System-level Approaches to Power Efficiency in FPGA-based Designs (Data Reduction Algorithms Case Study)

Submitted 6th February 2010; accepted 18th August 2010.

Pawel Piotr Czapski, Andrzej Śluzek

Abstract:

In this paper we present preliminary results on system-level analysis of power efficiency in FPGA-based designs. Advanced FPGA devices allow implementation of sophisticated systems (e.g. embedded sensor nodes). However, designing such complex applications is prohibitively expensive at lower levels so that, moving the designing process to higher abstraction layers, i.e. system-levels of design, is a rational decision. This paper shows that at least a certain level of power awareness is achievable at these higher abstractions. A methodology and preliminary results for a power-aware, system-level algorithm partitioning is presented. We select data reduction algorithms as the case study because of their importance in wireless sensor networks (WSN's). Although, the research has been focused on WSN applications of FPGA, it is envisaged that the presented ideas are applicable to other untethered embedded systems based on FPGA's and other similar programmable devices.

Keywords: *power awareness, FPGA, system-level, Handel-C, data reduction..*

1. Introduction

Sensor nodes are important examples of embedded systems. A typical sensor node (for either civilian or military applications) has a wireless communication unit, a processing unit, a sensing unit, and a power unit, [1]-[3]. Power resources of the sensor node are often limited or even irreplaceable in a field. This is a factor significantly constraining performances and power consumption of sensor nodes. Thus, processing devices with fixed architecture, e.g. microcontrollers (MCU) or digital signal processors (DSP), are still the most popular technique for sensor node implementations (due to their mature power and energy management).

However, advanced multimillion-gate reconfigurable architectures become incomparably more powerful (e.g. Altera FPGA chips, Xilinx FPGA chips, [4], [5]). Therefore, more attention has been recently paid to reconfigurable architectures, e.g. software-based processors (LatticeMico, Nios, MicroBlaze, PicoBlaze, XTensa), [4]-[7]. Although there are currently only a few wireless sensor node applications employing FPGA chips and in these applications FPGA is mostly used as a supporting processing unit (e.g. [8]-[11]) it is envisaged that by employing such reconfigurable processing units more flexible sensor nodes, adaptable to a wider range of scenarios (including unpredictable ones) can be developed.

A typical FPGA incorporates the main array of slices and I/O blocks, and a number of other hard cores, i.e. memory blocks, digital clock managers, encryption circuitries, and custom multipliers, [12]. Although, power and performances of FPGA are often compared to application-specific integrated circuits (ASIC), e.g. [12], [13], configurable interconnections and switching structures (indispensable to achieve programmability of FPGA) increase loads and, thus, power consumption, [14]. This is a drawback of FPGA and, therefore, a careful analysis of power characteristics is of a particular importance for FPGA-based designs.

Together with powerful FPGA devices, advanced high-level designing tools – e.g. compilers (Quartus, ISE), hardware description languages (Verilog, VHDL), system-level hardware description languages (such as Handel-C or Catapult C), etc. – are available so that complex processing units can be quickly synthesized and prototyped using FPGA's or other complex devices. However, the system-level techniques are not power-aware so that significant power and/or hardware overheads (comparing to tedious but efficient low-level techniques) may be introduced in designs developed at high-levels, e.g. [4], [5], [15].

The main objective of this paper is to show that a certain level of power awareness can be incorporated into the system-level design techniques with almost no overheads. We demonstrate it using results of several experiments on power optimization in FPGA designs. The experiments are conducted using Handel-C language and DK Design Suite. The results are obtained for the case study of data reduction algorithms. This is a deliberate choice since data reduction is one of the fundamental issues in wireless sensor networks (and other related areas).

The following sections of the paper are structured as follows: Section 2 overviews power consumption in FPGA and methods of power estimation and reduction. In Section 3, we present selected data reduction algorithms employed in untethered embedded systems (such as sensor nodes). Section 4, which is the core part of the paper, contains description of the experimental results obtained for those algorithms. We first demonstrate that hardware and power characteristics of FPGA designs can be represented sufficiently accurately at the system-level. Then, by using this observation, we show how power savings can be achieved by the system-level design partitioning. We focus on algorithms partitioned into domains that are run in parallel, but certain ideas for sequentially executable domains are presented in Section 5. Section 6 concludes the paper.

2. Power Consumption in FPGA

2.1. Dynamic and Static Power

Power consumption of CMOS devices, e.g. FPGA, consists of two components: static and dynamic, [12], [13], [16]-[19]. The dynamic power consumption of CMOS devices is caused by signal switching at the device transistors, [16], [18], [19]. Frequencies of signal switching are obviously related to the clock frequency. Hence, the dynamic power consumption of a multi-resource system is generally modelled as:

$$P = C_i^2 \cdot V_i \cdot f_i \tag{1}$$

where C_i, V_i, and f_i, represent correspondingly the capacitance, the voltage swing, and the clock frequency of the ith resource (e.g. [12], [13], [18], [19]).

The actual dynamic power of FPGA devices is, obviously, determined by the complexity of the implemented design. The design-dependent factors that contribute to the dynamic power are: *effective capacitance* of resources, *resources utilization*, and *switching activity* of resources, [12], [13], [20].

The *effective capacitance* represents the sum of parasitic capacitances of interconnected wires and transistors. The *resources utilization* reflects an obvious fact that FPGA provides more resources than usually required to implement a particular design (unused resources do not consume the dynamic power). The *switching activity* is the average number of signal transitions in a clock cycle. Generally, it is related to the clock frequency but it may also depend on other factors, e.g. temporal patterns of input signals. Hence, (1) can be rewritten as:

$$P = \sum_i C_i \cdot V_i^2 \cdot f_i \cdot U_i \cdot S_i \tag{2}$$

where U_i and S_i are the utilization, and the switching activity of individual resources.

The static power consumption is caused mainly by the leakage current between the power supply and the ground. The sub-threshold leakage current (depending on temperature and the threshold voltage V_{th}) dominates the leakage current, see [19].

Some researches, e.g. [12], show that the static power of modern FPGA's, e.g. the Virtex-II family (SRAM-based FPGA, 0.15 μm technology), ranges between 5 and 20% of the total dissipated power, depending on the temperature, the clock frequency, and the implemented design. However, since the static power of FPGA is mainly technology-dependent and it does not change with the design complexity, we do not discuss the static power issues in this paper.

2.2. Reduction of Power Consumption

According to [20], three approaches to FPGA dynamic power consumption reduction exist. First, changes can be done at the system-level, e.g. modifications to the algorithms used. Secondly, if the architecture of FPGA is already fixed, a designer may change the logic partitioning, mapping, placement and routing. Finally, if no such changes are possible, enhanced operating conditions (this includes changes in the capacitance, the supply voltage, and the clock frequency) may offer some improvements. For example, because of high capacitance of external connections, on-chip memories instead of off-chip memories are recommended, [19]. The capacitance may also be reduced by tight timing constraints, e.g. [12], [18], forcing place-and-route tools to choose resources with lower capacitance.

Reducing the supply voltage is the most effective mean to power consumption reduction (a quadratic term in (1) and (2)). However, lower supply voltages increase delays in circuits (that decreases performance) so it must be carefully balanced against any performance drop. Decreasing the clock frequency can also reduce power consumption. However, it may require changes to the design, especially for devices performing under predefined timing constraints.

2.3. Power Consumption Estimation in FPGA

Details of power consumption of FPGA can be obtained by real measurements or by simulation-based estimations, [12], [13], [20-22]. The real measurements provide the most accurate power information, e.g. [20], but the measured device must be a representative one. Power estimates that use the simulation-based approach are more convenient, but they provide only approximate results.

The majority of existing power estimation techniques are based on the switching capacitance and the corresponding factors such as the average switching activity and the average resource utilization, see [12], [13], [17]-[19], [20], [22], [23]. Such approaches are suitable for power consumption estimates of FPGA devices, where most of implemented designs are synchronous and driven by the system clocks.

3. Data Reduction Algorithms

3.1. Introduction

More complex embedded systems obviously have to process more data. Handling large amount of data becomes even more difficult when the data have to be transmitted wirelessly. 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, see [24]-[27]. Thus, the issue of data reduction (compression) becomes of the paramount importance.

Data reduction (compression) algorithms are either lossless or lossy, e.g. [28], [29]. Lossless techniques are used in applications that cannot tolerate any difference between the original and decompressed data. Generally, lossless compression techniques generate a statistical model of data and map data to bit strings based on the generated model. Lossy compression techniques provide much higher compression ratio by accepting distortion in the reconstruction process. In general, lossy compression techniques transform given data into a new data space using an appropriate basis function or functions, [30].

Compression algorithms can be evaluated using different criteria: the relative complexity, the memory requirements, CPU speed requirements, compression ratio, the distortion level, see [28], [29], [31].

3.2. Data Reduction in Wireless Sensor Networks

Data reduction is not commonly used in applications of wireless sensor networks. The major limitations are memory footprints and inadequate performances of processing units, see [31]-[33]. Therefore, the use of typical lossless data compression algorithms like LZO, BZIP2, PPMd (and other PC-based algorithms) is discouraged, see [31]. Nevertheless, there are some works on such algorithms used in sensor nodes of a limited power and performance (e.g. LZW in [34]). Other lossless data reduction algorithms used in sensor networks are Huffman and RLE coding, [35].

Some pre-processing techniques changing data descriptions and increasing compression ratio are also often used, [34, 35]. These use Burrow-Wheeler Transform (BWT) and Structured Transpose (ST) to reorder data before LZW coding, and decorrelation transforms such as Wavelet Transform (WT) to describe data structures before employing Huffman codes. However, the latter introduce some distortions due to employed lossy transformations.

To overcome the limitations of standard algorithms, novel compression schemes have been developed for wireless sensor networks, [25], [27], [31], [32], [36]-[40]. 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 some modifications.

Lossy data reduction in wireless sensor networks includes aggregation and approximation, [24], [27], [40]. Aggregation summarizes the measurements in the form of simple statistics, e.g. average, maximum, minimum, etc., over regular intervals. This is an effective way in reducing the data volume but rather crude for applications requiring detailed historical information, e.g. in surveillance and monitoring. Approximations (e.g. histograms, wavelets, discrete cosine transform, linear regression, etc.) are employed if data exhibit a large degree of redundancy.

There are also other methods of data reduction in wireless sensor networks, e.g. [26], [32], [41], [42]. They involve distributed processing and combine routing, data fusion and data aggregation so that they are not a subject of our discussion.

Although the shear data volume is the major issue in wireless sensor networks, directly affecting the communication capacity, e.g. [26], [34], there might be additional requirements for data reduction algorithms in such systems. For example, data reduction schemes are supposed to reduce communication latency, to enhance the energy efficiency (by reducing the energy consumed on data transmission, [25]) and to generally reduce the energy consumption, [26].

Altogether, energy awareness is (directly or indirectly) one of the main issues for data reduction algorithms in wireless sensor networks. Therefore, data reduction algorithms have been selected as the case study for this paper.

4. Experiments

In this section we discuss experiments on FPGA implementation of two data reduction algorithms. The main objective of these experiments is to prove that the dynamic power awareness of FPGA designs can be achieved at the system-level. The selected algorithms are *Huffman coding* and *Arithmetic coding*. Our further experiments have shown, nevertheless, that similar results have been obtained for other algorithms of diversified structures so that this case study exemplifies of what we believe is a useful technique of general applicability.

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, [35]. However, problems may arise if the alphabet of source data is not big enough, or with highly-skewed probabilities, or just a binary one (in-network data such as detection, classification, tracking, etc.) in the worst case, [28], [29]. This problem can be partially solved by building the extended alphabet (that has symbols grouped in blocks of two and more). However, this introduces exponential growth of the codebook.

Arithmetic coding is a better choice that assigns codewords to particular sequences without generating codes for all sequences of that length (as in the case of *Huffman coding*), [28], [29]. However, *Arithmetic coding* is much more tedious to implement. Thus, even if *Arithmetic coding* is a good candidate for wireless sensor networks, it has not been (to our knowledge) implemented yet in such applications. Nevertheless, our experiences show that *Arithmetic coding* may be a feasible choice for FPGA-based applications.

Dividing a design into a number of domains is a proven technique. The selected algorithms can be naturally decomposed into *compressors* and *decompressors* (although further partitioning of both *compressor* and *decompressor* is also later discussed). We use such decompositions as the major technique for optimizing the power consumption at the system-level of design, [43]. It is shown that by a proper algorithms partitioning and the corresponding selection of clock frequencies for the individual domains, a significant power savings can be obtained without analyzing the hardware-level details of the designs.

The algorithms are implemented at the system-level in Handel-C using DK Design Suite (a complete design environment for C-based algorithmic design entry, simulation and synthesis).

To investigate power efficiency of the designs, we compile them for RC203 development board, equipped in Xilinx Virtex-II FPGA (xc2v3000fg676-4). XPower (one of the accessories of Xilinx Integrated Software Environment (ISE)) is employed for hardware-level power consumption estimation. XPower provides the estimates using simulated data describing activity of the implemented design. Sensor nodes (and other similar systems) are often deployed in hardly predictable environments. Hence, we arbitrarily assume that activity rate of the implemented designs is 50%. To minimize differences between XPower estimates and the actual hardware implementations, we decided to use the external FPGA pins as the direct data inputs and outputs.

4.1. Methodology

Dynamic power consumption is proportional (see Equations (1) and (2)) to the size of the design and the clock frequency. However, the complexity of FPGA de-

signs is differently measured at different levels. The system-level (abstract) complexity is expressed by the number of equivalent NAND gates. For a given algorithm, its abstract complexity is therefore fixed (depending on the algorithm structure and the compiler's efficiency). Even if the algorithm is decomposed into several parts, its overall complexity is just a union of individual component complexities. Thus the system-level "dynamic power consumption" of a design can be defined in some non-defined units (NDU) as a product the equivalent NAND number and the clock frequency.

The hardware-level complexity is determined by the mapping the system-level structures (netlists) into the FPGA resources (slices, I/O blocks, interconnections, etc.). However, the mapping results may significantly vary for different clock frequencies, especially for decomposed algorithms that can be implemented within one domain or physically partitioned into several hardware domains.

Therefore, the fundamental question for system-level power estimates is whether the abstract complexity of algorithms (i.e. the number of equivalent NAND gates) can be used as a reliable factor for determining the actual dynamic power usage. Intuitively, the power grows with the number of NAND gates but it is important to evaluate fluctuations caused be mapping-and-placing differences (e.g. by using various clock frequencies), differences between single-domain and multi-domain implementations of a decomposed algorithm, etc. Since we have not found in the available sources experimental validation of this issue, Section 4.2 presents such a validation. It confirms feasibility of our approach, i.e. we can use the system-level complexity of designs to fairly accurately represent the power characteristics of the actual FPGA implementations.

4.2. Accuracy of the System-level Power Estimates

In this experiment we investigate hardware-level power characteristics of a decomposed algorithm (*Huffman coding* decomposed into *compressor* and *decompressor* is actually selected) implemented for various clock frequencies in either one or two domains. In order to avoid any distortion of results, we do not use any chip area constraints and we allow *map*, *place*, and *route* tools to perform unconstrained optimizations.

Huffman coding is selected because both parts (i.e. compressor and decompressor) in spite of their different inner structures have almost identical system-level complexities, i.e. the system-level dynamic power estimates would be similar. 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.

Huffman coding was implemented for data of 1bit width, the alphabet of 2 elements, and the sample size of 32 elements, but these parameters do not have any actual significance.

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

Figures 1 to 4 show exemplary layouts (which, as expected, are actually diversified) while Tables 1 to 3 show the related hardware-level estimates of dynamic power obtained by using XPower.

We can observe that the added 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 the rows 1, 2 and 3 of Tables 1 and 2 to the rows 1 and 4 of Table 3, correspondingly). The variations are below a 5% threshold.

The tables additionally show that the dynamic power consumption changes proportionally to the clock frequency change, as predicted in the system-level estimates. In spite of diversified physical layouts of the implementations (compare Figures 1 to 4) power characteristics of the design remain consistent.

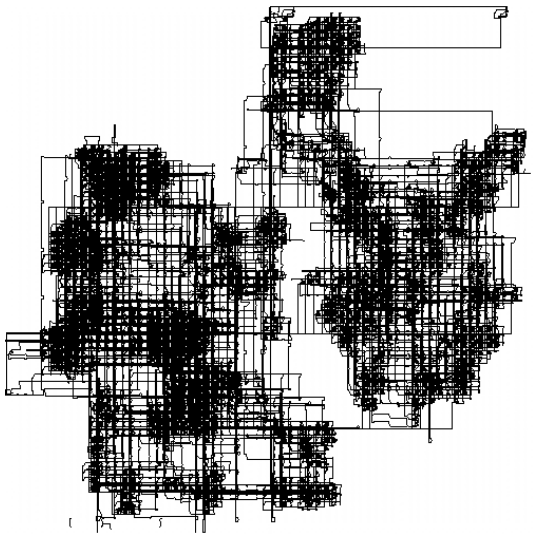

Fig. 1. Compressor (15MHz; on the right) and decompressor (15MHz; on the left) in an exemplary Design A – Huffman coding.

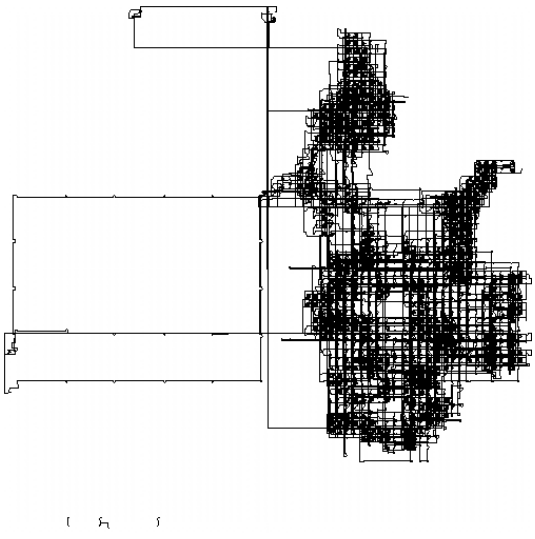

Fig. 2. Design B with only compressor (15MHz) – Huffman coding.

Table 1. 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 2. 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 3. The overall power consumption (decompressor/ compressor) – Design A.

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

The results of this experiment are the key to the system-level clock domain algorithm partitioning discussed below. They confirm that power consumption can be estimated at the system-level using the abstract complexity of the designs (e.g. the number of equivalent NAND gates) and the assumed clock frequency. Even though we cannot estimate the absolute values (which depend on the conversion ratio from non-defined units (NDU) to milliwatts - it should be determined individually for a given model of FPGA) the optimum clock frequencies for various domains and/or the best partitioning strategies can be found in this way.

4.3. Algorithm Partitioning into Parallel Domains – Approach

In the subsequent experiments, we use algorithm partitioning as a tool for power reduction. The same algorithms, i.e. *Huffman coding* and *Arithmetic coding* (actually their *compressors* and *decompressors*) are used as the case study. First, we focus on partitioning into domains that are run simultaneously (the alternative scenario is briefly discussed in Section 5).

The partitioning scheme is applied to *compressors* and *decompressors* of both algorithms. The *compressor* and *decompressor* are each divided into two domains performing simultaneously (more details in Subsection 4.4) and, based on their system-level characteristics, the most power-efficient clock frequencies are proposed for the domains.

Details of the system-level analysis of the designs are as follows:

System-level hardware complexity

The algorithm implemented at the system-level (DK Design Suite) is first compiled and synthesized to the netlist level. The system-level hardware complexity (resources) is estimated by the equivalent number of NAND gates used by the design. Such results are obviously platform-independent. Even though the synthesized designs are later targeted to a relevant hardware (using Xilinx ISE software) the resources are estimated at the system-level only.

When a domain is isolated from an algorithm, this domain is separately compiled and synthesized at the system-level to obtain the equivalent number of NAND gates. The complexity (i.e. the equivalent number of NAND gates) of the remaining algorithm is computed straightforwardly by subtracting the number of gates of the isolated domain from the whole algorithm. It has been verified experimentally that (at least in the implemented algorithms) 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 whether it is isolated or whether it is considered "the remaining part of the algorithm".

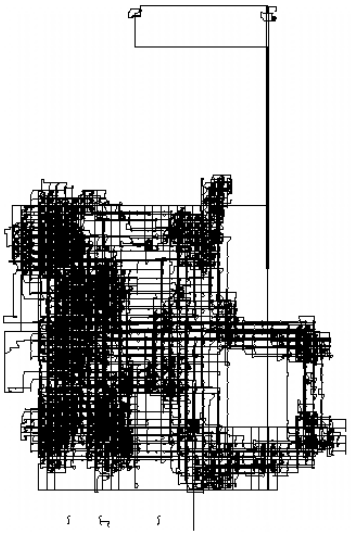

Fig. 3. Design B with only decompressor (15MHz) – Huffman coding.

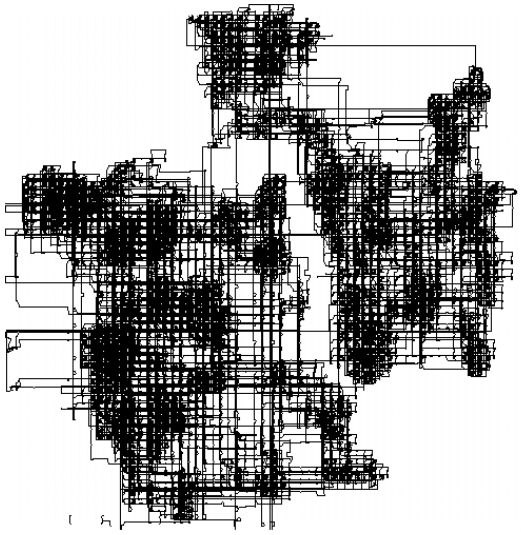

Fig. 4. Compressor (24MHz; on the right) and decompressor (6MHz; on the left) in an exemplary Design A – Huffman coding.

Processing time estimates

Processing time of a particular algorithm (or its domain) is also estimated at the system-level using debugging tools of DK Design Suite. A clock cycle is the basic unit of the time estimates. Because we assume a parallel run of the domains, the longer execution time (of the isolated domain or of the remaining algorithm) determines the overall processing time.

Power consumption estimates

Dynamic power consumption in FPGA is directly related to the hardware resources. We assume that the system-level complexity (i.e. the equivalent number of NAND gates) multiplied by the clock frequency describes the dynamic power utilisation expressed in NDU (non-defined units). The validity of this approach has been justified by the experiment described in Section 4.2.

It should be noted that such a power characteristics is platform-independent.

4.4. Algorithm Partitioning into Parallel Domains – Implementation

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 1sec of data gathering by certain sensors (e.g. the typical sampling frequency for magnetometers used in wireless senor networks is approx. 10-50 Hz) so they are reasonable, see [44]-[47]. We also arbitrarily decide that the width of processed data is 10bits which is typical resolution of analog-to-digital-converter (ADC) used in wireless sensor networks, [48].

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.

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* in one clock domain and *CodeSendDirect* in another clock domain. Moreover, we decided to implement a memory to store samples of input data (*SampleArray*) and the symbol code table (*SymbolCode*; for symbols encoding) in the same clock domain as *CodeSendDirect* (as the data are mostly accessed by *CodeSendDirect*). Hence, *BuildHuffTree* and *BuildHuffCode* have to access *SampleArray* and *SymbolCode* through channels. Block diagrams of the clock domain partitioning of the *Huffman coding compressor* is presented in Figure 5. The system-level characteristics of the design are given in Table 4.

The *decompressor* of *Huffman coding* consists of *BuildHuffTree* (building Huffman tree; however, it differs from *BuildHuffTree* used in compressor) and *CodeGet* (decoding symbols). 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 memory to store statistics of input data (*AlphArray*) and internal node structures of binary tree (*InterNodeArray*) in the same clock domain as *CodeGet*. *Hence*, *BuildHuffTree* has to access *AlphArray* and *InterNodeArray* through channels. Block diagrams of the clock domain partitioning of the *Huffman coding decompressor* are presented in Figure 6. The system-level characteristics of the design are given in Table 5.

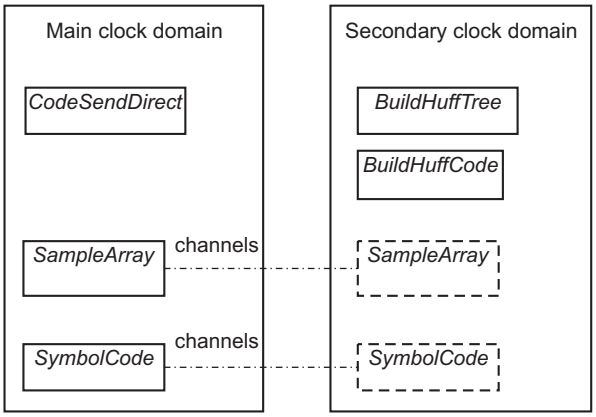

Fig. 5. Block diagram of Huffman coding compressor.

Table 4. Huffman coding (compressor) – hardware resources and processing time.

	[NAND gates equivalent]	Clock cycles
Complete compressor	214,634	-
Main clock domain	79,195	1,155
Secondary clock domain	135,439	20,352

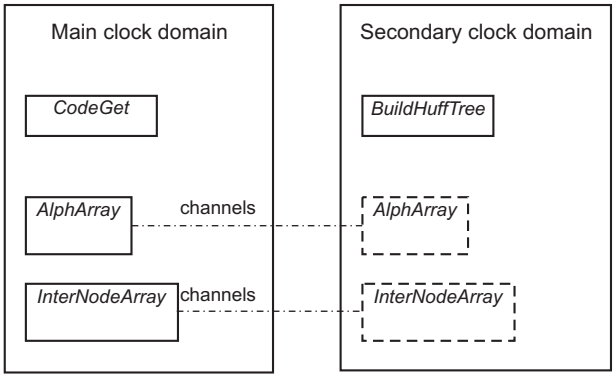

Fig. 6. Block diagram of Huffman coding decompressor.

Table 5. Huffman coding (decompressor) – hardware resources and processing time.

	[NAND gates equivalent]	Clock cycles
Complete compressor	130,724	-
Main clock domain	45,737	655
Secondary clock domain	84,987	14,666

Arithmetic coding

We have implemented the compressor of *Arithmetic coding* using the following functions: *vasPrbCount* (buil-

ding 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 symbols sequence to be encoded (so 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. Block diagrams of the clock domain partitioning of the *Arithmetic coding compressor* are presented in Figure 7. The design characteristics are shown in Table 6.

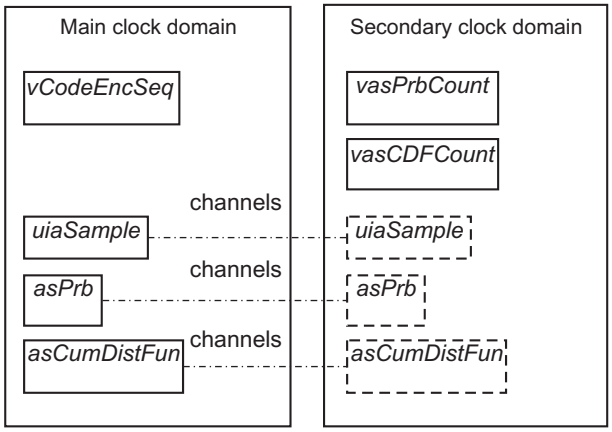

Fig. 7. Block diagram of Arithmetic coding compressor.

Table 6. Arithmetic coding (compressor) – hardware resources and processing time.

	[NAND gates equivalent]	Clock cycles
Complete compressor	231,666	-
Main clock domain	225,047	3,961
Secondary clock domain	6,619	5,350

The decompressor of our *Arithmetic coding* implementation consists of *vasCDFCount* (building the cumulative distribution function based on the probabilistic model of sample data) and *vCodeDecSeq* (decoding alphabet symbols or symbols sequences). 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 different 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. Block diagrams of the clock domain partitioning of the *Arithmetic coding decompressor* are presented in Figure 8. The characteristics of the design are given in Table 7.

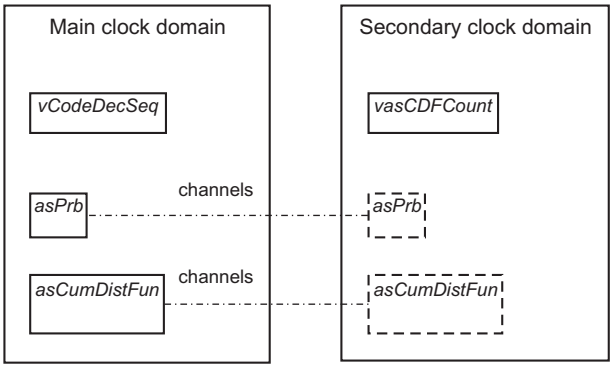

Fig. 8. Block diagram of Arithmetic coding decompressor.

Table 7. Arithmetic coding (decompressor) – hardware resources and processing time.

	[NAND gates equivalent]	Clock cycles
Complete compressor	303,114	-
Main clock domain	299,679	3,418
Secondary clock domain	3,435	3,204

Channels overhead

The resources estimates of a partitioned design might be distorted by the hardware needed for the inter-domain communication. To figure out the actual significance of these overheads, we have implemented the corresponding designs consisting of the channels only (actually, redundant channels that can transfer data samples of 32, 128, and 512 elements are implemented). The results, i.e. the equivalent numbers of NAND gates, are presented in Tables 8 and 9.

Table 8. Huffman coding – channel overheads.

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

Table 9. Arithmetic coding – channel overheads.

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

Tables 8 and 9 show that the channel overheads are insignificant compared to the compressor/decompressor logic (given in Tables 4 to 7). They are 0.24%, 0.40%, 0.22%, and 0.17%, of the compressor/decompressor logic of *Huffman* and *Arithmetic coding*, correspondingly. The additional hardware resources overheads for inter-domain clock synchronization are also included into these numbers.

The example (confirmed by similar experiments for other algorithms) shows that for moderate/large FPGA designs inter-domain communication overheads are negligible and they do not affect the system-level analysis of power characteristics.

4.5. Algorithm Partitioning into Parallel Domains – Analysis

Results of algorithm partitioning (using a two-domain partitioning) are presented in Tables 4 and 5 (*Huffman coding*) and in Tables 6 and 7 (*Arithmetic coding*). In both algorithms, the longer processing time of a domain defines the nominal clock frequency for the whole design (depending on the maximum acceptable processing time that cannot be exceeded). Any reduction of the clock frequency to an individual domain would correspondingly reduce the dynamic power (according to Equations 1 and 2). Therefore, we can estimate the power saving that can be achieved by slowing down the other domain that requires fewer clock cycles to complete its operation.

Huffman coding

In Table 4, the main domain needs only 1,155 clock cycles of execution time while the secondary domain requires 20,352 cycles (see Figure 5 for the domain details). When both domains are driven by the same clock frequency (i.e. the compressor design is not partitioned) the overall power consumption can be estimated (in some non-defined units (NDU)) as:

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

However, in the partitioned design 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. Thus, the power consumption for the main domain can be reduced to:

$$79{,}195 \times 0.0567 = 4{,}490.36\text{NDU}$$

while the secondary domain needs:

$$135{,}439 \times 1 = 135{,}439\text{NDU}$$

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

$$4{,}490.36 + 135{,}439 = 139{,}929.36\text{NDU}$$

i.e. 65.19% of the original 214,634NDU for the non-partitioned design. Almost 35% of the dynamic power is saved.

Following the same methodology for the *decompressor* (see Table 5 and Figure 6 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:

$$45{,}737 \times 0.0447 + 84{,}987 \times 1 = 87{,}031.44\text{NDU}$$

which is 66.58% of the power needed by the non-partitioned implementation (that needs 130,724NDU).

Arithmetic coding

Using the same approach for the *compressor* of *Arithmetic coding* (domain details in Figure 7) we can see in Table 6 that 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 power consumed by the non-partitioned algorithm driven by the nominal clock is:

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

while the total power estimate for the partitioned design is:

$$225{,}047 \times 0.7404 + 6{,}619 \times 1 = 173{,}243.80\text{NDU}$$

so 25.22% of power consumption has been saved compared to the non-partitioned design.

For the *Arithmetic coding decompressor* (details in Table 7 and in Figure 8), 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.

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

versus

$$299{,}679 \times 1 + 3{,}435 \times 0.9374 = 302{,}898.97\text{NDU}$$

The dynamic power reduction is only 0.07%.

5. Remarks on Sequential Partitioning

The algorithm partitioning framework discussed in Section 4 is applicable to algorithms where all fragments perform simultaneously (though possibly with diversified intensities, i.e. at various clock frequencies). This framework, nevertheless, may not give satisfactory power savings in some situations (e.g. for the *Arithmetic coding decompressor*). As seen in Table 7, both parts of the algorithm require almost the same processing time so that no matter what the domain sizes are, we cannot expect any spectacular power savings by parallel partitioning.

There are many algorithms, however, where not all fragments of a decomposed algorithm should be run continuously (i.e. processing is at least partially sequential). Since the dynamic power consumption depends on switching activities of the relevant resources, the idle fragments (i.e. those with temporarily very low switching activity) consume only negligible amounts of dynamic power. By exploiting this fact, further savings of the dynamic power are possible at the system-level.

Assume an algorithm partitioned into just two fragments X and Y that are executed sequentially. Let the corresponding domains D_x and D_y have their processing times of c_x and c_y clock cycles, correspondingly. The overall execution time for the whole algorithm can be, therefore, expressed as:

$$t = t_x + t_y = \frac{c_x}{f_x} + \frac{c_y}{f_y} \quad (3)$$

where f_x and f_y are the corresponding domain clock frequencies.

The power consumption can be hypothetically reduced if the clock frequency of the more hardware-intensive domain is reduced and (if we need to maintain the overall throughput of the system) the clock frequency of the other domain is correspondingly increased.

If the original frequencies are changed (by Δf_x and Δf_y, respectively) the overall execution time would changed and the following dependency can be straightforwardly obtained from Equation 3:

$$\Delta t = \frac{-\Delta f_x \cdot c_x}{f_x \cdot (f_x + \Delta f_x)} + \frac{-\Delta f_y \cdot c_y}{f_y \cdot (f_y + \Delta f_y)} \quad (4)$$

where Δt is the overall execution time increment due to frequency changes Δf_x and Δf_y.

If the processing time is preserved, the value of Equation 4 is zero, so that a simple expression can be obtained on how to simultaneously modify clock frequencies in both domains without affecting the processing time:

$$\frac{\Delta f_x \cdot c_x}{f_x \cdot (f_x + \Delta f_x)} = \frac{-\Delta f_y \cdot c_y}{f_y \cdot (f_y + \Delta f_y)} \quad (5)$$

Then, the recommended (i.e. minimizing the overall power consumption) frequencies can be found by optimizing *hardware*×*frequency* products under constraint specified by (5).

The proposed approach ignores several practical effects. First, the assumption about a zero dynamic power during the inactivity periods holds only approximately. Secondly, the static power that inherently exists in any FPGA device may further distort the validity of calculations. Therefore, the feasibility of this technique to the actual power consumption reduction has to be verified experimentally. The experiments are currently conducted, and the results will be presented in our future papers.

6. Conclusions

In this paper, we have proposed methods for optimizing the dynamic power consumption in FPGA device at the system-level of the designing process. It is calculated and verified experimentally that algorithm decompositions into simultaneously executed fragments (when combined with the appropriate choice of clock frequencies) may significantly reduce the dynamic power indeed. Moreover, as an additional/alternative tool, we propose a sequential algorithm partitioning that may further reduce the power consumption by changing clock frequencies of the relevant clock domains (without affecting the algorithm's throughput). It should be highlighted that the proposed ideas do not introduce any unintentional processing delays or significant hardware overheads.

Our estimations regarding power savings are intentionally based on the system-level results only. Therefore, the dynamic power savings should remain similar for a wide range of FPGA's and other devices. Only the ratio between the dynamic and static power, and the actual values in milliwatts will not be, obviously, device-independent.

Additionally (or rather primarily) we have also shown that the power characteristics of partitioned and non-partitioned designs estimated at the system-level are, in general, inherited at the hardware-level, in spite of variations in the actual layouts of the implemented designs. Thus, the validity of the proposed techniques has been justified experimentally.

Our experiments are focused on data reduction algorithms, namely *Huffman coding* and *Arithmetic coding*. Thus, certain properties of these algorithms have been (intentionally or not) revealed as well. 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 (more details are not discussed in this paper) it may be superior to *Huffman coding*.

Though our experimental works are based on data reduction algorithms implemented in FPGA devices, the same approach can be applicable to other algorithms and other configurable structures. And certainly wireless sensor networks are not the only area where the proposed framework can be useful.

We also express our hope that modern powerful FPGA devices will find a niche in wireless sensor networks and other energy-aware systems. In spite of a relatively high static power (e.g. 378mW of static power for Xilinx Virtex-II FPGA) they offer numerous advantages, e.g. if the design is large or at least moderate, the dynamic power dominates. For example, a design utilizing just 1/3 of Virtex-II FPGA slices may consume up to 533mW of the dynamic power. Thus, our efforts on dynamic power reduction do not seem baseless.

Finally, an important direction for the future results can be highlighted. In the conducted experiments, the algorithm partitioning has been done intuitively (based on our understanding on the algorithm's structure). Even though currently it seems to be the most typical (and the most convenient) approach, we believe that system-level algorithm partitioning for power consumption optimization is an interesting topic. Our experiments clearly reveal that, for example, partitioning into domains of opposite properties (large domains with slow clocks *versus* small domains with very fast clocks) is a recommended strategy for parallel partitioning. Moreover, we have found that (contrary to some believes) inter-domain communication resources in FPGA implementations are typically insignificant compared to the size of (moderate and large) designs. More interesting properties might be revealed when futher researches are conducted in this area.

AUTHORS

Pawel Piotr Czapski* - Nanyang Technological University, School of Computer Engineering, Block N4 #02a-32, Nanyang Avenue, Singapore 639798.
E-mail: pawel@czapski.eu.
Andrzej Śluzek - Nanyang Technological University, School of Computer Engineering.
* Corresponding author

References

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

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

[3] J. Feng, F. Koushanfar, 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.

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

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

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

[7] 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].

[8] 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.

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

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

[11] V. Tsiatsis, S.A. Zimbeck, 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.

[12] L. Shang, A.S. Kaviani, K. Bathala, "Dynamic Power Consumption in Virtex-II FPGA Family". In: *Proceedings of the 2002 ACM/SIGDA 10th International Symposium on Field-Programmable Gate Arrays*, 2002, pp. 157-164.

[13] V. Degalahal, 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.

[14] N. Rollins, 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.

[15] 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].

[16] S.J.E. Wilton, S.-S. Ang, 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.

[17] G.J.M. Smit, 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.

[18] P.J.M. Havinga, 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.

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

[20] H.G. Lee, S. Nam, N. Chang, "Cycle-Accurate Energy Measurement and High-Level Energy Characterization of FPGAs". In: *Proceedings of the 4th International Symposium on Quality Electronic Design*, 2003, pp. 267-272.

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

[22] K. Weiß, C. Oetker, I. Katchan, T. Steckstor, W. Rosenstiel, "Power Estimation Approach for SRAM-based FPGAs", In: *Proceedings of the 2000 ACM/SIGDA 8th International Symposium on Field Programmable Gate Arrays*, 2000, pp. 195-202.

[23] M. French, "A Power Efficient Image Convolution Engine for Field Programmable Gate Arrays". In: *International Conference on Military and Aerospace Programmable Logic Devices*, Washington, DC, USA, 2004.

[24] A. Deligiannakis, Y. Kotidis, 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.

[25] M. Chen, 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.

[26] M. Chen, 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.

[27] A. Deligiannakis, Y. Kotidis, "Data Reduction Techniques in Sensor Networks", *IEEE Data Engineering Bulletin*, vol. 28, no. 1, March 2005, pp. 19-25.

[28] K. Sayood, *Introduction to Data Compression*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2006.

[29] D. Salomon, *Data Compression - The Complete Reference*, 4th ed. London, UK: Springer-Verlag, 2007.

[30] T. Dang, N. Bulusu, W. Feng, "RIDA: A Robust Infor-

mation-Driven Data Compression Architecture for Irregular Wireless Sensor Networks". In: *Wireless Sensor Networks*, vol. 4373, K. Langendoen, T. Voigt, Eds. Berlin, Germany: Springer-Verlag, 2007, pp. 133-149.

[31] V. Jolly, S. Latifi, 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.

[32] N. Kimura, 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.

[33] K.C. Barr, K. Asanovic, "Energy-Aware Lossless Data Compression," *ACM Transactions on Computer Systems*, vol. 24, no. 3, pp. 250-291, August 2006.

[34] C.M. Sadler, M. Martonosi, "Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks". In: *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, 2006, pp. 265-278.

[35] J.P. Lynch, A. Sundararajan, K.H. Law, A.S. Kiremidjian, 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.

[36] H. Akcan, 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.

[37] A.T. Hoang, 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.

[38] P. J. Marron, R. Sauter, O. Saukh, M. Gauger, 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.

[39] D. Petrovic, R.C. Shah, K. Ramchandran, 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.

[40] A. Deligiannakis, Y. Kotidis, "Data Reduction Techniques in Sensor Networks", *IEEE Data Engineering Bulletin*, vol. 28, no. 1, March 2005, pp. 19-25.

[41] Y. Al-Obaisat, R. Braun, "On Wireless Sensor Networks: Architectures, Protocols, Applications, and Management". In: *Proceedings of the 1st IEEE International Conference on Wireless Broadband and Ultra Wideband Communication*, 2006.

[42] A. Ciancio, S. Pattem, A. Ortega, 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.

[43] P.P. Czapski, 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, 2007, pp. 148-154.

[44] 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.

[45] J. Ding, S.-Y. Cheung, C.-W. Tan, 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.

[46] J.R. Agre, L.P. Clare, G.J. Pottie, 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.

[47] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, 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.

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