# A cycle-accurate transaction-level modelled energy simulation approach for heterogeneous Wireless Sensor Networks

Mihai Galos, David Navarro, Fabien Mieyeville, Ian O'Connor
Institute de Nanotechnologie de Lyon, Ecole Centrale de Lyon
UMR CNRS 5270
{mihai.galos,david.navarro,fabien.mieyeville,ian.oconnor@ec-lyon.fr}@ec-lyon.fr

***Abstract*— Wireless Sensor Networks are networks made up of tiny sensor nodes, in medium to large quantities, from several tens to hundreds and even thousands. They are used in fields ranging from military, medical, to structural health monitoring for buildings for example. We have introduced an instruction set simulator in our IDEA1 WSN design framework to account for a fine-grained representation of the software running on the node hardware. We modelled the communication between the nodes' microcontrollers and their radio interface and microcontroller and sensor, respectively. This was done at transaction level by use of the SystemC simulation kernel. An application was developed, consisting of eight nodes compressing this abstract and sending it to their coordinator.**

## I. Introduction

Wireless Sensor Networks (WSN) are being deployed more and more in fields such as automotive [1], biomedical [2], avionics [3] or military [4]. They are comprised of sensor nodes (sometimes called "motes") which have the ability to sense an analogue value (such as temperature, pressure or light), do some sort of processing on it and send it to their neighbour(s) until eventually it gets to a coordinator node (also known as a "sink"). From there on, a computer can take the data and interpret it. Hence, the nodes do not act stand-alone but participate in a collaborative sensing scheme.

The internal structure of the node is presented in Figure 1. As can be seen, the node consists of a microcontroller, a radio transceiver interface, a sensor and a battery.

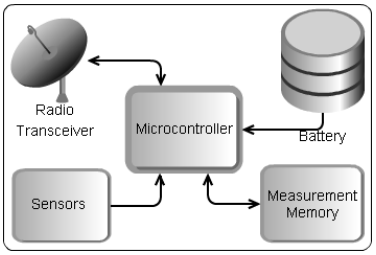


Fig. 1: WSN Node typical internal structure

The radio is the most power-consuming component of the node [5]. Certain radio transceivers like the MRF24J40 [6] have included most of the MAC and some of the DATA layers of the ISO/OSI stack as hard-coded support, effectively reducing the SPI communication time.

The mictrocontroller is the actual brain of the node, coordinating the tasks of the sensor, the radio interface and putting the node in sleep mode whenever idle. Sleeping is an important part of the node's activity, and is in fact the state it spends most of its time in, meaning a low duty cycle. This is because the most important component of the node is its battery. The node is required to run for long periods of time (2 or 3 years) and may even be in the human body, for example. Consequently, the task of replacing the battery is not trivial.

Because of the low power requirements of the node, as well as requirements pertaining to low cost, it is not powerful in terms of processing capabilities. Clock frequencies in excess of 20Mhz are seldom, they are 8 or 16-bit architecture with less than 16kB of RAM, while flash sizes rarely surpass 128kB.

We consider the 802.15.4 standard which is extensively used for both commercial and lab-assembled solutions. Because the most widely used network topologies are mesh and star [7] we consider these when we extended IDEA1's simulation framework.

Simulators built over the years fall into two categories [8]: Network simulators enhanced with node models (PowerTOSSIM [9], sQualNet [10], NS-2 [11], OMNet++ [12]) and node simulators enhanced with network models (Avrora [13] and SCNCL [14]). The former category fails to address low-level hardware considerations (physical node configuration, sensors, microcontroller type, etc.), making rough estimations concerning simulation results. The latter suffers from scalability problems (which in turn increases the simulation time) as well as from lack of heterogeneity support (support for different microcontrollers, different radio transceivers which may be present on the node).

IDEA1 [15] [16] is a validated Wireless Sensor Network design framework centered on energy consumption, inspired from SCNSL and written in C++ and SystemC. SystemC is widely used in the electronics community and offers support for behavioral modelling starting from microcontroller clock cycle, to node, and finally to network scale.

IDEA1 supports microcontrollers from Microchip (PIC) and Atmel (AVR) and radio transceivers such as CC2420 (Chipcon) and MRF24J40 (Microchip), and can simulate both slotted and unslotted CSMA-CA 802.15.4 modes.

In this paper, we present an extention to the original

IDEA1 implementation able to offer finer-grained simulation results through use of the Transaction-Level Modelling (TLM) scheme and the use of our custom-built Instruction Set Simulator (ISS) called Bliss.

## II. Node Hardware / Software Architecture

### A. Overview

In the original IDEA1 implementation, each node component had its own finite state machine implementation. We kept this concept, but we wanted to extend it to model the actual software running on the node. Different binary sizes of the software means different execution times which must be accounted for with a finite state machine approach. What is more, the interfacing of the microcontroller with the sensors and the radio transceiver needs to be TLM-modelled for the best possible energy estimation.

### B. Node Hardware Model

IDEA1 supports Atmel's AVR and Microchip's PIC architectures. With our present work, we added support for Texas Instrument's MSP430 family of microcontrollers, thus covering most of the microcontroller families present on WSN nodes.

We created the SystemC model for the node as is presented in Figure 2. The node is tied to a Proxy which in turn is connected to the Network model. The Network model allows to simulate the physical environment in which the nodes send their radio signals, and to account for variables such as attenuation, radio interference, and transmission time.

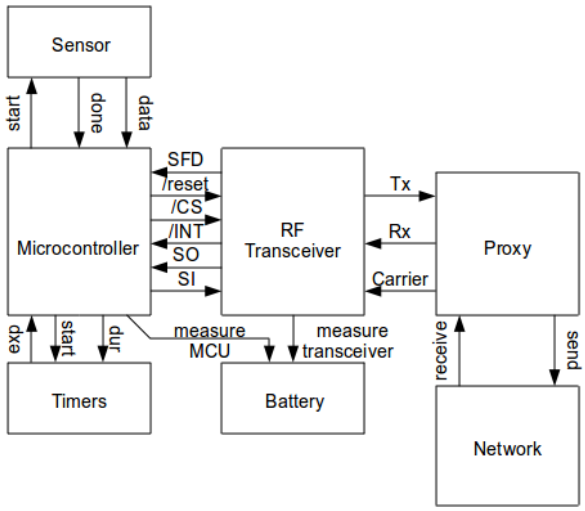


Fig. 2: Designed SystemC node model.

The current consumption for the Node Hardware Model is presented in Table I.

The microcontroller interfaces with the CC2420 radio transceiver through an SPI bus and with the sensor though control signals and a data bus. The battery block is responsible for logging the power consumption when a state change is performed at either radio transceiver or microcontroller part.

| Parameter | Quantity | Units |
|---|---|---|
| MCU frequency | 8 | Mhz |
| MCU active current | 4.2 | mA |
| CC2420 TX@-25dBm | 8.5 | mA |
| CC2420 TX@-15dBm | 9.9 | mA |
| CC2420 TX@-10dBm | 11 | mA |
| CC2420 TX@-5dBm | 14 | mA |
| CC2420 TX@0dBm | 17.4 | mA |
| CC2420 RX | 19.7 | mA |
| SPI Pins | 100 | uA/pin |

TABLE I: Node Hardware model current consumtion.

Knowing the amount of time a single bit takes to be transfered across the SPI interface (depending on SPI and MCU speed), we can compute precisely how much time a transfer lasts. The node hardware model takes into account such things as transceiver initialisation commands, writing to the Tx and reading from the Rx FIFO, as well as periodically repeating these tasks by the means of the Timer block.

Next, by knowing the supply voltage and the current drawn by each block, we have a refined estimate of the node's energy expenditure.

### C. Node Software model

If a functionality is defined by SystemC as being an untimed functional description, the software that makes up the functionality is the timed functional description. Here, the different processes' execution times are important, but we abstract the notion of how these processes communicate between themselves. They do so by means of the Bus Cycle accurate model and the Register Transfer Level (RTL) model.

A common approach would be to use TLM for the compilation unit to take into account the software running on the node in SystemC, based on the hardware implementation of the hardware unit. We developed a split-phase approach composed of a finite state machine representation of the software, coupled with metadata generated by an instruction set simulator. In this case, the node is not tied to an instruction set and has a generic characteristic. The ISS that we developed and coupled with the finite state machine (to have the node software model) is called Bliss.

## III. Bliss

Obviously, ISS-es are nothing new to the research field and other examples include Avrora [13] or MspSim [17], but suffer from two major drawbacks. First of all, they are only targetting one particular hardware architecture and secondly, they are not all written in the same programming language. We designed Bliss in C++ in order for the interfacing with IDEA1 to be easily done. It supports both Avr and Msp430 architectures, while PIC support is underway.

Bliss can simulate the operations (instructions) that a microcontroller executes and builds up a task profile consisting of the task's name and the number of clock cycles it requires. Next, knowing the MCU clock frequency, its state (active, sleep, idle) and the current associated with the state,

we can make a fine estimation on the energy it takes in order to perform each task. Hence, the RTL model of the microcontroller is accounted for.

This is fed as an input to the IDEA1 framework which handles the interaction with other nodes.

Bliss reads the actual modelled functionality in the output format of the compiler used to create the functionality for. Intrinsically, these means Bliss is compiler-dependant. For the AVR variant, Bliss reads the output of avr-gcc. Debugging information has to be included in the output file by specifying a "-gdwarf-2" linker flag. Concerning Msp430, the output of IAR Workbench (v5.10.1) is read. The code needs to be linked by the IAR compiler's linker with options "-yan" using the "elf/dwarf" output file format. The ELF produced is then fed into Bliss which begins simulating the software.

Some of the notable features of Bliss include simulating timer interrupts, simulating hardware interrupts as well as breaking on function entry (breakpoints).

In what concerns the actual Bliss simulation, the approach is different on AVR and MSP430, because of the former being an extended Harvard and the latter a Von Neumann architecture. On AVR, because of the flash and the RAM having different address spaces, the variables have to be fetched to internal registers and the operations performed on the registers. On the MSP430 side, data and flash memory share the same address space and operations can be made between a register and a memory location or even between two memory locations, for example. In both cases, after each instruction, the processor flags are updated and the interrupt timers are evaluated for under/overflow.

The Bliss simulation ends when an instruction jumping to its own address (an endless void loop) is reached. Output is written in the form of a C++ header file and consists of a structure mapping function names to their equivalent number of clock cycles.

In order to validate the correct amount of simulated clock cycles, we first compared Bliss with the clock cycles count taken from AvrStudio and IAR Workbench for several types of well-known algorithms. Because actual implementation can vary, and in order to test the de-facto, baseline implementation of these algorithms, we used C-Lab's WCET (worst-case estimation) benchmarks [18].

The results are presented in Table II.

Bliss AVR has furthermore been validated through comparison with the work of [19]. It is worth mentioning that in the case of sorting algorithms (BubbleSort, InsertSort), the time to sort the vectors depends on the actual values being sorted.

## IV. Results

In order to see actual simulation energy consumption data, we chose to send the text from the abstract of this article over radio from eight nodes to one coordinator, using a simple unslotted CSMA scheme. We wanted to see how much gain (if any), in terms of energy , would result by compressing it as opposed to sending it raw. The intention was to use our extended IDEA1 framework and simulate this scenario on 10 different compressions per node, repeating it 20 times.

Using data from the Bliss analysis of the compression time, we know it takes the number of clock cycles presented in Table III to compress and decompress the data.

| Parameter | Quantity | Units |
|---|---|---|
| Compress abstract | 407919 | clock cycles |
| Compress Time (8Mhz) | 50.98 | ms |
| Decompress abstract | 233195 | clock cycles |
| Decompress Time (8Mhz) | 29.14 | ms |
| Uncompressed size | 769 | bytes |
| Compressed size | 463 | bytes |
| Total energy mean per packet, compressed | 2719.8 (Node) 5214 (Coordinator) | uJ |
| Total energy mean per packet, uncompressed | 5015.2 (Node) 5403.2 (Coordinator) | uJ |

TABLE III: Parameters to send and receive the compressed/raw abstract of this paper.

As can be seen in Figure 3, Figure 4 and Table III, by compressing the abstract and sending it to the coordinator, a mean total of 2719 uJ per packet were needed by the node. The coordinator received and decompressed it using 5214 uJ. We can see how the compression algorithm increases the consumption of the microcontroller in in the active state. However, the node spends less time in sending the data because of it being of shorter length.

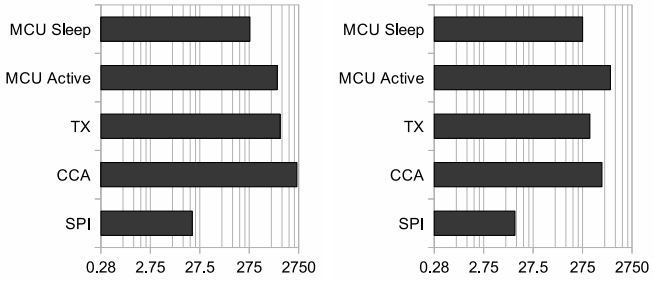


Fig. 3: Mean node energy requirements (uJ) to send the raw (left) and compressed (right) form of the abstract.

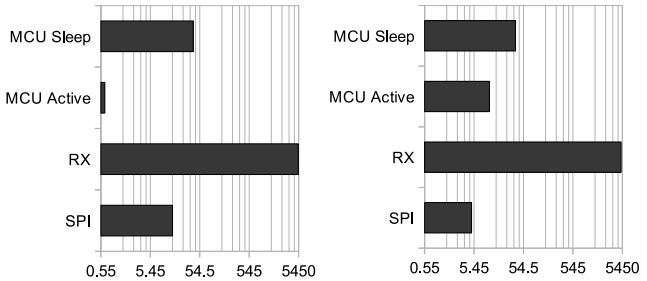


Fig. 4: Mean Coordinator energy requirements (uJ) to receive the raw (left) and compressed (right) form of the abstract.

| Algorithm | Bliss (AVR) | AvrStudio | Bliss(MSP430) | Iar Workbench | Mohan et al [19] |
|---|---|---|---|---|---|
| Bubble Sort(500) | 3,538,519 | 3,538,519 | 3,297,421 | 3,297,421 | 3,900,998 |
| InsertSort(10) | 1,449 | 1,449 | 1,221 | 1,221 | 1,978 |
| Fibonacci | 319 | 330 | 366 | 366 | 258 |
| Matrix Multiply([2][2]) | 2,724 | 2,724 | 2,219 | 2,219 | 2,318 |
| Matrix Multiply(20) | 526,130 | 526,130 | 378,164 | 378,164 | - |

TABLE II: Bliss Simumation Results

Because in the case of the compressed format the nodes spend less time in sending the data, the chance of a collission (and hence a radio backoff) is reduced. The nodes also spend less time in assessing the clear channel (CCA). The mean energy requirement for sending a packet compressed compared to sending it raw is reduced by a factor of 1.84.

## V. Conclusion and Perspectives

With this article, we presented an extension to the initial IDEA1 implementation. We included support for the MSP430 family microprocessors which we modelled in TLM. IDEA1 handles the TLM part concerning the communication of the microcontroller with the radio transceiver. Bliss, our custom built instruction set simulator, outputs the number of clock cycles for each function that makes up an application. Hence, Bliss handles the TLM part concerning the actual instructions executed by the microcontroller.

The current Bliss implementation takes instruction lengths from the datasheet directly as specified by the manufacturer. However, it does not take into account the pipelines present in the microcontrollers. One possible improvement would be to refine its output. Also, the current TLM scheme only supports MSP430. In future versions, we will offer support for both AVR and PIC.

## References


[1] H.-M. Tsai, W. Viriyasitavat, O. K. Tonguz, C. U. Saraydar, T. Talty, and A. MacDonald, "Feasibility of In-car Wireless Sensor Networks: A Statistical Evaluation," in *SECON*, 2007, pp. 101–111.

[2] A. Dhamdhere, V. Sivaraman, V. Mathur, and S. Xiao, "Algorithms for Transmission Power Control in Biomedical Wireless Sensor Networks," in *APSCC '08: Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 1114–1119.

[3] J. Allred, A. B. Hasan, S. Panichsakul, W. Pisano, P. Gray, J. Huang, R. Han, D. Lawrence, and K. Mohseni, "SensorFlock: an airborne wireless sensor network of micro- air vehicles," in *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2007, pp. 117–129.

[4] M. A. Hussain, P. Khan, and K. K. Sup, "Wsn research activities for military application," in *Proceedings of the 11th international conference on Advanced Communication Technology - Volume 1*, ser. ICACT'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 271–274. [Online]. Available: http://dl.acm.org/citation.cfm?id=1701955.1702006

[5] R. Wang, L. Zhang, and L. Cui, "Intelligent wakening scheme for wireless sensor networks surveillance," in *The 1st International Workshop on Cyber-Physical Networking Systems, in conjunction with INFOCOM 2011*, ser. CPNS '11, 2011.

[6] *Microchip MRF24J40 Data Sheet*, Microchip Technology Inc. [Online]. Available: ww1.microchip.com/downloads/en/DeviceDoc/DS-39776b.pdf

[7] A. Salhieh, J. Weinmann, M. Kochhal, and L. Schwiebert, "Power Efficient Topologies for Wireless Sensor Networks, International Conference on Parallel Processing," in *International Conference on Parallel Processing*, 2001.

[8] W. Du, D. Navarro, F. Mieyeville, and I. O'Connor, "Idea1: A validated system c-based simulator for wireless sensor networks." in *MASS*. IEEE, 2011, pp. 825–830. [Online]. Available: http://dblp.uni-trier.de/db/conf/mass/mass2011.html#DuNMO11

[9] V. Shnayder, M. Hempstead, B.-r. Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 188–200. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031518

[10] M. Varshney, D. Xu, M. Srivastava, and R. Bagrodia, "squalnet: An accurate and scalable evaluation framework for sensor networks," *Information Processing in Sensor Networks*, 2007.

[11] J. L. Font, P. Iñigo, M. Domínguez, J. L. Sevillano, and C. Amaya, "Analysis of source code metrics from ns-2 and ns-3 network simulators," *Simulation Modelling Practice and Theory*, vol. 19, no. 5, pp. 1330 – 1346, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X11000190

[12] C. Mallanda, A. Suri, V. Kunchakarra, S. S. Iyengar, R. Kannan, A. Durresi, and S. Sastry. Simulating wireless sensor networks with OMNeT++. [Online]. Available: http://csc.lsu.edu/sensor_web/final%20papers/OMNet++-IEEE-Computers.pdf

[13] *Avrora: scalable sensor network simulation with precise timing*, Apr. 2005. [Online]. Available: http://dx.doi.org/10.1109/IPSN.2005.1440978

[14] F. Fummi, D. Quaglia, and F. Stefanni, "A systemc-based framework for modeling and simulation of networked embedded systems." in *Forum on Specification, Verification and Design Languages (FDL)*, 2008, pp. 49–54.

[15] W. Du, F. Mieyeville, and D. Navarro, "IDEA1: A SystemC-based system-level simulator for wireless sensor networks," in *IEEE International Conference WCNIS2010*, W. Chen and S. Li, Eds., vol. 2. IEEE, June 2010, pp. 618–623.

[16] W. Du, D. Navarro, F. Mieyeville, and I. O'Connor, "IDEA1: a validated system-level simulator for wireless sensor networks," in *The 4th International Workshop on Wireless Sensor, Actuator and Robot Networks (WiSARN-Fall 2011)*, Valencia, Spain, Oct. 2011.

[17] J. Eriksson, A. Dunkels, N. Finne, F. sterlind, and T. Voigt, "Mspsim – an extensible simulator for msp430-equipped sensor boards," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, Delft, The Netherlands, Jan. 2007. [Online]. Available: http://www.sics.se/ adam/eriksson07mspsim.pdf

[18] C.-L. W. benchmarks. Available from http://www.c lab.de/home/en/download.html.

[19] S. Mohan, F. Mueller, D. Whalley, and C. Healy, "Timing analysis for sensor network nodes of the atmega processor family," in *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 405–414. [Online]. Available: http://dl.acm.org/citation.cfm?id=1048932.1049929