Energy Consumption Awareness for Resource-Constrained Devices: Extension to FPGA

Edgar M. Silva
Pedro Maló
Michele Albano*

*CISTER Research Centre

CISTER-TR-161104

2016

Energy Consumption Awareness for Resource-Constrained Devices: Extension to FPGA

Edgar M. Silva, Pedro Maló, Michele Albano*

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: mialb@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

The devices running embedded applications tend to be battery-powered, andthe energy efficiency of their operations is an important enabler for the wideadoption of the Internet-of-Things. Optimization of energy usage dependson modelling power consumption. A model-based simulation must considerparameters that depend on the device used, the operating system, and thedistributed application under study. A realistic simulation thus depends onknowledge regarding how and when devices consume energy. This paperpresents an approach to direct measurement of energy consumed in the differentexecution states of the device. We present the architecture and themeasurement process that were implemented. We provide a reference architecture,whose constituent parts can be implemented in different manners,e.g. the processing unit of the device can be the chip on a mote, or anField-Programmable Gate Array (FPGA) implementation. Details are givenregarding the setup of the experimental tests, and a discussion of the resultshints at which architecture is the best for each application under study. Thepresented methodology can be extended easily to new architectures and applications,to streamline the process of building realistic models of powerconsumption.

Energy Consumption Awareness for Resource-Constrained Devices: Extension to FPGA

Edgar M. Silva[1], Pedro Maló[1] and Michele Albano[2]

[1]*UNINOVA-CTS, Faculdade de Ciências e Tecnologia Universidade Nova de Lisboa, Portugal*
[2]*CISTER Research Unit, ISEP/INESC-TEC, Polytechnic Institute of Porto, Portugal*
E-mail: {ems; pmm}@uninova.pt; mialb@isep.ipp.pt

Received 16 September 2016; Accepted 16 November 2016;
Publication 22 November 2016

Abstract

The devices running embedded applications tend to be battery-powered, and the energy efficiency of their operations is an important enabler for the wide adoption of the Internet-of-Things. Optimization of energy usage depends on modelling power consumption. A model-based simulation must consider parameters that depend on the device used, the operating system, and the distributed application under study. A realistic simulation thus depends on knowledge regarding how and when devices consume energy. This paper presents an approach to direct measurement of energy consumed in the different execution states of the device. We present the architecture and the measurement process that were implemented. We provide a reference architecture, whose constituent parts can be implemented in different manners, e.g. the processing unit of the device can be the chip on a mote, or an Field-Programmable Gate Array (FPGA) implementation. Details are given regarding the setup of the experimental tests, and a discussion of the results hints at which architecture is the best for each application under study. The presented methodology can be extended easily to new architectures and applications, to streamline the process of building realistic models of power consumption.

Keywords: Simulation, Modeling, Resouce-Constrained Devices, Energy.

Journal of Green Engineering, Vol. 6_3, 229–256.
doi: 10.13052/jge1904-4720.631

1 Introduction

The Internet-of-Things (IoT) is an active field of research, since it is on the verge of the maturity needed to provide added value to both industrial processes and people's everyday life. The Cyber Physical Systems (CPSs) that make up the IoT address different application domains and services to target different IoT scenarios, which span from Smart Cities to Domotics (Smart Buildings), to Intelligent Transportation Systems, to eHealth, etc. A common aspect of these scenarios is that the involved devices tend to be embedded and resource constrained (low power, small processing power, limited storage capabilities, etc.). In particular, the energy available to the devices is limited, since in most scenarios they are powered by batteries. Energy saving is thus one of the most important research topics in this area.

Research efforts, like other human activities, proceed by trends. Past trends on energy saving were considering mainly wireless sensor networks, and focused on minimizing the energy spent for the communication activities of the devices, in particular by studying network protocols and Medium Access Control (MAC) layers. Within network protocols, the goal was to minimize the number of packets sent to perform a given activity in the sensor network as a whole; within MAC layers, the objective was to organize transmission and reception activities to maximize the time that the each device's wireless interface spent in the sleep state. More recent activities have generalized this vision in many ways. The focus has moved from wireless sensor networks to CPSs, which are devices not limited to data collection activities; the architecture of distributed systems and their Operating Systems (OSs) have been object of analysis, to maximize their energy efficiency; computations performed by devices has become part of the game, and for example it has been an important parameter in deciding which cryptographic algorithms can be used by constrained devices.

An accurate analysis of power consumption is a fundamental support for other Research and Development activities, since it is instrumental to predict expected devices lifetime and to allow developers to optimize energy consumption in distributed IoT applications. Two primary approaches have been followed up to now. Direct measurement is performed by engineering the devices to measure energy consumption while they are executing their distributed applications; this approach is accurate but it is very expensive since it involves engineering every single device involved, and it is not practical in large distributed applications. The second approach is related to simulation models, which are more practical and scale better, but which depend on direct measurement over smaller scenarios to collect the parameters used to enhance the realism of the model.

Direct measurement can be applied according to two main strategies, time-based sampling and event-based sampling. With first approach, the sampling process is executed periodically with a fixed period, while in the second approach the sampling execution is triggered by an event. The latter approach can be implemented in different manners [1] and has produced accurate results in some scenarios [2]. On the other hand, most current architectures for IoT provide straightforward support for the time-based sampling, and event-based sampling by means of conditional triggering over periodic interrupts, as will be shown in Section 5.

This paper follows the direct measurement approach, and considers the time-based sampling, since most devices would provide event-based sampling simulated on top of time-based sampling. A discussion on most common hardware, OSs and simulators is used to select OSs and platforms to be studied. An architecture is presented, and implemented over a testbed. An approach to the set up of experiments is given, and the results of basic experimental tests are presented, to showcase how this methodology can be applied to investigate energy efficiency.

2 Background Information

This section provides support to the paper by means of an acronyms table (Table 1) and a discussion on the hardware platforms, operating systems and simulators that appear most commonly in IoT scenarios. Anyway, a reader already knowledgeable regarding IoT hardware (Subsection 2.1), software (Subsection 2.2) and simulators (Subsection 2.3) can disregard the related subsections, and get directly to the discussion at the end of the section (Subsection 2.4). The conclusions drawn by the discussion drive the selection of the hardware and operating system that are targeted by the direct measurement activities described in the rest of the paper.

2.1 Hardware

Hardware technology for sensor nodes is changing due to advances in Micro-Electro Mechanical System (MEMS) technology, which have led to smaller and cheaper sensor nodes [3]. A wireless sensor node is composed by a

Table 1 Table of acronyms

Acronym	Full Text
ACM	Association for Computing Machinery
ADC	Analog-to-Digital Converter
ASIC	Application Specific Integrated Circuits
CPS	Cyber Physical System
CPU	Central Processing Unit
DIP	DIssemination Protocol
DYMO	DYnamic Manet On-demand protocol
FPGA	Field-Programmable Gate Array
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
LED	Light-Emitting Diode
LUT	Lookup Table
MAC	Medium Access Control
MEMS	Micro-Electro Mechanical System
nesC	Network Embedded System C
OS	Operating System
RAM	Random Access Memory
RF	Radio Frequency
SD	Secure Digital
SUT	System Under Test
VHDL	VHSIC Hardware Description Language
WSN	Wireless Sensor Network

micro-controller, memory, timer, transceiver, battery, sensing unit and Analog-to-Digital Converter (ADC) [4]. Figure 1 shows a simplified block diagram of a sensor node typical architecture.

Wireless sensor networks are nowadays based on an impressive number of different platforms [25] that provide hardware that can host sensor network

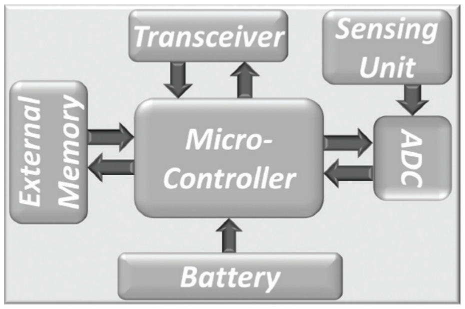

Figure 1 Typical sensor node architecture.

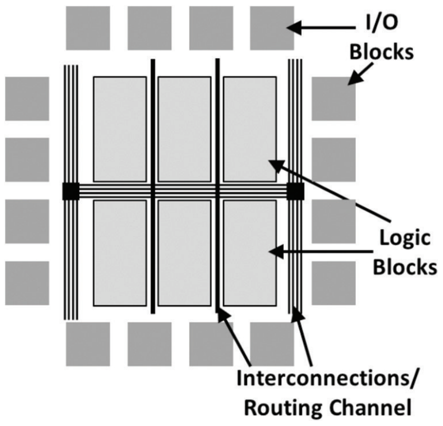

Figure 2 Typical FPGA architecture.

operating systems and applications. Lately, an alternative approach has been based on FPGA and reconfigurable hardware platforms (see Figure 2).

2.1.1 FPGA-based sensor networks

FPGA has been explored as the hardware for sensor networks for different goals.

First of all, an intermediate step between simulation and implementation on Application Specific Integrated Circuits (ASICs), also kwown as traditional hardware platforms, is based on hardware emulation, since it allows to produce a sensor platform with initial minimal investment [26]. It is thus possible to validate research solutions, programmed in Verilog or Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL) before implementing the solution in real hardware.

Moreover, FPGA is able to execute computationally intensive data processing more efficiently than low cost general-purpose microcontroller [27], and the solution resulting from the merging of FPGA with traditional sensor hardware is able to provide an efficient trade-off between low cost and efficient computations. In fact, FPGA platform can exploit the inherent parallelism in the tasks execution to speed-up the execution and end up with lower energy consumption. The work in [28] studied a sensor data processing platform providing a design matrix including tasks' partitioning, a low-complexity background subtraction, bilevel coding, and duty cycling, and it showed an energy reduction of up to a factor of 69 as compared with software solutions on traditional hardware.

Finally, when the simulation of extensive distributed systems is necessary, simulators run on traditional computers have limited computational power and bandwidth, and the implementation on FPGA-based platform enable higher computational effectiveness [29]. Thus, FPGA platform are useful for the extensive simulation of huge sensor networks, harvesting the advantages related to fully parallel hardwired architectures, reconfigurability, fast clock speed, and sophisticated design tools.

Different FPGA platforms were compared [30], to understand the advantages of each platform on its intended scenario. The Basys2 evaluation platform appears to be well accepted in the research community, for both its computational performance [31], low cost and reconfigurability [32].

2.2 Operating Systems

Several Operating Systems (OS) have been proposed to manage sensor nodes' constrained hardware resources in an efficient manner, and to allow the concurrent execution of multiple applications. According to the authors of [4, 5], the most popular OS for sensor nodes are TinyOS, Contiki, MANTIS, Nano-RK and LiteOS. The study conducted in [6], based on scientific and engineering databases including IEEE Xplore, ACM Digital Library and Science Direct, states that TinyOS and Contiki account for 81% and 9% respectively of the global references, and are considered in the following.

2.2.1 TinyOS operating system

TinyOS [7] is a multi-platform, component-based and open-source OS developed at the University of California (Berkley). It presents a footprint of about 400 bytes and it falls under the monolithic architecture class. TinyOS can support concurrent programs with low memory requirements and its execution model is event-based. TinyOS applications are designed as interaction between components, the latter being independent computational entities that expose one or more interfaces. The components are written and *wired* together using nesC (Network Embedded System C) [8], a component-based programming language based on C. Components present three computational abstractions: commands, events and tasks. Commands and events are mechanisms for inter-component communication, while tasks are used to express intra-component concurrency.

TinyOS simulates concurrency using TinyOS threads (TOSThreads), which were first introduced in [9]. TOSThreads are lightweight (context switches and system calls introduce an overhead of 0.92% or less [9]) and use either a non-preemptive First-In-First-Out (FIFO) scheduling or the Earliest Deadline First (EDF) scheduling [7]. These two algorithms present known disadvantages (FIFO's waiting time depends on task execution time and EDF does not produce a feasible schedule when tasks compete for resources), and it can be concluded that TinyOS does not provide solid real-time scheduling.

Efficient memory and type safety for TinyOS were introduced in [10] for Mica2, MicaZ and TelosB platforms. The presented mechanisms ensure safe execution by providing protection of Random Access Memory (RAM) against array and pointer errors, useful diagnostics, and recovery strategies. The internals make use of the Deputy concept, i.e. a resource to resource compiler that ensures type and memory safety. Current TinyOS provides communication support for communication through a number of routing protocols such as 6lowpan [17], DYMO [18] and DIP [19]. At the MAC layer, TinyOS provides implementation for a large number of protocols, comprising: a single hop TDMA protocol, a TDMA/CSMA hybrid protocol which implements Z-MAC's slot stealing optimization, B-MAC, and an optional implementation of an IEEE 802.15.4 complaint MAC [3]. To accomplish resource sharing, TinyOS uses two mechanisms, the Virtualization to provide independent virtual instances of a resource, and the Completion Event to handle sequential access to resources that cannot be virtualized.

TinyOS provides a single level file system, due to the assumption that for a given point in time only a single application is running. The limited amount of available memory supports this assumption. Other features supported by TinyOS are database support (TinyDB [11]), security for communications (TinySec [12]) and simulation of TinyOS applications (TOSSIM [13], described in Subsection 2.3.1). TinyOS widely adoption is also due to its extensive documentation, which can be found on the TinyOS home page (http://www.tinyos.net).

2.2.2 Contiki operating system

Contiki [20] is a lightweight operating system for memory-constrained device. While other operating systems need at least a few megabytes to run, Contiki can be executed with just 2 kilobytes of RAM. The OS is implemented in the plain C language, and the applications run under this OS are expected to be written in C. Contiki has been ported to a number of microcontroller architectures [22], comprising MSP430 and Atmel AVR.

The kernel of Contiki supports multiprogramming through a handful of mechanisms. Applications can be event-based, and both interrupt routines and the primary execution flow can dispatch events that cause the execution of event handlers. On top of this simple event-driven execution model, The OS provides multi-threading by means of Protothreads [21], which are stackless thread (state must be saved in the private memory of the process), cannot be preempted, and each protothread run until it puts itself into a waiting state. Finally, Contiki offers preemptive multi-threading, which is much more memory-consuming but provides each thread with an independent stack [20]. All these mechanisms requires deep user intervention, and Contiki cannot be considered to provide solid real-time scheduling.

Contiki has the particularity of being able to selectively reprogram parts of the on-chip flash memory, to flush out routines not under use and minimize used memory at any point in time. Contiki provides a full IP network stack by means of the μIP library [24], in addition to low-power standards such as 6lowpan, RPL, and CoAP [23].

Contiki is now managed and commercialized by Thingsquare, a company created by Contiki author, and it is thus well maintained. Finally, Cooja [16], described in Subsection 2.3.4, provides simulation environment for Contiki application, allowing to execute the same code on the simulated nodes, and the real ones, at least for some of the hardware platforms (see [22] for the list of platforms supported by Cooja).

2.3 Simulators

Usage of simulators allows to set up a measurement pipeline without working on the hardware of devices. The approach scales well because devices can be instantiated programmatically, while direct measurement needs preparation work on each and any physical device involved in the scenario. The simulators are the recipient for direct measurement results, since the latter are used to make the simulators more realistic.

This subsection presents the most widely used simulators tailored to wireless sensor networks. We are here disregarding general-purpose simulators such as network simulator 2 (ns-2) and network simulator 3 (ns-3), since they tend to have more complex simulation setup and can provide energy information regarding communication activities only.

2.3.1 TOSSIM

TOSSIM [13] is a C/C++ library included in the TinyOS framework, and it replaces low-level hardware components with simulated implementations, including models for CPUs, ADCs, clocks, timers, flash memories and radio components. The simulation is configured by writing a program that defines the scenario. The code used for the components can be the same nesC code that is deployed onto nodes. The scenario definition is written either in C++ or Python. The Python library allows the user to interact with the running simulation dynamically, however, at the cost of performance when obtaining the simulation results.

The hardware model employed by TOSSIM is quite abstract, and it is impossible to capture low-level details of timings and interrupts, which are important for precise power analysis. Also, the simulation is only supported for a single hardware platform, the MicaZ, making it a platform and OS-specific tool.

2.3.2 PowerTOSSIM

PowerTOSSIM [14] is an extension to TOSSIM that enables the estimation of node power consumption. It is shipped with a detailed model for hardware energy consumption of the Mica2 sensor node platform, built by extensive application-level benchmarking. Anyway, PowerTOSSIM is extensible and can be customized for different platforms, by mean of the same process of extensive application-level bechmarking. PowerTOSSIM authors ensure that it is able to achieve results within 13% of the power consumption of real hardware nodes [14].

2.3.3 Avrora

Avrora [15] is open-source and widely used, it is a cycle-accurate instruction level simulator, and scales to networks of up to 10,000 nodes. It is language and operating system independent, since it simulates the assembly code compiled from the source code of the applications. Applications are run without the need to specially adapt them for simulation.

Avrora presents support for sensor platforms such as Mica2 and MicaZ, and runs AVR elf-binary or assembly codes. Avrora is written in Java, each hardware component is represented as an object-oriented class, and each node is managed as an independent computational entity. It is possible to retrieve information from the application simulation both at runtime (e.g.: current LEDs state) and as a summary at the end of the simulation (e.g.: total energy consumption). Avrora is available from its sourceforge[1] repository, it is not actively maintained and it does not provide extensions for CPU architectures different than the AVRMCU cores, making it a platform-specific simulator.

[1]http://avrora.sourceforge.net/

2.3.4 Cooja

Cooja [16] is devoted to simulating the Contiki OS, as executed on on either TI-MSP430 or AtmelAVR microcontrollers. It enables simultaneous simulations at the network, operating system and machine code instruction set level, to verify application before being uploaded to sensor nodes. Each node can differ both in on-board software and simulated hardware. Contiki programs can be executed either by running compiled code directly on the host CPU, or by emulating the compiled program code in an instruction-level TI-MSP430 emulator.

2.4 Comparative Analysis

The information reported in this section were used to build up a comparison of the different hardware platforms, operating systems, and simulation environments.

Results of the comparison of the simulators are summarized in Table 2. The simulators operate at different levels. Within TOSSIM and Power TOSSIM, the applications are written in nesC and converted into the simulation code, and thus they are language and OS dependent [15], while Avrora and Cooja can simulate machine code and considered to be language and OS independent. TOSSIM does not have advanced power simulation. On the other hand, PowerTOSSIM introduces a detailed model for the hardware energy consumption built from real-life tests of the Mica2 platform, and it can be extended to other platforms. Avrora represents the hardware through Java classes and is capable of performing energy consumption simulations, but it supports the AVRMCU core only, is getting outdated and is not actively

Table 2 Simulators comparative analysis

Simulator	TOSSIM	PowerTOSSIM	Avrora	Cooja
Simulation level	Operating system	Operating system	Instruction level	Network, OS and machine code
Hardware representation	Abstract hardware model	Abstract hardware model	Object classes (Java)	Not available
Simulation interface	C++/Python	C++/Python	Java	Java
Energy consumption	No	Yes	Yes	No
Hardware platform	MicaZ	MicaZ	AVRMCU cores (Mica2 and MicaZ)	TI MSP430 cores, Atmel AVR cores

maintained. Concerning Cooja, no information on power consumption simulation was found, and it is believed to lack any kind of power consumption simulation tool.

Since the simulator that appears to be the best choice for energy simulation is PowerTOSSIM, which is focused on TinyOS, and since this latter operating system is the leader in terms of adoption at worldwide level [6], this discussion leads us to focus our measuring efforts on applications running over TinyOS. Regarding the hardware targeted by the measurement, it was opted to focus on a number of mainstream sensor platforms (MSP430 microcontroller on TelosB platform and Atmega128 microcontroller on Iris platform), and on FPGA-based platforms.

3 Energy Measurement Process

The proposed process, represented in Figure 3, is based on the basic formula that says that $P = VI$ (consumed power is equal to the tension multiplied by the current), and it involves direct measurement of both tension and current of a System Under Test (SUT). A SUT is a certain device (e.g. a microcontroller based or with a programmable logic design) that is executing a specified application and on which the direct measurement process is applied. As shown in the figure, the SUT is identified as 'Device + APP Under test'. The measurement process relies on two main blocks, called Circuits and Micro-Controller, which are used together to attain a reasonable resolution of consumed energy.

The Circuits block takes care of transforming the tension and current physical values into digital signals. An analysis of the operating values for the devices points out that the sensor nodes, powered by two batteries having

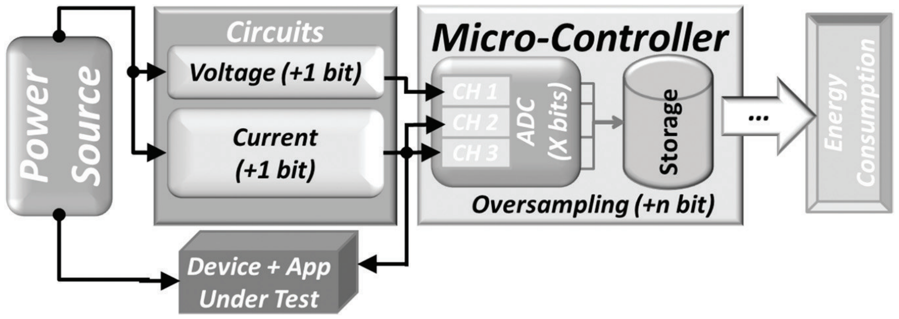

Figure 3 Energy measurement process.

Table 3 Arduino DUE – specifications (resumed)

Microcontroller	AT91SAM3X8E	SRAM	96 KB
Operating voltage	3.3 V	Clock speed	84 MHz
Digital I/O pins	54	Flash memory	512 KB
Analog inputs	16	ADC resolution	12-bit 1Msps

maximum voltage between 3 V and 3.3 V, can work as long as batteries can provide at least 2.1 V. Thus, the Circuit block was developed to measure voltages between 1.65 V to 3.3 V. For the current signal, no values can be excluded, and the signal is obtained using a common sensing resistor assembly (a low resistor value so it cannot interfere relevantly in the SUT energy consumption). For both elements, each measuring unit splits the signal in the middle into two part, and the following unit is tuned on the top or lower part of the signal, to gain an extra bit of definition for each measuring unit in the Circuit block.

The Micro-Controller produces the digital values corresponding to the current and tension analog signal. The block has got at least three ADC inputs, a good ADC transformer, a reasonable micro-controller clock speed and some storage capabilities. The block can easily transform the incoming analog signal into bits. It is also possible to add bits using software, using the oversampling technique. The theory says that collecting 4^n additional samples leads to getting *n* more bits in the measured values, but the implementation of this approach has the drawback of a direct impact on the sampling rate, and the application of the technique represents a trade-off to be considered at design time. Currently, the Micro-Controller block is implemented in a Arduino DUE (Table 3 summarizes its specifications) coupled with a Micro-SD card shield for storage.

The implemented SUT aim at a final precision of 14-bit for both voltage and current signals, obtained one bit by hardware, plus 12-bits given by the ADC and another one using the oversampling method. The sampling rate for the measurements ended up being 34 KHz (one sample every $\sim 29.5/\mu\text{s}$).

4 Tests

The tests were designed based on an analysis on the market share of IoT devices, and on the study on OS reported in Section 2. Five different devices were selected, four of them having similar features and compatible with both TinyOS and Contiki OS.

Table 4 Micro controller based devices selected to be tested

	Platform			Microcontroller		Memory		Radio
	Design	OSs		Ref.	Prog. Flash	Data RAM	Ext. Flash	RF Chip
XM1000				MSP430 @16 MHz	116 KB	8 KB	1 MB	
CM5000	TelosB	TinyOS	Contiki	MSP430 @8 MHz	48 KB	10 KB	1 MB	TI-CC2420
CM3300				MSP430 @8 MHz	48 KB	10 KB	1 MB	
XM2110	Iris			Atmega1281 @8 MHz	128 KB	8 KB	–	Atmel-RF230

Table 4 presents the characteristics of these four devices used in the experimental tests, in terms of their chipset, wireless interface, quantity of RAM and flash memory, and CPU speed. Apart from the characteristics presented in Table 4, the CM3300 device has got an amplifier to provide more power to the antenna. Since it is the market leader, all applications were executed over tinyOS (version 2.1.2).

Additionally, and as a competitor, a different device was also tested, based on a programmable logic design. This device, a development board (Basys2), is a ready-to-use hardware for digital circuit design, equipped with a Xilinx Spartan 3E-250 FPGA, onboard buttons, switches, leds, PS/2 and 8-bit VGA ports. The Spartan 3E presents the following main characteristics, among others: 250 K system gates, 5508 equivalent logic cells, 38 K distributed RAM bits, 12 dedicated multiplexers and a maximum of 172 user Input/Outputs.

Four simple applications were selected for this preliminary study. The simplest application (**Empty**) is a void application, and it helped studying how the devices take care of sleep states, and provide a baseline for the power consumption in the idle state. In the FPGA case this test consists on examining the energy consumption when the board is power on, since there is no operating system on this kind of devices. Since the most important job for sensor nodes is to sense the environment, the second application (**Timer 1000 ms**) studied how the timers are managed by the OS and the devices, by setting up and firing a timer with a fixed period of 1 second. Finally, three applications (**Blink Led 0, Blink Led 1, Blink Led 2**) were switching periodically on/off one of the three LEDs of the device, and they were used to study how these operations are scheduled, and how much power the LEDs consume. In the experiments, the LEDs were turned on and off with

Table 5 Programmed applications Size (bytes)

	Empty (ROM/RAM)	Timer 1000 ms (ROM/RAM)	Blink Led 0, 1, 2 (ROM/RAM)
CM3300	1320/6	2250/36	2420/56
XM1000	1244/6	2174/36	2344/56
XM2110	754/4	2088/33	2182/51
CM5000	1320/6	2250/36	2420/56

Table 6 Device (FPGA/Basys2) utilization summary

	Timer 1000 ms	Blink Led
Number of Slice Flip Flops	24 (<1%)	33 (<1%)
Number of used 4 input LUTs as logic	7 (<1%)	17 (<1%)
Number of used 4 input LUTs as route-thru	21 (<1%)	28 (<1%)

the *toggle* function of TinyOS. In the case of **Blink Led 0**, the LED stayed on for 500 ms, then off for 500 ms, and so on for CM3300, XM2110 and CM5000; the led stayed in the on and off states for 1s in the case of XM1000. In the **Blink Led 1** and **Blink Led 2** applications, all LEDs stayed on and off for 1s at a time.

Table 5 shows the memory footprint of programming each application on the respective device, on the other hand, Table 6 shows the amount of logic circuits used in the FPGA tests. Namely, the number of slice flip flops and Lookup Tables (LUTs), a slice is a block that can contain LUTs and flip-flops. A LUT is used to encode boolean functions by storing the truth table associated to that function.

In the Basys2 case, only appears the components utilization for two applications. This is due to the fact that FPGA has no need of OS, therefore no component is used the (**Empty**) test. And since all Leds available in the development board, Basys2, are of the same color the individual energy consumption is the same.

5 Results

The data collected from the tests, performed to telosB and Iris devices, is represented in the Figures 4–8. The results made it clear that these devices, which are sharing the same OS and applications, have got different energy consumption.

From the results regarding the Empty application, it appears a first difference between MSP430 controller and ATmega1281 (Iris). ATmega1281

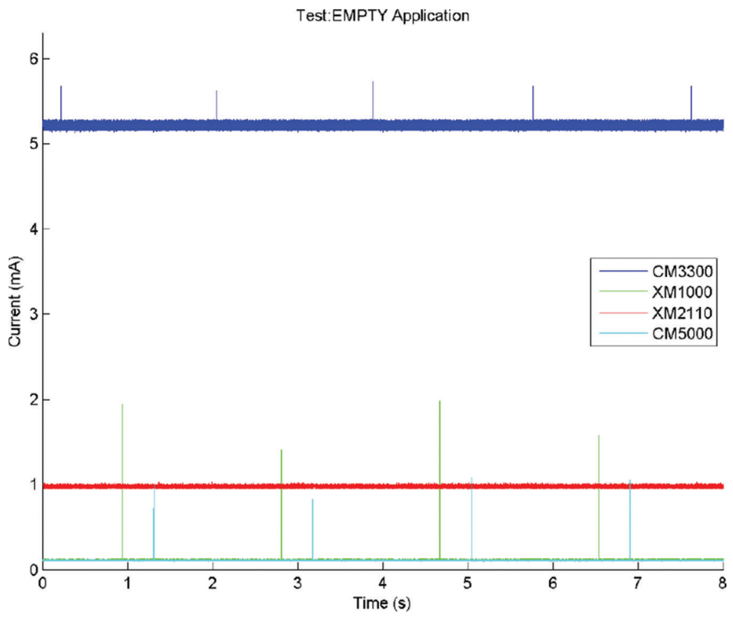

Figure 4 Current results for the empty application.

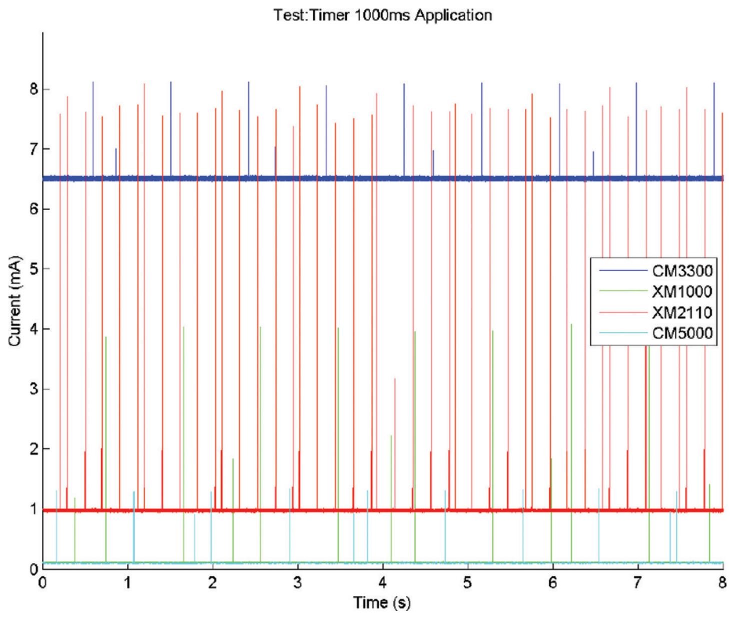

Figure 5 Current results for firing a timer each 1000 ms.

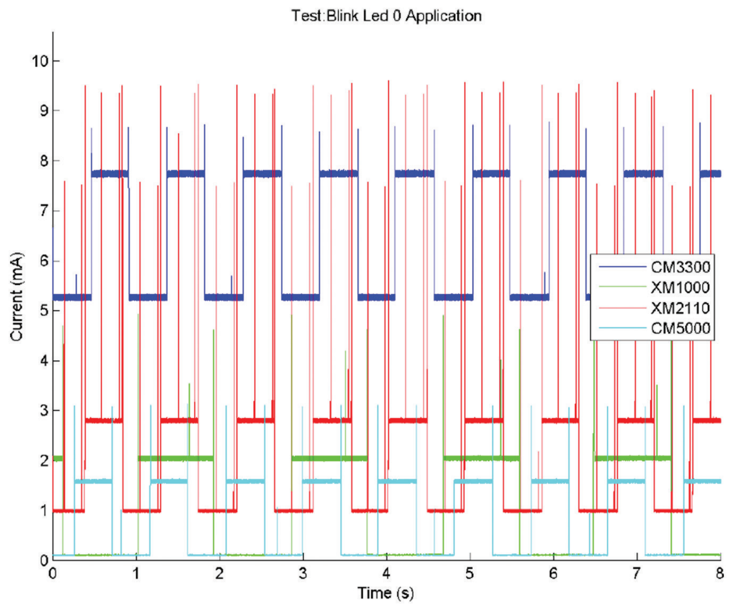

Figure 6 Current results for blink application (Led 0).

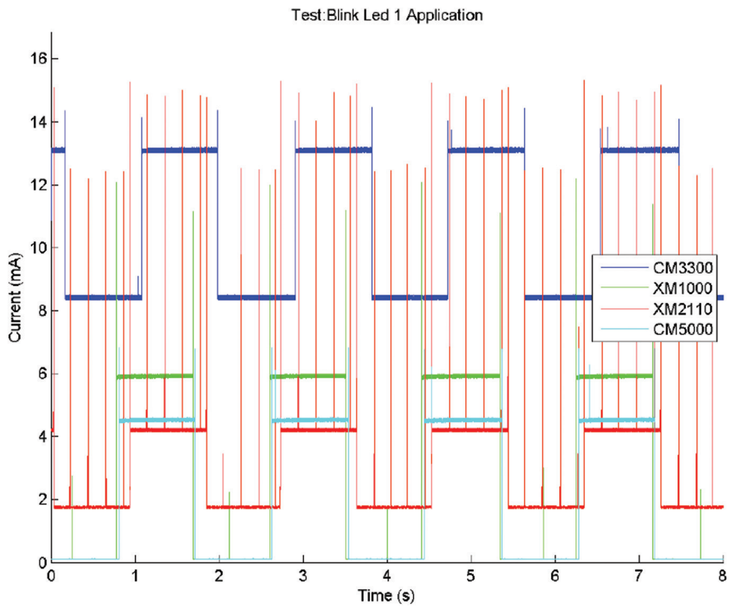

Figure 7 Current results for blink application (Led 1).

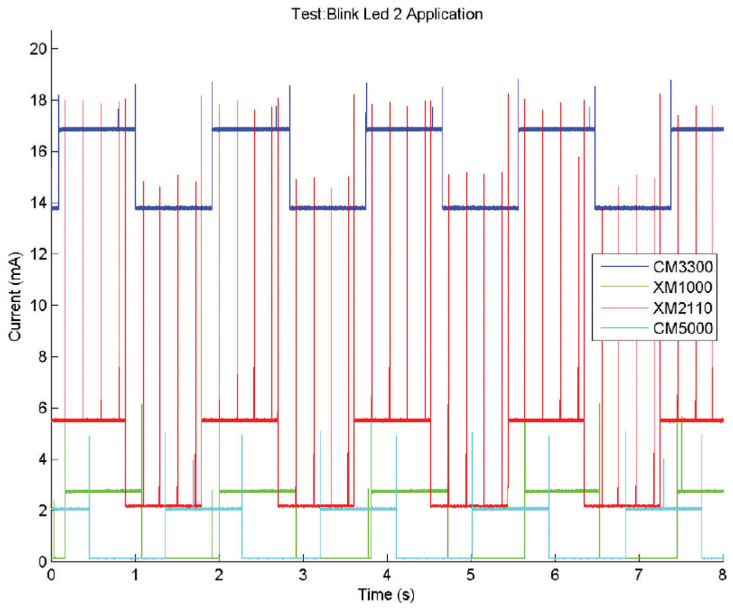

Figure 8 Current results for blink application (Led 2).

is put to sleep completely, while the MSP430 needs to wake every 1.85 s to verify if it has received any interrupt to process. This result is confirmed on the other graphs too, since the MSP430-based sensors have got more consumption spikes. The behaviour is related to the lack of a wake up interrupt of the TelosB microcontroller, causing it to sleep for a fixed period, and then waking up to verify if it has received data to process.

Another result from the Empty application is that the CM3300 devices features extra energy consumption of 5 mA (as mentioned in its datasheet) due to an amplifier used in wireless communication, even when the radio is not part of the picture. Thus, the CM3300 sensor has got no way to switch off the amplifier, at least with the current TinyOS libraries. XM1000 and CM500, which are both based on TelosB architecture, present similar consumption results when executing the Empty application, but anyway they feature lower energy consumption than the Iris device.

Figure 5 depicts the energy consumption when a 1000 ms timer is fired. From the data it is possible to conclude that energy consumption results are similar to those presented in the Empty application case. The main difference resides on additional peaks that are visible in all traces. This fact can be assigned to the microcontroller need to control the elapsed time to verify whether the timer period has ended.

The results from the Blink Leds applications shows that significant differences exist between devices, which are directly related to the led color used. Iris XM2110 presents for the three leds, whose colors are: Led 0 – red; Led 1 – green; and Led 2 – yellow. The TelosB motes present the same led configuration: Led 0 – red; Led 1 – yellow; and Led 2 – blue. The yellow in XM2110 (Led 2) has a different color shade than the yellow in TelosB motes (Led 1).

CM3300 is excluded from further analysis since it consumes always more energy due to its additional amplifier for wireless communication. Another results from Led 0 blinking application is that the Iris device appears to consume more energy than the others, due to a higher impact in energy consumption by the operating system in this type of devices (see Figure 4).

In the case of Led 1, the XM1000 is the mote that consumes more energy. XM2110's led color is of different shade, and the XM1000 presents better hardware features than the CM5000, which leads to a slightly higher consumption. The results in Figure 8 reinforce the analysis made in the previous cases: the Iris device has a different led but, when comparing to Figure 7, it presents a higher consumption (the yellow leds cases), and again the XM1000 has a higher consumption for the Blinking Led 2 application than the CM5000.

In order to compare different devices design, microcontroller and field-programmable approaches, three tests were made to the development board – Basys2. Tests follow the same principle as the previous ones, energy consumption analysis when an Empty, timer firing each 1000 ms, and blink a led applications are applied, but now to the Basys2. The data collected from the tests are presented in Figures 9–11.

The first thing that stands out, is the signal (current consumption) instability in all three tests. In this sense, the Basys2 development board presents some issues regarding the provided electrical power, and as it is visible in the tests, the signal takes the form of a sawtooth with is a variation around 5 mA, what is considerable.

Figure 9 reflects the inherent consumption of all development board components, and not only of the FPGA, when there is not an implemented application. The mean value, more or less, is about 37.5 mA and it will be used as a comparative value to identify the additional consumption imposed by the other two applications.

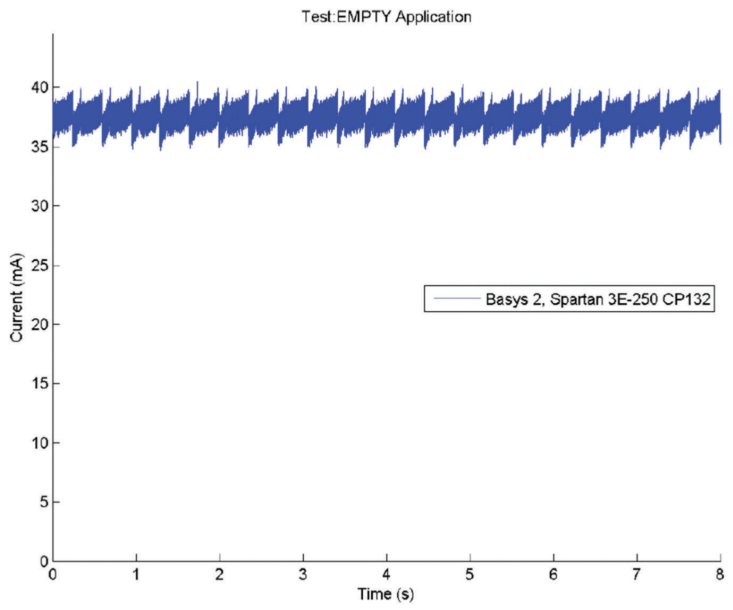

Figure 9 Current results for Basys2 without any application running.

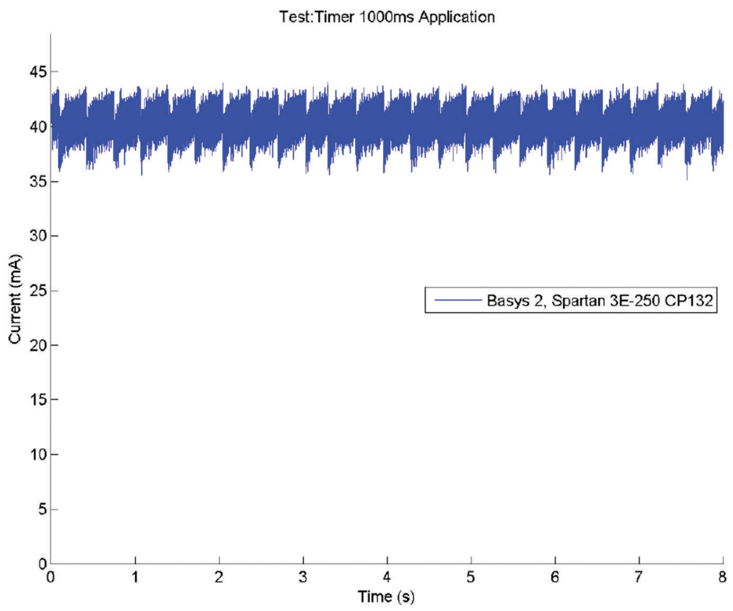

Figure 10 Current results for firing a timer each 1000 ms on Basys2.

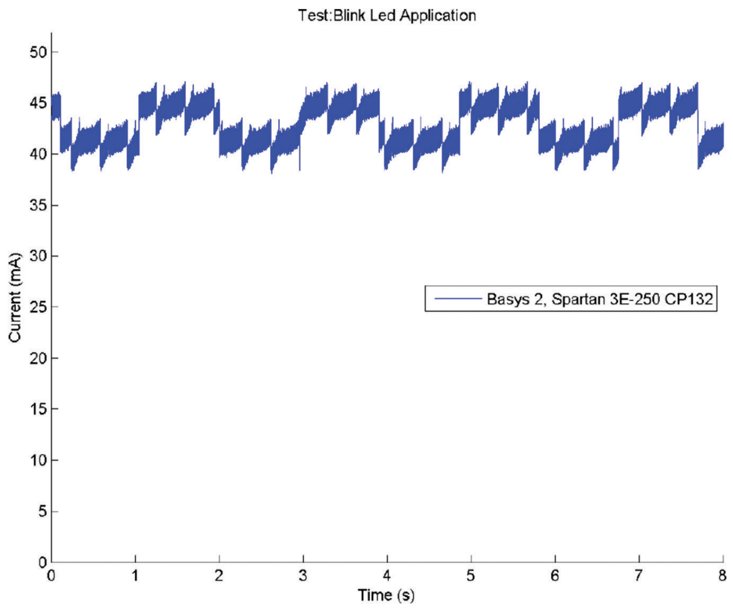

Figure 11 Current results for blink application running on Basys2.

As previously mentioned, an application in which a timer is fired each 1000 ms was tested in Basys2 (see Figure 10). The consumption associated to this application presents a average current value of 40.5 mA. Therefore, firing a timer each 1000 ms in a Basys2 board has an additional current impact of 3 mA.

The last test accomplished with the development board Basys2 reflects the consumption when a Led (of yellow colour) blinks every 1000 ms (see Figure 11). When the Led is off the current signal (average) presents the same value as the previous test, i.e. 40.5 mA. On the other hand, when the Led is on, the mean current value is 44.5 mA. Consequently, is possible to conclude that activating a Led has an impact of 4 mA in the Basys2. Note that the development board, Basys2, has eight leds all with the same colour. In this sense, no tests were made to the other seven leds, individually, since results would be the same.

Table 7 resumes the current consumption values obtained in each one of the tests presented. Comparing the results between microcontroller based devices (telosB and Iris) with the programmable logic design board (Basys2), it is clear that this last one consumes more energy than the majority of the others tested devices. First, it presents a higher energy consumption impact for just

Table 7 Current results summary vs applications (values in mA)

			Blink Leds [On]		
	Empty	Timer 1000 ms	0	1	2
CM3300	5.25	6.50	7.74 (Red)	13.1 (Yellow)	16.87 (Blue)
XM1000	0.10	0.12	2.04 (Red)	5.91 (Yellow)	2.74 (Blue)
XM2110	0.95	1.00	2.87 (Red)	4.28 (Green)	5.56 (Yellow)
CM5000	0.10	0.12	1.59 (Red)	4.52 (Yellow)	2.05 (Blue)
Basys2	37.5	40.5	44.5 (Yellow)	–	–

having it turned on (Basys2 37.5 mA versus CM3300 5.25 mA). Although, this difference can be associate to the Basys2 extra components, and not directly to Spartan 3E-250 FPGA. Second, and without reason for an extra consumption by extra components, is the difference between the Empty application and blink a Led (Yellow). The results shows that only the CM3300 presents a higher difference than the Basys2 (7.9 mA for CM3300, and 7 mA for Basys2).

Based on the results, it is possible for an application designer to select the device that has got the longer lifetime. For example, an application that can sleep for a long time and get back to work only when receiving an interrupt, could be deployed on an Iris. Another conclusion that can be drawn is that it would be better not to use devices equipped with the amplifier for most applications, since it is currently impossible to switch it off and thus any application on that platform would always consume a lot of energy.

6 Conclusions

A number of conclusions can be drawn from the results. It is hard for the programmer to have complete knowledge of the energy consumption of his applications, since many low level details are hidden by the OS, and the details of the operations executed by the OS are device-dependent. What started as a work to tune up the energy model for energy simulations with data taken from real hardware, ended up being a critical analysis of how the same application code is executed on different platforms. For example the results highlighted that a simple operation such as just blinking a led, has got different energy consumption impacts that can be predicted by performing proper platform analysis. The results also hinted that experiments must be devoted to verifying the OS capabilities, in the sense that some OS capabilities are not yet implemented on some hardware – such as switching off the amplifier of the radio on the CM3300 sensor – and this limitation has got an impact on energy consumption that cannot be predicted without proper experimentation on prototypes. Thus, it appears of the utmost importance to have access to data measured directly on the hardware devices with the correct configurations and software, to be able to predict energy consumption of complex applications in a realistic manner.

Future work involves the extension of the analysis to different hardware elements of the architecture (Analog-to-Digital and Digital-to-Analog Converters, network interface, etc). A further step will consider measuring the energy consumption in more complex operations (the sensor takes a measurement, converts it to digital, stores it, and sends it via the wireless interface) to compare it to the expected energy consumption related to the measurements done on the single hardware elements. Finally, we aim to define a complete architecture, in terms of hardware and software, to facilitate the direct measurement of energy consumption, to be distilled into information to be used in model-based simulations of energy consumption. The approach will streamline how data is included into the model-based simulators, to allow the application designer to receive realistic and accurate analysis of the consumed energy by simulation only.

Acknowledgments

This research work was partially supported by national funds provided via the Portuguese Agency "Fundação para a Ciência e a Tecnologia" (FCT) and funds provided by the European Commission in the scope of ECSEL/H2020-662189 MANTIS and ARTEMIS/FCT-332987 Arrowhead RTD projects and the UID/EEA/00066/2013 PEST (Strategic Plan for Science and Technology) project for the Centre of Technology and Systems (CTS). Furthermore, we would like to thank to João Rodrigues and José Gonçalves for their contribution in testing the devices.

References

[1] Pawlowski, A., Guzmán, J. L., Rodríguez, F., Berenguel, M., Sánchez, J., and Dormido, S. (2009). Simulation of greenhouse climate monitoring and control with wireless sensor network and event-based control. *Sensors* 9, 232–252.

[2] Ferre, J. A., Pawlowski, A., Guzmán, J. L., Rodríguez, F., and Berenguel, M. (2010). "A wireless sensor network for greenhouse climate monitoring," in *proceeding of the 5th International Conference on Broadband*

Communication, Information Technology and Biomedical Applications. Malaga, Spain.

[3] Yick, J., Mukherjee, B., and Ghosal, D. (2008). Wireless sensor network survey. *Comput. Networks*, 58, 2292–2330.

[4] Farooq, M. O., and Kunz, T. (2011). Operating systems for wireless sensor networks: a survey. *Sensors* 11, 5900–5930.

[5] Moschitta, A., and Neri, I. (2014). "Power consumption Assessment in Wireless Sensor Networks," *in ICT – Energy – Concepts Towards Zero – Power Information and Communication Technology*, ed. D. G. Fagas.

[6] Lajara, R., Pelegr-Sebasti, J., and Perez Solano, J. J. (2010). Power consumption analysis of operating systems for wireless sensor networks. *Sensors* 10, 5809–5826.

[7] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A. (2005). TinyOS: an operating system for sensor networks. *Ambient Intell.* 115–148.

[8] Gay, D., Levis, P., Von Behren, R., Welsh, M., Brewer, E., and Culler, D. (2003). The nesC language: a holistic approach to networked embedded systems. *PLDI* 38, 1–11.

[9] Klues, K. Mike Liang, C.-J., Paek, J., Musaloiu-E, Levis, P., Terzis A. et al. (2009). "TOSThreads: thread-safe and non-invasive preemption in TinyOS," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, 127–140.

[10] Cooprider, N., Archer, W., Eide, E., Gay, D., and Regehr, J. (2007). "Efficient memory safety for TinyOS," in *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, 205–218.

[11] Madden S. R., Franklin, M. J., Hellerstein, M., and Hong, W. (2005). TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30, 122–173.

[12] Karlof, C., Sastry, N., and Wagner, D. (2004). "TinySec: a link layer security architecture for wireless sensor networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, 162.

[13] Levis, P., Lee, N., Welsh, M., and Culler, D. (2003). "TOSSIM: accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, 126–137.

[14] Perla, E., Catháin, A. Ó., Carbajo, R. S., Huggard, M., and Goldrick, C. M. (2008). "PowerTOSSIM z: realistic energy modelling for wireless sensor network environments," in *Proceedings of the 3rd ACM Workshop on*

Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks, 35–42.

[15] Titzer, B. L., Lee, D. K., and Palsberg, J. (2005). Avrora: scalable sensor network simulation with precise timing, *Proceedings of the 4th international symposium on Information Processing in Sensor Networks*, Los Angeles, CA: IEEE Press.

[16] Osterlind, F., Dunkels, A., Eriksson, Finne, N., and Voigt, T. (2006). "Cross-Level Sensor Network Simulation with COOJA," in *Proceedings of 2006 31st IEEE Conference on Local Computer Networks*.

[17] Harvan, M. (2007). *Connecting Wireless Sensor Networks to the Internet-a 6lowpan Implementation for Tinyos 2.0*. Master thesis, Jacobs University Bremen, Germany.

[18] Thouvenin, R. (2007). *Implementing and Evaluating the Dynamic Manet on-Demand Protocol in Wireless Sensor Networks*. Master thesis, Aarhus University, Denmark.

[19] Lin, K., and Levis, P. (2008). "Data discovery and dissemination with dip," in *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008)*, Washington, DC: IEEE Computer Society.

[20] Dunkels, A., Gronvall, B., and Voigt, T. (2004). "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN 2004)*, Washington, DC: IEEE Computer Society.

[21] Dunkels, A., Schmidt, O., Voigt, T., and Ali, M. (2006). "Protothreads: simplifying event-driven programming of memory-constrained embedded systems," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems* (New York, NY: ACM), 29–42.

[22] Thingsware website. *Contiki Hardware*. Available at http://www.contiki-os.org/hardware.html

[23] Reusing, T. (2012). "Comparison of operating systems TinyOS and Contiki," in *Proceedings of the Seminar Sensor Nodes – Operation, Network and Application (SN)*, eds G. Carle, C. Schmitt, A. Klein, U. Baumgarten, and C. Söllner (Munich, Germany: Technical University of Munich).

[24] Dunkels, A. (2003). "Full TCP/IP for 8-bit architectures," in *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services* (New York, NY: ACM).

[25] Wikipedia website. *List of Wireless Sensor Nodes*. Available at https://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes

[26] Vam, D. P., Rimal, B. P., Maier, M., and Valcarenghi, L. (2016). "Design, analysis, and hardware emulation of a novel energy conservation scheme for sensor enhanced FiWi networks (ECO-SFiWi)." *IEEE J. Sel. Areas Commun*. 34, 1645–1662.

[27] Zhang, X., Heys, H. M., and Li, C. (2013). "FPGA implementation and energy cost analysis of two light-weight involutional block ciphers targeted to wireless sensor networks." *Mobile Netw. Appl*. 18, 222–234.

[28] Imran, M., Shahzad, K., and Ahmad, N. (2014). "Energy-efficient SRAM FPGA-based wireless vision sensor node: SENTIOF-CAM." *IEEE Trans. Circ. Syst. Video Technol*. 24, 2132–2143.

[29] Chen, Y. and Dinavahi, V. (2014). "Hardware emulation building blocks for real-time simulation of large-scale power grids." *IEEE Trans. Ind. Informat.* 10, 373–381.

[30] Aziz, S. M. and Pham, D. M. (2013). "Energy efficient image transmission in wireless multimedia sensor networks." *IEEE Commun. Lett*. 17, 1084–1087.

[31] Ghauri, S. A., Humayun, H., Ehsan ul Haq, M., and Sohail, F. (2012). "Implementation of Convolutional codes on FPGA," *IEEE International Conference for Internet Technology and Secured Transactions*, 175–178.

[32] Perera, M. D. R., Meegama, R. G. N, and Jayananda, M. K. (2014). "FPGA based single chip solution with 1-wire protocol for the design of smart sensor nodes." *J. of Sensors.* 2014, 11.

Biographies

E. M. Silva is researcher at UNINOVA-CTS Institute and Ph.D. student at Universidade Nova de Lisboa. Holds an MSc in Electrical and Electronic Engineering, speciality in Telecommunications. Has 5+ years of experience in national and international research projects, such as FP7-216420 CuteLoop, FP7-234344 CRESCENDO, FP7-318381 EAR-IT (project coordination team), FP7-288315 PROBE-IT (core project team) and FP7-662189

MANTIS. His primary area of expertise is related with Internet-of-Things (IoT); Hardware Design and Development; and Energy Consumption of Resource-Constrained Devices.

P. Maló, Professor at the Electrotechnical Engineering Department (DEE) of the Faculty of Science and Technology of Universidade Nova de Lisboa (FCT-UNL) and Senior Researcher at the UNINOVA Centre of Technology and Systems (CTS). He is graduated, with M.Sc. in Computer Science and a holds Ph.D. in Computer Engineering. Pedro's core research interests are the interoperability and integrability of (complex) systems with special emphasis on Cyber-Physical Systems / Internet of Things. Pedro has 15+ years practice in the management, research and technical coordination/development of research and innovation projects in ICT domains especially addressing data solutions, systems' interoperability and integration technologies. Pedro is an author of over 70 scientific publications published as book chapters, journal articles or conference papers.

M. Albano is Research Scientist in the CISTER Research Unit of the Polytechnic of Porto, Portugal, working on communication middleware for embedded systems with a focus on the application areas of industrial informatics, smart grids, and green wireless communications. He is a Founding Member of the Technical Committee on Green Communications and Computing (TCGCC). Michele received his degree from the University of Pisa, Italy, has been involved in more than 10 European research projects, and he acted as technical manager for CELTIC project Green-T and work package leader for FP7 IP ROMEO and ITEA2 CarCoDe. His works have been published in more than 70 international conferences and journals, he is on the editorial board of the International Journal of Social Technologies and of the Transactions on Emerging Telecommunication Technologies since 2011, and in 2015 he has been appointed as Editor in Chief for the Journal of Green Engineering.