Instruction Set Architecture
ISE	Integrated Software Environment

IXbar	Input Crossbar – switch connecting wire segment to CLB input
Long	wire that travels the length of FPGA chip (in vertical and horizontal dimensions)
LUT	Lookup Table
LZO	Lempel-Ziv-Oberhumer (data-reduction algorithm)
LZW	Lempel-Ziv-Welch (data-reduction algorithm)
MCU	Microcontroller
NAND	logical operator that consists of logical AND followed by logical NOT returning false value only if both operands are true (e.g. NAND gate – logic gate that simulates the function of the logical operator NAND)
NDU	Non-Descriptive Unit (e.g. a certain NDU amount of power or energy)
NRE	Non-Recurring Engineering (e.g. NRE cost)
OTP	One-Time Programmable (e.g. OTP device)
OXbar	Output Crossbar – switch connecting wire segment to CLB output
par	Handel-C keyword directing instructions to be executed in parallel
PCB	Printed Circuit Board
PIR	Passive Infrared
PPMd	Prediction by Partial Matching with Modifications (data-reduction algorithm)
RLE	Run-Length Encoding (data-reduction algorithm)
RTL	Register Transfer Level
SMA	Simple Moving Average (e.g. SMA filter)
SOPC	System-on-Programmable-Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSR	Small-Scale Reconfigurability
ST	Structured Transpose
TTM	Time to Market
UART	Universal Asynchronous Receiver-Transmitter
WSN	Wireless Sensor Network

WT	Wavelet Transform
ZRL	The Zurich Research Laboratory

LIST OF FIGURES

LIST OF TABLES

Abstract

Field programmable gate array (FPGA) processing units present considerably higher programming flexibility than other fixed architectures (e.g. microcontrollers (MCU's), digital signal processors (DSP's)). Although performances of FPGA are often compared to application-specific integrated circuits (ASIC's), the price for such a flexibility of programmable devices is a significantly higher power consumption, compared to other fixed-architecture processors.

Power consumption of FPGA implementations can be reduced at the low-level of design. However, for designs of moderate and high complexity such low-level approaches are tedious to implement and time-consuming. High (system) levels of design (e.g. algorithmic languages such as Handel-C) allow building systems of significantly higher complexity. Unfortunately, high-level design techniques have a limited (or no at all) ability to control power/energy properties of a design. **The objective of our work is, therefore, to investigate the system-level approaches to power (and energy) efficiency of FPGA-based devices**.

FPGA's dissipate static and dynamic power. However, only the dynamic power consumption is design-dependent, while static power consumption is mainly technology-dependent. Thus, we generally ignore the issues of static power reduction in the presented results.

First, we show that power and energy properties of FPGA-based designs can be estimated with a reasonable precision at the high level of designing process. Moreover, we show that the system-level partitioning of designs into several clock domains (typically used to improve performance only) does not noticeably affect power consumption and hardware resources compared to the equivalent low-level partitioning. These two observations are the foundations of further experiments on system-level approaches to power and energy efficiency.

We separately analyze the system-level parallel and sequential algorithm partitioning (in both cases employing the concept of multi-clock domains). It is shown that parallel algorithm partitioning can be optimized (by exploiting system-level estimates of domain sizes and timing) to provide substantial power consumption savings. Sequential partitioning was found a less efficient tool for reducing power and energy consumption of designs. However, we found that in sequentially partitioned designs power consumption losses can be minimized by selecting proper clock frequencies of a particular domain, if for certain reasons the domains must be run at diversified frequencies (which generally dramatically increases the overall energy usage).

Finally, we analyze the total consumption of data-processing and data-transmission energies in FPGA-based designs (which is a typical problem for wireless sensor network (WSN) applications). In general, hardware requirements (i.e. power and energy) of data processing algorithms grow proportionally to the amount of data processed concurrently, while the energy required for transmission is proportional to the volume of transmitted data. We show that by combining system-level algorithms properties and characteristics of transmission modules, substantial savings of the overall energy are achievable.

We believe that the proposed solutions will lead to more advanced system-level approaches to power and energy efficiency, i.e. development of tools incorporating low-level power and energy characteristics into high-level design methodologies. Such tools would have the ability to control low-level characteristics (e.g. power and energy consumption) of FPGA-based designs from the highest levels of abstraction.

CHAPTER I
INTRODUCTION

This chapter is a general introduction to our work on the system-level methods for power and energy efficiency of the programmable logic-based embedded systems. First, in Section 1.1, we define ubiquitous computing. Then, in Section 1.2, we introduce one of the most important (for this work) examples of ubiquitous computing, i.e. the wireless sensor network (WSN). In Section 1.3 field-programmable gate arrays (FPGA's) devices and microcontrollers (MCU's) are discussed from the WSN's perspective, i.e. their applications, advantages, and existing drawbacks. Finally, in Section 1.4, we present the scope, objectives and organization of the thesis.

1.1. Introduction

The term ubiquitous computing appeared at the beginning of the 90's and was introduced by Mark Weiser (1952-1999; widely considered to be the father of ubiquitous computing), [1], [2]. In his most cited quotation [1] and [3] he stated that "*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it*". Since then, there was a dramatic increase in the use of companion and embedded devices in the last decade. Computers are no longer stand-alone special-purpose machines operated by the experts only. Instead, they are present in networked environments ubiquitously.

In general, a ubiquitous environment is a collection of embedded, wearable, and handheld devices, wirelessly connected (possibly to a fixed network infrastructure, e.g. internet), [2], [4]. They have to be aware of their surroundings and be capable to provide services to and use services from other parties effectively, [5].

Although some scientists find ubiquitous and pervasive computing as separate terms, they agree that these words are almost synonyms, [1], [2]. If not, then pervasive computing refers to systems of small and mobile devices that are used for retrieving information anytime, anywhere, and on any device, [2], [6]. Moreover, the goal of ubiquitous computing is to hide computer architecture, while the pervasive computing is to create a technology that can invisibly assimilate into our everyday life, [2].

Requirements to ubiquitous computing are the effect of progress in distributed computing, [5]. Distributed computing requires remote communication, fault tolerance,

high availability, remote information access, and distributed security. Key issues in mobile computing are the cross product of the above issues with new ones, such as mobile networking, mobile information access, adaptive applications, energy awareness, and location sensitivity. Subsequently, key issues in ubiquitous computing are the cross product of the issues in mobile computing and new issues such as smart spaces, invisibility, localized scalability, and uneven conditioning of environments.

1.2. Wireless sensor networks

Wireless sensor networks (WSN's) are an example of ubiquitous computing. Their development started from military applications (Smart Dust, Net), [7]. Initially, WSN's were defined as large-scale, wireless, ad-hoc, multi-hop, unpartitioned networks of homogenous, tiny, and immobile sensor nodes. However, development of WSN's in the area of civilian applications (environmental and species monitoring, agriculture, industry, healthcare, etc.) have shown that such a definition is inaccurate, [7]. WSN's can be heterogeneous, mobile, with different network topologies, and may use existing network infrastructures. More generally, WSN's can be defined as networks of nodes with communication, computation, sensing, and even actuation abilities, [8], [9].

A sensor node may consist of a power, a sensing, a processing, a communication, and even an actuation unit, [8], [9]. The power unit is often a rechargeable battery, usually non-replaceable in the field, equipped with a voltage converter. Power harvesting techniques can be alternatively used. However, their low efficiency limits applicability, [7], [8], [9]. The sensing unit provides the processing and/or communication units with relevant signal data (analog and digital signals can be encountered), [8]. The processing unit, usually a microcontroller (MCU) (e.g. Crossbow WSN development platforms) and seldom a field-programmable gate array (FPGA), processes and passes data to a network, and may also perform some networking tasks, [10], [11], [12], [13], [14], [15]. Depending on the node concept, data can be aggregated, passed, or routed, [9], [14]. The communication unit is responsible for transmitting and receiving data. Some of the networking tasks may be embedded in the communication unit, especially if data has to be only routed without any prior processing, [13].

1.3. FPGA and MCU in WSN applications

1.3.1. Typical applications

To our knowledge, there are only few WSN (or related to WSN's) applications employing FPGA. Moreover, this programmable logic device is rather used as a supporting processing unit to the main processor, e.g. MCU.

Tyndal National Institute is developing a miniaturized modular platform for WSN's, [11]. This platform is build with an idea of stackable layers, each designed for a particular purpose, i.e. providing power to a device, sensing, processing, and communication. A layer equipped with FPGA is used for high-speed processing (computationally intensive signal processing, intelligent functionalities). Moreover, performing all tasks locally is possible by the relevant FPGA reconfiguration[1] that may also reduce wirelessly transmitted data. Employing FPGA was motivated by its computation capabilities and significant number of programmable inputs and outputs that may be used to control sensors and actuators, [10].

The Zurich Research Laboratory (ZRL) wireless sensor networking test bed consists of sensor units, a wireless network, a gateway connecting wireless sensors to the host environment, middleware supporting sensor data distribution to sensor applications, and sensor applications, [12]. Sensor units are equipped with FPGA that collects the sensors data, assembles the data into data frames, and sends the resulting data frames to the radio module.

In [13], the feasibility of networking functionality migration from a sensor unit to a radio unit is investigated. The radio unit is equipped with FPGA. However, the most computationally demanding tasks are performed by MCU located in the sensor unit.

Hybrid fine-grained integration of the fixed and reconfigurable logic is presented in [14]. The concept of small-scale reconfigurability (SSR) and optimum combination of such a hybrid logic system is investigated for power efficient adaptable WSN.

FPGA as a complete solution for a reconfigurable wireless communication system for small spacecraft is investigated in [16]. Such a programmable device is envisaged to incorporate all major communication and networking functionalities. Spacecraft wireless networks are not highly active. Therefore, it is envisaged that FPGA may replace other

[1] Swapping between different algorithms while having small FPGA footprint (not enough to accommodate all algorithms) so reducing power consumption.

computation components such as general purpose processors (GPP's), digital signal processors (DSP's), application-specific standard products (ASSP's), and application-specific integrated circuits (ASIC's).

1.3.2. Comparative analysis

A) FPGA advantages

Undoubtedly, the most important advantage of FPGA is reconfigurability, [16]. The same processing unit may be used for different functionalities. Reconfigurability may also be used for adaptability of a particular device to changing environmental conditions and network topologies.

FPGA has a number of user programmable inputs and outputs that help when interconnecting a significant number of different devices, e.g. sensors, actuators, [11]. FPGA designs do not require complex fabrication process compared to ASIC-based, [16]. This allows for low production volumes. Moreover, FPGA architecture allows for post-production configuration of the final product and even changes leading to the next product generation. Therefore, programmable devices are often used for prototyping, evaluation, and development of fixed-logic designs, [17]. Although the performance of FPGA is close to ASIC, they are far away in the field of power and energy efficiency. FPGA devices are significantly inferior, mainly because of additional hardware resources (switching transistors and configuration memory) needed to maintain reconfigurability.

B) FPGA Disadvantages

There are four general types of FPGA, i.e. static random access memory (SRAM)-based FPGA, electrically erasable programmable read-only memory (EEPROM)-based FPGA, FLASH-based FPGA, and Antifuse-based FPGA (does not support reconfiguration) [18].

SRAM-based FPGA is manufactured in a deep submicron technology due to required logic density. Hence, its power demands are significant, [8], [11], [19]. Moreover, noticeable amount of power is wasted in a standby and due to current leakages (caused by SRAM memory cells and switch transistors increasing the number of active elements). SRAM-based FPGA contains the reprogrammable device and the boot ROM

holding the configuration data. However, loading a configuration data at the power up requires significant amount of time.

EEPROM-based FPGA does not need boot ROM, and FLASH-based devices have smaller configuration memory compared to first one.

Antifuse-based FPGA does not require switch transistors or configuration memory, hence, the only required power is for the logic array. Therefore, the standby current is reduced significantly. However, this type of FPGA is only a one-time programmable (OTP) device.

C) MCU advantages

MCU's used in WSN applications are often equipped with various types of memory such as volatile and non-volatile, interfaces such as serial peripheral interfaces (SPI's), universal asynchronous receiver-transmitters (UART's), analog-to-digital and digital-to-analog converters (ADC's), (DAC's), counters, and timers, [8].

MCU's are also equipped with advanced power management circuits (various low power modes) and allow for diversified voltage and clock speed of the core, [8], [11], [14], [17], [19], [20].

Their specific instruction set architecture (ISA) allows them to perform complex signal processing computations, [13]. Moreover, some microcontrollers support multi-tasking, so implementing a simple operation system such as TinyOS is possible, [9].

D) MCU Disadvantages

MCU's are a type of application specific processor (ASP). They are designed with a specific ISA and data-path, that is, to perform efficiently relevant operations only. Moreover, they have considerably less inputs and outputs compared to FPGA's, [11].

1.4. Scope, objectives, and thesis organization

1.4.1. Scope

WSN's are a considerably new field of science. However, growing application diversity and the corresponding requirements significantly increase the pace of WSN's

development. Demands are also put on reducing non-recurring engineering (NRE) costs and time to market (TTM). Moreover, such devices are expected to be hardly visible and long lasting, which further complicates their development. Altogether, complexity of sensor nodes pushes their development from the low-(hardware) level to higher levels of abstraction.

Demands for flexibility, performance, longevity, and cost-related issues put on ubiquitous devices, suggest the usage of programmable logic devices, e.g. FPGA, as the processing unit. Technology advancements in the area of FPGA allow manufacturing of multimillion-gate devices, e.g. Altera FPGA chips, Xilinx FPGA chips. This fact additionally increases attention paid to reconfigurable architectures as the processing units, e.g. software-based processors (LatticeMico, Nios, MicroBlaze, PicoBlaze, XTensa), [21], [22], [23], [24]. Moreover, technology advancements are also observed in the area of high-level design techniques, e.g. compilers (Quartus, ISE), hardware description languages (Verilog, VHDL), and high-level hardware description languages such as Handel-C (named algorithmic HDL or algorithmic languages for clarity in the thesis), [21], [22], [25]. They allow for synthesis and prototyping of processing units in a relatively short time, skipping tedious low-level design techniques (with some additional power and hardware resources overheads, however), [21], [22], [25].

Undoubtedly, hardware description languages (HDL's) and the associated programming environments are already matured. Well established manufactures like Altera and Xilinx provide consumers with their programmable logic devices as well as with complete development environments, [21], [22]. Although the development processes using HDL's are considerably faster than transistor-level techniques, high-level HDL's (i.e. the system-level techniques) may shorten this process additionally. However, such languages, e.g. Handel-C, and the corresponding development environments, e.g. DK Celoxica, are still in their development stage.

Numerous academic groups and commercial vendors have attempted to create tools that convert a high-level language (HLL) into a HDL representation for register transfer level (RTL) synthesis (often referred to as 'C to HDL' or 'C to RTL' methodology) targeting either Verilog or VHDL HDL languages, [26]. Existing HLL's (allowing targeting FPGA at an algorithm level) are usually modifications of C or C++ programming languages, [26], [27], [28]. Since standard C lacks notations typical to hardware, such as parallelism or the passage of time, the concepts are introduced, either as extensions to the relevant language (or via 'pragmas') or are built into the tool and

annotated by a programmer against the standard C description. In effect, these allow a designer to use such a C-like programming language as a HDL. However, the differences of these C-like languages compared to standard C/C++ prevent simulation of the code outside specialized simulators. The range of tools (and associated C-like languages) that allow programming FPGA's is surprising, see Table 1, just to name a few.

Table 1. Tools (supporting C-like languages) for FPGA design.

Toolset	Vendor
NISC toolset [29]	University of California
Altium Designer [30]	Altium
Catapult [31]	Mentor Graphics
Cynthesizer [32]	Forte Design Systems
Agility Compiler [33]	Celoxica
DK Design Suite [34]	Agility Design Solutions (former Celoxica)
DIMEtalk [35]	Nallatech
Impulse C [36]	Impulse Accelerated Technologies
FpgaC [37]	an open source initiative
SA-C [38]	Colorado State University
Cascade [39]	CriticalBlue
Mitrion [40]	Mitrionics
C2R Compiler [41]	CebaTech
Mimosys Clarity [42]	Mimosys
HybridThreads Compiler [43]	University of Kansas

However, these high-level HDL's and the accompanying tools are focused on parallelization approaches, rather than abstraction, and lack the relevant development board libraries. To our knowledge, Agility Design Solutions (former Celoxica) is the only vendor providing complete high-level HDL programming environment, proprietary C-like language (Handel-C), and a number of development boards with platform abstract layer libraries. That is, DK Design Suite and RC development boards are often used by academia.

1.4.2. Objectives

We have observed that there is lack of system-level approaches to power and energy efficient development of programmable logic based designs. Although Handel-C (exemplar algorithmic HDL) allows for almost the same flexibility as C languages, it lacks the power and energy management schemes. Hence, the only way to achieve better power and energy efficiency of relevant designs is to deal with the design levels below the system level, that is, with the hardware levels. Therefore, we come up with several

approaches that allow achieving, to some extent, power and energy efficiency at the system-level.

Dividing a particular design into multi-clock domains is a well established technique to achieve better performances in the programmable devices arena. Although this operation is rather hardware-level oriented, algorithmic HDL's, such as Handel-C, allow for implementing a particular algorithm into various clock domains. Power and energy characteristics of such a multi-clock domain design partitioned at the hardware-level are easily recognizable. That is, performing such an operation carefully should not change power and energy characteristics of a design (assuming a similar number of hardware resources are employed). However, dividing a design into a number of clock domains at the system level is challenging since algorithmic HDL synthesis is focused on parallelization rather than on implemented logic, that is, on performance rather than on power and energy efficiency. It is also unknown how such a design partitioned at the system-level will behave in terms of hardware resources and power consumption. In general, the research approaches in our work to the system-level power and energy efficiency are based on multi-clock domain designs. Thus, the first of the objectives of the thesis is to investigate:

(i) **Relations between domains, i.e. clock domains, size of a domain, size of a design, power consumption**.

We investigate to what extent design partitioning at the system-level (using algorithmic language constructs) is as predictable as at the hardware-level in terms of hardware resources and power consumption. In other words, we verify whether dividing a design into multiple clock domains can be performed directly at the system-level.

Following the general thesis direction, i.e. the multi-clock domain approach, we subsequently investigate the ability of system-level algorithm partitionings to achieve power and energy efficiency. It is envisaged that some groups of (subject to partitioning) algorithms perform selected operations in parallel and/or sequentially (sometimes interchangeably). Therefore, the next research area of the thesis is:

(ii) **Issues of parallel and sequential algorithm partitioning**.

Data communication (receiving as well as transmitting) is found the most power and energy consuming operation among other operations performed by an embedded device. Thus, another objective of the thesis is to investigate:

(iii) **Energy efficiency of data communication from the perspective of data processing algorithms**.

The results presented in the thesis indicate that the proposed techniques give satisfactory results with acceptable power, energy, and hardware resources overheads. It is envisaged that with little effort our approaches can be integrated into lower levels of the design process, i.e. hardware description environments, allowing a further shortening of the development process.

1.4.3. Thesis organization

The thesis is divided into several chapters, each corresponding to a major topic of our work. In the beginning of each chapter a brief overview of its content is given, and the chapters are summarized with general conclusions.

In Chapter 2 we survey the literature related to the scope and objectives of our work. First, in Section 2.1, we present power and energy issues in FPGA chips themselves and in FPGA-based designs. Next, in Section 2.2, we survey sources on data processing in WSN's. We discuss sensing principles, sensor selection for WSN applications, application requirements to sensing devices, and data processing algorithms. However, the latter is only a brief introduction to the algorithms further analyzed in other parts of the thesis. In general, we do not analyze algorithm structures and how they perform since this is out of the thesis scope. In Section 2.3, we survey data-reduction algorithms used in WSN's. Finally, in Section 2.4, existing approaches to algorithm partitioning in FPGA designs are discussed.

In Chapter 3 the experimental setup is described. We overview high- and low-level development tools (programming environment, hardware targeting, etc.) used for the conducted experiments, and give general assumptions regarding the experiments. In this chapter, we also give a brief introduction to the high-level hardware description language, i.e. Handel-C, used in our experiments for the implementations of the algorithms.

Chapter 4 discusses relations between system- and low-level results of the algorithms implementations. In other words, we investigate whether system-level results can, at least to some extent, represent similar experiments performed at the hardware-level. Using this approach, tedious low-level implementations can be avoided by performing qualitatively the same experiments at higher levels of the design process. The results of these experiments are the basis for further system-level experiments.

In Chapter 5 we investigate relations between clock domains, size of a design, chip area constraints, and power consumption, i.e. what clock frequency is suitable for a particular design size in terms of power consumption, and how dividing designs into several clock domains influences power consumption. We also investigate dependencies between low-level design integration (a multi-clock domain design) and power consumption predictability at the system-level.

Chapter 6 presents experiments on a parallel algorithm partitioning. We base our investigations on selected data-reduction algorithms used in WSN applications. Parallel algorithm partitioning is discussed from the perspective of the power efficiency improvement.

In Chapter 7, a sequential algorithm partitioning, based on selected typical data processing algorithms used in WSN's, is investigated. We prove that selection of a clock frequency to such a sequentially partitioned algorithm (e.g. for a desired performance increase) must be performed extremely carefully. Otherwise, power and energy efficiency of a design may be strongly sacrificed. Although the chapter focuses on determining optimum domain clock frequencies in multi-clock domains designs rather than on improving power and energy efficiency, we show that the selection of clock frequencies influences power and energy properties of the design.

Chapter 8 approaches data processing and communicating issues in terms of energy consumption. The conducted experiments focus on the hardware resources required to implement the relevant data processing logic and the data volume to be communicated. In effect, significant energy efficiency improvement is approached.

In Chapter 9 we conclude our work and present feasible future works.

CHAPTER II
LITERATURE OVERVIEW

This chapter is the survey of the literature related to the scope and objectives of our work. In Section 2.1 we present power and energy issues in programmable devices, i.e. FPGA chips themselves, and in FPGA-based designs. However, the problems are addressed from the system-level view point, i.e. power and energy issues are not deeply studied at the hardware-level (transistor-level). In the following sections we present issues related to WSN's, which we considered a leading example of embedded systems. We chose surveillance applications as the typical application of WSN's. First, in Section 2.2, we survey sources on data processing in WSN's. Although the main focus is on algorithms, it is envisaged that other issues related to sensing are important to understand the subject. Therefore, issues like sensing principles, sensor selection for WSN applications, noise issues in sensing devices, and finally data processing algorithms, are discussed. Data processing algorithms are only briefly presented since the detailed analysis can be found in other parts of the thesis. Nevertheless, algorithm structures and how they perform are not analyzed at all since this is out of the thesis scope. In Section 2.3 we survey data-reduction algorithms used in WSN applications. Finally, existing approaches to algorithm partitioning in FPGA designs are discussed in Section 2.4.

2.1. Power and energy issues in FPGA-based designs

Power and energy efficiency may be analyzed at each of the design levels, i.e. ranging from a transistor-level to a system-level, [44]. Designing at the device-level, covering power issues at the circuit- and transistor-levels, was found the most important issue in the past decade. The device-level experts (circuits and layout) were responsible for power and energy, while the system-level experts (architectures, compilers, and operating systems) were designing for speed and performance. In recent years, however, there has been a growing interest in power and energy issues analyzed at higher levels (i.e. system-levels), e.g. [44], [45], [46], [47].

The basic building block of the current processors is a CMOS circuit, [44], [47], [48]. Each technology advancement in circuits that shrinks the transistor feature size by a factor of n, reduces the capacitance by n, and lowers the supply voltage by n^2, should reduce the dissipated power by a factor of n^3 assuming the same clock frequency.

However, new processor generations result in more complex and more performance-demanding designs. They use a higher clock frequency, a larger chip area, and more transistors, [44], [49]. In effect, there is a significant increase in the power dissipation and power density. Finally, power management policies aiming at the device-level become insufficient. Therefore, power issues of current processors become a key design constraint and propagate to higher levels of design, [44], [45], [46], [47].

2.1.1. Power and energy issues in design

The key to a proper design is to understand the conceptual difference between power-aware and low-power systems, [44]. The main goal in designing low-power systems is power minimization, while power-aware systems are designed to achieve particular power and energy properties.

- **Power-aware design *versus* power/energy minimization**

Decreasing power and energy does not have to be the main concern in power-aware designs. Actually, power and/or energy may even be increased. An example is the issue of decreasing the peak power in a processor. Typically, this is achieved by using a scheme that intentionally delays execution of some instructions to smoothen their distribution, so decreasing the peak of consumed power. However, such an approach may increase the execution time, and thus increase the energy consumption. Hence, this scheme is generally not suited for low-power designs.

- **Average power and maximum power**

Decreasing the average power does not have to reduce the maximum power. The average power dissipation, representing the power consumption distribution histogram, is computed over the entire execution time while the maximum power represents the peak value of such a histogram. Thus, decreasing the average power may sometimes increase the maximum power. Hence, these approaches are also not suitable to low-power designs.

- **Power efficiency and energy efficiency**

The integral of the power consumption over the execution time represents energy. It is well known that an improved power efficiency of a design may be obtained by decreasing the clock frequency and/or by reducing the power supply voltage of the

processor. However, this may degrade the performance of such a design, increasing the execution time, thus effectively increasing the energy consumption.

- **Power-constrained and energy-constrained design**

An energy-constrained design is one that is running under the constraints of a finite source of energy such as a battery. On the contrary, a power-constrained design is running on the infinite source of energy such as a solar battery. However, such an energy source is constrained by its power efficiency. Hence, the energy budget and the available power are totally different designing metrics.

- **Energy-constrained design and energy minimization**

Studies on batteries show that their properties are far from ideal capacitors, and the battery charge depends on other than capacity issues. Hence, the energy-constrained designs are focused on battery lifetime that does not correspond to energy-minimization.

2.1.2. Power consumption in FPGA

Devices fabricated in CMOS technology (e.g. FPGA) dissipate static and dynamic power, e.g. [44], [46], [47], [48], [50], [51].

- **Static power**

The leakage current between power supply and ground is the main source of the static power and includes the reverse biased PN-junction current, the sub-threshold leakage, the gate induced drain leakage, the punch through, and the gate tunnelling, [44]. The sub-threshold leakage current (that depends on temperature and the threshold voltage V_{th}) constitutes the majority of the leakage current.

In the past, a negligible level of the leakage current was the reason of not taking it into power consumption analysis. However, a significant static power increase can be expected due to the shrinking transistor size. Feature size decrease is generally accompanied by a reduction of the power supply voltage (V_{dd}). Hence, the threshold voltage has to be reduced to maintain or increase the performance. Eventually, the static power increases significantly since the sub-threshold leakage current grows exponentially with the threshold voltage decrease. The static power is independent of the device

activity, but it depends on the device area and temperature. Thus, the static power is present whenever a power is supplied to the CMOS device.

In addition, the thermal characteristic is also affected by the design shrinking. On-chip temperature may vary across the whole chip area. Its maximum values depend on the chip area and the maximum power dissipation. Therefore, by reducing the total maximum power the maximum temperature may be decreased, so influencing the static power.

It is claimed by some researchers, [50], that the static power of a typical FPGA device, e.g. Virtex-II family (SRAM-based FPGA, 0.15µm technology), is in the range of 5 up to 20% of the overall dissipated power (depending on the temperature, the clock frequency, and the implemented logic).

- **Dynamic power**

In a CMOS device, signal transitions at their transistors are the source of dynamic power dissipation, [44], [46], [47]. Frequencies of these transitions are obviously related to the clock frequency, i.e. the dynamic power consumption is generally modelled as:

$$P = \sum_i C_i \cdot V_i^2 \cdot f_i \qquad (1)$$

where C_i, V_i, and f_i, represent the capacitance, the voltage swing, and the clock frequency of the resource *i*, respectively, [44], [47], [50], [51]. The total dynamic power is the sum of the dynamic power of all resources.

FPGA programmability introduces additional design-dependent factors contributing to the dynamic power: the effective capacitance of resources, the resource utilization, and the switching activity of resources, [45], [50], [51].

The effective capacitance is the sum of original capacitance of the components and of parasitic effects caused by interconnection wires and transistors. The resource utilization represents the amount of resources that are not used after chip configuration. The average number of signal transitions in a clock cycle is represented by the switching activity. This generally depends on several factors, e.g. input signal patterns. Therefore, (1) can be expressed by:

$$P = V^2 \cdot f \cdot \sum_i C_i \cdot U_i \cdot S_i \qquad (2)$$

where *V* is the supply voltage, *f* is the clock frequency, and C_i, U_i, and S_i, represent the effective capacitance, the resource utilization, and the switching activity of each resource, respectively, [50].

2.1.3. Power characteristics of FPGA

A typical FPGA chip consists of (apart from its main array of slices and I/O blocks) a number of hard cores, i.e. memory blocks, digital clock managers, encryption circuits, custom multipliers, etc., [50].

An FPGA's power and performance are often compared to their ASIC counterparts, [50], [51]. However, the programmability of FPGA needs the interconnection structures with loading larger than for custom circuits, [52]. Moreover, the capacity load of signal nets over dedicated metal wires is additionally increased by signal buffers, pass transistors, and other programmable switching structures.

Such a flexibility of programmable devices compared to other processing units, with almost fixed architecture, leads to significant power consumption increase.

A) FPGA architecture – programmable fabrics (Virtex-II FPGA family chip example)

Virtex-II, which can be considered a typical FPGA chip (see Figure 1) consists of configurable logic blocks (CLB's) that are interconnected by a number of routing resources, [50]. Each CLB contains four slices, also referred as logic, where each slice consists of two 4-input lookup tables (LUT's), two flip-flops (FF's), and a diversity of dedicated circuits that accommodates more efficient implementations of some specific logic.

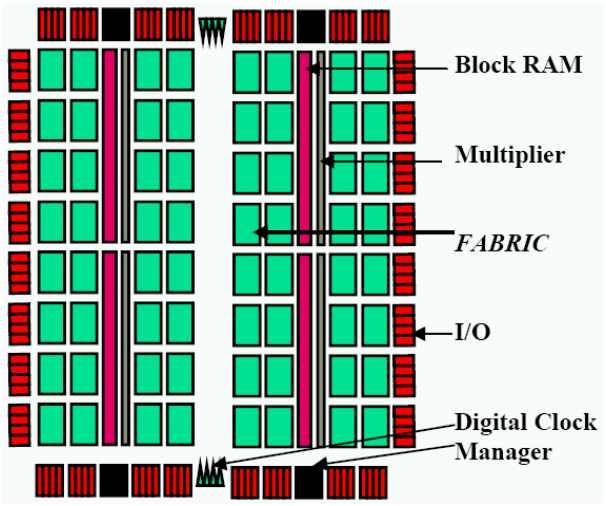

Figure 1. Architecture of Virtex-II FPGA chip, [50].

Virtex-II uses a segmented routing structure that minimizes the number of transistors and wires traversed by a signal reaching its destination. This routing architecture includes wires that travel along two CLB's (Double's), six CLB's (Hex'es), and the length of FPGA chip (Long's), in both the vertical and horizontal dimensions. There are also pass transistors and buffers associated with each set of wires. Moreover, there are two sets of switches that connect wire segments to inputs and outputs of each CLB, called input crossbars (IXbar's) and output crossbars (OXbar's), respectively. These interconnection resources add a significant power consumption to the power dissipated by other parts of the FPGA.

B) FPGA power consumption distribution (Virtex-II FPGA family chip example)

The dynamic power consumption distribution of a particular FPGA is determined by the effective capacitance, and by the utilization and the switching activity of the relevant resources.

Various techniques are used to estimate the effective capacitance of the FPGA resources, but the results are not significantly different; Tables 2 and 3 ([50] and [53]). For example, compare capacitance values of Double, Hex, and Long in Table 2 to the same resources in Table 3. They are within 90% accuracy. Capacitances of long lines and the global clock tree change with the width and the height of a device, while capacitances of other resources are the same among members of a particular device family.

Table 2. Effective capacitances of Virtex-II FPGA chip resources, [50].

Type	Resource	Capacitance [pF]
CLB interconnects	IXbar	9.44
	OXbar	5.12
	Double	13.20
	Hex	18.40
	Long	26.10
CLB logic	LUT inputs	26.40
	FF inputs	2.88
	Carry	2.68
Clocking	Global wiring	300
	Local	0.72

Table 3. Effective capacitances of Virtex-II FPGA chip resources, [53].

Resource	Capacitance [pF]
Embedded multiplier	1.196
Block select RAM	880
CLB	26
Long-line route	23
Hex-line route	18
Double-line route	13
Direct-connect route	5

Although numerous factors influence the resources utilization and the switching activity (e.g. design dependences, input patterns, etc.), experimental results consistently indicate that most of the dynamic power in current FPGA's is consumed by interconnects, and may constitute up to 60% of the total dissipated power, see Figure 2, [45], [46], [50], [51], [52], [53].

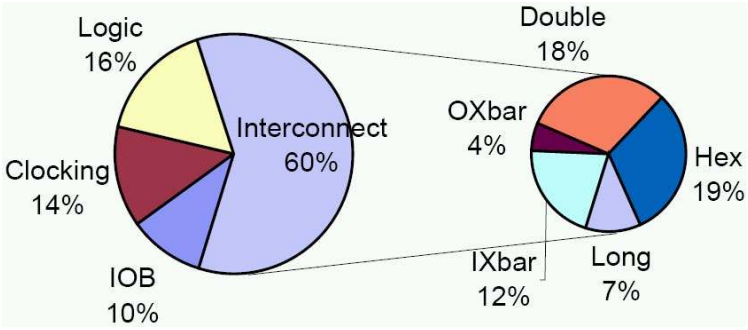

Figure 2. Typical dynamic power consumption distribution of Virtex-II FPGA chip, [50].

Moreover, a large portion of power, especially in arithmetic circuits, is wasted by unproductive signal transitions (caused by glitches), [46], [50], [52]. This is caused by spurious signal transitions on interconnect lines due to unequal logic or interconnect delays.

2.1.4. Power consumption estimation in FPGA

To ensure a proper functionality of a particular system, power consumption of the FPGA chip has to be estimated in early design steps, [49]. This is essential to design the appropriate printed circuit board (PCB), to provide an adequate power supply, and to ensure conditions for heat dissipation.

Real measurements as well simulation-based estimations can be used to obtain details of FPGA power consumption, [45], [49], [50], [51], [54].

Undoubtedly, the real measurements give the highest power measurement accuracy, however, the investigated device must be a representative one, [45]. Otherwise, measurements results may be distorted by some odd characteristics, e.g. during the manufacturing process.

The simulation-based power estimations are definitely much more convenient, [45]. However, they provide only approximate measurements.

Currently, power estimation approaches are mainly based on the switching capacitance and other corresponding factors, e.g. the average switching activity and the average resource utilization, [44], [45], [47], [48], [49], [50], [51], [53]. These power estimation approaches are found the most suitable for SRAM-based FPGA devices where the majority of designs are synchronous and driven by the system clock.

2.1.5. Means of power consumption reduction in FPGA

There are three major strategies to reduce power consumption in FPGA, [45]. First, power consumption may be reduced at the system-level, e.g. by algorithm modifications. Secondly, if the algorithm architecture cannot be modified, changes can be introduced to the logic partitioning, mapping, placement and routing. Finally, if there are no other possibilities, enhancing operation conditions is still a promising tool, e.g. changes to the load capacitances, the supply voltage, and the clock frequency.

Capacitance of on-chip connections is much lower than capacitance of external resources. Therefore, unutilized on-chip FPGA resources such as on-chip memory should be employed as much as possible, [44]. The capacitances may be also reduced by the timing constraints. They should be as tight as possible, and thus directing the place and route tools to choose resources with lower capacitances, [47], [50].

The supply voltage reduction gives the largest power consumption decrease (since it contributes a quadratic term to the power consumption). However, the supply voltage reduction increases delays, i.e. deteriorates performance.

Reduction of the clock frequency may also decrease power consumption. However, some design changes may be required. Otherwise, the same computations take more time leading eventually to the overall energy consumption increase.

2.1.6. Advanced power reduction techniques in FPGA

Recently, researchers, e.g. [55], tend to analyse power efficiency in more details. For example, they divide techniques for building power-efficient designs into five categories: computer-aided design (CAD), system, architecture, circuit, and process. For the sake of simplicity, we call the first two categories (CAD, and system) system-level techniques and the remaining ones hardware (or low)-level techniques.

CAD techniques refer to a diversity of enhancements made to the mapping, placing, and routing tools used to configure FPGA. System techniques consider high-level low-power techniques such as dynamic voltage control (scaling), e.g. [56], turning resources off when they are inactive, run-time reconfiguration (including algorithm modifications), device and architecture co-optimization, e.g. [57]. Architecture techniques consider functionality of the logic, input/output blocks, memory resources and the connectivity between such resources (e.g. region-constrained placement, [58]). Circuit techniques refer to how the logic and resources are implemented at the transistor level. Finally, process techniques consider the usage of new low-power process technologies offered by FPGA vendors.

Even though such techniques are common in lower-level design process, most of them cannot be employed at the algorithmic HDL level due to immaturity of such languages (e.g. Handel-C). Moreover, the results obtained by such techniques cannot be directly compared to our results (the methodologies of power reduction are not comparable). Therefore, the techniques are overviewed in this sub-section but they are not further discussed in the remaining parts of the thesis.

A) Low-power CAD techniques

Power consumption of FPGA devices can be affected by how CAD tools map an application to a programmable device. In general, such a mapping consists of five stages: high-level synthesis, technology mapping, clustering, placement, and routing. Each of the above steps can be optimized so that power efficiency of the final implementation is improved.

For example, in [59], [60] power-aware high-level synthesis algorithms are presented. In [59], the overall power reduction is obtained by minimizing the power of individual operations and minimizing the size of multiplexers. In [60], an algorithm of

programmable power supply is employed to minimize, given resource and timing constraints, power by assigning low V_{dd} to as many operations as possible. In [61], [62], [63], [64], [65], [66], [67], [68], low-power technology mapping algorithms are described. In these algorithms, power is minimized by absorbing as many high-activity nodes as possible and/or by minimizing node-duplication. In [65], [69], [70] and [71], low-power clustering techniques are presented. By absorbing as many small and high-activity nets as possible (where LUT's are packed into clusters) the proposed algorithms minimize power. [65], [72], [73] and [74] describe low-power place and route techniques that minimize power by reducing the distance between logic blocks. In [62], the leakage power is decreased by choosing low-leakage LUT configurations. Finally, [75] presents power-aware algorithms that map logical memories to the physical FPGA embedded memories. It allows minimizing dynamic power consumed by embedded memories by selecting the most power efficient ones.

B) System-level techniques

In this sub-section we present a diversity of system-level low-power design techniques that are used in current FPGA chips. Such techniques are divided into three categories: basic techniques, techniques involving run-time reconfigurability, and techniques used in soft-processors.

- Basic techniques

Using coarse-grained embedded blocks in FPGA appears to be a better solution than the fine-grained configurable logic blocks. It has been confirmed, [76], that the former approach is more power-efficient for the same functions. However, we must ensure that such a selection will not significantly increase the routing power consumption.

One of the most important system-level means to reduce the power consumption in FPGA is reduction of unproductive signal transitions, which can be obtained, for example, by pipelining, [46], [52]. Moreover, such a technique can straightforwardly be employed in high-level design (e.g. Handel-C), [46]. For example, this is achieved by implementing buffers (or increasing their length) in data processing algorithms, [25]. Such constructs force HDL tools to employ pipelining by using unused flip-flops.

A pipelined design has less logic between registers. Moreover, less logic between registers means the amount of interconnect between them is also reduced. Furthermore, pipelining breaks up long interconnects between registers that result in a smaller range of logic and interconnect delays. Consequently, fewer glitches occur and less dynamic power is dissipated during each cycle.

Additionally, pipelining may be implemented with almost no additional cost. Since many flip-flops within logic blocks of a design are unused, such flip-flops can be used for pipelining.

Pipelining is also used to reduce the energy per operation. Normally, it is employed to increase the clock frequency, hence to increase the number of operations per second. However, a pipelined design with the same clock frequency can perform a particular operation much faster than a non-pipelined design, i.e. consuming less energy (e.g. [77] reports 40% up to 90% energy per operation decrease in integer multiplication operations, CORDIC triple DES, and FIR filters).

Increase of the pipelining depth exponentially reduces the glitch-related power consumption. However, the additional latency is an unavoidable cost of pipelining, [46], [52]. Therefore, pipelining is a trade-off between power and energy reduction and additional latency.

Obviously, pipelining is easily and straightforwardly applicable to algorithmic HDL's. However, from the perspective of the dynamic power reduction, as discussed in Chapter 6, such a decomposition technique is practically equivalent to a parallel decomposition (see a note in Chapter 6.3). Therefore, even though pipelining may be used in our algorithm implementations, we do not analyze details of this technique in the thesis. From our perspective, it is just an example of a parallel decomposition (even though it sounds strange because, algorithmically, pipelining consists in sequential decomposition of calculations).

Word-length optimisation can be employed to obtain the best trade-off in speed, area, power consumption, flexibility, and accuracy. We can analyse the sensitivity of outputs in fixed-point hardware implementations to small errors caused by truncation or rounding internal variables. It has been found that power consumption can be reduced up to 98% (mean 87%) for adaptive filters and polynomial evaluations, [78], without significantly affecting the outputs.

Clock gating can be used to decrease power consumption by disabling the clock in the inactive regions of FPGA that prevents signal transitions. Such a technique can be combined with word-length optimisation, [79].

To further minimise power consumption (e.g. related to the temperature changes) dynamic voltage scaling can be used in FPGA supply voltage. It has been reported that in this way power consumption of various arithmetic circuits can be reduced from 4% up to 54%, [80].

- Run-time reconfigurability

If a design is used only temporarily, we can use run-time reconfigurability, as long as the energy reduction in execution is larger than the energy overheads for such a reconfiguration, [81]. If a device allows partial reconfiguration, further benefits can be obtained, [82].

- Low-power techniques for soft-processors (device and architecture co-optimization)

It has been found that by using the instruction set extension (obtained by iterative improvements) up to 40% energy reduction and 12% peak power reduction can be achieved for the MicroBlaze processor, [83].

A combination of power-aware scheduling and instruction recoding can be employed for optimising a soft processor at multiple abstraction levels. It has been reported that up to 74% power reductions can be obtained, [84].

C) Architecture and circuit techniques

Architecture and circuit levels influence directly the efficiency of mapping designs to FPGA, and the amount of circuitry used in the implementations.

For example, in [85] and [86], energy-efficient FPGA routing architectures and low-swing signalling technique for reducing power are presented. In [87], a technique reducing static and dynamic powers by reducing the number of configurable routing elements is described. In this approach, a new routing technique is proposed that utilizes a mixture of hardwired and standard programmable switches. In [88], a novel high-speed

routing switch with low-power and sleep modes is presented. In [89], the static power reduction is achieved by employing power-gating to the switches in the routing resources. In [90], power is reduced by optimizing the number of connections between the embedded modules and the routing resources, and by employing reduced supply voltage circuit techniques. In [91], authors employ a combination of various techniques (register file elimination, efficient instruction fetch) for a coarse-grained reconfigurable cell-based architecture.

D) Process techniques (device-level design: commercial devices)

Modern FPGA devices from such vendors like Altera and Xilinx incorporate diversity of low-power device-level technologies for improving power efficiency.

Both Altera and Xilinx employ triple gate oxide technology that allows for selection of three different gate thicknesses, thus optimizing the trade-off between static power and performance, [92], [93]. Although, the new medium thickness oxide transistors have slightly lower performances, power leaks are significantly smaller. Latest FPGA chips employ such transistors in the configuration memory and in switches controlled by this memory. Dynamic power is further reduced in new FPGA devices by using low-k dielectrics between metal layers. It reduces the parasitic capacitance and, in addition to smaller device geometries, the average node capacitance is reduced (reducing correspondingly the associated dynamic power). Further dynamic power reduction is obtained by lowering the supply voltage. Xilinx allows reducing the core supply voltage from 1.2V (in Virtex 4) to 1.0V (in Virtex 5), and Altera Stratix III allows selecting 1.1V for high performance and 0.9V for lower power consumption.

A number of architecture-level changes are made by Altera and Xilinx to their latest devices. They have already increased size of LUT's within the logic block, [93], [94]. For example, increasing the size of the basic logic element, from 4-input LUT's to 6 and 7-input LUT's reduces both static and dynamic power (more logic is implemented within each LUT so less routing between LUT's). Altera and Xilinx modified in their FPGA devices routing architectures that increase the number of neighbouring logic blocks (that can be reached in one or two hops only). Using routes with nearer hops reduces the average routing capacitance thus improving both the performance and power. Other architecture-level changes reducing overall power include the embedded memories, adders and multipliers, that are implemented as fixed-function embedded blocks. Such an

approach (opposite to implementations using the programmable fabrics) is more power-efficient since the circuitry that is normally required to provide the programmability is not needed.

Commercial FPGA CAD tools also incorporate a number of low-power techniques. For example, Altera Quartus II [94] and Xilinx ISE CAD tools [95] include detailed power models of various FPGA devices. Power-aware CAD techniques are incorporated into CAD flows. In Quartus II, minimizing the capacitance of high-activity signals is used to reduce the power consumption during mapping, placement, and routing. Power can also be reduced by optimizing the mapping to the embedded memories [75] and the embedded DSP blocks. In Xilinx ISE, tools minimizing the capacitance of high-activity signals are used for power reduction during placement and routing. Further dynamic power dissipation reduction is obtained by setting the configurations bits within partially used LUT's to minimize switching activity. Moreover, modern Altera and Xilinx tools ensure that unused logic circuitries are turned off for power savings.

Flash-based FPGA technology is a low-power alternative to SRAM-based solution. Flagship devices of such a technology are Actel's IGLOO devices. They are inherently more efficient since flash-based memory dissipates significantly less leakage power compared to SRAM memory. It is reported by Actel that their low-power FPGA devices dissipates 4 times less leakage power than other competitors, [96].

2.2. Data processing in WSN applications

Sensing, detection, classification, and tracking are typical operations in surveillance applications of WSN's, [97], [98], [99], [100], [101], [102], [103], [104]. The selection of appropriate data processing algorithms determines the overall performance, power, and energy efficiency of the whole system. Classification and tracking are rather specific to particular applications, but sensing and detection present common data processing properties. The last two are the subject of this section. For the thesis clarity, sensing and detection are investigated together and named as the sensing or data processing.

2.2.1. Sensing principles

A transducer is the main part of a sensor, [105]. In general, it converts a sensed quantity into a suitable voltage or current signal to be measured and processed. Such a signal may be further a subject of additional operations like conditioning and digital signal processing.

Typical sensors encountered in military and civilian surveillance applications include passive and active designs. While passive sensors are employed to detect and measure the signature of the object of interest (in various domains), active sensors perform similar actions by transmitting a signal and estimating how the target modifies, reflects, and/or scatters such a signal. General properties of passive and active sensors are presented in Table 4, [104].

Table 4. General properties of typical passive and active sensors used in surveillance WSN applications, [104].

Sensor	Type	Advantages	Disadvantages
Magnetic	Passive	Well defined far-field target phenomenology, discrimination of ferrous objects, no line-of-sight requirement	Poorly defined near-field target phenomenology, limited sensing range
Radar	Active	No line-of-sight requirement, operating through obstacles, estimates velocity, jamming resistant	Interferences
Thermal	Passive	Good sensitivity, good selectivity	Fresnel lens requirement, line-of-sight requirement
Acoustic	Passive	Long sensing range, high-fidelity, no line-of-sight requirement	Poorly defined target phenomenology, moderately high sampling rate, high complexity of signal processing
Chemical	Passive	No line-of-sight requirement, unique ability to detect gaseous compounds	Lack of availability for most of the chemicals
Electrical	Active	No line-of-sight requirement, non-contact sensing of non-ferrous, slow- or fast-moving, cool, quiet, odourless, steady, camouflage objects	Electrode placement requirement, nuisance parameters, interferences
Seismic	Passive	Long sensing range, no line-of-sight requirement	Signal propagation depends on ground composition, moderately high sampling rate, high complexity of frequency domain analysis
Optical	Passive	Long sensing range, high-fidelity	Poorly defined target phenomenology, line-of-sight requirement, high pixel sampling rate, high complexity of signal processing
Ultrasonic	Active	Multi-echo processing (sight beyond small obstacles)	Signal propagation depends on temperature and humidity, line-of-sight requirements, interferences

Sensors used in surveillance applications are typically used to measure movements (types of movement, accelerations, rotations or vibrations), forces (weight measurements, or forces/moments applied to an object or its part), light (diversity of wavelengths, light intensity, various changes to light over a time), temperature, humidity, sound (noise level, frequency spectrum, or various changes to sound over a time), and proximity/activity detection, [106]. We can also find in literature another categorization of sensors used in surveillance applications, based on the properties measured, i.e. sensors measuring physical properties (pressure, temperature, humidity, flow), motion properties (position, velocity, angular velocity, acceleration), contact properties (strain, force, torque, slip, vibration), presence (tactile/contact, proximity, distance/range, motion), biochemical properties (biochemical agents), and identification properties (personal features, personal identification data (ID)), [105].

Limited power sources of a typical sensor node discourage usage of active sensors, [105], [106]. Even passive solutions have to be power-efficient. This limits the performances of the sensors and results in a low quality of the produced data. Hence, sensor nodes are often equipped with sensors suffering from reading saturation due to granularity and range problems, long response time (insufficient for accurate signal extraction), electro-magnetic noise of the circuit board, thermal drift, and interference from a radio module (transceiver). Moreover, environments introduce additional noise sources, e.g. weather phenomena, targets that cannot be classified as objects of interest (e.g. animals), etc.

Although algorithms employed for filtering design-related and environment-related noises are different, they can be discussed jointly because their processing characteristics are similar.

Typically, sensor readings in surveillance applications present diversified, often unpredictable statistics, [107]. Models derived from such sensor data are not reliable enough to evaluate the performance of algorithms. Therefore, such evaluations are often carried out using data generated from simple parametric models only.

Although sensor selection is significant, none of the current sensors can detect the exact type of the object of interest.

2.2.2. Sensors selection for surveillance applications

Sensors should be selected in such a way that the best performance, lifetime, and cost of the system can be achieved, [104]. However, with a higher number of sensors more data are generated and requirements to process these readings are increased.

Moreover, measurements obtained by a single sensor node are usually not considered independent data, [102]. Although detection decisions are often performed by single nodes, classification decisions are usually a fusion of those single decisions as observations by individual nodes are not reliable enough. Thus, distributed detection, classification, and tracking are employed in WSN's. Additionally, multi-modal sensor nodes (i.e. equipped with various sensing devices) are used to improve sensing. A detection decision of such a multi-modal node is a fusion of decisions by individual sensors.

Problem formulation of a particular application is the crucial issue in a relevant sensor selection, [104]. Surveillance applications perform three fundamental tasks, i.e. target detection (discrimination between target's presence and absence), classification (identification whether the target belongs to one of several predefined classes) and tracking (maintaining the current position of the target).

The goal of target identification is to find a set of essential features with values specific to a particular object class. Although the identification process has to be performed in all typical signal domains (e.g. optical, mechanical, thermal, electrical, magnetic, and chemical), different aspects of the same domain may be detected by various sensors (e.g. measuring mechanical energy by microphones, accelerometers, or scales).

Most of the military surveillance applications assume ability to identify either three basic classes of targets, i.e. an unarmed person, an armed person (soldier), and a vehicle [99], [102], [104], [108], [109], [110], [111], or to detect only one class i.e. vehicles [112], [113]. In the latter case the ability to identify the vehicle type is assumed [113].

An unarmed person can be detected in thermal, mechanical (seismic or acoustic), electrical, chemical, and optical domains. It is assumed that the body of an unarmed person emits heat omni-directionally (as infrared radiation), impulsive signals of footsteps cause ringing at the natural ground frequencies, that acoustic signals of footsteps travel through the air with a different speed than seismic signals, and that complex chemical

trails are produced. Additionally, such a person reflects and absorbs light rays, and reflects and scatters optical, electro-magnetic, acoustic, and ultrasonic signals. However, the magnetic signature of such a person is negligible.

An armed person is often equipped with various ferro-magnetic objects (weapon and other metallic parts of a uniform). Therefore, such a person can be detected as a disturbance of the ambient (Earth's) magnetic field. Moreover, an armed person is expected to better reflect and scatter electro-magnetic signals, e.g. radar. Hence, the signal signature of an armed person is a subset of an unarmed one, [104].

A vehicle can be detected in thermal, mechanical (seismic or acoustic), electrical, magnetic, chemical, and optical domains. A thermal signature of a vehicle is particularly intensive in its characteristic hotspots, e.g. engine and exhaust. Seismic and acoustic signatures are caused by clicks and oscillations produced by mechanical parts of a vehicle. Its considerable metallic mass significantly disturbs ambient magnetic and electric field and reflects, scatters, and absorbs optical, electro-magnetic, acoustic, and ultrasonic signals. Additionally, a vehicle emits various gases as a side effect of combustion.

There are two general groups (Table 5) of comprehensive metrics used for sensor selection in surveillance applications of WSN's, [104], [111], [114]. While the first group is rather focused on design issues and costs, the other group addresses the sensor selection problem from the coverage, security, deployment, and sensing viewpoints.

Table 5. Sensor selection metrics.

First group	Second group
• Orientation invariance. Operation can be performed regardless of azimuthal and zenithal orientation of a sensor • Reasonable signal processing. The algorithm for data processing (signal detection, parameter estimation) should not be power or time consuming • Established. Sensors should be widely available on the market and well characterized • Reasonable size and cost. The integration of a sensor with the rest of hardware should be easy • Long sensing range. It allows turning a sensor node into sleep-mode between samples • No line-of-sight requirement • Co-locatable. Neighbour sensors should not interfere with each other • Passive operation. It allows a sensor to work in low-power operation mode, and make the node hard to detect • Reliability. A sensor should not provide false positive or negative readings	• Quantity of deployed sensors to provide required security level • Sensor detection model, and the way of determining the sensing coverage • The effect of terrain properties of the deployment area on the target detection • Sensor deployment in the area of interest • The weakest part of the coverage, and the way of the breach path discovery • The false alarm minimization and the decisions of the collaborative target detection improvement • The effects of the signal properties on the sensing coverage • The impact of the sensor scheduling on the sensing coverage • The effective communication and sensing range of sensors • Incremental sensors deployment

There is, therefore, a generally accepted agreement that magnetic, thermal, acoustic/seismic, and (in some rare cases) ultrasonic sensors are the most relevant to surveillance applications of WSN, [99], [102], [104], [108], [109], [110], [111], [112], [113].

2.2.3. Noise in typical sensing devices

A) Magnetic sensors

Magnetic sensors are used to detect vehicles or persons with ferrous objects, [102], [104], [112]. Detection is performed by measuring deflection of the magnetic field caused by movements of such a ferrous object. This requires sensing abilities with fine granularity within a wide range of signals that may eventually cause reading saturation. Additionally, a timely signal extraction is rather impossible due to the significant response latency, e.g. the time required for the circuitry to stabilize. Moreover, the circuitries introduce electro-magnetic noise lowering SNR, so increasing the computation cost of filtering. Changes in the ambient temperature also distort sensor readings. Finally,

the wireless communication module may interfere with the sensing circuitry (and with other sensing devices).

The response latency of a sensor is improved by preliminary estimates of such delays and their reduction to the acceptable level. The radio interference is eliminated by avoiding simultaneous operation of the wireless module and the sensing device. Other noises are suppressed by relevant data processing algorithms, e.g. a simple moving average (SMA) algorithm that acts like a finite impulse response (FIR) filter. A low-pass FIR filter is used to obtain reliable measurements of the magnetic field intensity. The thermal drift is eliminated by another SMA algorithm acting as a high-pass FIR filter. The low computational complexity of these techniques makes such data processing scheme suitable to various amplitude-based signals.

B) Thermal sensors

Thermal sensors are employed to detect movements of the object of interest, [99], [108]. A movement is indicated by changes in the thermal radiation. Such a sensor is built with passive infrared (PIR) sensing elements. Thermal sensors often incorporate a set of PIR elements, so their readings are not affected by the thermal drift. PIR performance is deteriorated by power supply fluctuations. However, this may be improved by a simple low-pass filtering. The weather and environmental conditions, including wind, temperature, humidity, moving objects (e.g. shaking leaves, rain drops, vehicles) are challenging to the sensor reliability. They introduce thermal disturbances triggering the sensor. Moreover, triggering events are diversified in their occurrences and frequencies. Nevertheless, thermal signatures of the object of interests often appear at significantly higher frequencies than such a noise. Therefore, a high-pass filter is sufficient to improve SNR of sensor readings. Typically, high-pass filters with low computational complexity such as infinite impulse response (IIR) and autoregressive moving average (ARMA) filters are employed.

C) Acoustic sensors

Acoustic sensors are used to distinguish among various types of objects of interest (e.g. a person, a vehicle), [102]. High sampling rates are required for effective detection and classification in such tasks (due to complex and diversified nature of acoustic signatures of object of interests). Unfortunately, computationally expensive techniques like FFT-based acoustic analysis cannot be employed in WSN applications (due to limited computation abilities of a sensor node), [102].

Before the signal of interest is analyzed, some basic filtering has to be applied, [102], [112], [113]. Acoustic readings are often distorted by other acoustic sources. Thus, deploying an acoustic sensor in unstable environments with high dynamics of environmental noise is challenging, [113]. However, SMA low-pass filtering [18], [115] and median filters [113] are found efficient in improving sensor readings.

2.2.4. Data processing algorithms

Detection (of an object of interest) is usually defined as sensing a value exceeding some threshold, [102], [112]. Design of such detection algorithms is obviously influenced by the sensor selection. Moreover, the detection efficiency is often degraded by additional operations performed to conserve power (which are needed in nodes with limited energy sources), [102], [104], [109], [113]. They may include non-continuous sampling (e.g. duty-cycling), energy-quality hierarchy (e.g. sensor triggering), and hysteresis filtering.

Duty-cycling is performed by cycling power of a relevant sensing system on and off with a frequency corresponding to the desired sampling. Triggering means that a low-power sensor (e.g. a thermal sensor) operates almost continuously and triggers other less power efficient sensing systems.

Other operations performed during sensing may include fusion of detection decisions, [102]. Moreover, data processing may be divided into several stages based on their computational complexity [115], i.e. preliminary sensing involves data processing operations with the lowest power (and performance) requirements.

Typical data processing algorithms used in surveillance WSN applications include SMA filters used as the low-pass filters, exponentially weighted moving average (EWMA) filters used as the low-pass filters, ARMA filters used as the high-pass filters, limiters, decimators, algorithms computing some data characteristics, constant false alarm

rate (CFAR) detectors, energy detectors, decision modules, and other designs of FIR and IIR low- and high-pass filters, [97], [98], [99], [100], [101], [102], [103], [104], [111], [112], [113].

Low-pass filters (including FIR, SMA, and EWMA filters) are used to reduce noise and improve SNR.

High-pass filters (including FIR, SMA, and ARMA filters) are employed for noise and thermal drift reduction and SNR improvement.

IIR low-pass filters acting as hysteresis filters perform operations over the sensor readings variance. Such hysteresis filters provide the fast-attack and the slow-decay response, i.e. non-constant phase shift. They prevent breaking a single detection into multiple smaller ones. However, such operations affect detection efficiency by causing longer decay time and non-linearly biasing duration estimations.

The limiter is a non-linear module which limits the magnitude of samples. Limiters may reduce the effect of noise outliers.

Decimators down-sample the sampling rate to the application requirements.

Data characteristics are required in sensing, detection, and classification processes. They include mean, variance, moving variance, and mean deviance over sensor readings.

CFAR detectors are used to estimate the signal duration (e.g. by employing the Neyman-Pearson detector). They output two values, i.e. *true* while the target passes by the sensor, and *false* otherwise.

Energy detectors determine and estimate the energy over sensor readings. Results of these operations may be used for event detection.

Decision modules estimate the target presence, may categorize the target, or just pass the data processing results to other modules for further processing.

2.3. Data-reduction in WSN applications

With technology advancements, applications of embedded systems become more sophisticated, where the need for more data to be processed increases as well. Unfortunately, the battery technology advancement is not as fast as in other technology areas. The data processing problems become even more severe when the data have to be passed wirelessly to another party. Some researchers report that the cost of sending one bit of data over a certain distance is as high as the cost of 3000 CPU instructions executed

locally, [116], [117], [118], [119]. They suggest it is much more energy-efficient to spend some CPU time to reduce the size of data to be sent.

2.3.1. Introduction to data-reduction

We may categorize data-reduction or data compression algorithms into two main groups, i.e. lossless and lossy, [120], [121].

Lossless compression techniques involve no loss of information within data being processed, [120], [121]. The original data can be recovered exactly from the compressed data. This technique is used in applications that cannot tolerate any difference between the original and decompressed data mainly. Generally, lossless compression techniques generate a statistical model of data and map data to bit strings based on the generated model, [122].

Lossy compression techniques introduce loss of information within data being processed, [120], [121]. Hence, the data cannot be recovered or reconstructed exactly from the compressed data. However, lossy compression techniques allow much higher compression ratios by accepting distortion in the reconstruction process. Generally, lossy compression techniques transform given data into a new data space using an appropriate basis function or functions, [122].

We can evaluate compression algorithms in various ways, [120], [121], [123]. A compression algorithm can be evaluated by its relative complexity, the memory required for its implementation, requirements regarding CPU speed, the obtainable compression ratio, and how closely the reconstructed data approximate the original, e.g. the distortion introduced by compression.

2.3.2. Typical WSN data-reduction algorithms

Data-reduction is not commonly used in applications of WSN's. Major limitations to applicability of data compression algorithms are memory footprints and processing unit performance requirements, [123], [124], [125]. Therefore, the use of typical lossless data compression algorithms such as Lempel-Ziv-Oberhumer (LZO), basic zip with modifications (BZIP2), prediction by partial matching with modifications (PPMd), and other PC-based algorithms is rather discouraged, [123]. However, there are some works on employing such data compression algorithms, e.g. Lempel-Ziv-Welch (LZW), to

power and performance limited devices, [126]. Nevertheless, a universal data-reduction scheme for all WSN's is practically impossible to develop due to diversity of data gathered, [117], [123], [127], [128].

Some dedicated compression schemes have been developed especially for WSN's to overcome to some extent the above-mentioned limitations, [117], [119], [123], [124], [129], [130], [131], [132]. These are: coding by ordering, pipelined in-network compression, and differential coding lossless schemes, and some low-complexity video compressions schemes such as JPEG with certain modifications.

Other lossless data-reduction algorithms commonly used in sensor networks are Huffman, LZW, and run-length encoding (RLE) coding, [126], [133]. Moreover, some techniques to change data description, and increase compression ratio, before data compression are also often used, [126], [133]. These use Burrow-Wheeler transform (BWT) and structured transpose (ST) to reorder data before LZW compression, and decorrelation transforms such as wavelet transform (WT) to describe structures in data before Huffman compression. However, the latter introduce some distortions due to lossy transformations.

Lossy compression algorithms used in WSN's include aggregations and approximations, [116], [119]. Aggregation summarizes the measurements in the forms of simple statistics, e.g. average, maximum, minimum, etc., that are transmitted to the base station over regular intervals. Aggregation is found an effective way in reducing the volume of data but rather crude for applications requiring detailed historical information, e.g. surveillance or monitoring. Approximation is a less intrusive form of data reduction, e.g. histograms, wavelets, discrete cosine transform, linear regression, etc., employed (if data exhibit a large degree of redundancy) to replace the underlying data by an approximate signal tailored to the application needs.

There are also other means of data-reduction in WSN's [118], [124], [134], [135], however they are not subject of our discussion. They involve distributed processing and combine routing, data fusion, and data aggregation.

2.3.3. Data-reduction requirements in WSN's

The main objective of currently used data-reduction algorithms is to reduce the data volume (that directly influences communication capacity), [118], [126]. Only minor efforts are put on the computation energy, [117], [118], [126].

The diversified nature of embedded systems, especially untethered and wireless ones, sets new challenges for data-reduction algorithms. Data-reduction schemes are supposed to reduce communication latency and gain energy efficiency (by reducing the energy consumed on data transmission), [117]. The global objective, however, is to reduce the overall energy consumption, [118]. This may affect, for example, the selection of matching compression and decompression algorithms since both operations are often not performed by the same algorithm (and decompression is usually less costly in terms of energy), [125].

Altogether, energy awareness is one of the main requirements in WSN data-reduction algorithms, next to low complexity and a small memory footprint.

2.4. Algorithm partitioning of FPGA-based designs

FPGA-based designs may be partitioned at two different levels of design, i.e. at low-level (often referred to a hardware-level) and at higher levels (e.g. at system-levels), [27], [136], [137], [138], [139], [140], [141], [142], [143], [144]. The low-level design partitioning is often in a form of design decomposition into multiple clock domains, while the latter approach addresses algorithm partitioning. Although these approaches differ significantly, their common goal is the performance improvement. Algorithm partitioning may also lead to a multi-domain decomposition, but this is not a strict rule. Undoubtedly, hardware-level multi-clock domain techniques are well established, and such a partitioning is also possible at the HDL-level description, i.e. at higher levels. However, methods and techniques for design partitioning based on algorithm partitioning, [27], [136], [137], [138], [139], [140], [141], [142], [143], [144], are still in their development stage.

In simple designs, e.g. [138], [142], [143], with a single FPGA chip, algorithm partitioning is used to decompose a particular algorithm, to isolate those parts that may cause performance deterioration, and finally to parallelize majority of operations. Partitioning may also be used to more efficiently re-use components of an algorithm and to employ partial FPGA reconfiguration. Moreover, the decomposed algorithm can be easier and faster evaluated against various metrics.

In more advanced designs, e.g. [136], with a number of FPGA chips, algorithm partitioning may be employed to partition a particular algorithm (especially if a single chip does not provide enough logic resources) in a way that the whole system may be

implemented on a number of programmable devices in a seamless (from the user perspective) way.

In designs with diversified types of processing devices, [137], [142], e.g. FPGA and DSP chips, algorithm partitioning is helpful in balancing the computational load between these processors. Balancing is also used to trade off the need for high-speed, low power consumption, and a small physical size of the system, [142].

In reconfigurable computers, [137], [140], [141], (i.e. computers with a GPP and one or more programmable devices) algorithm partitioning may be employed to divide some operation executions between the host (i.e. GPP) and FPGA chip(s).

Algorithm partitioning is also used for hardware/software partitioning for system-on-programmable-chip (SOPC) and reconfigurable computers, [27], [141]. In both cases, partitioning is used to accelerate the critical parts of algorithms by moving them to hardware (e.g. FPGA chip).

Academia and industry groups working on tools for design process automation are also interested in algorithm partitioning, [136], [139], [141], [144].

2.5. Chapter summary

In this chapter we have presented a general survey on literature related to the scope and objectives of the thesis. Even though some of the topics are not directly related to the thesis, we believe they are important to understand our work. In particular, we believe that a better understanding of underlying issues of data sensing and processing in sensor networks helps to justify the selection of implemented algorithms, and the proposed assumptions regarding timing and other characteristics of designs.

With a diversity of existing sensors, surveillance applications can be deployed in diversified environments. However, the most typical sensors for such applications are magnetic, thermal, and acoustic. Therefore the experiments reported in the thesis are often based on typical characteristics (regarding timing, data processing algorithms, etc.) of these sensors.

Until recently, data-reduction schemes used in WSN applications have focused mainly on reducing data volumes, while power and energy issues have been neglected. It is believed that by exploiting these issues, while taking into account the general properties of data-reduction algorithms, power and energy efficiency of FPGA-based designs in WSN can be improved.

Although both the static and the dynamic power consumption in FPGA-implemented designs can be optimized, the reduction of static power can be generally obtained only by technological improvements of a particular FPGA chip (or by switching to programmable devices manufactured in a different technology). These areas are beyond the scope of the thesis. However, the dynamic power consumption can be potentially shaped at various levels of design, including the system-level. This is one of the objectives of the presented work.

Algorithm partitioning of hardware-based designs (including FPGA implementations) is an emerging area. However, in this work we do not focus on performance improvements that can be achieved by partitioning. Assuming the available partition (which may or may not be optimized from the performance perspective) of an algorithm and a target hardware platform (FPGA device) we attempt to achieve a better power and energy efficiency using such a partitioning.

CHAPTER III
EXPERIMENTAL SETUP

This chapter describes tools and equipment used to conduct our experiments. In Section 3.1 we give a brief overview of high- and low-level programming tools used for hardware targeting, and a brief description of the hardware platform. Section 3.2 is a short introduction to Handel-C, an algorithmic programming language used in our experiments for targeting hardware at the high-level of design. In the final Section 3.3, we discuss what results can be obtained using high- and low-level tools, and how these results are interpreted in further discussions. This section also defines general assumptions and notions required to understand the conducted experiments. Other assumptions and notions, specific to particular experiments only, are defined in the corresponding chapters.

3.1. Software tools and development platform

3.1.1. Software

Two different hardware-programming environments are used to conduct our experiments, i.e. DK Design Suite 4.0 SP1 and Xilinx Integrated Software Environment (ISE) 7.1i, [21], [34].

DK Design Suite (i.e. a complete design environment for C-based algorithmic design entry, simulation, and synthesis) allows targeting hardware at the high-level. Using this programming environment we code particular problems (i.e. algorithms) in an algorithmic HDL, Handel-C. Resulting designs are synthesized to the electronic design interchange format (EDIF) netlist used for further low-level implementations.

Xilinx ISE is employed to target hardware at the low-level (using EDIF netlist), i.e. to map, place, and route a design for a particular FPGA chip. Xilinx ISE is also used to assign area constraints to such a design (i.e. to fix particular implementation parts to a particular area on a chip). When a design is mapped, placed, and routed, the FPGA chip can be physically programmed (configured) and power properties of a particular design can be investigated by using XPower (one of the accessories of Xilinx ISE), [145]. XPower allow estimating dynamic power consumption of an implemented algorithm, i.e. consumed by clock, logic, and signal resources.

3.1.2. Hardware, and algorithms verification and validation

The hardware platform used in our experiments is the RC203 development board (by Celoxica) equipped with a Xilinx Virtex-II FPGA (part: xc2v3000fg676-4), Figures 3 and 4, [146]. Table 6 lists some other development boards and processors for comparison.

Verification of the implemented algorithms is done at the system-level. Hardware validation of the implemented algorithms is, in general, not the subject of the thesis. However, hardware validation is performed for selected algorithms, especially if on-board devices are used. Based on the tests described in Chapter 4, we can assume that hardware validation does not introduce qualitative or quantitative changes to the results obtained by the system-level verification of conducted experiments.

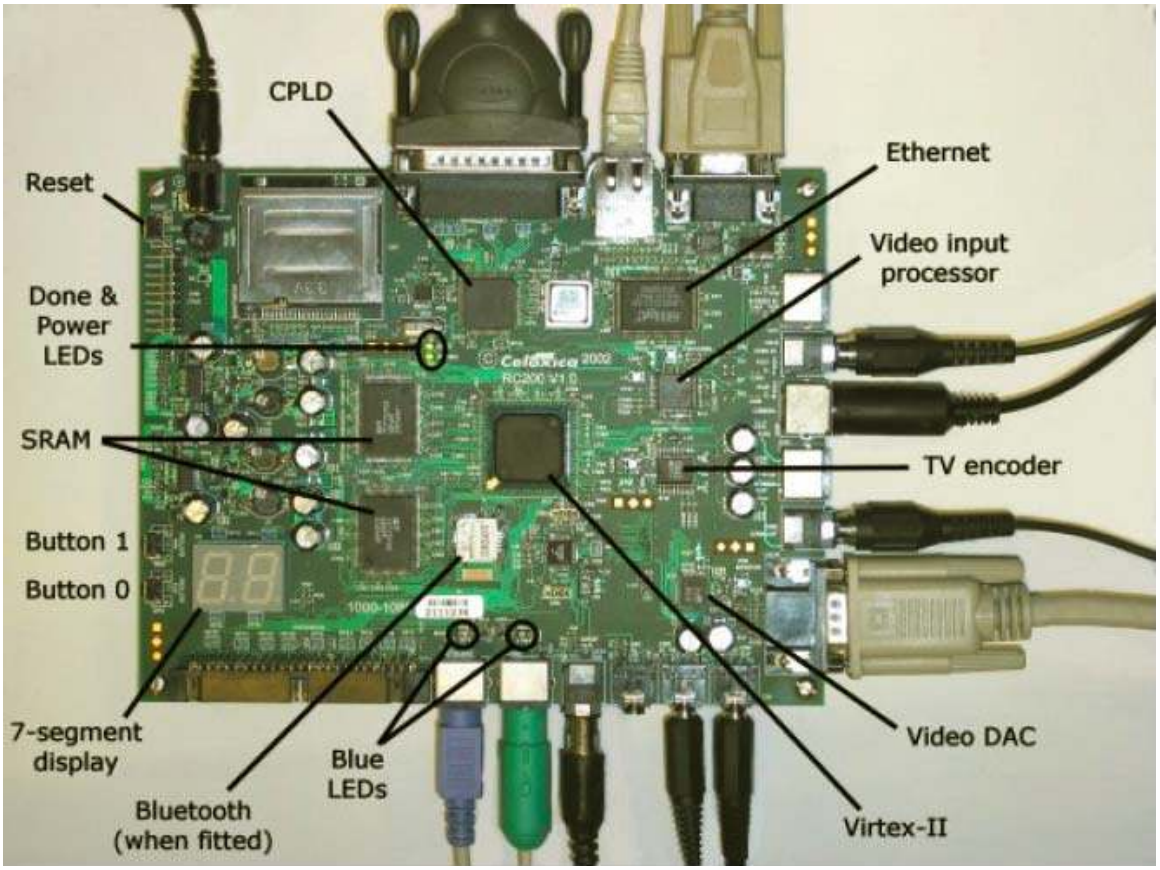

Figure 3. Devices on exemplary RC200 (same as RC203, except FPGA chip) development board, [146].

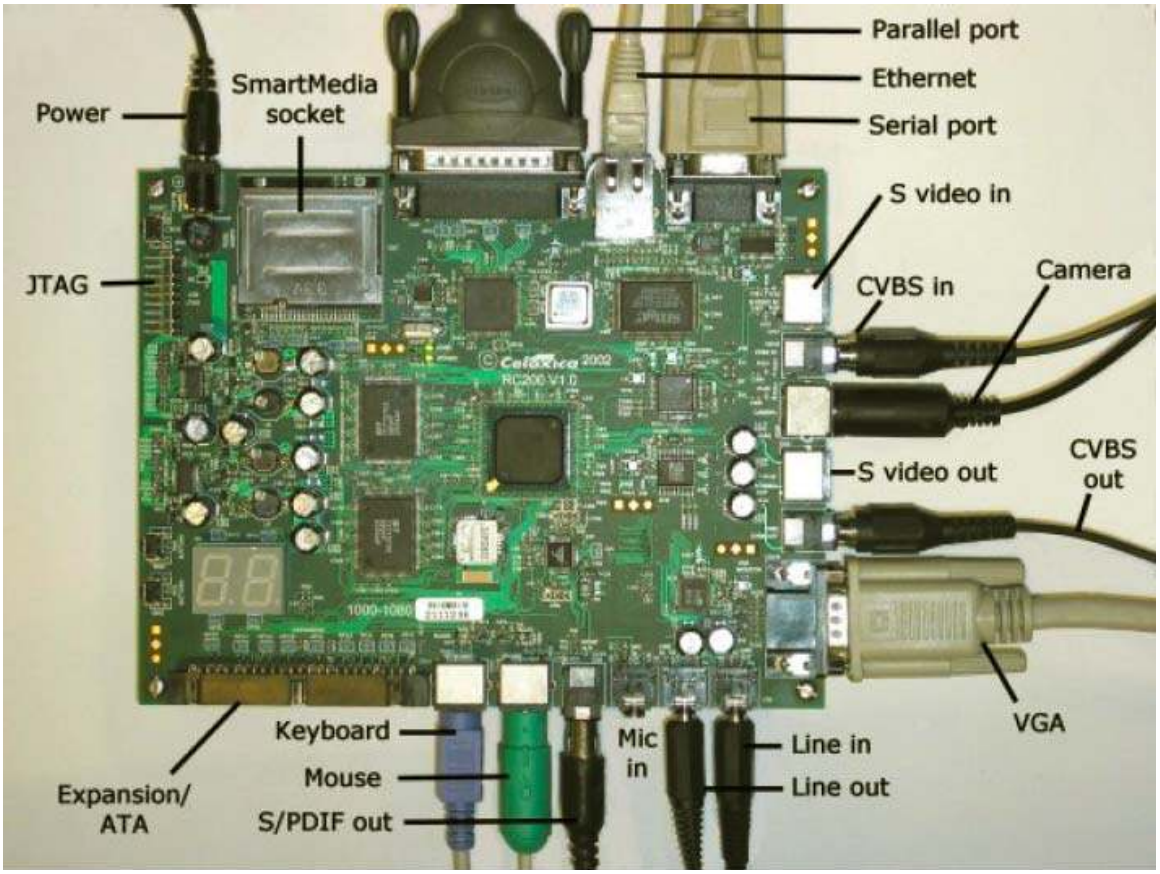

Figure 4. Connectors on exemplary RC200 (same as RC203, except FPGA chip) development board, [146].

Table 6. Comparison of Celoxica development boards, [147].

	FPGA chip properties				
Development board name	**Family**	**Device**	**Package**	**Speed grade**	**Part**
RC100	Xilinx Spartan-II	xc2s200	fg456	5	XC2S200FG456-5
RC200	Xilinx Virtex-II	xc2v1000	fg456	4	XC2V1000-4-FG456
RC203	Xilinx Virtex-II	xc2v3000	fg676	4	XC2V3000-4-FG676
RC1000	Xilinx Virtex	xcv1000	bg560	6	V1000BG560-6

3.2. Introduction to Handel-C

Handel-C is a type of algorithmic HDL. This is a rich subset of C, with non-standard extensions to control hardware instantiations and parallelism, [25]. Moreover, some features of the C language, not appropriate to hardware implementations, are removed. Handel-C has much of the syntax of conventional C and includes all common C language features that are necessary to describe complex algorithms in hardware. However, floating point data types are omitted, as is the case for other languages used to describe hardware. Floating point data types are supported through additional external libraries, that, unfortunately, need large amounts of hardware resources.

Although sequential programs can be written in Handel-C, parallelism is recommended to gain the maximum performance from the target hardware. Handel-C parallelism is a true parallelism. This means that two instructions commanded to execute in parallel will perform at exactly the same time instant by two separate pieces of hardware. This is different from the time-sliced parallelism of GPP's.

Programs written in Handel-C are implicitly sequential, i.e. a sequence of instructions must be executed in such an exact order. If instructions are supposed to be executed in parallel, a *par* keyword must accompany those instructions.

Handel-C controls the flow of a program by providing mechanisms similar to conventional C. For example, a particular code can be executed conditionally or a block of code can be repeated a number of times.

Using Handel-C we can express our algorithms without specific knowledge of the underlying hardware. In a way, Handel-C is to hardware what a conventional high-level language is to GPP assembly language.

DK Design Suite generates the hardware specification (in a form of an EDIF netlist) directly from Handel-C source programs, i.e. there is no intermediate interpreting layer typical to assembly languages targeting GPP.

3.3. General assumptions and notions on results

The results of the conducted experiments are obtained from two applications. Results related to experiments at the system-level are produced by DK Design Suite, while Xilinx ISE is used to produce low-level results. Using results of both levels, we estimate hardware resources requirements, processing time requirements, and power consumption properties of particular algorithms or their parts. We are aware that the system-level measurements are less accurate. Nevertheless, the accuracy is acceptable as shown in the following chapters. Moreover, the system level estimates can be obtained much faster.

In all subsequent experiments implemented algorithms are investigated against typical device and hardware-programming environment settings. That is, we do not force DK Design Suite or Xilinx ISE to perform against area or performance only. This is to balance implementations tradeoffs.

Implemented algorithms do not occupy the whole chip area, and the unused parts of the device are left on (e.g. clock gating is not used). If unused clock nets were switched

off, the experimental results would remain qualitatively similar although the numerical values of power/energy estimates e.g. the hardware inactivity coefficient, see the analysis in Section 7.2.5) would.

3.3.1. Hardware resources requirements

The hardware resources requirements are described at the system-level by estimating the equivalent number of NAND gates (or flip-flops) and at the hardware-level by determining the number of FPGA slices. The latter number shows the physical usage of a particular FPGA chip.

The equivalent number of NAND gates used by a particular design is obtained by compiling and synthesizing the design at the system-level using DK Design Suite. These results remain the same if a design is compiled and synthesized for another FPGA chip. To obtain the physical usage of a particular FPGA, the synthesized design is targeted to the hardware using Xilinx ISE.

3.3.2. Processing time requirements

The processing time requirements of a particular algorithm or its relevant parts are investigated at the system-level using the debugging tools of DK Design Suite. If the investigated data processing algorithm is data dependent (in term of the processing time), we chose the worst case scenario. This can be obtained by using specially generated data files.

3.3.3. Power consumption estimates

We estimate power consumption at the system- and low-levels. However, the first estimation technique is based on some additional assumptions.

In the general, the number of clock cycles represents the processing time of a particular algorithm or its parts. However, this also indirectly determines the clock frequency for the corresponding implementation of this algorithm or its parts, if all parts of the algorithm (or the algorithm itself) are performing under the same time constraints, and the maximum acceptable processing time should remain the same regardless the number of clock cycles. Hence, more clock cycles required by a part of a particular

algorithm (or the algorithm itself) correspond to a higher frequency, i.e. a higher dynamic power consumption, according to Equation (1).

From Equation (1), it can be seen that the dynamic power consumption of a particular design is proportional to its hardware area. Thus, if we neglect the static power (that is inherently there) and assume a certain clock frequency, the results from the system-level implementation (i.e. the equivalent number of NAND gates, latches, etc.) estimate the dynamic power consumption in some non-descriptive units (NDU's) that is hardware independent. Then, if we apply a certain clock frequency to the low-level hardware estimates (e.g. the number of slices) these hardware resources would represent the actual power consumption. Such an assumption is important in comparing the system-level power consumption estimates to low-level power consumption estimates. The validity of such an approach is further justified by the experiments described in Chapter 4.

Power consumption estimation at the low-level is obtained by XPower, [145], similarly to other works on the power estimates in FPGA relying upon FPGA vendor tools to perform power estimations instead of actual measurements. However, such power estimation tools require simulation data describing activity rate of nets of the implemented design. Embedded systems such as sensor nodes are often deployed in hard to predict environments. Nodes often process data of highly diversified or unpredictable pattern. In effect, usage of FPGA hardware resources of an implemented algorithm is also diversified in terms of time and resource location. Therefore, we arbitrarily assume the nets activity rate of our designs at 50% level as a fair approximation. Others value can be used without any loss of generality and without invalidating the proposed methods. However, too small values are not recommended because they increase the relative contribution of the dynamic power of the unused section of the FPGA (see the next paragraph).

XPower provides the overall power measurements so that the actual dynamic power of a design only (i.e. excluding unused section of FPGA) cannot be measured. However, the dynamic power of unused parts is considered negligible (excluding extremely small designs that are not discussed in the thesis). The validity of this approach has been additionally verified by the results presented in the updated Section 7.2.5 where the switching activity of the unused parts of FPGA has been estimated in 0.0007-0.005 range (the highest values for the lowest clock frequencies). Thus, the assumption on a small impact of the dynamic power of unused parts of FPGA seems reasonable. This

power becomes just a small additive bias. This approach has been adopted throughout most of the thesis.

The only part of the thesis where the dynamic power of unused parts is taken into account is Chapter 7. In Chapter 7, we analyze the *ratio* between dynamic powers of inactive and active designs. The presence of an additive bias may change the ratio so that the analysis has been presented on how to exclude the effects of the unused parts (Sections 7.2.4 and 7.2.5).

3.4. Chapter summary

In this chapter we have presented software and hardware platforms for the conducted experiments.

A brief overview of high- and low-level tools used in our experiments is given. In particular, Handel-C is introduced in this chapter. We focus on advantages that such an algorithmic HDL can offer in hardware targeting.

We have also discussed how to interpret results obtained from our experiments, i.e. hardware resources requirements, processing time requirements, power consumption estimates. Only general assumptions and notions common to all conducted experiments are given. If a particular experiment requires additional assumptions and/or notions, they will be discussed in the corresponding chapter.

CHAPTER IV
POWER ESTIMATES IN SYSTEM- AND LOW-LEVEL EXPERIMENTS

This chapter is a discussion on relations between system- and low-level dynamic power estimation results in experiments on algorithm implementations. In Section 4.1 we introduce the basis of our experiments. Section 4.2 investigates whether hardware-level estimates can, at least to some extent, be represented by similar experiments performed at the system-level. These experiments investigate both designs partitioned into a number (two) of clock domains, and non-partitioned designs. Although this is a common-sense assumption that system- and low-level power estimates should be correspondingly related, we have not found any sources confirming it experimentally.

4.1. Introduction to conducted experiments and general assumptions

These experiments are based on a selected data reduction algorithm used in WSN applications, i.e. *Huffman coding*, [126], [133]. We investigate how design decomposition and diversified clock frequencies of clock domains affects the overall power consumption in the corresponding hardware implementation. In order to avoid any distortions of results, we do not use any chip area constraints and allow map, place, and route tools to perform unconstrained optimizations. Moreover, we decided to use the external FPGA pins as direct data inputs and outputs for *Huffman coding*, i.e. we do not implement any ADC/DAC library for RC203 on-board devices. This is to avoid any additional result distortions that may arise due to non-*Huffman coding* logic implementations.

For these experiments, *Huffman coding* (decomposed into *compressor* and *decompressor*) is intentionally selected. Both domains (*compressor* and *decompressor*) perform completely different tasks, i.e. compressing and decompressing data, so that their algorithmic structure is diversified. Even though, both domains occupy similar amounts of estimated system-level (latches, i.e. FF's and memory bits) and low-level (slices) hardware resources (in spite of their different inner structure), see Table 7.

More details on *Huffman coding* are given in Chapter 6.

Table 7. System- and hardware-level complexities – *Huffman coding*.

	Latches (FF+memory bits)	**Slices**
Design B with only decompressor	357+1254=1611	1555 (10%)
Design B with only compressor	283+1148=1431	1291 (9%)
Compressor and decompressor – Design A	2805+106=2911	2865 (19%)

Note: In Design A, the *compressor* and *decompressor* are implemented within the same module. In Design B, they are implemented in a separate module each (more in Chapter 4.2).

It can be noticed that there are certain differences between system- and low-level estimates for *compressor* and *decompressor*, i.e. *compressor* (1431 latches) has a 12% lower complexity than *decompressor* (1611 latches) for the system-level estimates, while for the low-level estimates *compressor* (1291 slices) and *decompressor* (1555 slices) differ by 20%. This may be considered a significant difference, but we should not neglect the fact that our approach corresponds to the highest abstraction layer, i.e. the system-level design. Such an approach would obviously decrease precision, but a shorter time-to-market (so more complex designs are possible) is achieved at the same time.

Compressor and *decompressor* parameters are chosen in the way allowing low hardware utilization, up to 20% of the chip area. This is to give the system freedom to map, place, and route tools in achieving the most suitable utilization of available resources, and to decrease the clock frequency selection effects on such resources utilization within a chip. To obtain a moderate hardware utilization, *Huffman coding* was implemented for data of 1bit width, the alphabet of 2 elements, and the sample size of 32 elements. Any modification of these values would proportionally change the size of both *compressor* and *decompressor* so that the hardware utilization would increase/decrease, but the relative sizes of both modules (and the ratio between their power consumptions) will not be affected. However, too large values may increase the design size to the point where the compiler has to perform a constrained optimization (e.g. area minimization only). Then, our assumptions on unconstrained design optimization would be violated and the results might be biased. Therefore, the actual values of *Huffman coding* parameters are of secondary importance but they cannot be increased beyond the limitations of the available FPGA device.

This chapter investigates only two clock domains decomposition. However, this is the foundation (confirming preliminary assumptions on the correspondence between low- and system-level power estimates) to further discussions on estimating low-level power consumption using system-level results for designs with more than two clock domains, see Chapter 5. We show that the dynamic power consumption can be obtained directly from system-level results using simple derivations (shown in Section 4.2).

4.2. Results of the Experiments

In Design A, the *compressor* and *decompressor* are implemented within the same module but in two separate clock domains. In Design B, they are implemented in a separate single-domain module each (in Chapter 6 we further decompose *compressor* and *decompressor* into more parts).

The dynamic power consumption (see Equation (1)) of a particular design is proportional to its hardware area. By neglecting the static power (that is inherently there) and assuming a certain clock frequency, results from the system-level implementation (i.e. the equivalent number of NAND gates or the number of latches) would represent the dynamic power consumption in some non-descriptive units (NDU's) that are hardware independent. Then, if we assume the same clock frequency for the low-level hardware estimates (the number of slices) these hardware resources will correspond to the power consumption estimated at the low level (XPower).

Multiple variants of both designs have been hardware-implemented using diversified clock frequencies (minimum and maximum clock frequencies defined by the platform limitations). Although certain variations in the physical layouts of the implementation are unavoidable, we expected that the hardware-level power estimates would consistently correspond to the system-level estimates.

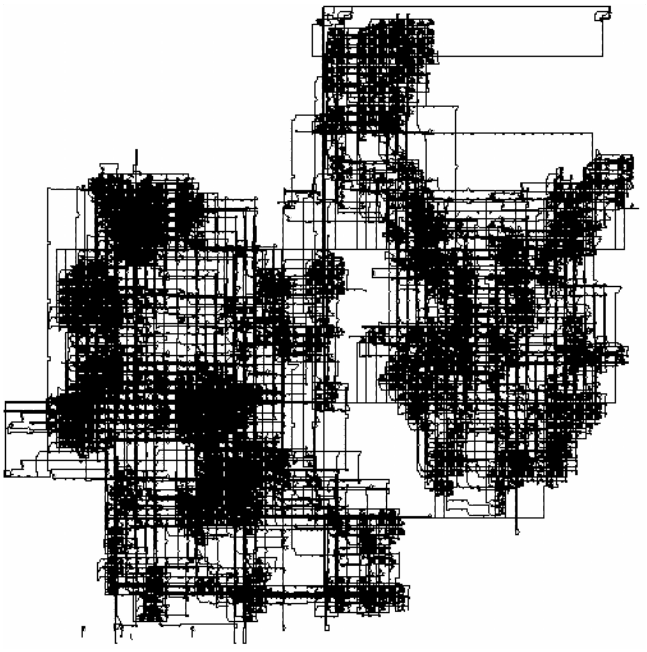

Figure 5. *Compressor* (15MHz; on the right) and *decompressor* (15MHz; on the left) in an exemplary Design A – *Huffman coding*.

Typical results of the experiments are given in Figures 5 to 8 and in Tables 8 to 11. The total dynamic power (clock, logic, signals) of relevant implementations is determined by XPower.

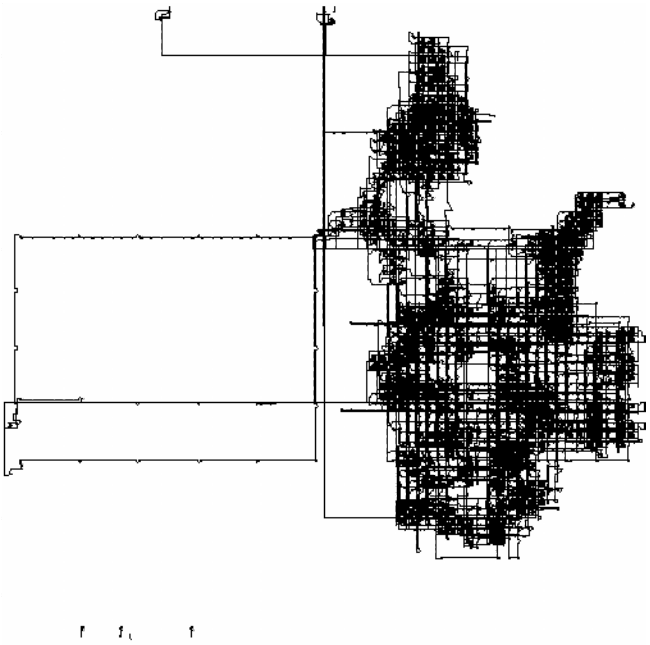

Figure 6. Design B with only *compressor* (15MHz) – *Huffman coding*.

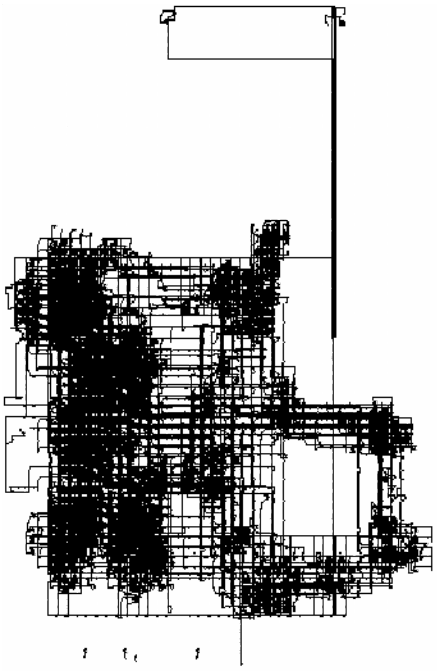

Figure 7. Design B with only *decompressor* (15MHz) – *Huffman coding*.

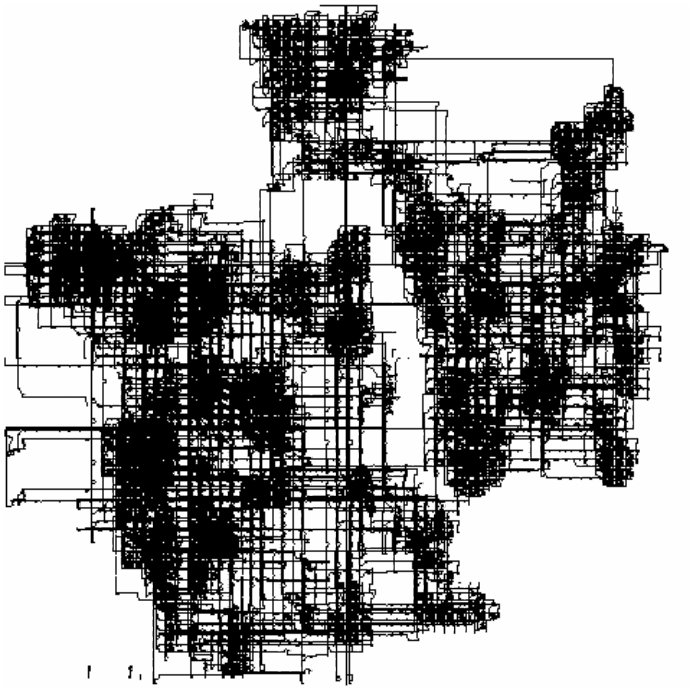

Figure 8. *Compressor* (24MHz; on the right) and *decompressor* (6MHz; on the left) in an exemplary Design A– *Huffman coding*.

Table 8. Only *decompressor* – Design B.

Clock frequency [MHz]	**Total dynamic power (clock+logic+signals) [mW]**
6	1.57+5.31+13.66=20.54
15	1.04+13.06+33.73=47.83
24	1.67+20.89+54.46=77.02

Table 9. Only *compressor* – Design B.

Clock frequency [MHz]	**Total dynamic power (clock+logic+signals) [mW]**
6	1.04+5.06+12.43=18.53
15	1.04+12.43+30.91=44.38
24	1.67+19.88+49.40=70.95

Table 10. The overall power consumption (*decompressor*/*compressor*) – Design A.

Decompressor clock frequency [MHz]	**Compressor clock frequency [MHz]**	**Total dynamic power (clock+logic+signals) [mW]**
15	15	1.97+25.48+65.79=93.24
18	12	2.48+25.65+68.03=96.16
22	8	2.48+25.84+67.71=96.03
24	6	2.53+25.95+65.75=94.23

Table 11. The overall power consumption (*decompressor*/*compressor*) – Design A.

Decompressor clock frequency [MHz]	**Compressor clock frequency [MHz]**	**Total dynamic power (clock+logic+signals) [mW]**
15	15	1.97+25.48+65.79=93.24
12	18	2.57+25.40+64.98=92.95
8	22	2.82+25.25+64.00=92.07
6	24	2.77+25.19+63.07=91.03

Although we provide the dynamic power consumption decompositions (into power consumed by clock, logic, and signals resources) such a detailed analysis of particular components of the dynamic power is not needed in our experiments. These numbers are given to indicate that even though the overall dynamic power consumption corresponds to the system-level estimates, it does not have to be predictably distributed into particular components of the dynamic power. The hardware-programming tools can choose the most suitable implementation parameters. Details of the resources selection may vary from algorithm to algorithm, so the dynamic power consumption may be differently distributed.

We can observe that the combined total dynamic power consumption of separately implemented *compressor* and *decompressor* is almost the same as the total dynamic power consumption for the design with both *compressor* and *decompressor* (compare first and the last rows of Tables 8 and 9 to the last rows of Tables 10 and 11, and second rows of Tables 8 and 9 to the first rows of Tables 10 and 11). This also applies to a certain extent if we compare components (clock, logic, signals) of the total dynamic power. This shows that power consumption of the whole system (i.e. Design A – combined *compressor*/*decompressor*) can be also viewed from the perspective of system elements (i.e. Designs B). For example, for any clock frequencies we can combine dynamic power consumptions of individually designed *compressor* and *decompressor* to obtain the total dynamic power consumption of the system consisting of *compressor* and *decompressor*. Tables 10 and 11 additionally show that the total dynamic power consumptions changes correspondingly to the clock frequency changes. Moreover, in spite of diversified physical layouts of the implementations (compare Figures 5 to 8) power characteristics of the design remain consistent.

The obvious fact is that dynamic power consumption of programmable logic changes linearly with frequency (according to (1)). Thus, from the results of the above experiments we can envisage that the total dynamic power consumption of the whole system can be estimated directly from the system-level results. Such estimations are obviously valid for FPGA chip of a particular type only.

We can obtain the low-level-to-system-level dynamic power consumption coefficient (for a particular algorithm and a particular FPGA), *k*, by dividing experimentally obtained power consumptions (for a particular frequency) of a design by its system-level hardware resources. For *decompressor* of Design B (15MHz) this coefficient is:

$$k = 47.83/1611 = 29.690 \cdot 10^{-3} \qquad (3.A)$$

Repeating the same computations, however, for the *compressor* (Design B – 15MHz), we get:

$$k = 44.38/1431 = 31.013 \cdot 10^{-3} \qquad (3.B)$$

Then, the total dynamic power consumption for Design B (6MHz for *decompressor* and 24MHz for *compressor*) would be obtained by summing the estimated power consumptions of the *decompressor* and the *compressor* using the coefficient *k* calculated for *decompressor* (Eq. 3.A):

$$(6/15) \cdot k \cdot 1611 = 19.13\,\text{mW} \qquad (4)$$

$$(24/15) \cdot k \cdot 1431 = 67.98\,\text{mW} \qquad (5)$$

to give the total dynamic power 87.11mW. This result is close to the result from the low-level estimation, i.e. 91.03mW (see the last row of Table 11).

If we use the coefficient *k* calculated for *compressor* (Eq. 3.B) the total dynamic power consumption is estimated as 97.68mW. Regardless *compressor* or *decompressor* is used as the reference, we get results of satisfactory accuracy (107% and 96% of the low-level power estimate, correspondingly).

To further verify the validity of the above approach, we estimate the power consumption for Design A with arbitrarily selected 15MHz clock (both *compressor* and *decompressor*). The total dynamic power obtained (for *k* based on the *decompressor*) is:

$$k \cdot 2911 = 86.43\,\text{mW} \qquad (6)$$

that is also close to the low-level measurements, i.e. 93.24mW (the first rows of Tables 10 and 11). The same computations repeated for *k* based on the *compressor* provide (according to Equation (6)) the total dynamic power estimate of 90.28mW. Again, the accuracy of results (regardless *compressor* or *decompressor* is used in estimations) is within 93-97% range.

Even though the achieved accuracy is far from ideal, the results are estimated at the system-level that represents the design in a very abstract manner.

It should be additionally mentioned that the XPower measures of the dynamic power are actually for the whole FPGA device. Even though both the compressor and

decompressor are of similar sizes (so that the effects of the power coming from the unused parts are similar) the results obtained by excluding the unused parts might be even better. Means for estimating the dynamic power of the unused part will be presented in Section 7.2.5.

4.3. Chapter summary

In this chapter we have confirmed that power consumption can be estimated at the system-level using the abstract complexity of the designs (hardware resources equivalents e.g. the number of latches or the number of equivalent NAND gates) and the assumed clock frequency. Obtainable results reasonably well estimate the low-level measurement of the power consumption. Thus, tedious low-level implementations can be skipped for power consumption analysis. The estimates would remain the same in both partitioned and non-partitioned implementations because no significant power overheads have been observed when a particular algorithm is decomposed into clock domains (further discussion in Chapters 5 and 6).

The results of these experiments are the key to further investigations on power and energy properties of particular designs at the system-level.

CHAPTER V
RELATIONS BETWEEN SIZE OF DESIGN, CLOCK DOMAINS, AND POWER CONSUMPTION

This chapter complements the results presented in Chapter 4. We discuss relations between the number of clock domains, size of a design, chip area constraints, and dynamic power consumption. Several ideas and their experimental verifications, mostly low-level implementation results, are presented. The main goal of these experiments is to provide an evidence that it is feasible to propose a multi-domain design decomposition as a tool for the dynamic power savings, and to estimate the amount of such savings. Since the decomposition can be performed at the system-level, a laborious low-level analysis can be potentially avoided.

Section 5.1 describes the basics of conducted experiments and the concept of design decomposition into several clock domains. In Section 5.2 we investigate how dividing a design into several domains affects power consumption and how to establish the most suitable (in terms of power consumption) clock frequencies for a design of a particular size. We also investigate the issue of overheads that may result from design decompositions into clock domains.

5.1. Introduction and general assumptions

The presented experiments are based on one of the most typical data processing algorithm used in WSN applications, i.e. EWMA filter, [97], [98], [99], [100], [101], [102], [103], [104], [111], [112], [113]. The computational properties of the selected algorithm are of secondary importance and any other algorithm can be selected instead. Our designs actually consist of multiple copies of the same filter (working in parallel). This is to generate designs of a required size and hardware complexity within an FPGA chip. The hardware uniformity of the design reduces other factors (e.g. diversity of employed hardware resources) that may distort results of the conducted experiments. A parallel implementation of multiple filters is used as a testbed only. In other words, the implemented designs may not have any practical applications.

The purpose of the experiments is to determine: (1) how the power efficiency changes with the size of a design (i.e. the number of copies of EWMA filter) and (2) how

the dynamic power can be reduced when a design is decomposed into several domains driven by various clock frequencies.

Since no computational/numerical properties of the selected algorithm are exploited in the presented experiments, we believe that similar results can be expected for FPGA implementations of other data-processing algorithms.

In the experiments, we use a "shell design", Figure 9, consisting of ADC and DAC circuits (to interface the filter(s) with the external world) to which a certain number of copies of EWMA filter are added, Figure 10. Depending on the requirements, ADC/DAC and several copies of the filter are implemented in a single or several clock domains. However, there is only one ADC/DAC per the whole design (see Figure 9). The filtered signal is actually an audio signal which is in-sampled (ADC), processed by a number of filters, and finally out-sampled (DAC).

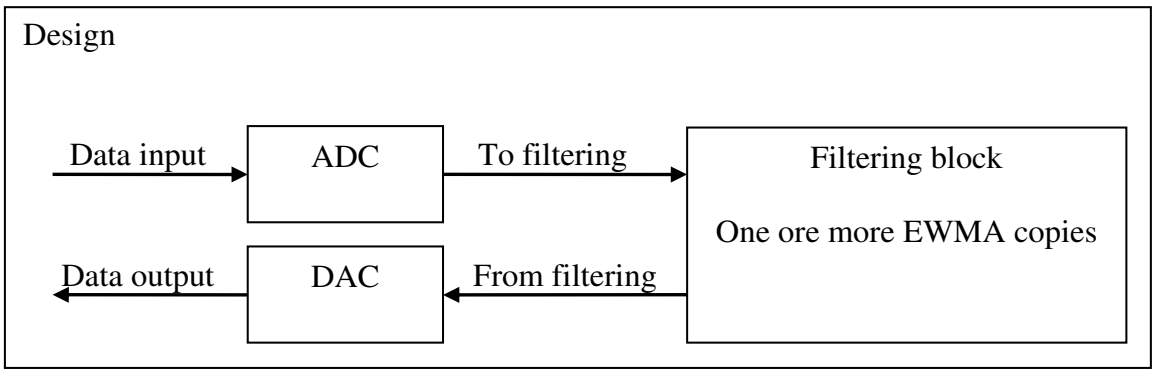

Figure 9. Design implementation ("shell design") consisting of ADC, DAC, and a certain number of EWMA filter copies; an example of a single clock domain design.

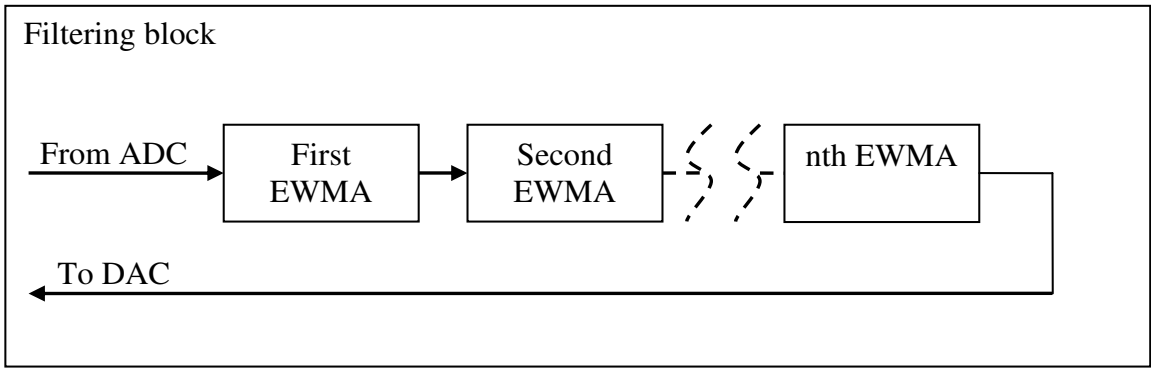

Figure 10. Functional implementation of filtering block; an example of a single clock domain design.

5.2. Experimental results

5.2.1. Power consumption and clock frequency

To obtain designs of diversified sizes, we have implemented designs with 3, 6, 9, and 24 copies of the same EWMA filter. These filters are implemented in parallel (which may not have any practical applications), and they form (from the user perspective) a single filtering block, see Figure 9. Such a filtering block is fed by data from a single ADC, and filtered data are fed back to a single DAC (see Figure 10). The designs occupy 7%, 11%, 16%, and 36% of slices available within the FPGA, respectively. These numbers include overheads of ADC and DAC circuits (that occupy 3% of the slices). As a reference, we have also implemented a design with only ADC and DAC circuits.

According to (1) the dynamic power consumption should increase linearly with changes of clock frequency. Although this is an obvious fact, the experimental confirmation is required for the sake of the subsequent experiments. In particular, we have to obtain coefficients (slopes of lines) describing the ratio between the total dynamic power consumption[2] and clock frequencies. For the designs with only ADC/DAC, and with 3, 6, 9, and 24 copies of EWMA filter, the coefficients[3] are equal to 1.0934, 2.0731, 3.1132, 4.4961, and 11.5, respectively, see Figure 11.

[2] Total dynamic power is the sum of dynamic power of clock, logic, and signal resources.
[3] Power figures are determined using dynamic power consumption estimates derived from XPower. That is, power consumption figures are based on the estimated low-level results.

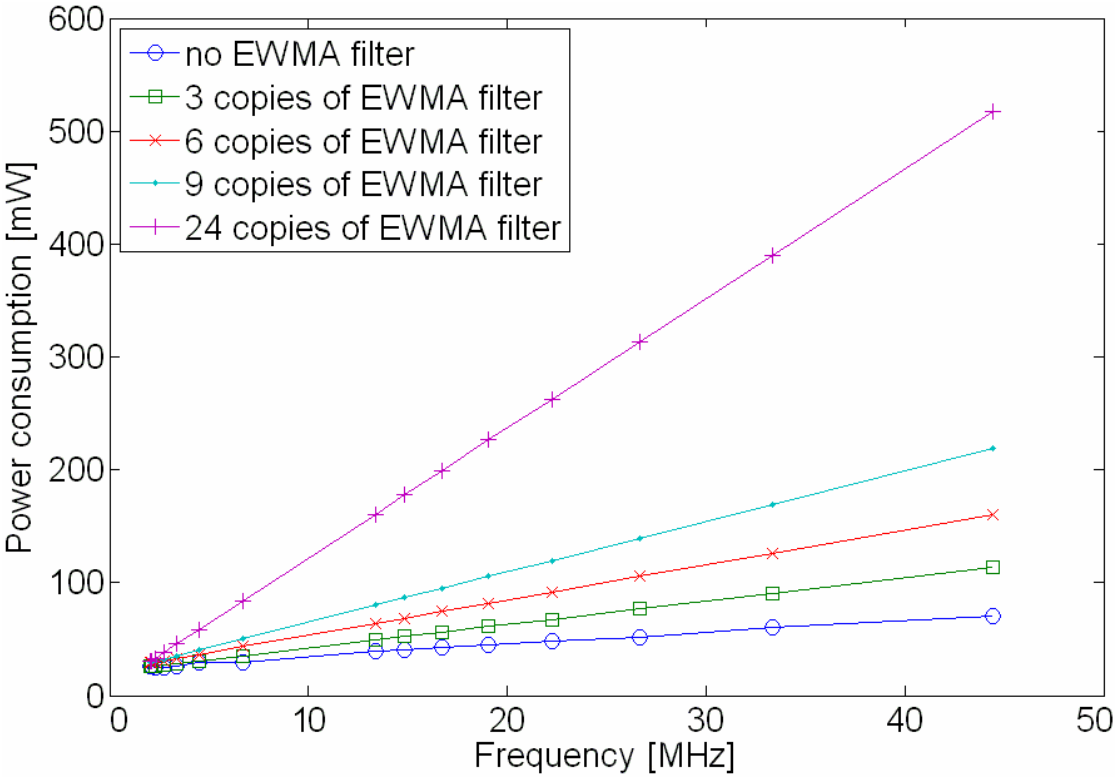

Figure 11. Relations between dynamic power consumption *versus* clock frequency and design size.

5.2.2. Multiple clock domains

This experiment investigates whether decomposition of designs into multiple clock domains (at the system-level) can be used as a tool for power consumption savings.

The multiple clock domains are implemented at a high-level of the design process, i.e. using Handel-C. We use channels or interfaces for communication between clock domains. The former mechanism is built into Handel-C and synthesized with the essential synchronization and hand-shaking (data integrity) circuits. The latter mechanism, which is also built into Handel-C, supports only signal interconnections and the hand-shaking circuit must be implemented additionally.

The single-clock domain design consists of a certain number of copies of the EWMA filter and ADC/DAC circuits. In the designs with 2, 3, 4, and 5 clock domains, the first domain contains only ADC/DAC circuits, while the other clock domains contain equal numbers of EWMA filters. All designs incorporate, altogether, the same number of EWMA filters and one ADC/DAC circuit. For example, if the single-clock domain design consists of 24 copies of EWMA filter, the two-domain design consists of 24 copies of EWMA filter in the second domain (the first domain contains only ADC/DAC circuits),

the three-domain design consists of 12 copies of EWMA filter in the second and the third domain, etc., Figures 12, 13, 14.

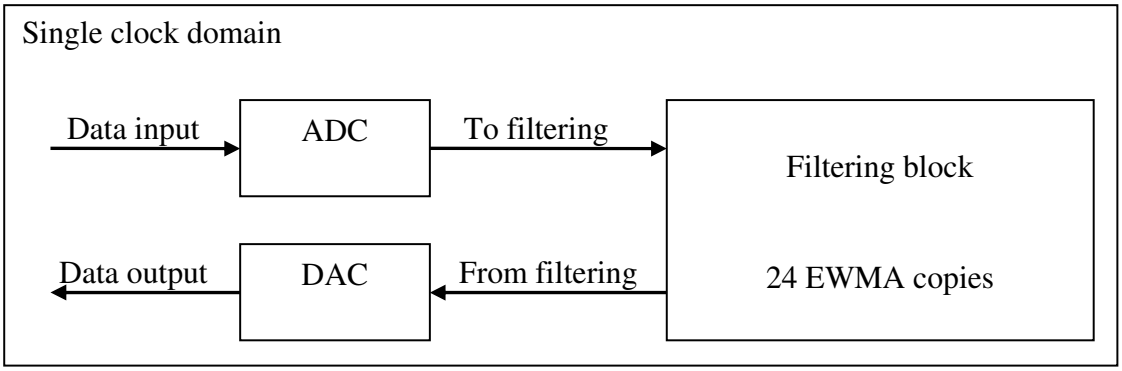

Figure 12. An example of a single clock domain design.

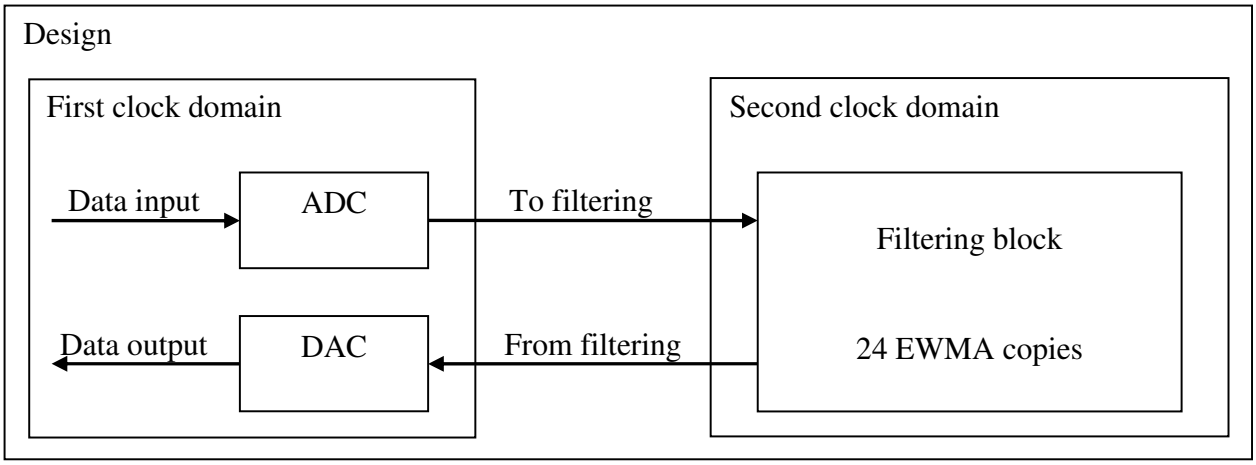

Figure 13. An example of a two clock domains design.

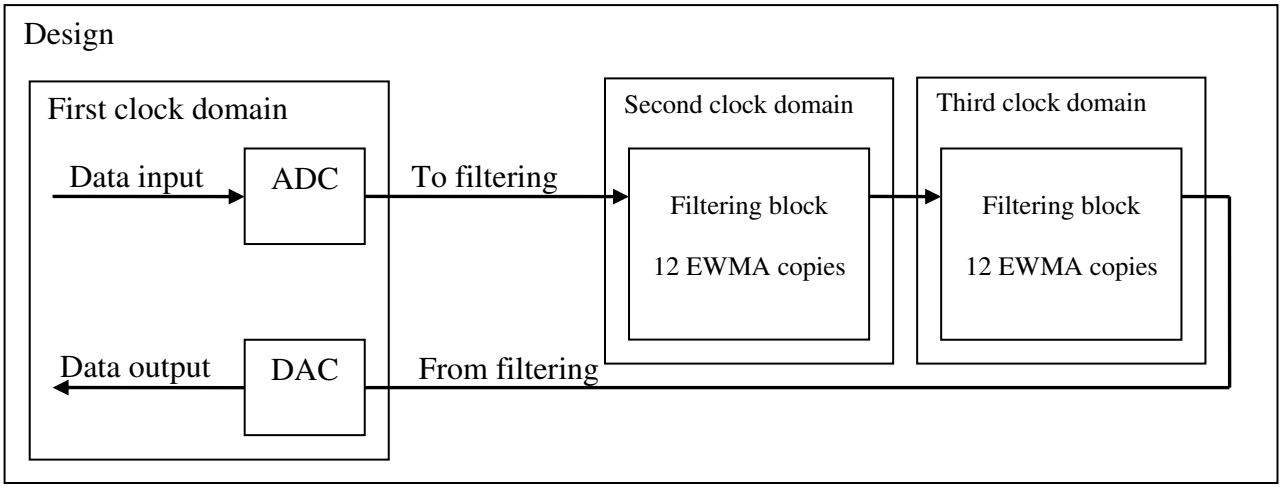

Figure 14. An example of a three clock domains design (12 EWMA copies per domain).

Clock domains are functionally (and physically) connected so that data sampled in the first clock domain are sent to the second clock domain, processed there, and sent to the next clock domain for further processing. The last clock domain also performs processing and then sends the data back to the first clock domain (ADC/DAC circuit) where the data are out-sampled. Channels and interfaces used for clock domain interconnections are 18bit wide.

To investigate power consumption and hardware resources overheads due to channel interconnections, we have implemented designs with 1 up to 5 clock domains, and with 12, 24, and 48 copies of EWMA filter altogether. Designs with 2 up to 5 clock domains have the same number of EWMA filters in each domain. Initially, all designs are clocked with the same frequency, i.e. 44.3MHz. The results of implementations are presented in Tables 12, 13, and 14.

Table 12. Design with 12 copies of EWMA filter; clock frequency 44.3MHz.

No of clock domains	Total dynamic power consumption [mW]	No of used slices (utilization)
1	296	2900 (20%)
2	310	3054 (21%)
3	315	3295 (22%)
4	291	3289 (22%)
5	296	3352 (23%)

Table 13. Design with 24 copies of EWMA filter; clock frequency 44.3MHz.

No of clock domains	Total dynamic power consumption [mW]	No of used slices (utilization)
1	533	5294 (36%)
2	486	5444 (37%)
3	536	5769 (40%)
4	498	5896 (41%)
5	495	5999 (41%)

Table 14. Design with 48 copies of EWMA filter; clock frequency 44.3MHz.

No of clock domains	Total dynamic power consumption [mW]	No of used slices (utilization)
1	833	10052 (70%)
2	847	10204 (71%)
3	874	10775 (75%)
4	890	10904 (76%)
5	868	10956 (76%)

By comparing the first row of each table to the other rows, hardware overheads due to channels can be estimated. For designs with 5 clock domains, and with 12, 24, and 48 EWMA filters, the overheads are 452, 705, and 904 slices, correspondingly. It can be noticed that even for a large design occupying 76% of available slices, hardware

overheads due to channels are rather small, i.e. less than 10%. Moreover, we can notice that there are practically no power consumption overheads, even for the largest design with 48 copies of EWMA filter, due to the number of domains. The latter result is not straightforwardly expected, since for such a high resource utilization, map-place-and-route tools might have encountered problems with achieving the desired performances. Thus, the results (for multi-domain designs) are satisfactory in a sense that decomposition of a design into multiple clock domains in general does not increase the dynamic power consumption.

In the subsequent experiments, we investigate whether a multi-clock domain decomposition can be used to reduce the power consumption. Various designs with 5 clock domains have been implemented using either channels or interfaces (see next sub-section). Functionally, these designs are the same as designs described in previous sections.

In each implementation, a different number of EWMA filters is used in an individual domain. We have implemented designs with 3, 6, and 9 EWMA filters in a domain. Additionally, the designs were implemented with various clock frequencies in each domain. The basic clock frequency is equal to 44.3MHz. This is the maximum clock frequency that the implemented circuits of EWMA filters can be clocked with.

To simplify the notation, we represent the clock frequencies of domains by the *clock frequency division factor* (CFDF). For example, CFDF = 11111 means that all domains are clocked with the same basic frequency. CFDF = 12111 means that the second domain is clocked with the basic frequency divided by 2 (i.e. downsampled to 22.15MHz) while the remaining domains are still clocked with the basic frequency.

First, we tried to estimate power consumption savings based on the results described previously. Since the total power consumption of domains interconnection is insignificant, we derived from the Figure 11 results some empirical equations[4] describing power consumption (p_i) against clock frequency for designs of various sizes. We straightforwardly assumed that the total dynamic power consumption of EWMA filters can be obtained by subtracting power consumption of the design with only ADC/DAC from power consumption of the design with the filters. For designs with only 3, 6, and 9 copies of EWMA filters, these equations are, respectively, as follows:

$$p_3(f) = 0.9797 \cdot f + a \qquad (7)$$

[4] First-degree linear equations are derived directly from the Figure 11 (by taking some its characteristic points).

$$p_6(f) = 2.0198 \cdot f + a \qquad (8)$$

$$p_9(f) = 3.4027 \cdot f + a \qquad (9)$$

where a[5] is a certain additive value.

It can be noted that the above numbers correspond to the proportionality factors obtained in Section 5.2.1 for the designs with ADC/DAC components, i.e.:

0.9797 = 2.0731-1.0934,

2.0198 = 3.1132-1.0934,

3.4027 = 4.4961-1.0934.

According to the above equations, if the clock frequency is reduced by half (i.e. from 44.3MHz to 22.15MHz) the estimated power consumption for each domain (clocked by the lower frequency) with 3, 6, and 9 copies of EWMA filter, should be reduced by 22mW, 45mW, and 75mW, respectively.

A) Reduction of power consumption – using channels

Results of the actual experiments (designs with clock domains interconnected by channels) are given in Tables 15, 16, and 17.

Table 15. Design with 3 copies of EWMA filter per domain.

CFDF	Total dynamic power consumption [mW]	Actual dynamic power saving [mW]	Estimated dynamic power saving [mW]
11111	289	-	-
12111	266	23	22
12211	236	53	44
12221	209	80	66
12222	184	105	88

Table 16. Design with 6 copies of EWMA filter per domain.

CFDF	Total dynamic power consumption [mW]	Actual dynamic power saving [mW]	Estimated dynamic power saving [mW]
11111	489	-	-
12111	472	17	45
12211	377	112	90
12221	324	165	135
12222	284	205	180

[5] Corresponds to a location of a curve in a diagram.

Table 17. Design with 9 copies of EWMA filter per domain.

CFDF	Total dynamic power consumption [mW]	Actual dynamic power saving [mW]	Estimated dynamic power saving [mW]
11111	678	-	-
12111	671	7	75
12211	522	156	150
12221	458	220	225
12222	381	297	300

As seen from the tables, results obtained in the experiment are close to the estimated power consumption savings (actually, the power saving are sometimes even higher than predicted). However, there are some cases of more erratic results for designs, with 6 and 9 copies of EWMA filter in each domain. These might be attributed to additional routing resources needed when a design occupies a larger portion of the FPGA area (designs with 6 and 9 EWMA filters in each domain that utilizes 41% and 59% of available slices, correspondingly). Nevertheless, in general a significant power consumption reduction is possible, and the power savings can be estimated at the system-level.

B) Reduction of power consumption – using interfaces

As an alternative, we also analyzed designs with multi-clock domains interconnected using interfaces. One of the reasons for using interfaces at higher-levels of the design process is to integrate a particular design with other designs that are already synthesized at lower levels (e.g. in EDIF or VHDL format) or provided as intellectual property (IP) cores. Moreover, such separately synthesized designs can be assigned chip area constraints for mapping, placing, and routing. This can be done at the lower level of design process, e.g. floorplanning. Again, we investigate how effectively the multi-clock domain can reduce the dynamic power consumption (although now the domains are interconnected by interfaces).

To conduct this experiment we implemented the same designs with 5 clock domains as previously. However, the designs are physically divided into separate modules, each representing a particular clock domain. Such decomposition was done at the high-level of design process, i.e. Handel-C, and each module was synthesized into a separate EDIF file, subsequently used for mapping, placing, and routing. Moreover, we use a top-level design as a wrapper for all modules. The top-level design is functionally equivalent to the design with no EWMA filters from the experiment in Section 5.2.1.

Hence, the wrapper performs in-sampling data, passes data to the (physically separated) second module, receives data from the last module, and out-samples data. Moreover, the wrapper defines interconnections between other clock domains (modules) by specifying relevant interfaces. However, the top-level is not involved in the inter-domain communication. The top-level module is synthesized with settings of Xilinx Virtex-II FPGA (part: xc2v3000fg676-4), and other modules are synthesized as cores (this is required by map, place, and route tools).

The area constraints are assigned only to the modules (clock domains) containing EWMA filters (see Figure 15) since they represent the large majority of the hardware resources used.

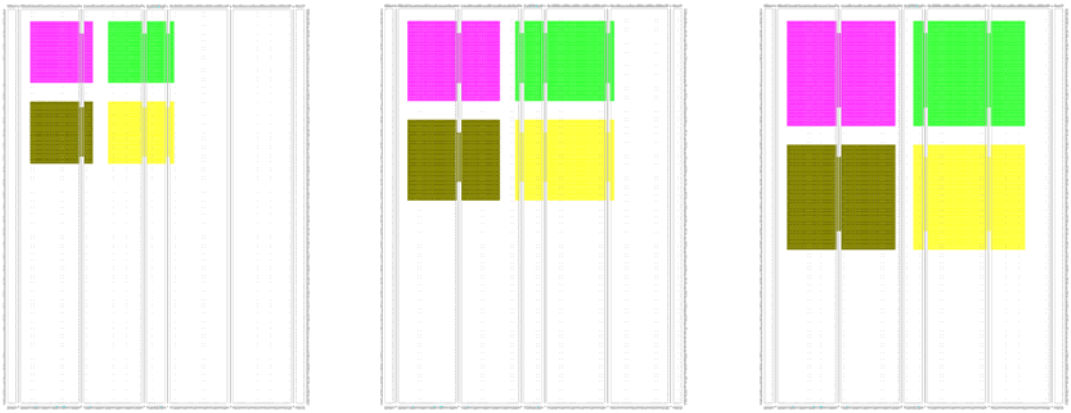

Figure 15. Area constraints for clock domains with 3, 6, and 9 copies of EWMA filter per domain, respectively.

These clock domains are located close to each other to avoid routing overheads due to data flow. However, we leave some space between them and away from the chip borders for additional routing within a particular clock domain. Moreover, we assign area constraints of about 125-130% of the space required by a particular clock domain (in order to enable efficient routing).

Results of the experiment for the multi-domain designs with interfaces and area constraints are given in Tables 18, 19, and 20.

Table 18. Design with 3 copies of EWMA filter per domain.

CFDF	Total dynamic power consumption [mW]	Dynamic power saving [mW]
11111	267	-
12111	235	32
12211	202	65
12221	162	105
12222	127	140

Table 19. Design with 6 copies of EWMA filter per domain.

CFDF	Total dynamic power consumption [mW]	Dynamic power saving [mW]
11111	450	-
12111	383	67
12211	311	139
12221	238	212
12222	172	278

Table 20. Design with 9 copies of EWMA filter per domain.

CFDF	Total dynamic power consumption [mW]	Dynamic power saving [mW]
11111	645	-
12111	548	97
12211	438	207
12221	330	315
12222	222	423

As we see from the tables, results obtained in this experiment, for all sizes of designs, are much better than the estimated power consumption savings from Part (A), i.e. when channels are used. It should be highlighted, however, that interfaces are used with the domains individually optimized at the hardware level (including, for examples, area constraints). Such mechanisms cannot be used at the system level so the eventual power savings are obviously better than by using channels (and the system-level approach). Qualitatively, nevertheless, the results follow the same as in case of using channels.

5.3. Chapter summary

In this chapter we have presented multi-clock domains approach as a tool for power consumption savings. We have also shown that such power consumption savings can be estimated at the system-level with the precision accurate enough to replace the low-level estimates. However, power consumption savings by employing multi-domains approach can be often better than estimated[6] by incorporating low-level techniques (e.g. precompiled individual domains interconnected by interfaces).

By estimating dynamic power consumption at the system-level, tedious low-level implementations can be skipped for power consumption analysis. Therefore, our further experiments (presented in the following chapters) are based mainly on the system-level estimates. Such a system-level approach will allow investigating particular problems in significantly shorter time and more variants of such problems can be considered.

[6] Derived from empirical equations.

CHAPTER VI
PARALLEL PARTITIONING OF ALGORITHMS

This chapter presents some ideas and their experimental verifications on parallel partitioning of FPGA-implemented algorithms from the power efficiency perspective. We show that with such an approach, a certain level of power awareness can be incorporated into the system-level of the design process. In other words, this chapter shows how power savings can be achieved by analyzing designs partitioned at higher levels of the design process. We primarily exploit the concept of multiple clock domains. The system-level estimates of the power requirements and savings are directly based on the results presented in Chapters 4 and 5, i.e. the dynamic power is assumed proportional to the size of designs represented by the number of equivalent gates, and to the frequency of the system clock.

Section 6.1 is an introduction to conducted experiments; the corresponding results are presented in Section 6.2.

6.1. Introduction and general assumptions

The conducted experiments are based on selected typical data reduction algorithms, i.e. *Huffman coding*, *Arithmetic coding*, [120], [121], [126], [133]. The first one is a typical algorithm used in WSN applications. The latter one is chosen for its prospective applicability to WSN's, as explained below.

Huffman coding is a popular algorithm for embedded systems due to its simplicity, low hardware and performance requirements, and the nature of data to be stored or communicated, [133]. However, problems may appear if the alphabet of the source data is not big enough, if the probabilities are highly skewed, or if a binary alphabet is used (e.g. detection, classification, tracking, etc.) in the worst case, [120], [121]. This problem can be partially solved by building the extended alphabet (that has symbols grouped in blocks of two and more).

Arithmetic coding is a preferred choice that assigns codewords to particular sequences without generating codes for all sequences of the corresponding length (as *Huffman coding* does), [120], [121]. However, *Arithmetic coding* is much more tedious to implement. Thus, even if *Arithmetic coding* is a good candidate for WSN's, it has not been (to our knowledge) implemented yet in such applications. Nevertheless, our

experiments show that *Arithmetic coding* may be a feasible choice for FPGA-based applications.

The general assumption is that the algorithms can be decomposed into isolated fragments that can be run independently, i.e. without any data exchange during the execution cycle. Data compression algorithms, that can be decomposed into *compressors* and *decompressors*, are quite a natural selection, but the experiments go further – the actual partitioning is done separately for the *compressors* and *decompressors* of the implemented algorithms.

It is envisaged that qualitatively similar results can be obtained for any other algorithms suitable for such a partitioning.

In this chapter we focus on partitioning into fragments that can be run simultaneously (parallel partitioning). The alternative scenario, where algorithms are partitioned into fragments executed sequentially is discussed in Chapter 7.

6.2. Experiments

6.2.1. Algorithm partitioning into parallel domains

The objective of the algorithm partitioning is to reduce the dynamic power of FPGA implementations. When an algorithm is partitioned into autonomous fragments, each fragment can be implemented as a separate domain and the corresponding pieces of hardware can perform simultaneously. We investigate how to select the optimum clock frequencies for each domain so that the overall processing timing is preserved and the dynamic power consumption is minimized.

In the selected algorithms, i.e. *Huffman coding* and *Arithmetic coding*, compressing and decompressing are independent operations run at unrelated timing regimes so that any clock frequencies can be applied to both parts (resulting in the corresponding dynamic power consumption). This problem was analysed (for *Huffman coding*) in Chapter 4.

In this chapter we discuss partitioning of *compressors* and *decompressors* of both algorithms. The *compressor* and *decompressor* are partitioned (each into two domains performing simultaneously - more details in Section 6.2.2) with a straightforward assumption that the operation (compression or decompression) should be completed within predefined time constraints. We show that based on the system-level

characteristics of the domains, the optimum clock frequencies can be proposed for the domains in order to minimize the dynamic power.

The system-level characteristics are: (A) the hardware resource estimates and (B) the processing time estimates.

A) Hardware resources estimate

When a domain is isolated from an algorithm, the domain is separately compiled and synthesized at the system-level to obtain the equivalent number of NAND gates (or latches – in this chapter we arbitrarily use the number of gates as the hardware complexity measure). The complexity (i.e. the equivalent number of NAND gates) of the remaining part of the algorithm is computed straightforwardly by subtracting the number of gates of the isolated domain from the whole algorithm.

This is a very simple approach and, generally, the results do not depend on which domain is isolated, i.e. in case of two domains the complexity of any domain is practically the same no matter if it is isolated or considered "the remaining part of the algorithm".

B) Processing time estimate

A clock cycle is the basic unit of the time estimate at the system-level. When an algorithm is partitioned, each domain has its own execution time (number of clock cycles). Since the domains are run in parallel, the overall processing time is determined by the longest execution time. In case of two domains, it will be either the processing time of the isolated domain or the time of the remaining part of the algorithm. If the algorithm should be executed within a certain time $\boldsymbol{T}$, the system clock frequency would be determined as the ratio of the overall processing time (i.e. the number of clock cycles) over the time $\boldsymbol{T}$.

6.2.2. Implementations details

To deal with certain limitations of DK Design Suite, we use samples of 32 elements, and sequences of 4 symbols for *Arithmetic coding*. These values correspond to 1 second of data gathered by some actual sensors (e.g. the typical sampling frequency for magnetometers used in WSN's is approximately 10-50Hz) and thus they are reasonable,

see [102], [108], [109], [112]. We also arbitrarily decide that the width of processed data is 10bits, i.e. a typical width of ADC used in WSN applications, [15].

Memories required by data reduction algorithms are implemented within the FPGA so that large capacitances of external connections are avoided. Such an approach does not distort the results since the FPGA-based memory is used only for the essential operations, and we do not store more than one sample of input or output data.

Partitioning is done arbitrarily based on the algorithm's structure. In general, the algorithm partitioning would be based on the individual properties of algorithms and this issue (that belongs, in our opinion, to the theory of algorithms) is not investigated in the thesis.

A) Huffman coding

The *compressor* of *Huffman coding* consists of *BuildHuffTree* (building Huffman tree), *BuildHuffCode* (building Huffman code), and *CodeSendDirect* (encoding symbols) functions.

BuildHuffTree and *BuildHuffCode* are executed for every new sample, and *CodeSendDirect* is executed for every new symbol to be encoded. Therefore, we decided to put *BuildHuffTree* and *BuildHuffCode* together in the same (secondary) clock domain, and *CodeSendDirect* in the main clock domain. Moreover, we decided to implement a memory to store samples of input data (*SampleArray*) and the symbol code table (*SymbolCode*; for symbol encoding) in the main clock domain (together with *CodeSendDirect* as the data are mostly accessed by *CodeSendDirect*). Hence, *BuildHuffTree* and *BuildHuffCode* have to access *SampleArray* and *SymbolCode* through channels. The block diagram of the clock domain partitioning of the *Huffman coding compressor* is presented in Figure 16.

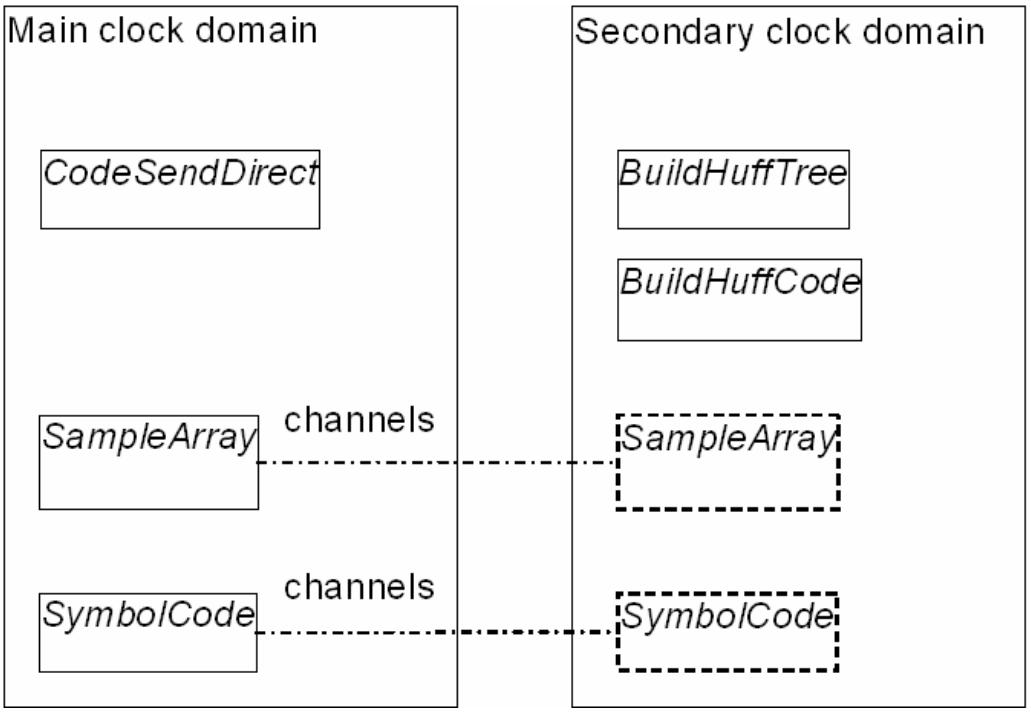

Figure 16. Block diagram of *Huffman coding compressor*.

The *decompressor* of *Huffman coding* consists of *BuildHuffTree* (building Huffman tree; it differs from *BuildHuffTree* used in *compressor*) and *CodeGet* (decoding symbols) functions. The first function is executed for each new sample and the latter one is executed for each new code to be decoded into a symbol. Hence, they are in different clock domains. Moreover, we decided to implement a memory to store statistics of input data (*AlphArray*) and internal node structures of binary tree (*InterNodeArray*) in the same clock domain as *CodeGet*. Therefore, *BuildHuffTree* has to access *AlphArray* and *InterNodeArray* through channels. The block diagram of the clock domain partitioning of the *Huffman coding decompressor* is shown in Figure 17.

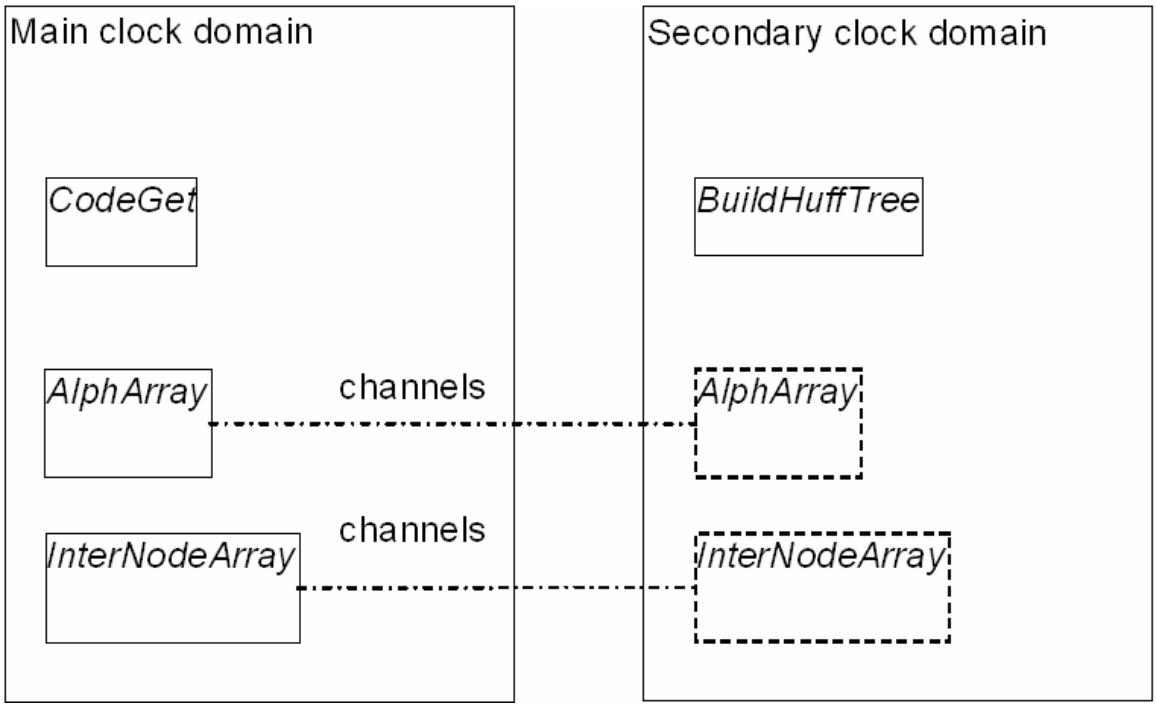

Figure 17. Block diagram of *Huffman coding decompressor*.

The system-level hardware complexity and processing times of the designs are given in Tables 21 and 22.

Table 21. *Huffman coding* (*compressor*) – hardware resources and processing time.

	[NAND gates equivalent]	**Clock cycles**
Complete compressor	214634	-
Main clock domain	79195	1155
Secondary clock domain	135439	20352

Table 22. *Huffman coding* (*decompressor*) – hardware resources and processing time.

	[NAND gates equivalent]	**Clock cycles**
Complete decompressor	130724	-
Main clock domain	45737	655
Secondary clock domain	84987	14666

B) Arithmetic coding

We have implemented the *compressor* of *Arithmetic coding* using the following functions: *vasPrbCount* (building a probabilistic model of sample data), *vasCDFCount* (building a cumulative distribution function based on the probabilistic model of sample data) and *vCodeEncSeq* (encoding the alphabet symbols or sequences of symbols). *VasPrbCount* and *vasCDFCount* are executed for each new sample (so they are in the same clock domain) and *vCodeEncSeq* is executed for each new symbol or sequence of

symbols to be encoded (so it is located in the other clock domain). The memories storing a sample of input data (*uiaSample*), storing the probabilistic model of input data (*asPrb*), and storing the cumulative distribution function of input data (*asCumDistFun*) are implemented in the same clock domain as *vCodeEncSeq*. Thus, *vasPrbCount* and *vasCDFCount* have to access *uiaSample*, *asPrb*, and *asCumDistFun* through channels. The block diagram of the clock domain partitioning of the *Arithmetic coding compressor* is presented in Figure 18.

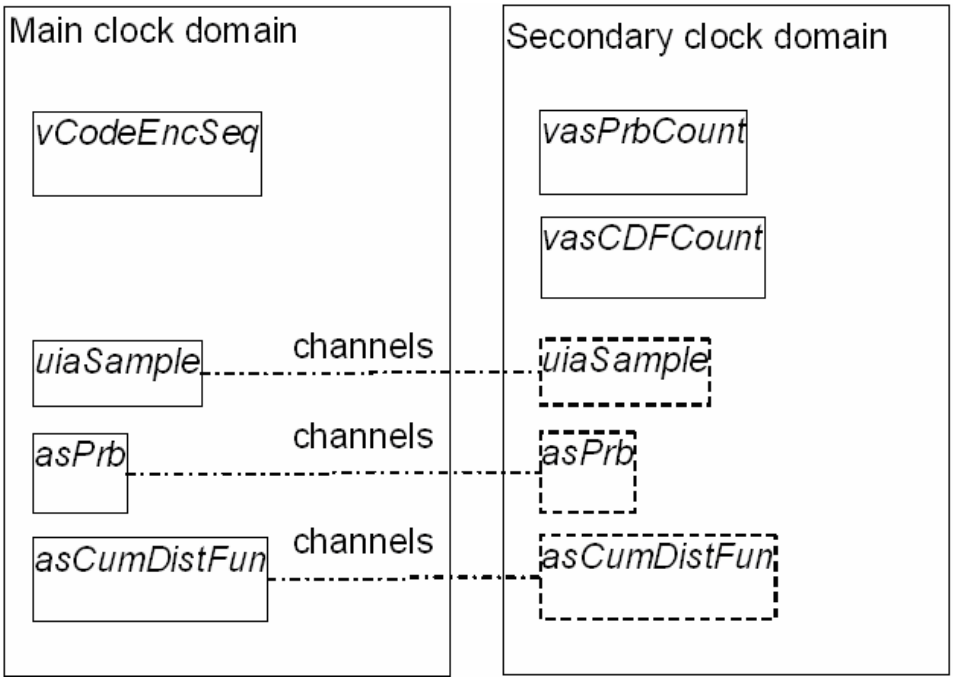

Figure 18. Block diagram of *Arithmetic coding compressor*.

The *decompressor* of our *Arithmetic coding* implementation consists of *vasCDFCount* (building the cumulative distribution based on the probabilistic model of sampled data) and *vCodeDecSeq* (decoding alphabet symbols or symbol sequences) functions. The first function is executed for each new sample and the latter one is executed for each new code to be decoded into a symbol or a sequence of symbols. Therefore, we decided to place each function in separate clock domains. Moreover, the memories storing the probabilistic model of input data (*asPrb*) and storing cumulative distribution function of input data (*asCumDistFun*) are implemented in the same clock domain as *vCodeDecSeq*. Hence, *vasCDFCount* has to access *asPrb* and *asCumDistFun* through channels. The block diagram of the clock domain partitioning of the *Arithmetic coding decompressor* is presented in Figure 19.

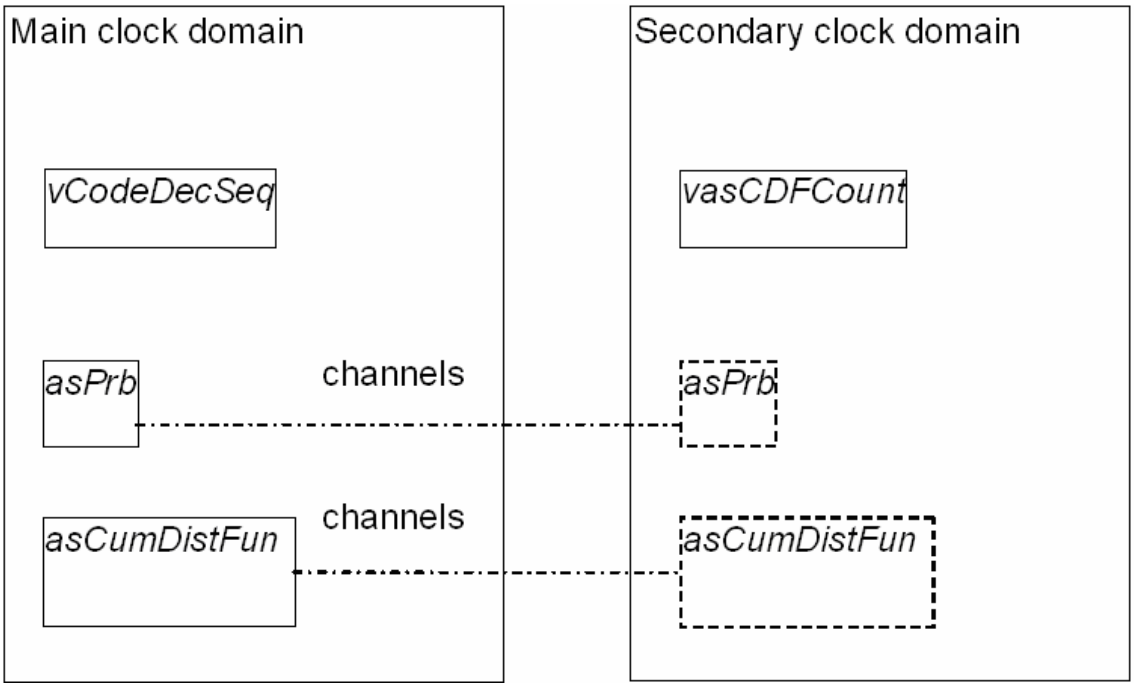

Figure 19. Block diagram of *Arithmetic coding decompressor*.

The system-level hardware complexity and processing time of the designs are shown in Tables 23 and 24.

Table 23. *Arithmetic coding* (*compressor*) – hardware resources and processing time.

	[NAND gates equivalent]	Clock cycles
Complete compressor	231666	-
Main clock domain	225047	3961
Secondary clock domain	6619	5350

Table 24. *Arithmetic coding* (*decompressor*) – hardware resources and processing time.

	[NAND gates equivalent]	Clock cycles
Complete decompressor	303114	-
Main clock domain	299679	3418
Secondary clock domain	3435	3204

The presented partitioning of *compressors* and *decompressors* can be considered an example and a guideline how other FPGA-implemented algorithms should be partitioned into simultaneously run domains.

C) Complexity of the design and channels overheads

To estimate hardware resources overheads due to inter-domain communication, we have implemented the designs consisting of the channels only. Actually, we have implemented designs with channels that might be required to transfer more data (data

samples of 32, 128, and 512 elements). Results, i.e. the equivalent numbers of NAND gates, are presented in Tables 25 and 26.

Table 25. *Huffman coding* – channel overheads.

Sample size	32	128	512
Compressor [NAND gates equivalent]	216	228	240
Decompressor [NAND gates equivalent]	680	764	848

Table 26. *Arithmetic coding* – channel overheads.

Sample size	32	128	512
Compressor [NAND gates equivalent]	728	812	896
Decompressor [NAND gates equivalent]	478	538	598

Tables 25 and 26 show that the channel overheads are insignificant compared to the *compressor/decompressor* logic complexity (given in Tables 21 to 24). They are 0.24%, 0.40%, 0.22%, and 0.17%, of the *compressor/decompressor* logic of *Huffman* and *Arithmetic coding*, correspondingly.

Therefore, the channel hardware overheads are actually added to the complexity of the domains (arbitrarily assuming they split equally between both domains).

6.2.3. Results

Results of algorithm partitioning (using a two-domain partitioning described in Section 6.2.2) are presented in Tables 21 and 22 (*Huffman coding*) and in Tables 23 and 24 (*Arithmetic coding*).

In both algorithms, the longest processing time of a domain defines the nominal clock frequency for the whole design (corresponding to the maximum acceptable processing time that cannot be exceeded). Any reduction of the clock frequency in an individual domain would correspondingly reduce the dynamic power (according to (1)).

A) Huffman coding

As shown in Table 21, the main domain of *Huffman coding compressor* needs only 1,155 clock cycles of execution time while the secondary domain requires 20,352 cycles (see Figure 16 for the domain details). When both domains are driven by the same

clock frequency (i.e. the *compressor* design is effectively not partitioned) the overall power consumption can be estimated (using some non-descriptive units (NDU's)) as:

$$(79{,}195+135{,}439)*1 = 214{,}634\text{NDU} \quad (10)$$

However, the main domain can be run at the frequency equal to only 5.67% of the nominal clock frequency (1,155/20,352 = 0.0567) and can still complete its operation within the same time as the secondary domain. Thus, the power consumption for the main domain can be reduced to:

$$79{,}195*0.0567 = 4{,}490.36\text{NDU} \quad (11)$$

while the secondary domain (which is driven by the original clock frequency) needs:

$$135{,}439*1=135{,}439\text{NDU} \quad (12)$$

Therefore, the total power consumed by the partitioned design is equal to:

$$4{,}490.36 + 135{,}439 = 139{,}929.36\text{NDU} \quad (13)$$

We can notice that (13) is only 65.19% of the original 214,634NDU (see (10)) of the non-partitioned design. 34.81% of the dynamic power is saved.

Following the same methodology for the *decompressor* of *Huffman coding* (see Table 22, and Figure 17 for the domain details) we conclude that 84,987 equivalent gates of the secondary domain should be driven by the nominal clock frequency while 45,737 gates of the main domain need only 4.47% of that frequency (655/14,666 = 0.0447). Therefore, the power consumption of the partitioned design can be expressed as:

$$45{,}737*0.0447 + 84{,}987*1=87{,}031.44\text{NDU} \quad (14)$$

which is 66.58% of the power needed by the non-partitioned implementation of the *decompressor* that needs 130,724NDU of the dynamic power. In this case 33.42% of the dynamic power has been saved by reducing the clock frequency of the main domain.

B) Arithmetic coding

Following the same methodology as for the *Huffman coding*, we can see in Table 23 (domain details in Figure 18) that for the *compressor* of *Arithmetic coding* 6,619 gates of the secondary domain should be driven by the nominal clock, while 225,047 gates of the main domain can be driven by 74.04% of the nominal frequency (3,961/5,350 = 0.7404).

Thus, the total dynamic power consumed by the whole *compressor* driven by the nominal clock (i.e. the design is effectively not partitioned) is:

$$(225{,}047+6{,}619)*1 = 231{,}666\text{NDU} \quad (15)$$

while the total power estimate for the partitioned design is:

$$225{,}047*0.7404 + 6{,}619*1 = 173{,}243.80\text{NDU} \quad (16)$$

In this case, 25.22% of power consumption has been saved compared to the non-partitioned design.

For the *Arithmetic coding decompressor* (details in Table 24 and in Figure 19), the main domain (consisting of 299,679 gates) determines the nominal clock frequency, and the secondary domain (only 3,435 gates) needs 93.74% of the frequency.

The power savings are very insignificant in this case, i.e.:

$$(299{,}679+3{,}435)*1 = 303{,}114\text{NDU} \quad (17)$$

for the non-partitioned design *versus*:

$$299{,}679*1 + 3{,}435*0.9374 = 302{,}898.97\text{NDU} \quad (18)$$

for the partitioned design. The dynamic power reduction is only 0.07%.

6.3. Chapter summary

In this chapter, we have proposed a system-level method for the dynamic power reduction in FPGA devices. The method is based on the algorithm partitioning into independent domains that are executed simultaneously. Such a decomposition of the algorithm implementation is combined with the appropriate choice of clock frequencies for the individual domains so that the dynamic power can be reduced. It should be highlighted that the proposed method does not introduce any processing delays (i.e. the whole algorithm is completed within the same time as before the partitioning) or any significant hardware overheads. The complementary problem of sequential partitioning of algorithms is discussed in Chapter 7.

It should be noted that pipelining, which functionally is a sequential decomposition of an algorithm, should be considered a parallel decomposition technique (as defined in this thesis) from the perspective of power consumption. In pipelining, individual domains of the algorithm are run simultaneously (and have to satisfy similar timing constraints as in case of parallelly decomposed algorithms) although they process data from different datasets (samples). Therefore, the results presented in this chapter apply to pipelined implementations of algorithms.

Our estimates of power savings are intentionally based only on the system-level results because Chapters 4 and 5 provide justifications for such an approach. Therefore, the estimates of dynamic power savings would be similarly obtained for a wide range of

FPGA's and other similar devices. However, the ratio between the dynamic and static power consumptions will not be, obviously, device-independent.

Our experiments are focused on two data reduction algorithms, namely *Huffman coding* and *Arithmetic coding*. Therefore, certain properties of these algorithms have been indentified as additional conclusions from the conducted experiments. In particular, contrary to the existing believes, we found that *Arithmetic coding* is a feasible candidate for FPGA-based data reduction embedded systems. In certain scenarios (small source alphabet size or with skewed probabilities) it may be even superior to *Huffman coding*.

CHAPTER VII
SEQUENTIAL ALGORITHM PARTITIONING

It is obvious that a number of data processing algorithms used in WSN applications do not continuously process data during the whole execution time. Thus, particular sections of such an algorithm (or even the whole algorithm) can be decomposed into fragments that are run sequentially.

In this chapter, we discuss such a sequential partitioning of algorithms from the perspective of power and energy efficiency. The objective is to minimize the dynamic power consumption while maintaining the overall processing time of the algorithms. The algorithms are partitioned into sequentially run clock domains (for easier analysis of results, we use only two domains in our experiments) clocked by their corresponding frequencies. Since we assume a constant overall processing time, it means that each domain is given a time slot (shorter or longer – depending on the domain frequency) but the overall duration of all slots is constant.

We show that selection of diversified clock frequencies for a sequentially partitioned design, though feasible, must be performed extremely carefully. Otherwise, power and energy efficiency of the implementation may deteriorate.

Our experiments are based on some typical data processing algorithms used in sensor nodes for sensing, detection, and classification (such as SMA-based filters, energy event SMA-based detectors, EWMA-based energy computing, data difference and data ratio EWMA-based event detectors) and for computing data characteristics (e.g. variance, estimated and definition-based moving variance, mean deviance), see [97], [98], [99], [100], [101], [102], [103], [104].

The algorithm partitioning (including hardware resources estimates) is generally analysed at the system-level (Handel-C) of the design process. However, certain power- and energy-related parameters (in particular the hardware inactivity coefficient – more in Sections 7.1.1 and 7.2.4) can be obtained only at hardware-level so that the algorithms are compiled to that level. Using the results from the system and hardware levels, we evaluate efficiency of the proposed approaches to the power and energy reduction.

General assumptions and methodologies of the experiment methodology are overviewed in Section 7.1. The actual experimental results on sequential algorithm partitioning are presented in Section 7.2. In Section 7.3, a feasibility of automated algorithm partitioning is discussed.

7.1. Assumptions and methodology

7.1.1. Algorithm partitioning

We assume that sequentially partitioned algorithms are divided into just two domains (for more domains the principles of the proposed approach are the same) D_x and D_y that can be run one after another. Therefore, the hardware resources, processing time, power and energy estimates of such a design will be estimated as follows:

A) Hardware estimates

Hardware estimates are obtained at the system-level. When a design is divided into two domains, D_x and D_y, each domain is separately compiled and synthesized so that h_x and h_y values (each representing the hardware resources for the corresponding domain in a form of equivalent NAND-gate numbers, or numbers of latches) are obtained.

B) Processing time and clock frequencies

In order to provide realistic time constraints, we should first estimate the processing time of the whole algorithm. That may depend on the application constraints and complexity of the input data, but we assume that such an estimate is available and the algorithm should be completed within *t* time. This time is divided into two slots: t_x (when D_x is run) and t_y (when D_y is run). Obviously, $t = t_x + t_y$.

Alternatively, the processing time of each domain can be represented by the number of clock cycles, i.e. c_x and c_y clock cycles are needed to run domains D_x and D_y, correspondingly.

Therefore, we can obtain the clock frequencies (f_x and f_y, respectively) needed for both domains in order to satisfy the time constraints:

$$t = t_x + t_y = \frac{c_x}{f_x} + \frac{c_y}{f_y} \qquad (19)$$

If any of these frequencies are changed (by Δf_x and Δf_y, respectively) the resulting increment Δt of the overall execution time would be expressed as follows:

$$\Delta t = \frac{-\Delta f_x \cdot c_x}{f_x \cdot (f_x + \Delta f_x)} + \frac{-\Delta f_y \cdot c_y}{f_y \cdot (f_y + \Delta f_y)} \qquad (20)$$

If the overall processing time must be preserved, the value of (20) is zero so that a simple dependency can be obtained on how to simultaneously modify clock frequencies in both domains without affecting the overall processing time:

$$\frac{\Delta f_x \cdot c_x}{f_x \cdot (f_x + \Delta f_x)} = \frac{-\Delta f_y \cdot c_y}{f_y \cdot (f_y + \Delta f_y)} \qquad (21)$$

C) Power and energy estimates

According to (2) the dynamic power consumption depends on the switching activity of relevant resources. For the active hardware, the switching activity is usually approximated by a percentage of the clock frequency (e.g. 50% often assumed in this reports). However, if a particular hardware is inactive (no data is being processed) there is still some switching activity (e.g. clock inputs of the components) and dynamic power is still consumed, see [50], [148], [149], [150], [151], [152].

Therefore, two parameters are used to model the switching activity during the period of inactivity. The first one is *device inactivity coefficient* α describing the relative dynamic power consumption (calculated for the whole device) during the period of inactivity of the implemented design. The actual value of the *device inactivity coefficient* is both design-dependent and device-dependent (see Section 7.2.4). The other parameter is *design inactivity coefficient* αd which describes the relative dynamic power consumption of a design only during its period of inactivity. If *design inactivity coefficients* are known for a design partitioned into D_x and D_y domains, its dynamic power consumption is proportional to:

$$P \sim \begin{cases} h_x \cdot f_x + h_y \cdot f_y \cdot \alpha d_y & \text{during } t_x \text{ period} \\ h_x \cdot f_x \cdot \alpha d_x + h_y \cdot f_y & \text{during } t_y \text{ period} \end{cases} \qquad (22)$$

where αd_x and αd_y are the *design inactivity coefficients* separately obtained for D_x and D_y domains.

Therefore, the average power consumption of the algorithm is proportional to:

$$P_{avg} \sim \left(h_x \cdot f_x + h_y \cdot f_y \cdot \alpha d_y\right) \cdot \frac{t_x}{t} + \left(h_x \cdot f_x \cdot \alpha d_x + h_y \cdot f_y\right) \cdot \frac{t_y}{t} \qquad (23)$$

where, obviously, $t = t_x + t_y$.

Based on (22) or (23), the energy consumption during the execution cycle of the algorithm can be straightforwardly estimated as proportional to:

$$E \sim \left(h_x \cdot f_x + h_y \cdot f_y \cdot \alpha d_y\right) \cdot t_x + \left(h_x \cdot f_x \cdot \alpha d_x + h_y \cdot f_y\right) \cdot t_y = P_{avg} \cdot t \qquad (24)$$

Because we assume that the total processing time t is constant, energy minimization is equivalent to the minimization of average power consumption.

We investigate whether the power/energy consumption can be reduced by modifying the clock frequencies of both domains according to (19) and (21) so that the values of (22) and/or (24) are minimized.

7.1.2. Implementation details

We arbitrarily assume the overall processing time t of the implemented algorithms is equal to 300μs so that the algorithm can be executed within 1/1000 of the average human reaction time. This assumption is based in the envisaged typical application, i.e. WSN's. In sensor networks, the human observer should be usually notified faster than within the human reaction time. Thus, even if the communicated data are delayed 1000 times (due to communication delays, limited bandwidth and processing power, wireless protocol delays, etc.) the observer will be still notified on time.

The decided width of processed data is 10bit (a typical width of ADC used in applications of WSN's), [15]. Moreover, we arbitrarily assume the processed data samples consist of 32 elements (due to DK Design Suite certain limitations). Such a data sample length also corresponds to the amount of data captured within a second by some sensors used in WSN applications, e.g. [102], [108], [109], [112].

7.2. Results

7.2.1. Selected algorithms and their partitioning

The first two columns of Table 27 provide names and functionality of the implemented algorithms. As mentioned previously, the algorithms are partitioned into

two domains only. Although further partitioning of a particular domain might be possible, we do not discuss such a multilevel partitioning that would be too tedious to follow (and the basic mechanisms remain the same).

The first domain D_y is an isolated fragment autonomously performing a certain data processing operation (the names and functionalities of the isolated domains are also given in Table 27). The remaining part of the algorithm is considered D_x domain.

Table 27. Sequential algorithm partitioning (functional results).

Algorithm	Function of the algorithm	Name of the isolated domain D_y	Function of the isolated domain
EngDetectorTS_simple	energy SMA-based event detector	*EwmaFilter_simple*	EWMA-based filter (simple)
EnergyEwmaR	EWMA-based energy computing	*EwmaFilterR*	EWMA-based filter (Roberts)
EwmaDetectorDiffPrm	data difference EWMA-based event detector	*EwmaFilterR*	EWMA-based filter (Roberts)
EwmaDetectorRatioPrm	data ratio EWMA-based event detector	*EwmaFilterR*	EWMA-based filter (Roberts)
SmaFilter	SMA-based filter	*Mean*	Data mean
VarDef	Data variance (definition-based)	*Mean*	Data mean
MoveVarE	Data moving variance (estimated)	*VarEstim*	Data variance (estimated)
MoveVarD	Data moving variance (definition-based)	*VarDef*	Data variance (definition-based)
MeanDev	Data mean deviance	*Mean*	Data mean

7.2.2. Hardware requirements

The system-level estimates of the hardware requirements (equivalent NAND gates) for the selected algorithms and their isolated domains are given in Table 28. It should be noted that functionally identical domains may have different hardware requirements within different algorithms (e.g. *Mean* domain in *SmaFilter* and *VarDef* algorithms) since the inner algorithm complexity may vary.

Table 28. Hardware requirements for the selected algorithms (the system-level estimates).

Algorithm	Complete algorithm [NAND]	D_x domain [NAND]	D_y domain [NAND]
EngDetectorTS_simple	59894	13789	46105
EnergyEwmaR	59874	13599	46275
EwmaDetectorDiffPrm	95492	2942	92550
EwmaDetectorRatioPrm	99507	6957	92550
SmaFilter	16424	9305	7119
VarDef	45580	38527	7053
MoveVarE	188329	132502	55827
MoveVarD	97835	52766	45069
MeanDev	17082	10029	7053

7.2.3. Processing time

The total processing time of each investigated algorithm is assumed the same regardless the data pattern. Thus, if the processing time of the algorithms or of an individual domain varies (e.g. variable initialization, starting and ending condition, etc.) we assume the worst-case processing time.

Initially, we assume the same clock frequency (the basis frequency) for both domains. The basis clock frequency is computed using the following simple formula:

$$f = \frac{c}{t} \qquad (25)$$

where t is the total processing time (we assume 300µs for all algorithms) and c is the total number of clock cycles in both domains ($c = c_x + c_y$).

Processing times, numbers of clock cycles, and basis clock frequencies of the investigated algorithms are given in Table 29.

Table 29. Processing times, clock cycles, and basis clock frequencies.

Algorithm	**Clock cycles in total**	**Clock cycles of D_x domain**	**Clock cycles of D_y domain**	**Basis clock frequency [MHz]**	**Processing time for D_x domain [µs]**	**Processing time for D_y domain [µs]**
EngDetectorTS_simple	384	115	269	1.28	89.84	210.16
EnergyEwmaR	330	37	293	1.1	33.63	266.37
EwmaDetectorDiffPrm	449	173	276	1.4967	115.59	184.41
EwmaDetectorRatioPrm	449	173	276	1.4967	115.59	184.41
SmaFilter	197	66	131	0.65667	100.51	199.49
VarDef	280	187	93	0.93334	200.35	99.65
MoveVarE	255	64	191	0.85	75.29	224.71
MoveVarD	324	95	229	1.08	87.96	212.04
MeanDev	280	187	93	0.93334	200.35	99.65

7.2.4. Device inactivity coefficient

The device inactivity coefficient, α, denotes the ratio between the dynamic power used in low switching activity and high switching activity of an FPGA device with an implemented design. We arbitrarily assume that high activity is represented by the switching activity at 50% level, while the low activity denotes 0% level of the switching activity. However, any arbitrary value can be used without any loss of generality. To obtain this coefficient, a particular design is targeted to the hardware (using Xilinx ISE)

and details regarding the power consumption[7] are obtained from XPower. The algorithms are implemented as single-domain designs. However, the same experiments can be conducted for any individual domain of each design as well.

Using XPower measurements, we can only measure dynamic power for the whole device and, thus, only the *device inactivity coefficient* can be obtained. Therefore, we decided to estimate values of *design inactivity coefficients* from similar designs of various sizes (by using multiple copies of the same domain) and, additionally, from the same design driven by various clock frequencies. Then, we assume that the *design inactivity coefficients* for a given *design* (or its individual *domain*) can be approximated.

In Equations (22)-(24) we use the system-level size of domains (i.e. use equivalent NAND gates) which obviously not always corresponds to the hardware-level size (i.e. the number of slices) of the design. From the experiments (some of them presented in Chapter 4 and 5) we have concluded, nevertheless, that such accuracy is generally sufficient. However, because the following experiments are based on the hardware implementations, we use the hardware utilization percentage as the design size measure. We assume that for any size change in terms of NAND gates (the system-level) there is a correspondingly similar change in terms of occupied slices (the hardware-level).

The results of the first experiment are given in Table 30. The device inactivity coefficient is obtained for several selected designs driven by various clock frequencies. Due to algorithms implementation limitations we were not able to obtain this coefficient for the complete range of clock frequencies.

Table 30. Hardware inactivity coefficients for selected algorithms and selected clock frequencies.

Hardware utilization [%]	**Clock frequency [MHz]**					**Design**
	2	6	8	10	16	
2	0.63	0.56	0.52	0.49	0.45	*SmaFilter*
4	0.6	0.56	0.51	0.46	0.44	*MeanDev*
10	0.54	0.51	0.49	-	-	*VarDef*
4	0.65	0.59	0.57	0.55	0.49	*EnergyEwmaR*
19	0.41	0.32	0.33	-	-	*MoveVarD*
15	0.43	0.34	0.35	-	-	*MoveVarE*

The general conclusion from Table 30 is that the device inactivity coefficient decreases with the increase of either the clock frequency and/or the hardware area.

In the next experiment, we investigate the size- and frequency-dependency of the device inactivity coefficients for the designs of the same structure. We selected for this

[7] Dynamic power consumption.

experiment two algorithms, i.e. *SmaFilter_OptLngPth*, *MeanDev_OptLngPth*, that are modified versions of *SmaFilter* and *MeanDev*. These modified algorithms are optimized against the length path[8] that allows targeting hardware with significantly higher clock frequencies. To estimate different hardware area occupancies, algorithms are replicated in hardware as the array of functions (1, 8, and 16 copies of the algorithm in a particular design). The designs of various sizes are driven by the clock frequency ranging from 2 to 64MHz.

Results of this experiment for *SmaFilter_OptLngPth* and *MeanDev_OptLngPth*, are given in Tables 31-33, and 34-36, correspondingly.

Table 31. Device inactivity coefficient – 1 copy of *SmaFilter_OptLngPth* algorithm.

Clock frequency [MHz]	Device inactivity coefficient	Hardware utilization [%]
2	0.61	1
8	0.55	1
16	0.48	1
32	0.31	1
64	0.31	1

Table 32. Device inactivity coefficient – 8 copies of *SmaFilter_OptLngPth* algorithm.

Clock frequency [MHz]	Device inactivity coefficient	Hardware utilization [%]
2	0.28	7
8	0.19	7
16	0.18	7
32	0.17	7
64	0.15	7

Table 33. Device inactivity coefficient – 16 copies of *SmaFilter_OptLngPth* algorithm.

Clock frequency [MHz]	Device inactivity coefficient	Hardware utilization [%]
2	0.23	14
8	0.17	14
16	0.16	14
32	0.16	14
64	0.14	14

Table 34. Device inactivity coefficient – 1 copy of *MeanDev_OptLngPth* algorithm.

Clock frequency [MHz]	Device inactivity coefficient	Hardware utilization [%]
2	0.58	1
8	0.5	1
16	0.43	1
32	0.28	1
64	0.27	1

[8] Selection of relevant FPGA resources (at low-level) to reduce the length of combinatorial path between resources.

Table 35. Device inactivity coefficient – 8 copies of *MeanDev_OptLngPth* algorithm.

Clock frequency [MHz]	Device inactivity coefficient	Hardware utilization [%]
2	0.29	9
8	0.22	9
16	0.2	9
32	0.19	9
64	0.18	9

Table 36. Device inactivity coefficient – 16 copies of *MeanDev_OptLngPth* algorithm.

Clock frequency [MHz]	Device inactivity coefficient	Hardware utilization [%]
2	0.22	18
8	0.18	18
16	0.18	18
32	0.17	18
64	0.17	18

Device inactivity coefficient of a design can be approximately modelled using the fundamental Equations (1) and (2). We assume the following parameters characterizing both the design and the FPGA chip:

H_U: hardware utilization coefficient.

S_{AD}: the average switching activity of a working design (e.g. 0.5).

S_{UD}: the switching activity (averaged over the design area) during the inactivity period; the number is much smaller, e.g. 0.1, since only some components, e.g. those directly connected to the clock, experience any signal switching.

S_{uncf}: the switching activity averaged over the whole area of the unused part of the FPGA; this number is assumed very small (e.g. 0.01) since very few components of the unused (unconfigured) part of the FPGA are connected to any signals.

Using the above symbols, we can estimate the dynamic power consumption of a design implemented in a device as:

$$P_{DA} \sim H_U \cdot S_{AD} \cdot f + \left(1 - H_U\right) \cdot S_{uncf} \cdot f$$

where f indicates the clock frequency.

The dynamic power consumption of an inactive design would be:

$$P_{DU} \sim H_U \cdot S_{UD} \cdot f + \left(1 - H_U\right) \cdot S_{uncf} \cdot f$$

The ratio between both values is obviously the theoretical estimate of the *device inactivity coefficient* of the design:

$$\alpha = \frac{P_{DU}}{P_{DA}} = \frac{H_U \cdot S_{UD} + \left(1 - H_U\right) \cdot S_{uncf}}{H_U \cdot S_{AD} + \left(1 - H_U\right) \cdot S_{uncf}} \qquad (26)$$

Exemplary results given in Table 37 show that the device inactivity coefficient gradually decreases for larger designs (providing the other characteristics of the design do not change).

Table 37. Device inactivity coefficient changes for a hypothetical FPGA and a design of gradually increased size (assumed: $S_{AD} = 0.5$, $S_{UD} = 0.1$ and $S_{uncf} = 0.01$).

Hardware utilization coefficient H_U	**Device inactivity coefficient** α
0.01 (1%)	0.732
0.05 (5%)	0.420
0.1 (10%)	0.322
0.2 (20%)	0.259
0.5 (50%)	0.216

The content of Table 37 qualitatively corresponds to the experimental results observed in Tables 31-36, i.e. the device inactivity coefficient decreases with the design size.

However, Equation (26), which does not depend on the clock frequency, does not explain why the device inactivity coefficient diminishes for higher clock frequencies (as seen in Tables 31-36), although the increment ratio is not high (less than 2 for the frequencies 32x higher).

The proposed explanation is to assume that for higher frequencies, the value of S_{uncf} (the switching activity of the inactive design) is decreased. Equation (26) indicates that a lower value of S_{uncf}, with the other parameters unchanged, decreases the device inactivity coefficient. The calculations presented in the following Section 7.2.5 confirm this assumption, although we are not able to provide a fully credible physical explanation of this fact.

7.2.5. Design inactivity coefficient from device inactivity coefficient

Assuming the device inactivity coefficient is known for designs consisting of several replicas of the same domain, we can estimate the design inactivity coefficient for the design consisting of a single domain (the values needed in Equations (22)-(24)).

If the device inactivity coefficient is calculated using XPower for the assumed switching activity S_{AD}, there are three parameters in Equation (26), i.e. H_U (hardware utilization coefficient), S_{UD} (switching activity during the inactivity period) and S_{uncf} (switching activity of the unused part of the FPGA).

Tables 31-36 contain hardware inactivity coefficients for the designs with 1, 8 and 16 copies of the same domain (always assuming 50% switching activity S_{AD}) so that a system of three equations can be formed based on Equation (26):

$$\begin{cases} H_U(0.5\alpha_1 - S_{UD}) = (1-\alpha_1)(1-H_U)S_{uncf} \\ 8H_U(0.5\alpha_8 - S_{UD}) = (1-\alpha_8)(1-8H_U)S_{uncf} \\ 16H_U(0.5\alpha_{16} - S_{UD}) = (1-\alpha_{16})(1-16H_U)S_{uncf} \end{cases} \quad (27)$$

However, Tables 31-36 provide also the approximate values of H_U so that only two parameters are unknown and, instead of Equation (27), a system of two linear equations can be created using data from only two implementations (e.g. 1 and 16 copies of the domain):

$$\begin{cases} 0.5H_{U1}\alpha_1 = (1-\alpha_1)(1-H_{U1})S_{uncf} + H_{U1}S_{UD} \\ 0.5H_{U16}\alpha_{16} = (1-\alpha_{16})(1-H_{U16})S_{uncf} + H_{U16}S_{UD} \end{cases} \quad (28)$$

where $H_{U1}(\alpha_1)$ and $H_{U16}(\alpha_{16})$ are known hardware utilization coefficients (device inactivity coefficients) for both implementations

Using either of the above systems of equations, the value of S_{UD} (switching activity during the inactivity period) can be calculated, from which the *design inactivity coefficient* αd is found from Equation (26) as follows:

$$\alpha d = \frac{H_U \cdot S_{UD}}{H_U \cdot S_{AD}} = \frac{S_{UD}}{S_{AD}} \quad (29)$$

As an example, we calculate the design inactivity coefficients for *SmaFilter_OptLngPth* (Alg_1) and *MeanDev_OptLngPth* (Alg_2) algorithms (shown in Tables 31-36) using Equation (28) in two variants (i.e. with 1 and 8 copies, and with 1 and 16 copies).

Table 38. Design inactivity coefficient – *SmaFilter_OptLngPth* and *MeanDev_OptLngPth* algorithms.

	SmaFilter_OptLngPth		***MeanDev_OptLngPth***	
Clock [MHz]	**1-8 copies**	**1-16 copies**	**1-8 copies**	**1-16 copies**
2	0.171	0.172	0.179	0.186
8	0.129	0.131	0.167	0.154
16	0.119	0.125	0.162	0.162
32	0.143	0.147	0.178	0.164
64	0.118	0.127	0.163	0.165

The results in Table 38 indicate that the design inactivity coefficient, in general, does not depend on the clock frequency (although some fluctuations exist, apparently resulting from very approximate estimates of the hardware utilization coefficient H_U).

When the same design is implemented in another FPGA platform, the values in Equation (29) would generally remain the same (assuming a similar design-to-device methodology) though some fluctuations of S_{UD}, determined by technological differences, may exist. Since a very accurate estimate of design inactivity coefficient is not critical (see Section 7.2.6) we assume that approximate parameters of a system-level model of sequential decompositions can be established from an exemplary implementation of a design, and subsequently used in other designs of similar structure/complexity.

Additionally, Table 39 shows the estimates of S_{uncf} (switching activity of the unused part of the FPGA) used in modelling the *device inactivity coefficient* (Section 7.2.4). The results fully correspond to the proposed explanation of lower *device inactivity coefficient* for higher clock frequencies. We postulated the lower value of S_{uncf} for higher clock frequencies, and the content of Table 39 confirms this assumption.

Table 39. S_{uncf} (switching activity of the unused part of the FPGA) estimated from *SmaFilter_OptLngPth* and *MeanDev_OptLngPth* algorithms.

	SmaFilter_OptLngPth		***MeanDev_OptLngPth***	
Clock [MHz]	**1-8 copies**	**1-16 copies**	**1-8 copies**	**1-16 copies**
2	0.0051	0.0056	0.0042	0.0047
8	0.0048	0.0048	0.0034	0.0035
16	0.0037	0.0035	0.0024	0.0024
32	0.0014	0.0013	0.0007	0.0008
64	0.0012	0.0012	0.0007	0.0007

7.2.6. Power and energy optimization

A) Average power consumption against clock frequencies changes

In this experiment we investigate how the average power (i.e. energy) of particular designs is related to the simultaneous clock frequency changes in both domains, and whether the power efficiency can be improved by such changes. We estimate the total average dynamic power P_{avg} using (23), with the clock frequencies simultaneously changing so that the total processing time is preserved (according to (21)).

The figures depict power changes expressed in some non-descriptive units (NDU's). The energy changes are not shown because for the constant total processing time the energy is always proportional to the average power (see (24)).

The values of the design (domain) inactivity coefficients are calculated using device inactivity coefficients for selected frequencies and hardware utilizations (similarly to the examples given in Tables 30-36 and Table 38).

Results of this experiment are presented in Figures 20-28.

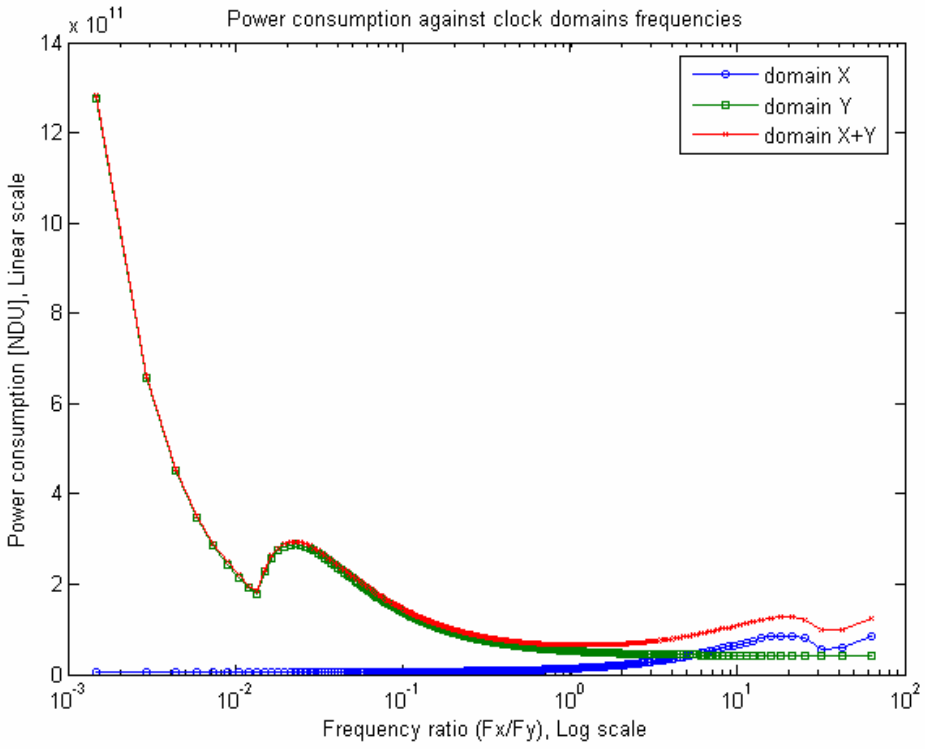

Figure 20. Average power consumption in *EngDetectorTS_simple* algorithm.

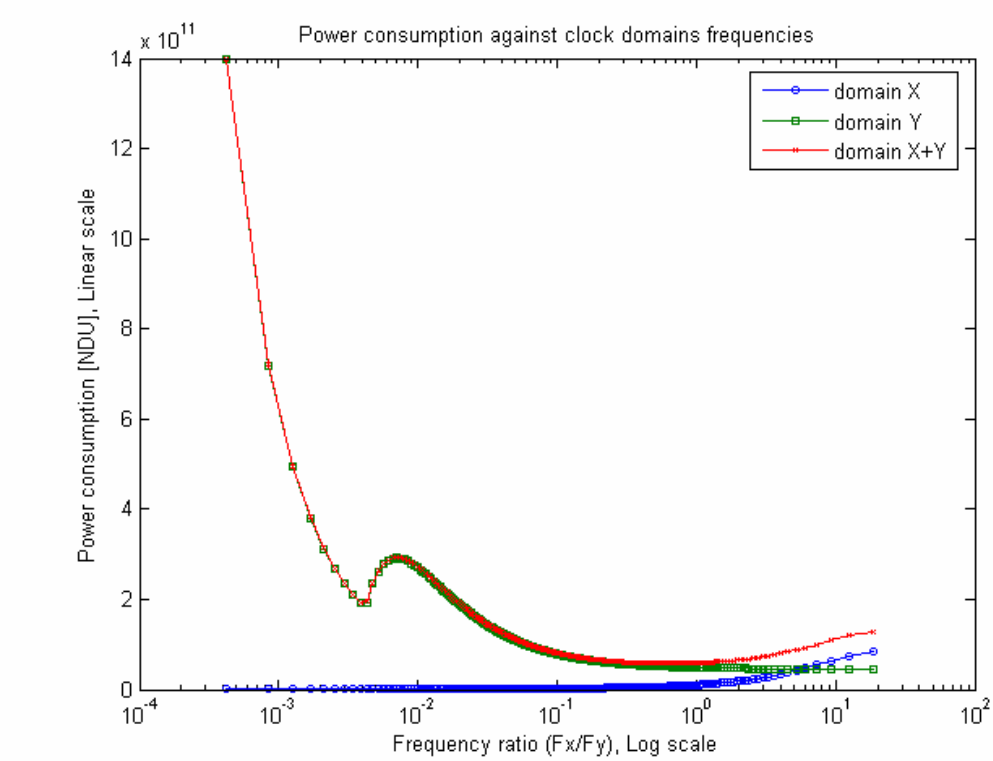

Figure 21. Average power consumption in *EnergyEwmaR* algorithm.

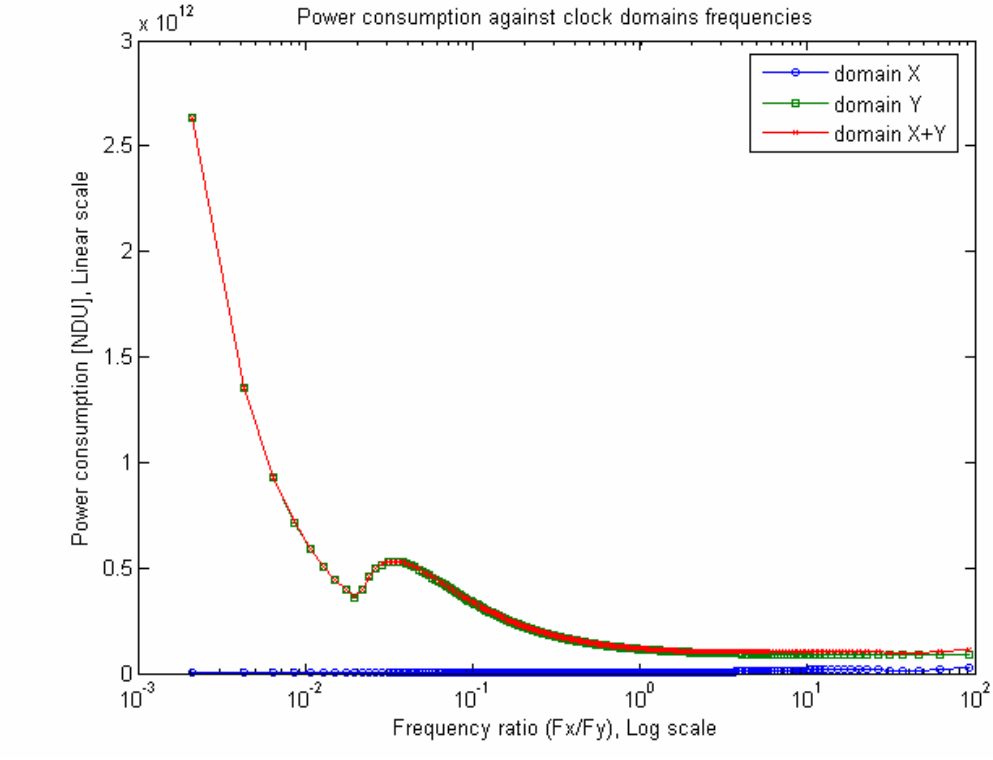

Figure 22. Average power consumption in *EwmaDetectorDiffPrm* algorithm.

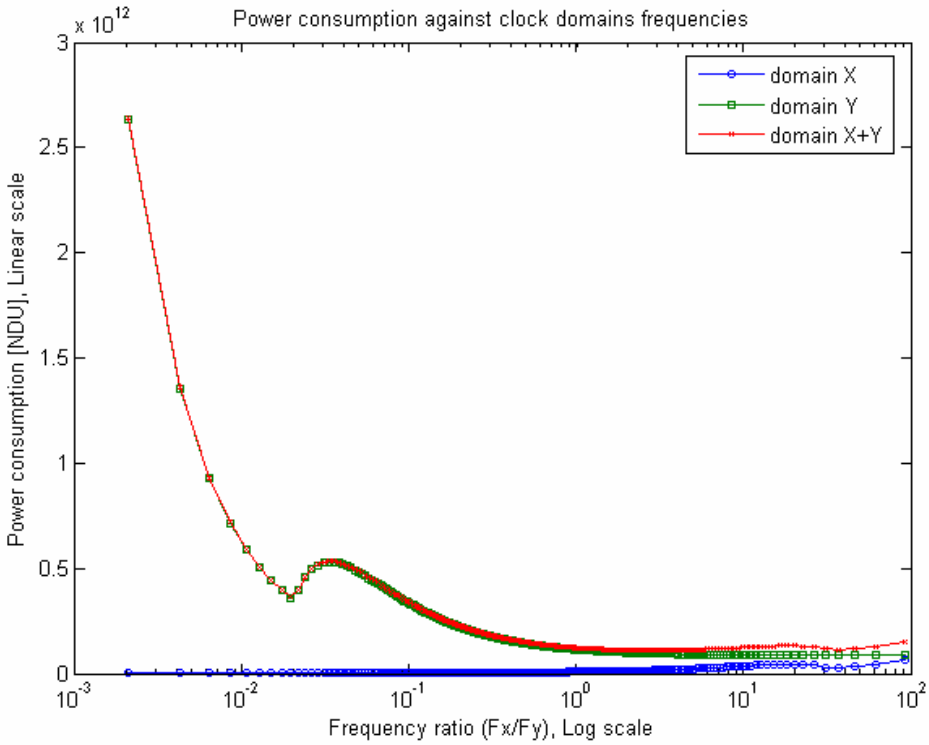

Figure 23. Average power consumption in *EwmaDetectorRatioPrm* algorithm.

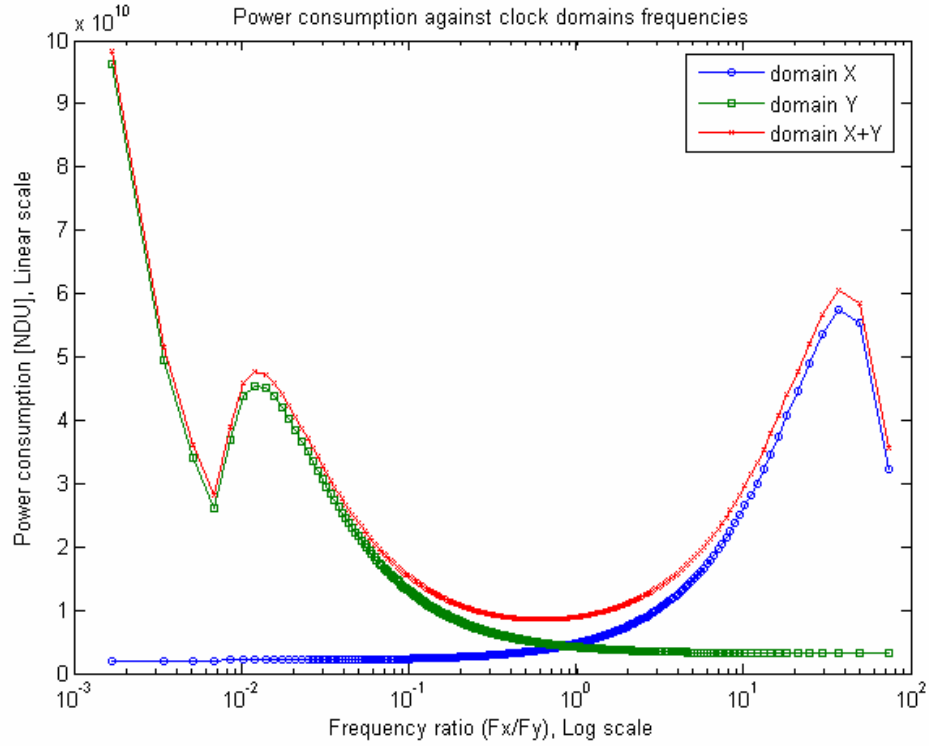

Figure 24. Average power consumption in *SmaFilter* algorithm.

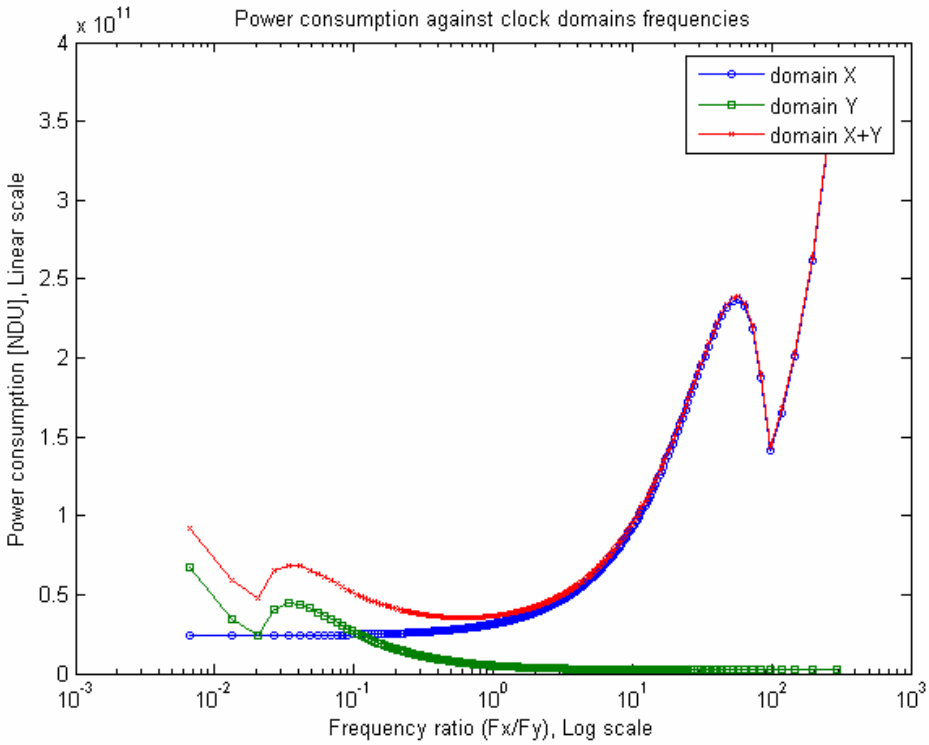

Figure 25. Average power consumption in *VarDef* algorithm.

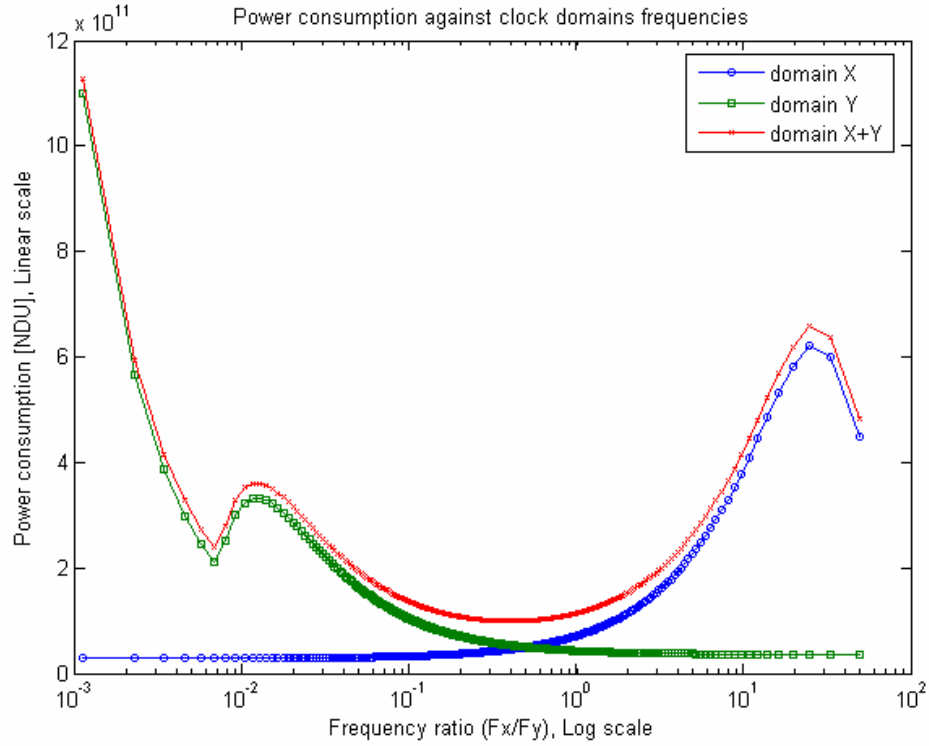

Figure 26. Average power consumption in *MoveVarE* algorithm.

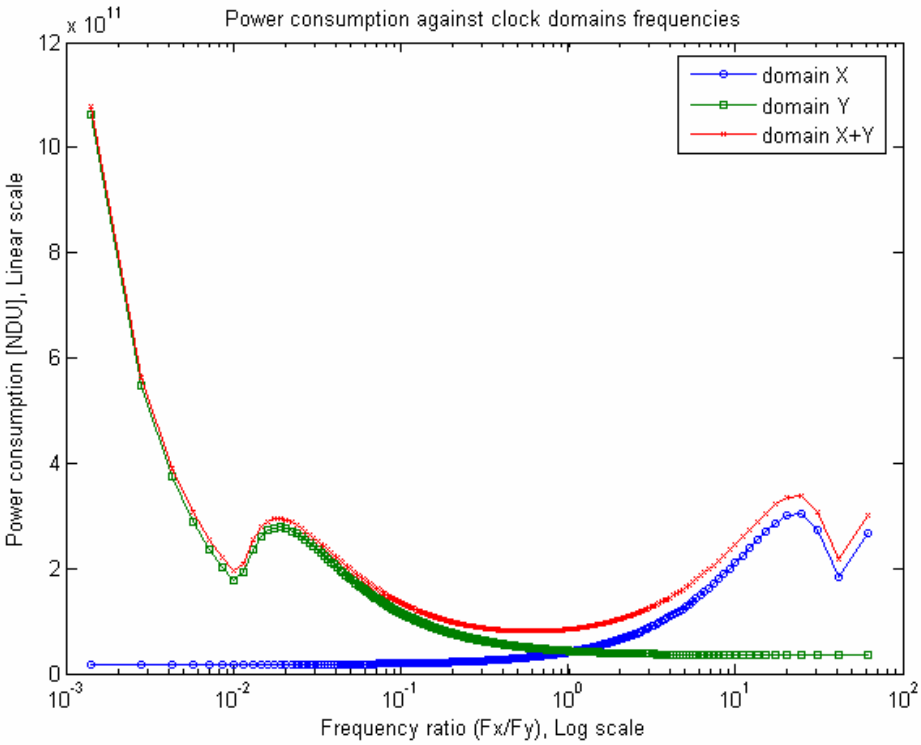

Figure 27. Average power consumption in *MoveVarD* algorithm.

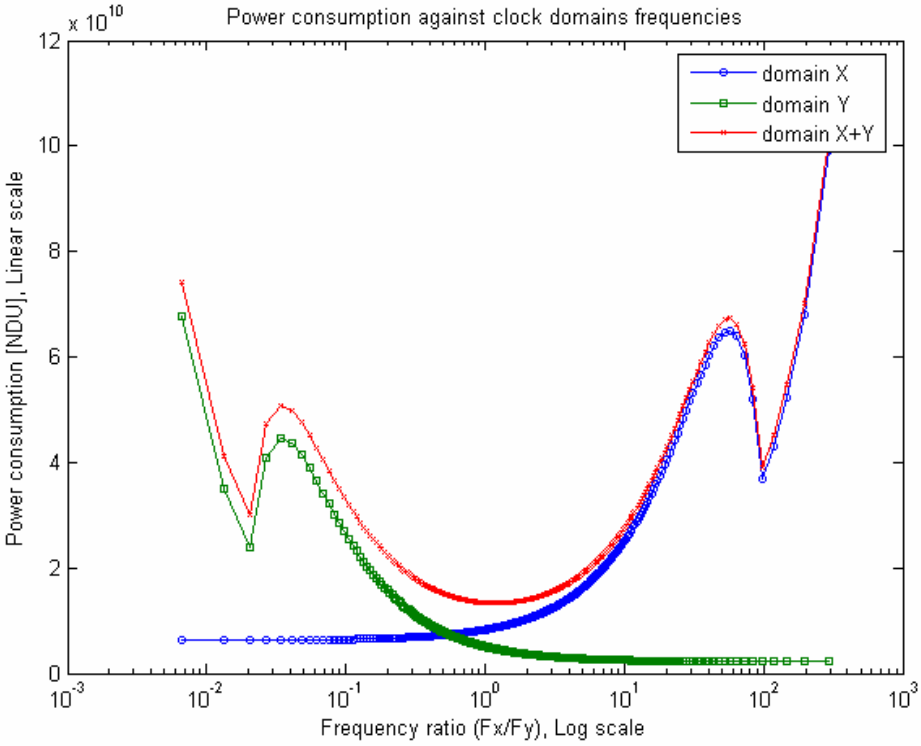

Figure 28. Average power consumption in *MeanDev* algorithm.

The most obvious conclusion from the presented figures is that the power (energy) efficiency is almost not improvable, i.e. the minimum of average power (energy)

consumption is close to the 10^0 frequency ratio point (the basis clock frequency of the algorithm partitioning). We believe this is a general property of sequential partitioning if the constant overall processing time is required or assumed.

Nevertheless, another interesting conclusion can be drawn from the presented results. It can be observed that each investigated design has two other minimum points (left and right to the central minimum at approx.10^0 frequency ratio) of the power (energy) consumption. Although power consumption is higher there than for the central minimum, we can still consider these minima significant.

If there is a need to decrease processing time of one domain (without affecting the overall processing time) we recommend selecting the clock frequencies corresponding to one of these two external minimum points. In this way, we minimize the power (energy) loses caused by a forced slowdown of one domain.

The actual locations of those external minimum points fluctuate and strongly depend on the relative sizes and processing times (numbers of clock cycles) of the domains. However, their existence is an important fact that can help to minimize energy loses in sequentially decomposed multi-domain designs with diversified clock frequencies.

B) Errors in power estimates

This experiment is related to the results given in Part A. We investigate to what extent the power estimates are sensitive to incorrect estimates of the design inactivity coefficients. Two algorithms, i.e. *SmaFilter* and *MeanDev*, are selected to illustrate the effects. The estimation error is the difference in average power between the values computed using the correct design inactivity coefficients, and the values of the coefficients taken from Table 38 (i.e. from functionally similar designs of different complexity).

The results are presented in Figures 29 to 32.

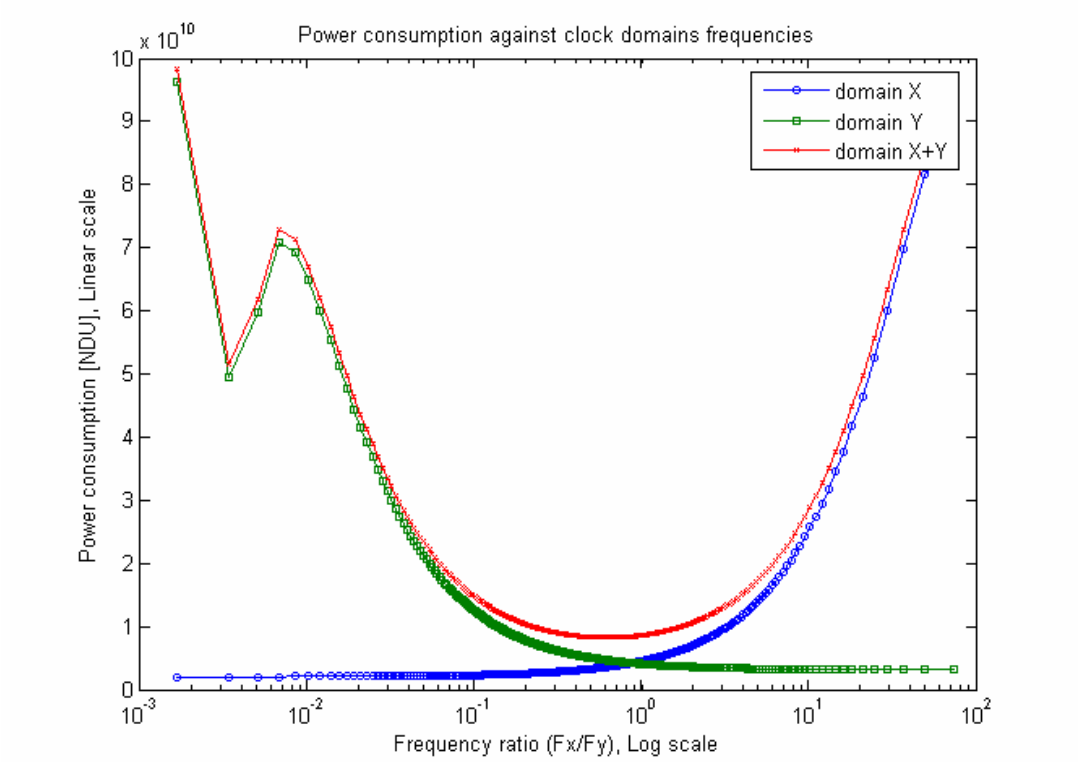

Figure 29. Average power consumption estimate of *SmaFilter* algorithm (using the design inactivity coefficient of *SmaFilter_OptLngPth*).

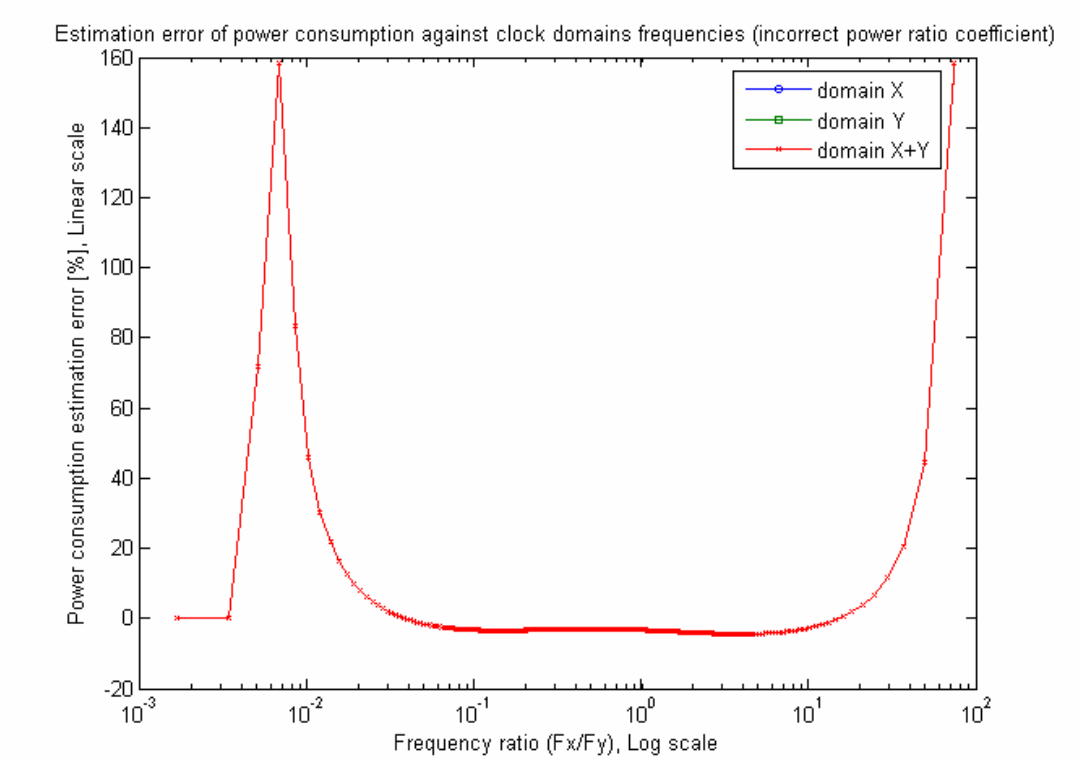

Figure 30. Estimation error of *SmaFilter* average power consumption (difference between Figures 24 and 29).

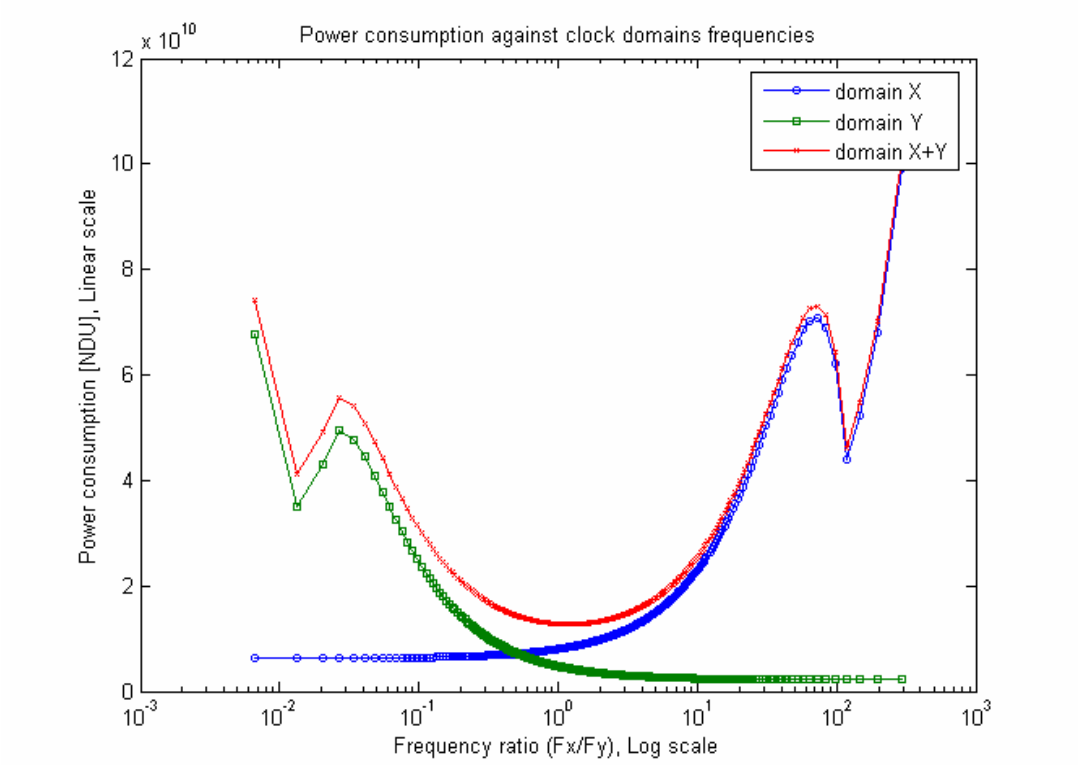

Figure 31. Average power consumption estimate of *MeanDev* algorithm (using the design inactivity coefficient of *MeanDev_OptLngPth*).

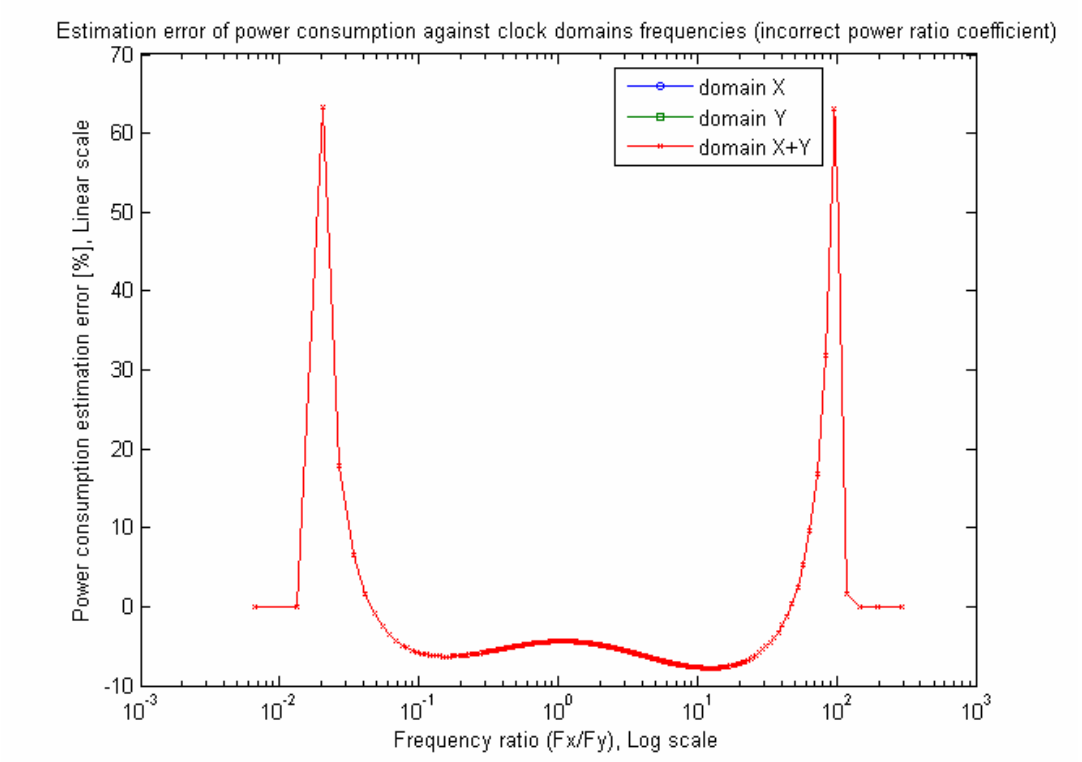

Figure 32. Estimation error of *MeanDev* average power consumption (difference between Figures 28 and 31).

If we compare Figures 24 and 29 (presenting *SmaFilter* power consumption) and Figures 28 and 31 (presenting *MeanDev* power consumption) we see that the local minima and maxima (left and right to the middle point) are at very similar coordinates. This confirms that power and energy characteristics of sequentially partitioned algorithms are not too sensitive (in terms of their profiles) to the incorrect values of design inactivity coefficients. However, the power estimation errors are distributed not so predictably. For *SmaFilter* the maxima of the estimation error are about the local maxima of power (energy) consumption, while for *MeanDev* they are at other locations. It is also not surprising. We can hardly expect very accurate power estimates if the design inactivity coefficients are not accurate, even if these inaccurate values yield qualitatively similar profiles of power consumption.

C) Average power consumption for negligible design inactivity coefficients

In this numerical experiment we investigate the situation when the design inactivity coefficients (αd) are negligible (i.e. can be approximated by zeroes). It should be noted that such a model can be used to describe FPGA devices that can temporarily switch off unused domains (nets).

We arbitrarily select two algorithms i.e. *EngDetectorTS_simple* and *SmaFilter*. For the design inactivity coefficients equal to zero, the dynamic power consumption according to (22) for a design with two clock domains (D_x, D_y) is proportional to:

$$P \sim \begin{cases} h_x \cdot f_x & \text{during } t_x \text{ period} \\ h_y \cdot f_y & \text{during } t_y \text{ period} \end{cases} \qquad (30)$$

Therefore, the average power consumption of the design is proportional to:

$$P_{avg} \sim h_x \cdot f_x \cdot \frac{t_x}{t} + h_y \cdot f_y \cdot \frac{t_y}{t} \qquad (31)$$

where, obviously, $t = t_x + t_y$.

The results for the selected algorithms are given in Figures 33 and 34. Even though the general assumptions are the same as in the previous experiment, the results are qualitatively very different from Figures 20-28.

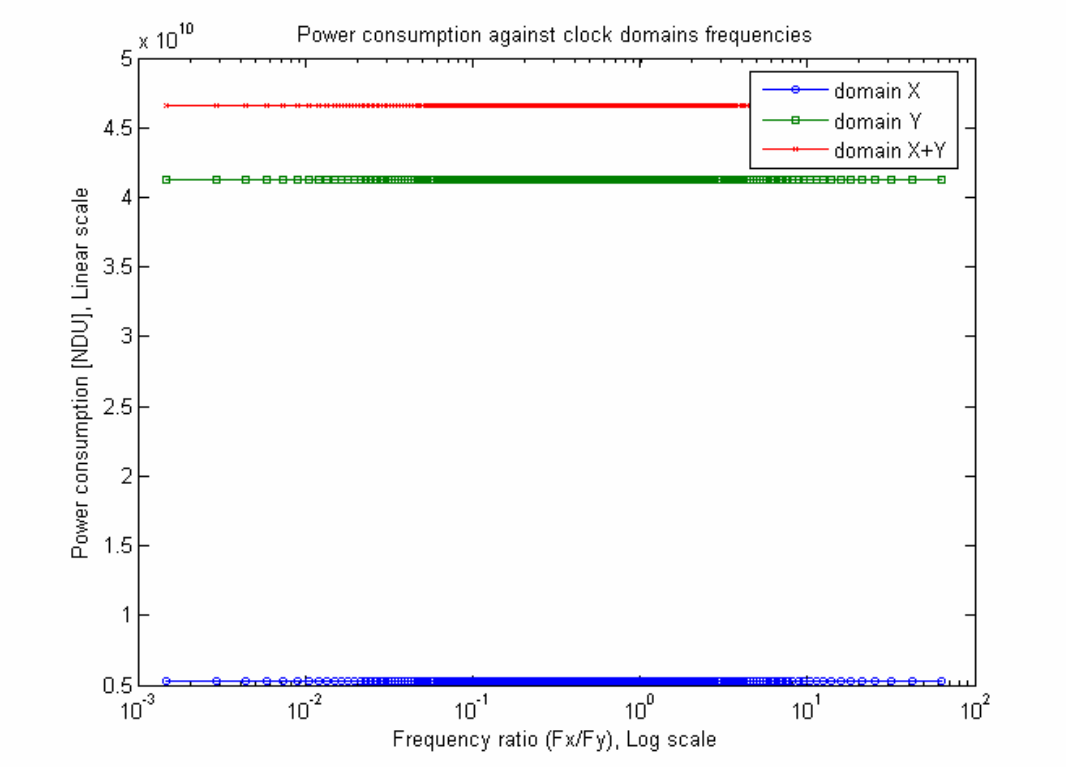

Figure 33. Average power consumption in *EngDetectorTS_simple* algorithm for $\alpha d = 0$.

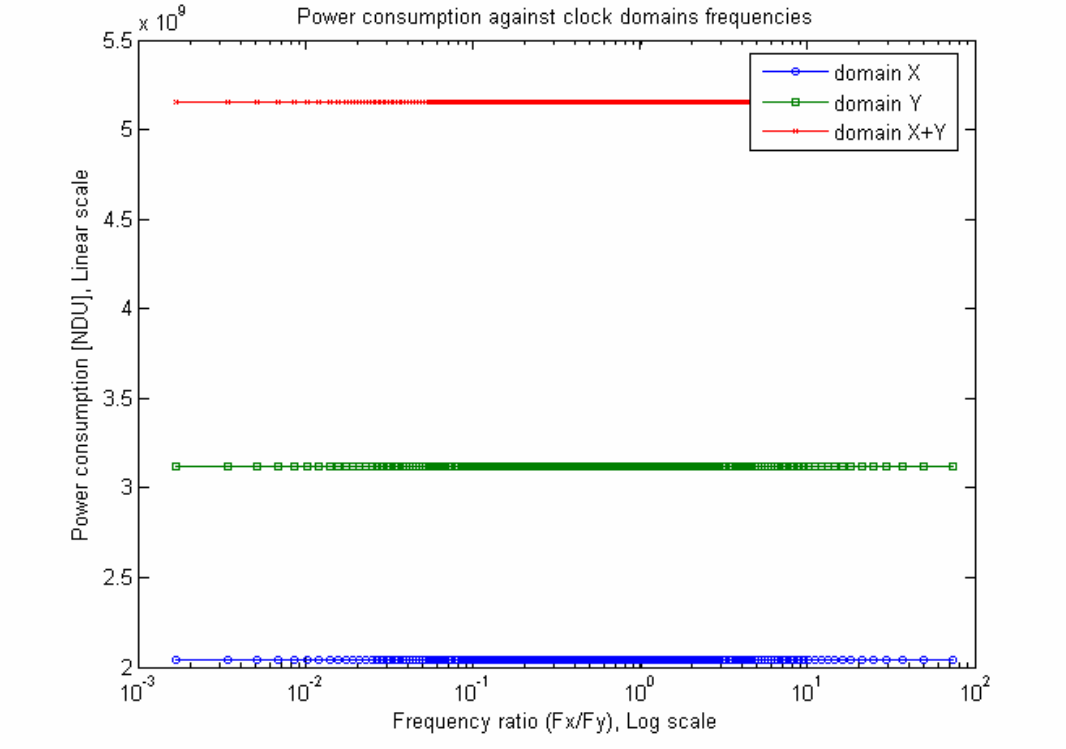

Figure 34. Average power consumption in *SmaFilter* algorithm for $\alpha d = 0$.

Clearly, the average power consumed by designs with αd equal to zero is the same regardless of the selected clock frequencies. However, it should not be neglected that the

peak dynamic power of the individual domains will change with the domains' clock frequencies changes.

Moreover, for zero *αd* coefficients the energy consumption (based on (30) and (31)) can be straightforwardly estimated as proportional to:

$$E \sim h_x \cdot f_x \cdot t_x + h_y \cdot f_y \cdot t_y = P_{avg} \cdot t \qquad (32)$$

Since $t_x = \frac{c_x}{f_x}$ and $t_y = \frac{c_y}{f_y}$, (29) can be rewritten into:

$$E \sim h_x \cdot c_x + h_y \cdot c_y \qquad (33)$$

Therefore, according to (33), the dynamic energy consumption for zero *αd* coefficients remains constant regardless of clock frequency changes (since hardware resources and numbers of clock cycles do not change).

This result can be used to build systems where the current minimization is needed (with FPGA's allowing shutdown of inactive clock domains). In other words, it is applicable to systems with energy sources of given capacity and limited power efficiency. To minimize the current drained from such an energy source with a limited power, we should follow:

$$h_x \cdot f_x = h_y \cdot f_y \qquad (34)$$

For example, if domain D_x is larger (in terms of hardware resources) than D_y, a slower clock is suggested to D_x. Then, the peak power is decreased, while the average power consumption and the energy consumption are maintained. This is somehow similar to the discussion in Chapter 6, where we suggest such a solution to improve power efficiency of parallel partitioning.

It can be noted that a similar recommendation can be proposed for two-domain designs with non-zero design inactivity coefficients, i.e. $h_x \cdot f_x(1-\alpha d_x) = h_y \cdot f_y(1-\alpha d_y)$.

7.3. Chapter summary and practical recommendations

In this chapter we have presented principles of the sequential algorithm partitioning targeting the power/energy performances improvement. We have shown the importance of a proper clock frequency selection to so partitioned designs, especially if we have to increase or reduce the execution time of a selected domain without affecting the overall processing time. Although it has been found that the sequential algorithm partitioning is not giving straightforward power/energy consumption savings,

considerably power/energy losses can be avoided by selecting proper clock frequencies for the corresponding domains.

Although the results of the presented experiments are not as spectacular as hoped, they are in our opinion useful for the practice of design implementation using FPGA (and similar) devices.

We have shown that the **device** inactivity coefficient is strongly design-dependent and frequency-dependent and, but it has been also found that:

a) Algorithms of similar structure are expected to have similar **design** inactivity coefficients regardless the clock frequency and (at least approximately) regardless the device used.

b) Power characteristics (as a function of frequency changes) of a design remain qualitatively similar even if the design inactivity coefficient is not accurately estimated.

c) The results can be used to model systems matching the properties of energy sources (i.e. a battery of a given capacity and limited power efficiency). The method could be particularly useful, if design inactivity coefficients are equal to zero (e.g. FPGA that can shutdown inactive domains, or FPGA with very low inactivity coefficients) so that inaccuracies in the coefficients do not distort the results.

Therefore, it seems feasible to introduce additional functionalities in the system-level hardware design tools. In particular, estimates of the design inactivity coefficients for standard or representative designs can be provided (although obtaining them can be a tedious process). Given such data (and the system-level estimates of the design timing and hardware complexity) users could easily investigate if and how sequential partitioning combined with diversified clock frequencies can improve power characteristics of their designs.

We believe that with some efforts at hardware-level, a useful tool can be created allowing automatic dynamic power analysis at the system-level, thus further reducing time-to-market (TTM) and development costs.

CHAPTER VIII
DATA PROCESSING AND TRANSMISSION

In this chapter we investigate the issue of energy efficiency in FPGA-based embedded systems performing both data processing and transmission (WSN's are the primary intended application). We present and discuss several experiments comparing the energy used for the local data processing *versus* the energy needed to transmit the processed data. The goal is to minimize the total energy spent on processing and transmission of the data. The energy efficiency is generally addressed at the system-level of the design process (based on the results presented in the previous chapters) but a certain number of low-level experiments have been conducted to further verify the system-level results.

In Section 8.1 we introduce the methodology of the conducted experiments and their detailed specifications. Results are discussed in Section 8.2.

8.1. Introduction

As stated in Section 2.2, typical operations performed by a sensor node are gathering, processing, and storing data produced by the node's environment. Such data are typically produced in large quantities so that the processing problems are exacerbated if the data have to be received or transmitted. In some applications (see Section 2.3) the energy needed to wirelessly transmit one bit over a certain distance may be equal to the energy used to execute 3000 CPU instructions. Therefore, reduced forms of data (e.g. their statistical characteristics) are often transmitted instead.

The crucial issue in FPGA-based embedded systems, such as sensor nodes, is how to effectively perform all operations (i.e. both processing and transmission) under tight power and communication (e.g. range, bandwidth) constraints. It is, therefore, important to determine efficient implementations of data-processing algorithms and efficient ways of communicating data. In the following sections we analyze various aspects of this problem.

We base our experiments on selected algorithms used to compute data characteristics in typical WSN applications, e.g. mean, variance (estimated and definition-based) and mean deviance (see Section 2.2.4). However, it is envisaged that qualitatively similar results would be obtained for other algorithms used in such applications.

8.1.1. Setup

The experiments are conducted using the same development tools described in previous chapters (i.e. Handel-C and DK Design Suite). According to the results presented in Chapters 5 to 7, we estimate the energy efficiency at the system-level (considering results from DK Design Suite and some theoretical derivations). A certain number of hardware-level results are used as an additional verification step. Estimates of energy related to the wireless transmission are based on characteristics of Chipcon CC1000, one of typical wireless modules in WSN applications, [153].

8.1.2. Parameters of data processing algorithms and data transmission

A typical pattern of data acquisition, processing and transmission is assumed in the experiments, i.e.:

- Data are gathered through a digital input of *INDATAWIDTH* resolution (number of bits).
- The total number of *SAMPLELENGTH* data samples is processed in a single round of the process, i.e. *INDATAWIDTH*×*SAMPLELENGTH* indicates the volume of acquired data.
- The data processing algorithm processes locally (i.e. within *SAMPLELENGTH* data samples) *SAMPLELENGTH_LCL* samples of data and converts them into a single sample of the output data. *SAMPLELENGTH_LCL* is often (e.g. for averaging filters) referred to as the processing window. Thus, the data are downsampled with the ratio:

$$\frac{SAMPLELENGTH}{SAMPLELENGTH\_LCL} \qquad (35)$$

- The assumed width of the transmitted output data is *OUTDATAWIDTH*, i.e. the overall volume of transmitted data *OUTVOL* equals:

$$OUTVOL = \frac{SAMPLELENGTH}{SAMPLELENGTH\_LCL} \cdot OUTDATAWIDTH \qquad (36)$$

We arbitrarily decide that the width of the input data (i.e. *INDATAWIDTH*) is 10bits, which is a typical width of ADC used in WSN applications, [15]. The *OUTDATAWIDTH* value depends on the selected algorithm (see Section 8.2). Moreover, we arbitrarily assume that $SAMPLELENGTH = 128$ (due to certain limitations of DK

Design Suite). This value corresponds to a typical number of samples acquired within 1 second of data gathering by some sensors (e.g. typical sampling frequency for magnetometers used in WSN is up to 100-200 Hz, [102], [108], [109], [112]).

8.1.3. Power and energy estimates

A) Total processing power

The total processing power of a particular data-processing algorithm is computed at the system-level only. According to (1) and previously discussed results, the dynamic power is proportional to the hardware area (the occupied logic) and to the number of clock cycles required to process data (if the algorithm should be completed within a predefined time, the number of clock cycles determines the clock frequency). Hence, the total processing (dynamic) power required for data processing can be expressed as:

$$P_{total} \sim hwa \cdot cc \qquad (37)$$

where *hwa* is the hardware area used (the number of the equivalent NAND gates), and *cc* is the number of clock cycles required to process data.

When necessary, the hardware-level power estimates are measured by XPower.

B) Processing energy

The processing energy is straightforwardly computed using:

$$E_{process} = P_{total} \cdot t_{exec} \qquad (38)$$

where t_{exec} represents the data processing time. Because t_{exec} is assumed fixed, the processing energy is proportional to the processing power.

C) Static power

The total processing power and the processing energy estimated at the system-level are related to the dynamic power consumption only (i.e. to the amount of hardware used – equivalent NAND gates). At hardware level, we incorporate the estimates of static power as well. For a particular FPGA chip, the static power usage is fixed so that it can be considered an offset value to the dynamic power. The bias is particularly strong for small designs, where most of the power consumption is static. For moderate and large designs,

the dynamic power may be as high as the static power or even higher, [154]. However, the static power is just a constant additive component to the total energy (assuming a constant data processing time). In particular, it does not change the ratio between (dynamic) processing energy and transmission energy.

D) Transmission energy

The total energy used to send all output data is obviously modelled as:

$$E_{send} = OUTVOL \cdot E_{bit} \qquad (39)$$

where *OUTVOL* is the overall volume of transmitted data (see Section 8.1.2) and E_{bit} is the energy required to transmit 1 bit of data.

Our exemplary estimates of E_{bit} are based on the Chipcon CC1000 specifications (for 433MHz, 3V, and 25°C) given in Table 40.

Table 40. Selected parameters of Chipcon CC1000, [153].

Data rate (max) [kbps]	76.8
TX current consumption (max), 10dBm, [mA]	26.7

According to Table 40, the energy to transmit 76800bits in 1 second with 10dBm of the transmitter power is equal to. $3V \cdot 26.7 \cdot 10^{-3} A \cdot 1s = 0.0801J$. Hence, the energy required to send 1bit within 1 second is equal to $0.0801J \div 76800 = 1.043 \cdot 10^{-6} J$ or 1.043μJ.

E) Total energy

The total energy, E_{total}, spent on data processing and transmission is simply the summation of both energies:

$$E_{total} = E_{process} + E_{send} \qquad (40)$$

8.1.4. Other general assumptions

The purpose of the experiments is to minimize the total energy consumption by establishing the best proportion between implementation complexity (processing energy) and the volume of transmitted data (transmission energy).

Data processing algorithms are implemented in the hardware-optimized manner. However, each change to a design, e.g. variable width of data, number of processed sample, etc. affects both the hardware requirements and the transmission energy. For example, by increasing *SAMPLELENGTH_LCL* we decrease the volume of output data (and often the number of clock cycles) while the hardware area increases.

The experiments are conducted in two phases, at the system-level and at the hardware-level (selected experiments only). In both phases we observe changes of the hardware resources, the clock frequency (or the number of clock cycles), and the volume of output data. We arbitrarily assume the total processing time of the implemented algorithms equals to 300μs so that the algorithm can be executed 1000 times within the average human reaction time. However, the total processing time is assumed constant even if we vary *SAMPLELENGTH_LCL* value.

At the system-level, the processing power/energy is expressed using NDU (non-descriptive units). Therefore, we also assume a certain NDU energy required to send 1bit of data in the estimates at the system-level. At the hardware-level, the proper units for power and energy (i.e. W, J) are used. Otherwise the experiments are identical.

However, due to compiler limitations, the hardware-level experiments are conducted fewer times. This level is generally used only to further verify the experiment assumptions.

8.2. Results

The selected algorithms are: *Mean* (data mean), *VarEstim* (data variance, based on variance estimation), *VarDef* (data variance, based on variance definition), and *MeanDev* (data mean deviance) data processing algorithms. As mentioned before, *INDATAWIDTH* is fixed (10 bits). *OUTDATAWIDTH* for *Mean*, *VarEstim*, *VarDef*, and *MeanDev*, are 10, 20, 21, and 11bits, correspondingly.

According to the assumptions (and in order to simplify calculations) the results (regarding power, energy, clock frequency, and the volume of output data) are normalized to 1 second. Then, curves representing the total processing power and the processing energy in the system-level estimations are obviously identical.

8.2.1. System-level experiments

For a selected algorithm, changes of the hardware area and the corresponding changes of other parameters can be obtained by varying *SAMPLELENGTH_LCL*. The system-level results for *Mean*, *VarEstim*, *VarDef*, and *MeanDev* algorithms are presented in Tables 41 and 42.

Table 41. System-level results of changing *SAMPLELENGTH_LCL* – *Mean*, *MeanDev*.

		Mean		*MeanDev*	
		NAND gates	**Clock cycles**	**NAND gates**	**Clock cycles**
SAMPLELENGTH_LCL	2	6535	704	14656	1216
	4	7568	672	17146	1120
	8	8677	656	20042	1072
	16	9862	648	23618	1048
	32	11123	644	28418	1036
	64	12460	642	35546	1030
	128	13873	641	47242	1027

Table 42. System-level results of changing *SAMPLELENGTH_LCL* – *VarEstim*, *VarDef*.

		VarEstim		*VarDef*	
		NAND gates	**Clock cycles**	**NAND gates**	**Clock cycles**
SAMPLELENGTH_LCL	2	48390	832	41153	1216
	4	56396	736	45644	1120
	8	65042	688	50660	1072
	16	74328	664	56552	1048
	32	84524	652	64016	1036
	64	94820	646	74468	1030
	128	106026	643	90788	1027

It can be noted that (expectedly) the size of designs grows with the increase of *SAMPLELENGTH_LCL* while the number of clock cycles (i.e. the clock frequency if we want to maintain the same total processing time) decreases rather insignificantly. Thus, the dynamic processing energy would also increase. Clock cycles increase is indicated by the fact that with wider processing window (i.e. *SAMPLELENGTH_LCL*) more data is processed concurrently, so there are less controlling commands (e.g. *if*, *while*, etc.) executed, that finally leads to a decrease in the total number of clock cycles required.

From the results for *Mean* and *VarDef*, Figures 35 and 36 depicting the total processing power, the processing energy, the transmission energy, and the total energy (processing+transmission) as functions of *SAMPLELENGTH_LCL* have been created.

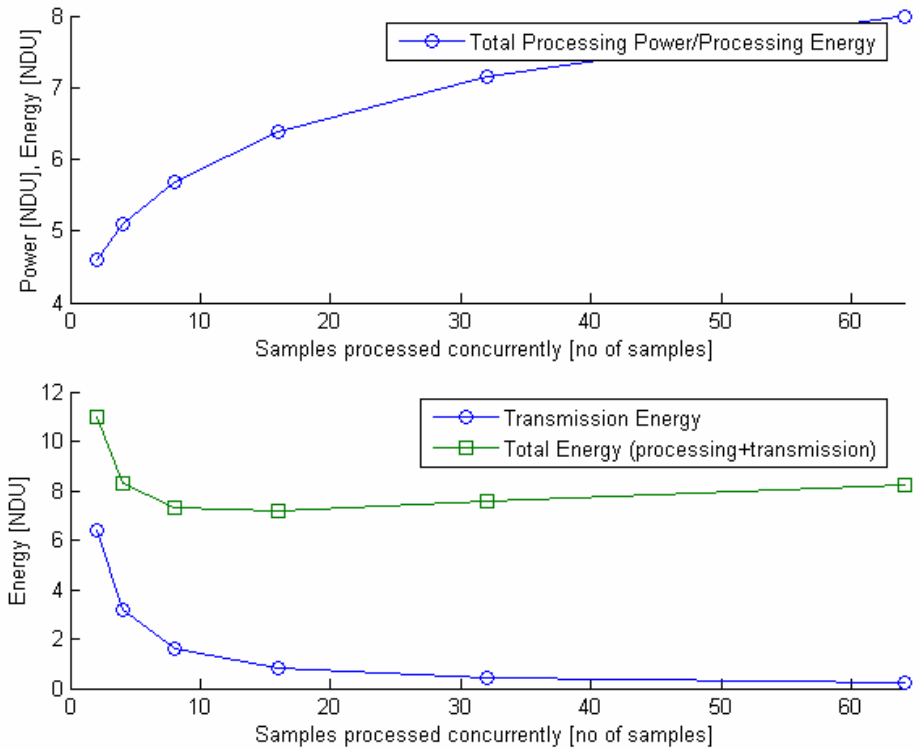

Figure 35. *Mean* – top: processing power and energy, bottom: transmission and total energy.

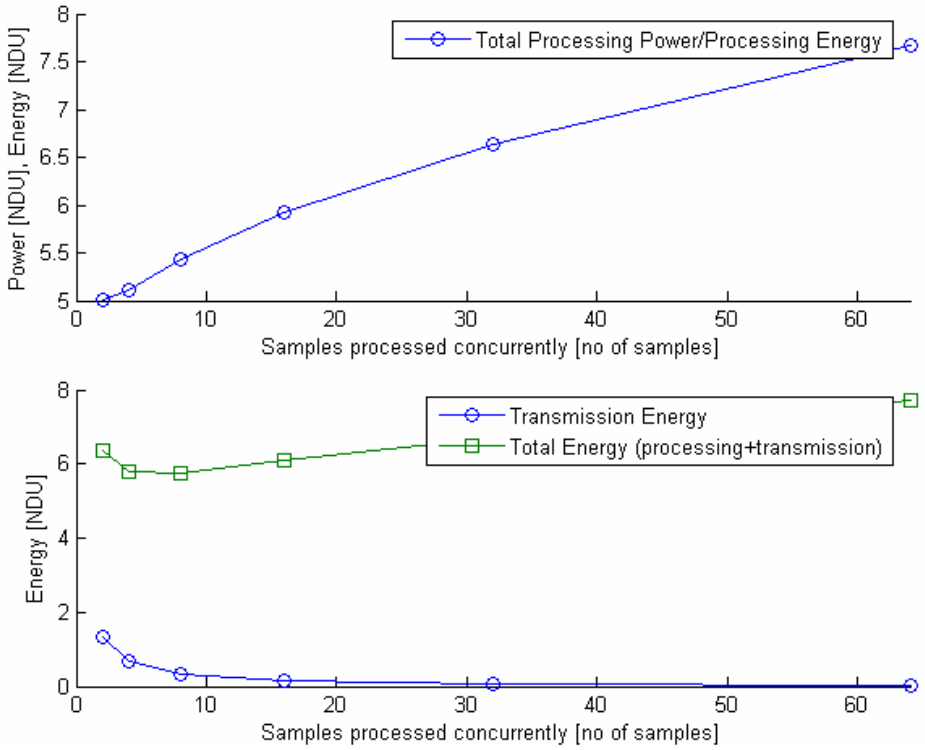

Figure 36. *VarDef* – top: processing power and energy, bottom: transmission and total energy.

The figures show that *SAMPLELENGTH_LCL* (the amount of samples processed at once) increases the total processing power and the processing energy (the top part of

each figure) increases. However, the transmission energy (the bottom part of each figure) obviously decreases. Thus, the key to energy efficiency is the ratio between both energies for changing *SAMPLELENGTH_LCL*. The optimum *SAMPLELENGTH_LCL* value is where the sum of both curves reaches its minimum (i.e. the total energy curves).

In Figures 35 and 36, the transmission energy is expressed using arbitrarily selected non-descriptive units (NDU). For the values provided, the optimum size of *SAMPLELENGTH_LCL* is approx. 16 for the *Mean* algorithm (see Figure 35) and approx. 8 for *VarDef* algorithm (see Figure 36).

The system-level experiments do not include the static power (and the corresponding static energy consumption). However, the additive offset of the static energy should not significantly shift the optimum values of *SAMPLELENGTH_LCL*.

We decided to choose the same two most representative data processing algorithms, i.e. *Mean* and *VarDef*, for further hardware-level energy estimations.

8.2.2. Hardware-level experiments

Experiments at the hardware-level are divided into two scenarios. In both scenarios, we search for the optimum *SAMPLELENGTH_LCL* providing the minimum total energy. We also, somehow arbitrarily, assume that the throughput of the wireless module can meet any requirements on the volume of transmitted data, and that the energy per bit remains the same, i.e. the energy efficiency of the wireless module remains constant regardless of the throughput.

In the first scenario we calculate power consumptions of selected designs using XPower results for several clock frequencies (as previously, we incorporate the dynamic power of unused nets). Power consumptions for other clock frequencies are interpolated.

However, the implemented algorithms occupy a small part of the chip area. *Mean* algorithm occupies just 1-2% of the available slices for *SAMPLELENGTH_LCL* ranging from 2 to 128, while for *VarDef* algorithm, the occupancy ranges from 10 to 14% for the same range of *SAMPLELENGTH_LCL*. As a result, even if the power of unused nets is included, the dynamic power consumption of the implemented algorithms for the investigated FPGA chip (Xilinx Virtex-II FPGA; part: xc2v3000fg676-4) and for low clock frequencies (i.e. not exceeding 10-15 MHz) is very small (up to 10mW) compared to the transmission energy. Thus, the curve of the processing energy is virtually horizontal, if the same drawing scales as for the transmission energy are used. The results

are shown in Figures 37 and 38. Therefore, within the analyzed ranges of the *SAMPLELENGTH_LCL* and frequencies, the minimum total energy can be obtained just by increasing the size of *SAMPLELENGTH_LCL*. It should be noted that by incorporating the static power of the device (i.e. 378mW) we do not change the above conclusion.

A straightforward observation is, therefore, that for very small designs (and for typical transmission energies) we should minimize just the volume of transmitted data for the total energy minimization.

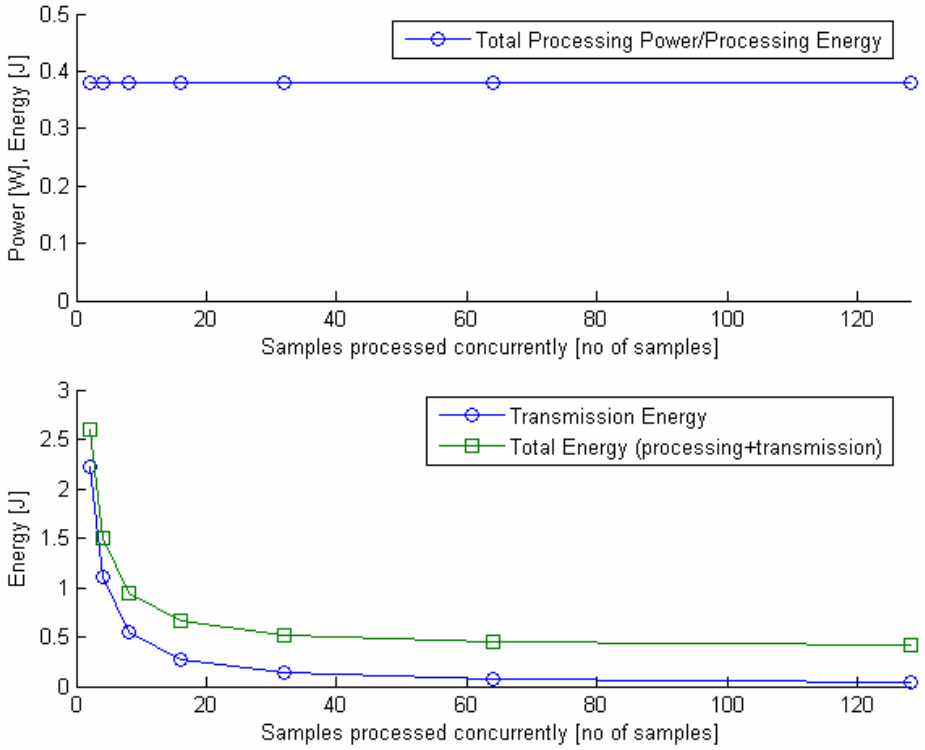

Figure 37. *Mean* (scenario 1) – top: processing power and energy, bottom: transmission energy and total energy.

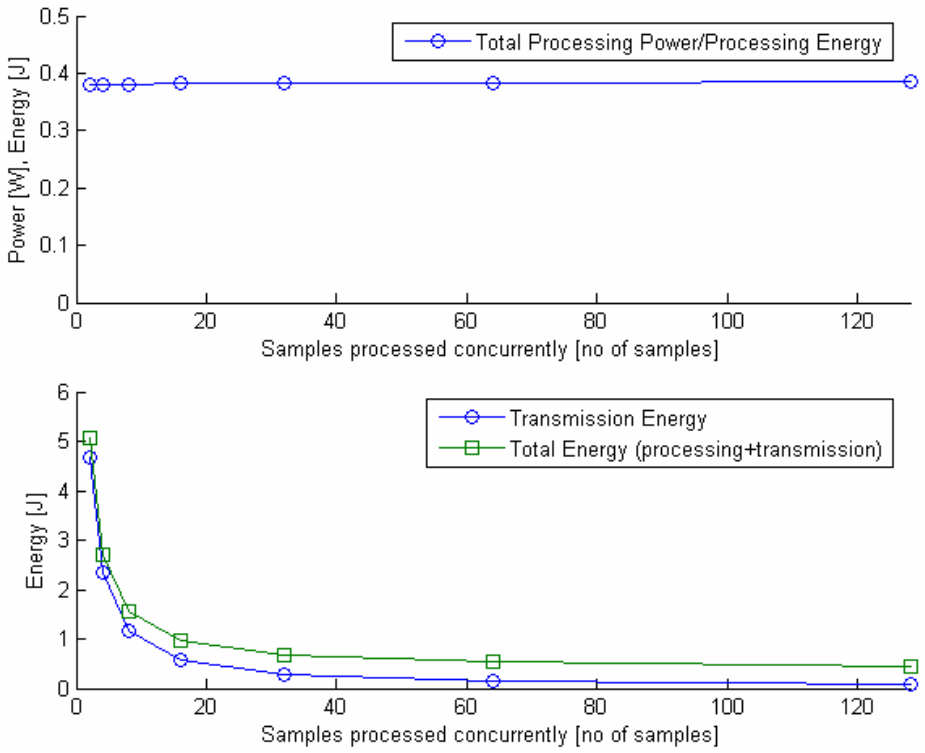

Figure 38. *VarDef* (scenario 1) – top: processing power and energy, bottom: transmission energy and total energy.

In order to verify the validity of this method in more realistic conditions, in the second scenario we artificially increase the dynamic power of the implemented designs. This can be obtained by implementing multiple copies of the same algorithm (all copies processing the same data) and by the frequency increase. Altogether, the dynamic power consumption has been increased 100 times so that the proportions between dynamic and static energies are similar to designs with a large chip area usage, [154], and comparable to the transmission energy. In real applications, such large implementations with higher clock frequency may represent designs with higher processing requirements (e.g. sophisticated data processing algorithms to be executed within the same time constraints).

Alternatively we could assume much smaller transmission energy, but such an assumption would be less realistic taking into account characteristics of currently existing wireless transmission modules.

The results of scenario 2 for *Mean* and *VarDef* algorithms are given in Figures 39 and 40.

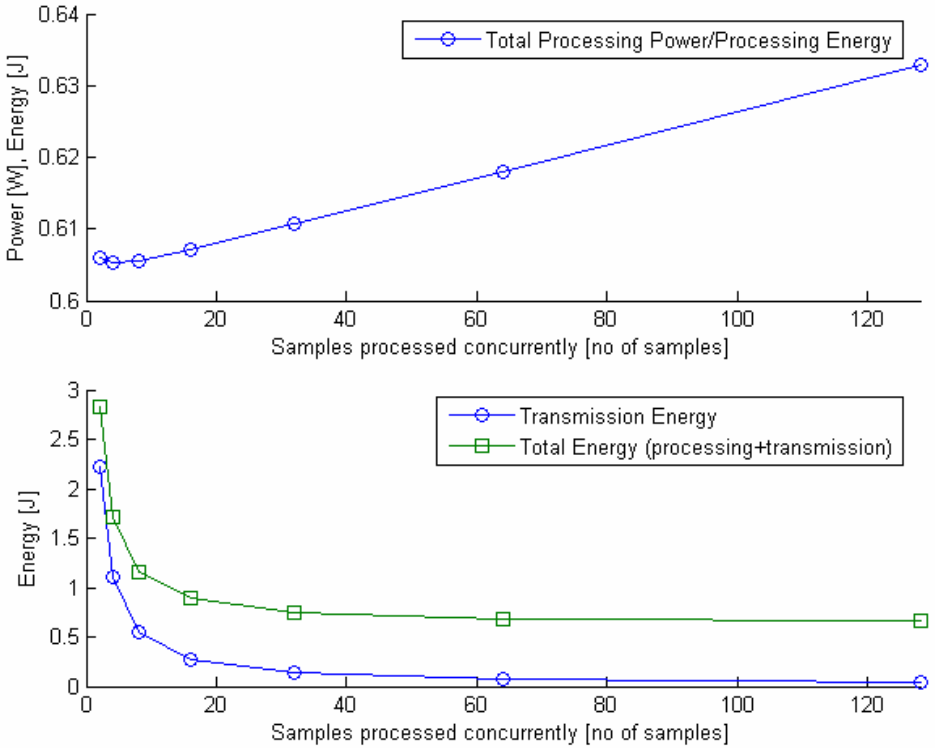

Figure 39. *Mean* (scenario 2) – top: processing power and energy, bottom: transmission energy and total energy.

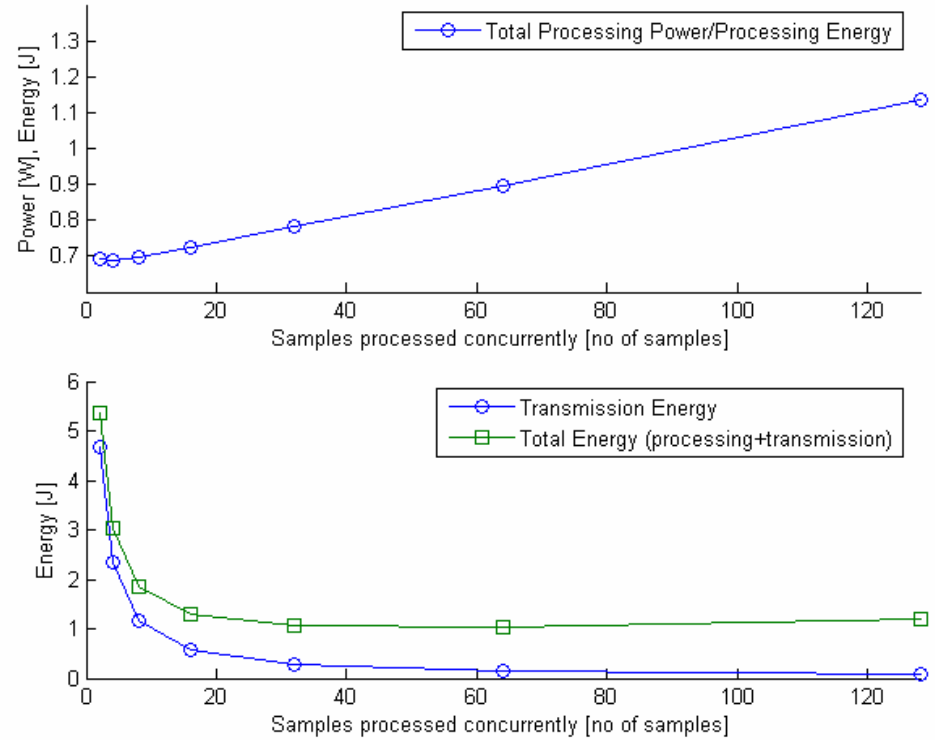

Figure 40. *VarDef* (scenario 2) – top: processing power and energy, bottom: transmission energy and total energy.

We can see from the above figures that it is possible to establish the optimum *SAMPLELENGTH_LCL* for which the total energy (i.e. processing and transmission) is minimized. For *Mean* and *VarDef* algorithms, these optimums values are within (or above) 60-70 range.

To estimate the improvement of the energy efficiency, we compare the total energy of a particular design for the starting value of $SAMPLELENGTH\_LCL = 2$, and for the optimal value.

In scenario 1, the optimal (i.e. as long as possible) *SAMPLELENGTH_LCL* may give the total energy savings up to 82.73% for *Mean* algorithm and up to 89.53% in case of *VarDef* algorithm. It should be remembered, however, that in this scenario the contribution of the (dynamic) processing energy to the total energy is negligible.

In scenario 2, the experiments show similar total energy savings, i.e. 75.71% and 80.54%, for *Mean* and *VarDef* algorithms, respectively, using the optimal (i.e. within 60-70 range) *SAMPLELENGTH_LCL*.

8.3. Chapter summary

In this chapter we have discussed the total energy efficiency of FPGA-based embedded systems wirelessly transmitting unprocessed and/or processed data, e.g. sensor nodes with FPGA as the processing unit. In particular, we have analyzed relations between the volume of data in a particular data processing algorithm and the overall energy spent on the data processing and transmission.

We have verified (using typical to WSN data processing algorithms) that in order to improve energy efficiency (e.g. of a sensor node) the optimum number of data samples should be processed simultaneously and converted into a single sample of output data. However, the experimental results should be looked at in a more general way. The fundamental issue in the presented energy minimization method is the ratio between the processing energy changes and the transmission energy changes when the number of simultaneously processed data samples increases. There might be cases when the processing energy increases disproportionally slower than the transmission energy decreases (as shown in Scenario 1 experiments). Then, the total energy might have no obvious minimum and the number of simultaneously processed samples should be increased to the largest feasible values. This is an important observation in cases when the static power dominates the overall power consumption in an FPGA chip.

We believe that results of our experiments are not limited to FPGA-based designs only, and may be generalized to other embedded systems with wireless transmission modules.

CHAPTER IX
CONTRIBUTIONS AND FUTURE WORKS

In general, this thesis is focused on development of energy-efficient FPGA-based designs using system-level (i.e. algorithmic level) methodologies. This problem is approached mainly by employing system-level decompositions of algorithms, and by investigating power and energy characteristics of such decompositions. Typically, power and energy characteristics can be determined at the low (hardware)-level of the design process. The proposed system-level methods allow power and energy modelling (and optimization) of designs without executing the most tedious and time-consuming operations of design implementations.

The contributions of the thesis are overviewed in Section 9.1. Obviously, the thesis does not present a complete development platform that can be used for energy-efficient designing of FPGA implementations. However, the achieved contributions are important steps towards such platforms. Thus, Section 9.2 addresses the issue of potential practical exploitation of the contributions.

9.1. Contributions

The most typical application area for which the results of our work are intended is development of FPGA-based wireless sensor networks (WSN's). Therefore, the conducted experiments are based on popular algorithms used in data sensing and processing. However, the algorithm's efficiency (or computational properties) was not the focus of the thesis. We just selected a representative sample of algorithms with diversified low- and high-level complexity. Therefore, it is believed that results obtained for such a selection of algorithms are relevant to much wider areas of applications.

We can identify the following topics in which novel/innovative results have been proposed and proven feasible.

9.1.1. System-level power estimates

There is an intuitively straightforward believe that complexity and size of an algorithm determine the power/energy consumption of the algorithm's hardware

implementations. We have experimentally verified that power and energy properties of FPGA-based designs can be sufficiently accurately estimated at the system-level. In Chapters IV and V it has been shown that:

a) The equivalent number of NAND gates is a sufficiently accurate estimate of the dynamic power consumption for practically any design (regardless its hardware and algorithmic complexity). This is an intuitively obvious observation, but (to my best knowledge) no such analysis has been presented in the available sources. Subsequently, a product of the NAND gates equivalent by the number of clock cycles (assuming a predefined execution time of the algorithm) is proportional to the dynamic power consumption used by such a design.

b) Algorithm partitioning (i.e. design decomposition into multiple domains) does not change the power estimates. First, the NAND equivalent estimates of partitioned designs accurately correspond to the actual complexity of the corresponding pieces of hardware (regardless hardware distribution due to design partitioning). Secondly, overheads for domain interconnections (interfaces or channels) are generally insignificant. Such estimates are accurate when the system-level approach is systematically used in the design process. However, by incorporating certain low-level mechanisms (an example in a form of precompiled domains interconnected by interfaces has been discussed) further savings can be achieved. Nevertheless, such savings are also proportional to the estimates, although with a higher proportionally factor (up to 40% obtained for the above-mentioned example).

The above results are the foundation for the further system-level analysis.

9.1.2. System-level design partitioning

Parallel and sequential algorithm partitioning are two general directions of system-level design decomposition, and their applicability depends on the structure and computational properties of the partitioned algorithm. Although the objective of both approaches is to improve the power/energy efficiency of algorithms (under predefined timing constraints) the obtained results are qualitatively different.

In all our experiments the algorithms have been partitioned only into two domains (for a better clarity of results) but the identical approach is applicable to partitioning into any number of domains.

In parallel partitioning, a simultaneous execution of several operations (i.e. several domains running concurrently) is assumed, while the most time-consuming operation determines the reference value for the overall processing time. We have confirmed that power efficiency of algorithms can be improved by allocating different clock frequencies to individual domains according to their hardware complexity and execution time (i.e. the number of clock cycles). Parallel partitioning is particularly efficient (in terms of dynamic power consumption reduction) for algorithms decomposed into domains of opposite properties, i.e. large domains with low processing time *versu*s small domains with a large number of clock cycles.

In sequential algorithm partitioning, a design is divided into several clock domains running sequentially according to the order of executed operations. The overall processing time is assumed constant so that clock frequencies of individual domains are adjusted in the way preserving that overall processing time.

Theoretical results indicate that no energy saving can be expected by sequential partitioning if the domains do not consume any dynamic power during the inactivity periods. In practice, however, inactive domains of an algorithm consume some dynamic power (we have experimentally found the values for selected implementations). It was verified that in such a scenario the minimum energy consumption is usually obtained when all domains are driven by similar clock frequencies (i.e. effectively the design does not need partitioning). When significantly different clock frequencies are applied to individual domains, the energy consumption dramatically increases. However, we also identified certain critical clock frequencies exist for which the loss of energy efficiency is relatively small. Such clock frequencies are strongly recommended if an individual domain of a sequentially partitioned design must be run at higher (or lower) speed than the remaining part of the algorithm.

9.1.3. Energy optimization in data processing and transmission

Additional improvements of energy consumption in WSN applications can be achieved by optimizing the energy usage for both processing data and wireless transmission the results. In typical algorithms of sensor networks, temporal data characteristics are calculated (data aggregation) so that by increasing the number of concurrently processed data samples the volume of transmitted data (i.e. the transmission energy) is reduced. We observed, and proved in Chapter VIII, that for a particular

algorithm a relevant sample length gives significant energy consumption reduction with negligible or no at all power consumption overheads.

9.2. Future works

In our opinion, the future works related to this thesis results should focus on system-level tools and methods for automatic (or semi-automatic) design of energy-efficient FPGA-based embedded systems, with processing and communication capabilities (i.e. wireless sensor nodes). We identify three major directions of these future works based on the results presented in the thesis:

(i) **Further power/energy optimization techniques incorporating deep-sleep modes and device reconfiguration**.

Advanced FPGA devices allow various sleep modes during which the power consumption is dramatically reduced. We believe that certain aspects of such an option should be used for power/energy optimization of FPGA designs. In particular, it should be investigated how turning into sleep mode and leaving such a mode influences timeliness of data processing algorithms, i.e. whether the energy savings during the sleep mode period exceed the energy losses.

Although we believe that online device reconfiguration (including partial reconfiguration) opens new possibilities (e.g. adaptive algorithms to environmental changes) to FPGA-based wireless sensor nodes, it may also compromise timeliness of data processing algorithms (reconfiguration time can be significant, compared to the existing time constraints) and increase the energy consumption. Therefore, the proposed methods have to be investigated or even re-developed to incorporate the reconfigurability issues without compromising temporal performance of implemented algorithms.

(ii) **Development of tools that incorporate power and energy optimization into the system-level design of FPGA implementations**.

Such tools would allow power and energy modelling and optimization of a particular design at the high-level of design process. The tools should be device-independent (i.e. not involving low-level design techniques) but easily applicable to any typical device. They should be also user friendly.

In an idealized scenario, it is envisaged that a user only specifies the algorithm (in a form of an algorithmic-language sequential source code), defines its time constraints, and selects a target device. The system would automatically identify possible

partitionings (in particular parallel partitionings) and implement the algorithm in the most energy-efficient way.

(iii) **Power/energy-efficient routing protocols for FPGA-based wireless sensor networks (WSN's)**.

In FPGA-based designs, unlike in DSP and MCU, the energy efficiency depends not only on how a particular algorithm performs but also on how it is implemented. Therefore, there should be some interaction between individual nodes so that power and energy is used effectively and the overall performance of the whole network (or its part) is optimized. We believe there is a strong need for new routing protocols incorporating power and energy properties of FPGA-based nodes.

LIST OF PUBLICATIONS

[1] A. Sluzek, P. Annamalai, Md. S. Islam, and P. Czapski, "Towards Wireless Sensor Networks with Enhanced Vision Capabilities," in *Proceedings of the Twenty First Autumn Meeting of Polish Information Processing Society*, 2005, pp. 9-18.

[2] A. Sluzek, P. Annamalai, Md. S. Islam, and P. Czapski, "Towards Wireless Sensor Networks with Enhanced Vision Capabilities," *Annales UMCS Informatica AI*, vol. 4, pp. 6-19, 2006.

[3] P. P. Czapski, "A Survey: MAC Protocols for Applications of Wireless Sensor Networks," in *Proceedings of the 2006 IEEE Region 10 Conference*, 2006, pp. 14-17.

[4] P. P. Czapski and A. Sluzek, "Power Optimization Techniques in FPGA Devices: A Combination of System- and Low-Levels," *Proceedings of World Academy of Science, Engineering and Technology*, vol. 22, pp. 313-319, July 2007.

[5] P. P. Czapski and A. Sluzek, "Power Optimization Techniques in FPGA Devices: A Combination of System- and Low-Levels," *International Journal of Electrical, Computer, and Systems Engineering*, vol. 1, no. 3, pp. 148-154, 2007.

[6] P. P. Czapski and A. Sluzek, "A Survey on System-Level Techniques for Power Reduction in Field Programmable Gate Array (FPGA)-Based Devices," in *Proceedings of the Second International Conference on Sensor Technologies and Applications*, 2008, pp. 319-328.

[7] P. P. Czapski and A. Sluzek, "Experiments on Data Processing Algorithms: Energy Efficiency of Wireless and Untethered Field Programmable Gate Array (FPGA)-Based Embedded Systems," in *Proceedings of the First International Conference on Electronic Design*, 2008.

[8] P. P. Czapski and A. Sluzek, "Experiments on Data Processing Algorithms: Energy Efficiency of Wireless and Untethered Field Programmable Gate Array (FPGA)-Based Embedded Systems," *International Journal of Information and Communication Technology*, vol. 2, no. 1/2, pp. 4-18, 2009.

[9] P. P. Czapski and A. Sluzek, "System-Level Power Awareness in FPGA-Based Designs (Data Reduction Algorithms Case Study)," to appear in *Journal of Automation, Mobile Robotics and Intelligent Systems (JAMRIS).*

REFERENCES

[1] M. Muhlhauser and I. Gurevych, "Introduction to Ubiquitous Computing," in *Human computer interaction: concepts, methodologies, tools, and applications*, P. Zaphiris, Ed. City University of London, UK: Medical Information Science Reference, 2008, pp. 1-20.

[2] G. Biczok, K. Fodor, B. Kovacs, and A. Szabo, "Pervasive Computing – An Overview," *Internal journal of Budapest University of Technology and Economics*, vol. LVIII, no. 3, pp. 2-7, 2003. [Online]. Available: http://www.omikk.bme.hu. [Accessed August 21, 2008].

[3] M. Weiser, "The Computer for the 21st Century," in *Human-computer interaction: toward the year 2000*, R. M. Baecker, J. Grudin, W. A. S. Buxton, and S. Greenberg, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 933-940.

[4] P. Stroud and Sheikh I. Ahmed, "A Survey on Current and Future Pervasive Computing Devices and Applications," in *Proceedings of the International Conference on Wireless Networks*, 2004, pp. 887-890.

[5] S. K. S. Gupta, W.-C. Lee, A. Purakayastha, and P.K Srimani, "An Overview of Pervasive Computing," *IEEE Personal Communications*, vol. 8, no. 4, pp. 8-9, August 2001.

[6] S. Acharya, "Application and Infrastructure Challenges in Pervasive Computing," in *Proceedings of the National Science Foundation Workshop on Context-Aware Mobile Database Management*, 2002, pp. 1-3.

[7] K. Romer and F. Mattern, "The Design Space of Wireless Sensor Networks," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54-61, December 2004.

[8] M. A. M. Vieira, C. N. Jr. Coelho, D. C. Jr. da Silva, and J. M. da Mata, "Survey on Wireless Sensor Network Devices," in *Proceedings of the Emerging Technologies and Factory Automation*, 2003, pp. 537-544.

[9] J. Feng, F. Koushanfar, and M. Potkonjak, "System-Architectures for Sensor Networks Issues, Alternatives, and Directions," in *Proceedings of the IEEE International Conference on VLSI in Computers and Processors*, 2002, pp. 226-231.

[10] B. O'Flynn, et al., "The Development of a Novel Miniaturized Modular Platform for Wireless Sensor Networks," in *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks*, 2005, pp. 370-375.

[11] S. J. Bellis, K. Delaney, B. O'Flynn, J. Barton, K. M. Razeeb, and C. O'Mathuna, "Development of Field Programmable Modular Wireless Sensor Network Nodes for Ambient Systems," *Computer Communications*, vol. 28, no. 13, pp. 1531-1544, August 2005.

[12] D. Bauer, S. Furrer, S. Rooney, W. Schott, H. L. Truong, and B. Weiss, "The ZRL Wireless Sensor Networking Testbed," IBM Zurich Research Laboratory, Ruschlikon, Switzerland, Tech. Rep. RZ 3620 (# 99630), 2005.

[13] V. Tsiatsis, S. A. Zimbeck, and M. B. Srivastava, "Architecture Strategies for Energy-Efficient Packet Forwarding in Wireless Sensor Networks," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2001, pp. 92-95.

[14] J. Lach, D. Evans, J. McCune, J. Brandon, and L. Hu, "Power-Efficient Adaptable Wireless Sensor Networks," presented at the International Conference on Military and Aerospace Programmable Logic Devices, Washington, DC, USA, 2003.

[15] Crossbow Technology, Inc., "Product Catalog," *Crossbow Technology : Wireless : Home Page*, 2008. [Online]. Available: http://www.xbow.com. [Accessed: September 06, 2008].

[16] P. P. Chu and M. Kifle, "A Reconfigurable Communication System for Small Spacecraft," National Aeronautics and Space Administration, Cleveland, OH, USA, Tech. Rep. NASA TM-2004-212534, 2004.

[17] Y. Manoli, S. Ramachandran, S. Jayapal, S. Bhutada, and R. Huang, "Energy Reduction Strategies for Sensor-Node-on-a-Chip," presented at the 4. GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze", Zurich, Switzerland, 2005.

[18] M. Wang, "Bridging the Wi-Fi/Embedded Divide," CommsDesign, p. 1+, April 05, 2005. [Online]. Available: http://www.commsdesign.com. [Accessed: September 06, 2008].

[19] K. M. Razeeb, S. Bellis, B. O'Flynn, J. Barton, K. Delaney, and C. O'Mathuna, "A Hybrid Network of Autonomous Sensor Nodes," in *Proceedings of the Second European Union Symposium on Ambient Intelligence*, 2004, pp. 69-70.

[20] M. A. M. Vieira, "BEAN: Brazilian Energy-Efficient Architectural Node," Master thesis, Universidade Federal de Minas Gerais, Belo Horizonte, Brasil, 2004.

[21] Xilinx, Inc., "Product Selection Guides," *FPGA and CPLD Solutions from Xilinx, Inc.*, 2008. [Online]. Available: http://www.xilinx.com. [Accessed: September 06, 2008].

[22] Altera Corporation, "Altera Product Catalog," *Altera - FPGA, CPLD, ASIC and Programmable Logic*, 2008. [Online]. Available: http://www.altera.com. [Accessed: September 06, 2008].

[23] Lattice Semiconductor Corporation, "LatticeMico Product Brochure," *FPGA and CPLD solutions from Lattice Semiconductor*, 2008. [Online]. Available: http://www.latticesemi.com. [Accessed: September 06, 2008].

[24] Tensilica, Inc., "XTensa Product Brief," *Tensilica: Configurable and Standard Processor Cores for SOC Design*, 2008. [Online]. Available: http://www.tensilica.com. [Accessed: September 06, 2008].

[25] Agility Design Solutions, Inc., "Handel-C Language Reference Manual," *Agility Design Solutions :: Algorithm to Implementation. Fast. :: MATLAB to C :: C to FPGA*, 2008. [Online]. Available: http://www.agilityds.com. [Accessed: September 06, 2008].

[26] R. A. Klein and R. Moona, "Migrating Software to Hardware on FPGAs," in *Proceedings of the IEEE International Conference on Field-Programmable Technology*, 2004, pp. 217-224.

[27] M. Petko and G. Karpiel, "Hardware/Software Co-design of Control Algorithms," in *Proceedings of the IEEE International Conference on Mechatronics and Automation*, 2006, pp. 2156-2161.

[28] R. Wain, I. Bush, M. Guest, M. Deegan, I. Kozin, and C. Kitchen, "An Overview of FPGAs and FPGA Programming; Initial Experiences at Daresbury," CCLRC Daresbury Laboratory, Daresbury Warrington, Cheshire, UK, Tech. Rep. DL-TR-2006-007, 2006.

[29] University of California, "NISC Toolset User Guide," *NISC Technology*, 2008. [Online]. Available: http://www.ics.uci.edu/~nisc. [Accessed: September 06, 2008].

[30] Altium, Ltd., "Altium Designer," *Altium Limited > Home*, 2008. [Online]. Available: http://www.altium.com/products/altiumdesigner. [Accessed: September 06, 2008].

[31] Mentor Graphics, Corp., "High Level Synthesis," *Mentor Graphics :: The EDA Technology Leader*, 2008. [Online]. Available: http://www.mentor.com/products/esl/high_level_synthesis. [Accessed: September 06, 2008].

[32] Forte Design Systems, "Cynthesizer Closes the ESL-to-Silicon Gap," *Forte Design Systems: Market and technology leader for ASIC & SoC high-level design | Cynthesizer: The Industry Leader in ESL Synthesis*, 2008. [Online]. Available: http://www.forteds.com/products/cynthesizer.asp. [Accessed: September 06, 2008].

[33] Celoxica, Ltd., "Agility Compiler," *Celoxica - The Technology Leader in C Based Electronic Design and Synthesis*, 2006. [Online]. Available: http://www.celoxica.com/products/agility/default.asp. [Accessed: October 18, 2006].

[34] Agility Design Solutions, Inc., "DK Design Suite," *Agility Design Solutions :: Algorithm to Implementation. Fast. :: MATLAB to C :: C to FPGA*, 2008. [Online]. Available: http://www.agilityds.com/products/c_based_products/dk_design_suite/default.aspx. [Accessed: September 06, 2008].

[35] Nallatech, Inc., "Product Details," *High Performance FPGA Computing Solutions for Defense and HPC - Nallatech*, 2008. [Online]. Available: http://www.nallatech.com/?node_id=1.2.2&id=19. [Accessed: September 06, 2008].

[36] Impulse Accelerated Technologies, "Impulse CoDeveloper C-to-FPGA Tools," *Impulse Accelerated Technologies - Software Tools for an Accelerated World*, 2008. [Online]. Available: http://www.impulsec.com/products_universal.htm. [Accessed: September 06, 2008].

[37] An Open Source Initiative, *FPGA C Compiler*, 2008. [Online]. Available: http://fpgac.sourceforge.net. [Accessed: September 06, 2008].

[38] Colorado State University, *SA-C Overview*, 2008. [Online]. Available: http://www.cs.colostate.edu/cameron/SACoverview.html. [Accessed: September 06, 2008].

[39] CriticalBlue, "Overview," *CriticalBlue | Home*, 2008. [Online]. Available: http://www.criticalblue.com/products/cascade.html. [Accessed: September 06, 2008].

[40] Mitrionics AB, "The Mitrion Software Development Platform," *Mitrionics*, 2008. [Online]. Available: http://www.mitrionics.com/?page=products_platform. [Accessed: September 06, 2008].

[41] Cebatech, Inc., "C2R Compiler," *CebaTech*, 2008. [Online]. Available: http://www.cebatech.com/c2r_compiler. [Accessed: September 06, 2008].

[42] Mimosys AG, "Clarity," *MIMOSYS - Home*, 2008. [Online]. Available: http://www.mimosys.com/htm/prod.htm. [Accessed: February 09, 2008].

[43] University of Kansas, "HybridThreads Compiler," *HybridThreads Compiler - Hybridthreads Project*, 2008. [Online]. Available: https://wiki.ittc.ku.edu/hybridthread/HybridThreads_Compiler. [Accessed: September 06, 2008].

[44] O. S. Unsal and I. Koren, "System-Level Power-Aware Design Techniques in Real-Time Systems," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1055-1069, July 2003.

[45] H. G. Lee, S. Nam, and N. Chang, "Cycle-Accurate Energy Measurement and High-Level Energy Characterization of FPGAs," in *Proceedings of the Fourth International Symposium on Quality Electronic Design*, 2003, pp. 267-272.

[46] S. J. E. Wilton, S.-S. Ang, and W. Luk, "The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays," in *Field Programmable Logic and Application*, Vol. 3203, J. Becker, M. Platzner, and S. Vernalde, Eds. Berlin, Germany: Springer-Verlag, 2004, pp. 719-728.

[47] P. J. M. Havinga and G. J. M. Smit, "Low Power System Design Techniques for Mobile Computers," University of Twente, Department of Computer Science, Enschede, Netherlands, Tech. Rep. ISSN 1381-3625, 1997.

[48] G. J. M. Smit and P. J. M. Havinga, "A Survey of Energy Saving Techniques for Mobile Computers," University of Twente, Department of Computer Science, Enschede, Netherlands, Tech. Rep. Moby Dick, 1997.

[49] K. Weiß, C. Oetker, I. Katchan, T. Steckstor, and W. Rosenstiel, "Power Estimation Approach for SRAM-based FPGAs," in *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays*, 2000, pp. 195-202.

[50] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic Power Consumption in Virtex-II FPGA Family," in *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-Programmable Gate Arrays*, 2002, pp. 157-164.

[51] V. Degalahal and T. Tuan, "Methodology for High Level Estimation of FPGA Power Consumption," in *Proceedings of the 2005 Conference on Asia South Pacific Design Automation*, 2005, pp. 657-660.

[52] N. Rollins and M. J. Wirthlin, "Reducing Energy in FPGA Multipliers Through Glitch Reduction," presented at the International Conference on Military and Aerospace Programmable Logic Devices, Washington, DC, USA, 2005.

[53] M. French, "A Power Efficient Image Convolution Engine for Field Programmable Gate Arrays," presented at the International Conference on Military and Aerospace Programmable Logic Devices, Washington, DC, USA, 2004.

[54] N. Chang and K. Kim, "Real-Time per-Cycle Energy Consumption Measurement of Digital Systems," *Electronics Letters*, vol. 36, no. 13, pp. 1169-1171, June 2000.

[55] J. Lamoureux and W. Luk, "An Overview of Low-Power Techniques for Field-Programmable Gate Arrays," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems*, 2008, pp. 338-345.

[56] J. L. Nunez-Yanez, V. Chouliaras and J. Gaisler, "Dynamic Voltage Scaling in a FPGA-Based System-on-Chip," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2007, pp. 459-462.

[57] L. C. Wong, P. F. Li, Y. Lin and L. He, "Device and Architecture Co-optimization for FPGA Power Reduction," in *Proceedings of the Forty Second Design Automation Conference*, 2005, pp. 915-920.

[58] S. E. Esmaeili and N. I. Khachab, "Efficiency Of Components' Region-Constrained Placement To Reduce FPGA's Dynamic Power Consumption," in *Proceedings of the Fourteenth IEEE International Conference on Electronics, Circuits and Systems*, 2007, pp. 857-860.

[59] D. Chen, J. Cong, and Y. Fan, "Low-Power High-Level Synthesis for FPGA Architecture," in *Proceedings of the Low Power Electronics and Design Conference*, 2003, pp. 134-139.

[60] D. Chen, J. Cong, and J. Xu, "Optimal Module and Voltage Assignment for Low-Power," in *Proceedings of the Asia South Pacific Design Automation Conference*, 2005, pp. 850-855.

[61] M. J. Alexander, "Power Optimization for FPGA Look-Up Tables," in *Proceedings of the ACM International Symposium on Physical Design*, 1997, pp. 156-162.

[62] J. H. Anderson, F. N. Najm, and T. Tuan, "Active Leakage Power Optimization for FPGAs," *IEEE Transaction on Computer-Aided Design*, vol. 25, no. 3, pp. 423-437, March 2006.

[63] D. Chen, J. Cong, F. Li, and L. He, "Low-Power Technology Mapping for FPGA Architectures with Dual Supply Voltages," in *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*, 2004, pp. 109-117.

[64] A. H. Farrahi and M. Sarrafzadeh, "FPGA Technology Mapping for Power Minimization," in *Proceedings of the International Workshop on Field-Programmable Logic and Applications*, 1994, pp. 167-174.

[65] J. Lamoureux and S. J. E. Wilton, "On the Interaction between Power-Aware CAD Algorithms for FPGAs," in *Proceedings of the IEEE International Conference on Computer Aided Design*, 2003, pp. 701-708.

[66] H. Li, S. Katkoori, and W.-K. Mak, "Power Minimization Algorithms for LUT-based FPGA Technology Mapping," *ACM Transaction on Design Automation of Electronic Systems*, vol. 9, no. 1, pp. 33-51, January 2004.

[67] C-C. Wang and C-P Kwan, "Low Power Technology Mapping by Hiding High-Transition Paths in Invisible Edges for LUT based FPGAs," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1997, pp. 1536-1539.

[68] Z-H. Wang, E-C. Liu, J. Lai, and T-C. Wang, "Power Minimization in LUT-based FPGA Technology Mapping," in *Proceedings of the ACM Asia South Pacific Design Automation Conference*, 2001, pp. 635-640.

[69] D. Chen and J. Cong, "Delay Optimal Low-Power Circuit Clustering for FPGAs with Dual Supply Voltages," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2004, pp. 70-73.

[70] H. Hassan, M. Anis, A. El Daher, and M. Elmasry, "Activity Packing in FPGAs for Leakage Power Reduction," in *Proceedings of the Design Automation and Test in Europe*, 2005, pp. 212-217.

[71] A. Singh, G. Parthasarathy, and M. Marek-Sadowski, "Efficient Circuit Clustering for Area and Power Reduction in FPGAs," *ACM Transaction on Design Automation of Electronic Systems*, vol. 7, no. 4, pp. 643-663, 2002.

[72] B. Kumthekar and F. Somenzi, "Power and Delay Reduction via Simultaneous Logic and Placement Optimization in FPGAs," in *Proceedings of the Design Automation and Test in Europe*, 2000, pp. 202-207.

[73] K. Roy, "Power-Dissipation Driven FPGA Place and Route under Timing Constraints," *IEEE Transaction on Circuits and Systems*, vol. 46, no. 5, pp. 634-637, May 1999.

[74] N. Togawa, et al., "A Simultaneous Placement and Global Routing Algorithm for FPGAs with Power Optimization," in *Proceedings of the Asia Pacific Conference on Circuits and Systems*, 1998, pp. 125-128.

[75] R. Tessier, V. Betz, D. Neto, A. Egier, and T. Gopalsamy, "Power-Efficient RAM Mapping Algorithms for FPGA Embedded Memory Blocks," *IEEE Transaction of Computer- Aided Design*, vol. 26, no. 2, pp. 278-289, February 2007.

[76] I. Kuon and J. Rose, "Measuring the Gap between FPGAs and ASICs," *IEEE Transaction on Computer-Aided Design*, vol. 26, no. 2, pp. 203-215, February 2007.

[77] S. J. E. Wilton, S-S. Ang, and W. Luk. "The Impact of Pipelining on Energy per Operation in Field Programmable Gate Arrays," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2004, pp. 719-728.

[78] G. Constantinides, "Word-Length Optimization for Differentiable Nonlinear Systems," *ACM Transaction on Design Automation of Electronic Systems*, vol. 11, no. 1, pp. 26-43, March 2006.

[79] W. G. Osborne, W. Luk, J. G. F. Coutinho, and O. Mencer, "Power and Branch Aware Word-Length Optimisation," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2008, pp. 338-345.

[80] C. T. Chow, et al., "Dynamic Voltage Scaling for Commercial FPGAs," in *Proceedings of the IEEE International Conference on Field Programmable Technology*, 2005, pp. 173-180.

[81] W. G. Osborne, W. Luk, J. G. F. Coutinho, and O. Mencer, "Reconfigurable Design with Clock Gating," in *Proceedings of the International Symposium on Systems, Architectures, Modelling and Simulation*, 2008, pp. 187-194.

[82] J. Noguera and I. O. Kennedy, "Power Reduction in Network Equipment Through Adaptive Partial Reconfiguration," in *Proceedings of the International Conference on Field Programmable. Logic and Applications*, 2007, pp. 240-245.

[83] P. Biswas, et al., "Performance and Energy Benefits of Instruction Set Extensions in an FPGA Soft Core," in *Proceedings of the International Conference on VLSI Design*, 2006, pp. 651-656.

[84] R. Dimond, O. Mencer, and W. Luk, "Combining Instruction Coding and Scheduling to Optimize Energy in System-on-FPGA," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006, pp.175-184.

[85] V. George, H. Zhang, and J. Rabaey, "The Design of a Low Energy FPGA," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 1999, pp. 188-193.

[86] M. Meijer, R. Krishnan, and M. Bennebroek, "Energy Efficient FPGA Interconnect Design," in *Proceedings of the Conference on Design and Test in Europe*, 2006, pp. 1-6.

[87] S. Sivaswamy, G. Wang, C. Ababei, K. Bazargan, R. Kastner, and E. Bozargzadeh, "HARP: Hard-Wired Routing Pattern FPGAs," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, 2005, pp. 21-29.

[88] J. H. Anderson and F. N. Najm, "A Novel Low-Power FPGA Routing Switch," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2004, pp. 719-722.

[89] Y. Lin, F. Li, and L. He, "Routing Track Duplication with Finegrained Power-Gating for FPGA Interconnect Power Reduction," in *Proceedings of the Asia South Pacific Design Automation Conf*erence, 2005, pp. 645-650.

[90] E. Kusse and J. Rabaey, "Low-Energy Embedded FPGA Structures," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 1999, pp. 155-160.

[91] S. Khawam, et al., "The Reconfigurable Instruction Cell Array," *IEEE Transaction on VLSI Systems*, vol. 16, no. 1, pp. 75-85, January 2008.

[92] Altera, *Quartus II Handbook*, Chapter 10, Vol. 3, Altera, 2007.

[93] Xilinx, Inc., *Power Consumption in 65nm FPGAs*, Xilinx, Inc., 2007.

[94] Altera, *Quartus II Handbook*, Chapter 9, Vol. 2, Altera, 2007.

[95] Xilinx, "Optimizing FPGA power with ISE design tools," *Xcell Journal*, no. 60, pp. 16-19, First Quarter 2007.

[96] Actel, *IGLOO Handbook*, Actel, 2008.

[97] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh, "Monitoring Volcanic Eruptions with a Wireless Sensor Network," in *Proceedings of the Second European Workshop on Wireless Sensor Networks*, 2005, pp. 108-120.

[98] R. R. Brooks, P. Ramanathan, and A. M. Sayeed, "Distributed Target Classification and Tracking in Sensor Networks," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1163-1171, August 2003.

[99] T. He, et al., "An Overview of the VigilNet Architecture," in *Proceedings of the Eleventh IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2005, pp. 109-114.

[100] L. Girod and M. A. Roch, "An Overview of the Use of Remote Embedded Sensors for Audio Acquisition and Processing," in *Proceedings of the Eighth IEEE International Symposium on Multimedia*, 2006, pp. 567-574.

[101] D. Li, K. D. Wong, Y. H. Hu, and A. M. Sayeed, "Detection, Classification and Tracking of Targets in Distributed Sensor Networks," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 17-30, March 2002.

[102] L. Gu, et al., "Lightweight Detection and Classification for Wireless Sensor Networks in Realistic Environments," in *Proceedings of the Third International Conference on Embedded Networked Sensor Systems*, 2005, pp. 205-217.

[103] Q. Wang, W.-P. Chen, R. Zheng, K. Lee, and L. Sha, "Acoustic Target Tracking Using Tiny Wireless Sensor Devices," in *Proceedings of the Second International Workshop on Information Processing in Sensor Networks*, 2003, pp. 642-657.

[104] A. Arora, et al., "A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking," *Computer Networks*, vol. 46, no. 5, pp. 605-634, December 2004.

[105] F. Lewis, "Wireless Sensor Network," in *Smart Environments: Technology, Protocols and Applications*, D. J. Cook and S. K. Das, Eds. New York, NY, USA: Wiley, 2004.

[106] M. Beigl, A. Krohn, T. Zimmer, and C. Decker, "Typical Sensors Needed in Ubiquitous and Pervasive Computing," in *Proceedings of the First International Workshop on Networked Sensing Systems*, 2004, pp. 153-158.

[107] Y. Yu, D. Estrin, M. Rahimi, and R. Govindan, "Using More Realistic Data Models to Evaluate Sensor Network Data Processing Algorithms," in *Proceedings of the Twenty Ninth Annual IEEE International Conference on Local Computer Networks*, 2004, pp. 569-570.

[108] J. R. Agre, L. P. Clare, G. J. Pottie, and N. P. Romanov, "Development Platform for Self-Organizing Wireless Sensor Networks," in *Proceedings of the SPIE Conference on Unattended Ground Sensor Technologies and Applications*, 1999, pp. 257-268.

[109] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler, "Design of a Wireless Sensor Network Platform for Detecting Rare, Random, and Ephemeral Events," in *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks*, 2005, pp. 497-502.

[110] A. Arora, et al., "ExScal: Elements of an Extreme Scale Wireless Sensor Network," in *Proceedings of the Eleventh IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2005, pp. 102-108.

[111] S. de Vlaam, "Object Tracking in a Multi Sensor Network," Master thesis, Delft University of Technology, Delft, Netherlands, 2004.

[112] J. Ding, S.-Y. Cheung, C.-W. Tan, and P. Varaiya, "Signal Processing of Sensor Node Data for Vehicle Detection," in *Proceedings of the Seventh International IEEE Conference on Intelligent Transportation Systems*, 2004, pp. 70-75.

[113] B. P. Flanagan and K. W. Parker, "Robust Distributed Detection Using Low Power Acoustic Sensors," in *Proceedings of the SPIE Conference on Unattended Ground Sensor Technologies and Applications VII*, 2005, pp. 73-80.

[114] T. Onel, E. Onur, C. Ersoy, and H. Delic, "Wireless Sensor Networks for Security: Issues and Challenges," in *Advances in Sensing with Security Applications*, Vol. 2, J. Byrnes, Ed. Il Ciocco, Italy: Springer, 2006, pp. 95-119.

[115] H. Wang, J. Elson, L. Girod, D. Estrin, and K. Yao, "Target Classification and Localization in Habitat Monitoring," in *Proceedings of the 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2003, pp. IV-844-847.

[116] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, "Compressing Historical Information in Sensor Networks," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004, pp. 527-538.

[117] M. Chen and M. L. Fowler, "The Importance of Data Compression for Energy Efficiency in Sensor Networks," in *Proceedings of the 2003 Conference on Information Sciences and Systems*, 2003.

[118] M. Chen and M. L. Fowler, "Data Compression Trade-Offs in Sensor Networks," in *Proceedings of the SPIE Conference on Mathematics of Data/Image Coding, Compression, and Encryption VII, with Applications*, 2004, pp. 96-107.

[119] A. Deligiannakis and Y. Kotidis, "Data Reduction Techniques in Sensor Networks," *IEEE Data Engineering Bulletin*, vol. 28, no. 1, pp. 19-25, March 2005.

[120] K. Sayood, *Introduction to Data Compression*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2006.

[121] D. Salomon, *Data Compression – The Complete Reference*, 4th ed. London, UK: Springer-Verlag, 2007.

[122] T. Dang, N. Bulusu, and W. Feng, "RIDA: A Robust Information-Driven Data Compression Architecture for Irregular Wireless Sensor Networks," in *Wireless Sensor Networks*, Vol. 4373, K. Langendoen and T. Voigt, Eds. Berlin, Germany: Springer-Verlag, 2007, pp. 133-149.

[123] V. Jolly, S. Latifi, and N. Kimura, "Energy-Efficient Routing in Wireless Sensor Networks Based on Data Reduction," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, 2006, pp. 804-812.

[124] N. Kimura and S. Latifi, "A Survey on Data Compression in Wireless Sensor Networks," in *Proceedings of the International Conference on Information Technology: Coding and Computing*, 2005, pp. 8-13.

[125] K. C. Barr and K. Asanovic, "Energy-Aware Lossless Data Compression," *ACM Transactions on Computer Systems*, vol. 24, no. 3, pp. 250-291, August 2006.

[126] C. M. Sadler and M. Martonosi, "Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks," in *Proceedings of the Fourth International Conference on Embedded Networked Sensor Systems*, 2006, pp. 265-278.

[127] C. Tang and C. S. Raghavendra, "Compression Techniques for Wireless Sensor Networks," in *Wireless Sensor Networks*, C. S. Raghavendra, K. M. Sivalingam, and T. Znati, Eds. USA: Springer, 2004, pp. 207-231.

[128] M. L. Fowler and M. Chen, "Perspectives on Data Compression for Estimations from Sensors," in *Proceedings of the SPIE Conference on Mathematics of Data/Image Pattern Recognition, Compression, and Encryption with Applications IX*, 2006, pp. 631505.1-631505.11.

[129] H. Akcan and H. Bronnimann, "Deterministic Data Reduction in Sensor Networks," in *Proceedings of the 2006 IEEE International Conference on Mobile Adhoc and Sensor Systems*, 2006, pp. 530-533.

[130] A. T. Hoang and M. Motani, "Collaborative Broadcasting and Compression in Cluster-based Wireless Sensor Networks," in *Proceedings of the Second European Workshop on Wireless Sensor Networks*, 2005, pp. 197-206.

[131] P. J. Marron, R. Sauter, O. Saukh, M. Gauger, and K. Rothermel, "Challenges of Complex Data Processing in Real World Sensor Network Deployments," in *Proceedings of the ACM Workshop on Real-World Wireless Sensor Networks*, 2006, pp. 43-48.

[132] D. Petrovic, R. C. Shah, K. Ramchandran, and J. Rabaey, "Data Funneling: Routing with Aggregation and Compression for Wireless Sensor Networks," in *Proceedings of the First 2003 IEEE International Workshop on Sensor Network Protocols and Applications*, 2003, pp. 156-162.

[133] J. P. Lynch, A. Sundararajan, K. H. Law, A. S. Kiremidjian, and E. Carryer, "Power-Efficient Data Management for a Wireless Structural Monitoring System," in *Proceedings of the Fourth International Workshop on Structural Health Monitoring*, 2003, pp. 15-17.

[134] Y. Al-Obaisat and R. Braun, "On Wireless Sensor Networks: Architectures, Protocols, Applications, and Management," in *Proceedings of the First IEEE International Conference on Wireless Broadband and Ultra Wideband Communication*, 2006.

[135] A. Ciancio, S. Pattem, A. Ortega, and B. Krishnamachari, "Energy-Efficient Data Representation and Routing for Wireless Sensor Networks Based on a Distributed Wavelet Compression Algorithm," in *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks*, 2006, pp. 309-316.

[136] R. Patriquin and I. Gurevich, "An Automated Design Process for The CHAMP Module," in *Proceedings of the IEEE 1995 National Aerospace and Electronics Conference*, 1995, pp. 417-424.

[137] N. Nguyen, K. Gaj, D. Caliga, and T. El-Ghazawi, "Implementation of Elliptic Curve Cryptosystems on a Reconfigurable Computer," in *Proceedings of the 2003 IEEE International Conference on Field-Programmable Technology*, 2003, pp. 60-67.

[138] L. Charaabi, E. Monmasson, M.-A. Nassani, and I. Slama-Belkhodja, "FPGA-based Implementation of DTSFC and DTRFC Algorithms," in *Proceedings of the Thirty First Annual Conference of IEEE Industrial Electronics Society*, 2005, pp. 245-250.

[139] Z. K. Baker and V. K. Prasanna, "Automatic Synthesis of Efficient Intrusion Detection Systems on FPGAs," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 4, pp. 289-300, October-December 2006.

[140] U. D. Patel, C. Brambora, and P. Ghuman, "Lessons from Adaptive Level One Accelerator (ALOA) System Implementation," National Aeronautics and Space Administration: Goddard Space Flight Center, Maryland, DC, USA, Tech. Rep. 20010086478, 2001.

[141] N. P. Ngoc, G. Lafruit, J.-Y. Mignolet, G. Deconinck, and R. Lauwereins, "QOS Aware HW/SW Partitioning on Run-time Reconfigurable Multimedia Platforms," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2004, pp. 84-92.

[142] J. E. Scalera, et al., "Reconfigurable Object Detection in FLIR Image Sequences," in *Proceedings of the Tenth Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002, pp. 284-285.

[143] S. Cichon, M. Gorgon, and M. Pac, "Handel-C Design Enhancement for FPGA-based DV Decoder," in *Reconfigurable Computing: Architectures and Applications*, Vol. 3985, K. Bertels, J. M. P. Cardoso, and S. Vassiliadis, Eds. Berlin, Germany: Springer-Verlag, 2006, pp. 128-133.

[144] H. Ruckdeschel, H. Dutta, F. Hannig, and J. Teich, "Automatic FIR Filter Generation for FPGAs," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Vol. 3553, T. D. Hamalainen, A. D. Pimentel, J. Takala, and S. Vassiliadis, Eds. Berlin, Germany: Springer-Verlag, 2005, pp. 51-61.

[145] Xilinx, Inc., "XPower," *XPower Help*, 2007. [Online]. Available: http://www.xilinx.com/itp/xilinx10/help/iseguide/mergedProjects/xpower/xpower.htm#html/xp_b_overview.htm. [Accessed: September 06, 2008].

[146] Celoxica, Ltd., *Platform Developer's Kit: RC200/203 Manual*, Celoxica, Ltd., 2005.

[147] Celoxica, Ltd., "RC Series Platforms," *Celoxica - The Technology Leader in C Based Electronic Design and Synthesis*, 2006. [Online]. Available: http://www.celoxica.com/products/agility/default.asp. [Accessed: October 18, 2006].

[148] K. Arshak, E. Jafer, and C. Ibala, "Power Testing of an FPGA Based System Using Modelsim Code Coverage Capability," in *Proceedings of the IEEE Design and Diagnostics of Electronic Circuits and Systems*, 2007, pp. 1-4.

[149] E. Todorovich, E. Boemo, F. Angarita, and J. Vails, "Statistical Power Estimation for FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2005, pp. 515-518.

[150] F. A. Aloul and A. Sagahyroon, "Estimation of the Weighted Maximum Switching Activity in Combinational CMOS Circuits," in *Proceedings of the 2006 IEEE International Symposium on Circuits and Systems*, 2006, pp. 2929-2932.

[151] J. A. Clarke, A. A. Gaffar, and G. A. Constantinides, "Parameterized Logic Power Consumption Models for FPGA-based Arithmetic," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2005, pp. 626-629.

[152] C. Gillies and G. Quan, "An Overview of Dynamic Power Consumption Estimation Methodologies for FPGAs," University of South Carolina, Columbia, SC, USA, 2007.

[153] Chipcon AS, "CC1000 Datasheet," *ZigBee, RF transmitter, transceiver and receiver from Chipcon*, 2007. [Online]. Available: http://www.chipcon.com. [Accessed: March 13, 2007].

[154] P. P. Czapski and A. Sluzek, "Power Optimization Techniques in FPGA Devices: A Combination of System- and Low-Levels," *International Journal of Electrical, Computer, and Systems Engineering*, vol. 1, no. 3, pp. 148-154, 2007.