Design Framework for Heterogeneous Hardware and Software in Wireless Sensor Networks

David Navarro, Fabien Mieyeville, Wan Du, Mihai Galos, Nanhao Zhu, Ian O'Connor

Design Framework for Heterogeneous Hardware and Software in Wireless Sensor Networks

David Navarro, Fabien Mieyeville, Wan Du, Mihai Galos, Nanhao Zhu, Ian O'Connor
Université de Lyon, Institut des Nanotechnologies de Lyon (INL), UMR5270 - CNRS, Ecole Centrale de Lyon, Ecully, F-69134, France
david.navarro@ec-lyon.fr, fabien.mieyeville@ec-lyon.fr, wan.du@ec-lyon.fr, mihai.galos@ec-lyon.fr, nanhao.zhu@ec-lyon.fr, ian.oconnor@ec-lyon.fr

***Abstract*- Wireless Sensor Networks are composed of many autonomous resource-constrained sensor nodes. Constrains are low energy, memory and processing speed. Nowadays, several limitations exist for heterogeneous Wireless Sensor Networks: various hardware and software are hardly supported at design and simulation levels. Meanwhile, to optimize a self-organized network, it is essential to be able to update it with new nodes, to ensure interoperability, and to be able to exchange not only data but functionalities between nodes. Moreover, it is difficult to make design space exploration, as accurate hardware-level models and network-level simulations have very different (opposite) levels. We propose a simulator –based on SystemC language- that allows such design space explorations. It is composed of a library of hardware and software blocks. More and more sophisticated software support is implemented in our simulator. As trend is to deploy heterogeneous nodes, various software levels have to be considered. Our simulator is also thought to support many levels: from machine code to high level languages.**

Keywords-wireless sensor network; WSN; simulation; model; systemC

I. INTRODUCTION

Many applications use communicating and distributed sensory systems, such as for example environmental data collection, security monitoring, logistics or health [1]. These radiofrequency-based communicating systems are called Wireless Sensor Networks (WSN). Wireless Sensor Networks are large-scale networks of resource-constrained sensor nodes (electronic systems). Limited resources are of different kinds: energy, memory, processing and data-rate. Indeed, these autonomous systems have to ensure a so long autonomy that processing architecture and communications data-rate have to be very low. Sensor nodes cooperatively monitor and transmit data, such as temperature, vibration, pressure etc. They are typically composed of one or more sensors, a 8-bit or 16-bit microcontroller, a few Kbytes non-volatile memory, a low data-rate (often 250 Kbits/s) radiofrequency transceiver and a light battery. Fig. 1 shows a typical sensor node architecture.

A lot of hardware platforms exist (for example Crossbow, Ember, Meshnetics, Zolertia) and several devices are widely used: ATMEL, Texas Instruments or Microchip for microcontrollers, Texas Instruments, ATMEL, Freescale, or ST-Microelectronics for radiofrequency transceivers. Linux systems composed of 32-bit RISC processors exist – like the well known Crossbow's Stargate platform - but prohibitive energy consumption relegates these products to the border of the Wireless Sensor Networks field.

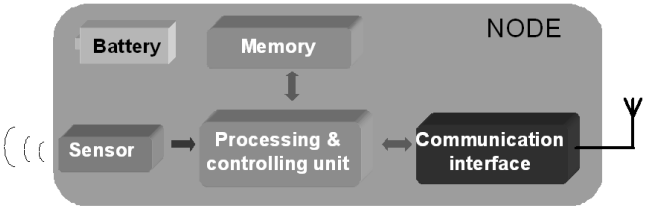

Figure 1. Wireless sensor node hardware architecture

We do not consider such systems, and we do focus on long autonomy systems. Low power constraint and large number of existing devices oblige us to think about dedicated (heterogeneous support) accurate simulator and programming tools.

Wireless Sensor Networks interconnect (topologies and network hierarchy) is inspired from wireless telecommunication and computer networks. We only focus on the often used IEEE 802.15.4 standard [2] that is widespread in Wireless Sensor Network commercial or custom platforms. Although complex topologies exist, such networks are dedicated to low power and low data rate applications, mainly for physical and environmental remote measurements. Most used topologies are also star or mesh networks [3].

Wireless Sensor Networks design is a difficult task, because the system designer has to develop a network with low level (at sensor node) hardware and software constraints. Computer-Aided-Design (CAD) tools would also be required to make system-level simulations, taking low-level parameters into account.

As presented in [4], many simulators have been developed over the last few years [5-9], but most of them are restricted to specific hardware or precisely focus on either network level or node level. They can be broadly divided into two categories: network simulators enhanced with node models (e.g., NS-2 [7] and OMNeT++ [8]), and node simulators enhanced with network models (e.g., Avrora [9], or SCNSL [10]). In the first category, simulators are not sensor platform specific and they are too high level for

 ISBN: 978-1-61208-166-3

hardware considerations. Precision problems are recurrent. In the second category, simulators are better suited for electronic system designers, requiring precise low level models for top-down (network to node) approach, but they suffer from too low-level aspects. Scalability and simulation time are problematic. Instruction Set Simulators have the same drawbacks. We propose a fast simulator of the last category.

Heterogeneous support means at first to be able to program several devices with a single compiler (C-level programming is nowadays the most used for Wireless Sensor Networks [11]), and to allow not only data but functionality exchange between nodes. As wireless sensor nodes are not often accessible, they have to be able to compile by themselves. Dynamic reconfiguration is also required. Many solutions exist nowadays; they require Operating Systems or Virtual Machines. The most known Operating Systems are TinyOS [11], Contiki [12], SOS [13], but they don't support heterogeneous firmware update. Indeed, they all use monolithic binary updates, which are architecture-specific. The most popular Virtual Machines for Wireless Sensor Networks are Maté [14], Darjeeling [15], VMStar [16], and ContikiVM [17]. They interpret a bytecode that is higher level than machine code. The drawback of these solutions is that big energy overhead is required to interpret and execute each bytecode instructions. It is well known that most of the power consumption in these systems is due to radiofrequency devices. So, a dedicated solution would be to consider in-situ compilation that minimizes code size transmission, in order to match the Sensor Networks constraints.

In this paper, we present a hIerarchical DEsign plAtform for sensOr Networks Exploration called IDEA1. It is characterized with SystemC simulation kernel, and a graphical interface to make it easier to use. Section II details its architecture, particularly in terms of hardware, software and network models. Section III details simulator user interface and results.

II. MODELS DETAILS

Our simulator is inspired from the SCNSL library [10], a networked embedded systems simulator. It is written in SystemC and C++. SystemC is widely used in electronics community; it is part of the classical design flow. SystemC based on an event-driven simulator kernel, and this language permits to model hardware and software at same time, in the so-called co-simulation. As Fig. 2 shows, three main models exist: nodes (in SystemC), node-proxy (in SystemC) and network (in C++). C++ is used to model the network in terms of connectivity and communication characteristics; and proxies that make input/output interfaces between nodes and network. Simulation occurs in two steps: a gcc compilation creates the network, that is also static, and then the SystemC kernel runs the simulated time. It would be possible to simulate a dynamic (moving) network, but simulation time would be largely affected.

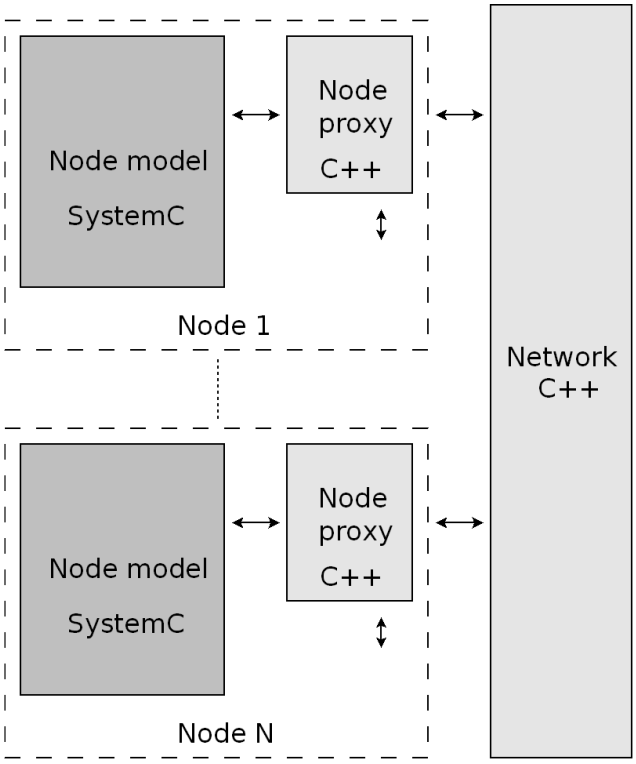

Figure 2. Wireless sensor network model

Node model is detailed in Fig. 3. It is composed of hardware and software parts. This physical layer is also detailed below for hardware and software parts.

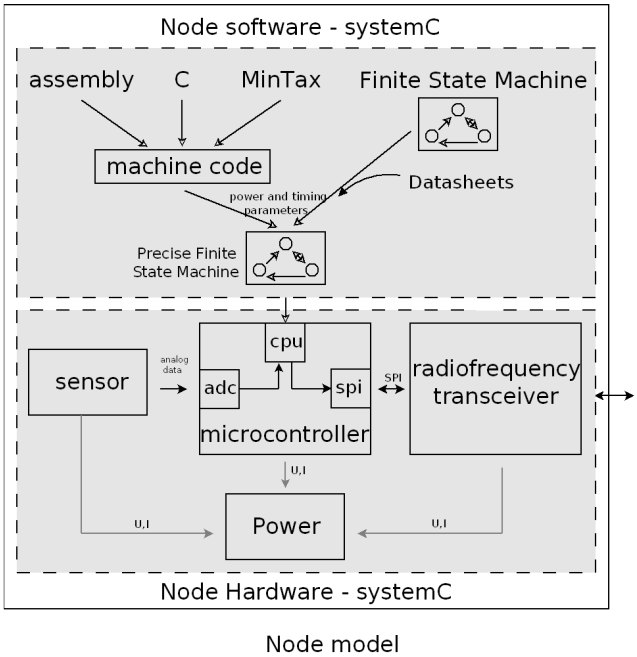

Figure 3. Wireless sensor node architecture model

A. Hardware model

Hardware part embeds classical wireless sensor nodes devices: a sensor, a microcontroller, a radiofrequency transceiver and a battery. Hardware devices models are detailed with electrical and timing parameters.

As battery discharge is based on chemical non linear reactions, we prefer - for the moment - to define a simpler power module that monitors instantaneous and average power and energy. A generic battery life time can be computed.

Sensor is modeled with its transfer function that gives sensor output voltage versus physical input. This transfer function can be composed of integrations during time, or error deviations.

Microcontroller is the processing and controlling unit. Depending on application, an external flash memory is sometimes used, its usage really impacts power consumption. Sensor data is captured by an Analog to Digital Converter (ADC) that is typically a 10-bit successive approximation converter. It gives the best balance between speed and accuracy. Concerning communication interface that enables dataflow output, it is done by a classical serial communication, hardware supported by means of a serial peripheral interface (SPI) block. The processing part of the microcontroller is a simple 8-bit or 16-bit datapath organized around a light arithmetic and logic unit (ALU). In power-aware Wireless Sensor Nodes, processing power of that element is at maximum a few tens of MIPS, coupled with specific low power architectures.

Next, data are output from the node by a radiofrequency transceiver. This complex device allows generating a high frequency carrier in order to propagate data over the air. The carrier depends on country norms, the most typical free frequencies that are used are 433MHz, 868MHz, 916MHz, 2.4GHz. Due to market explosion concerning embedded products, and to small size of antenna, 2.4GHz radiofrequency transceivers are nowadays mostly used in embedded systems. Data have to be organized in packets. These packets allow to route data towards the right node, to ensure data integrity, while respecting a given communication protocol. The radiofrequency transceiver model contains different working states (receive, transmit, idle, sleep), and several operating modes.

At the whole, several hardware devices have been modeled (Table I). These hardware devices are interchangeable in order to model different existing or novel hardware platforms. Simulator enables user to test an application on several hardware devices to find the solution that best fits requirements, such as data-rate and energy. ATMEL ATMega128 and Microchip PIC16LF88 are well known low power microcontrollers. Texas Instruments CC2420 and Microchip 24J40 are the most used transceivers, as their carrier is 2.4GHz, and they support the IEEE 802.15.4 standard and the ZigBee stack. Sensors are often used ones. The first is a light sensor of the Crossbow Mica platform, the other one is a widespread temperature sensor.

TABLE I. LIST OF MODELEDHARDWARE DEVICES

Microcontrollers		ATMEL ATMega128 Microchip 16LF88
Radiofrequency transceivers		Texas Instruments CC2420 Texas Instruments CC1000 Microchip MRF24J40 Nordic nRF24L01
Sensors	**Temperature**	National Semiconductor LM35DZ
	Light	Clairex CL9P4L

B. Software model

Software has to be considered on two different aspects:

- Portability: executable (machine) code is specific to each precise hardware architecture.
- Level: many different languages exist, thus enabling different levels of coding, from assembly to high level languages.

For these two reasons, we have decided to support heterogeneous multiple software levels. As Fig. 3 shows, the software input can be at state machine level or at programming language level.

1) State Machine level

The software running on microcontroller is divided into different tasks (states), such as data processing, analog to digital conversion (ADC) and communication (SPI). The execution time of each task is calculated according to its datasheet typical values. For example, the time taken by PIC16LF88 to configure and launch ADC is taken into account (hardware delays such as the 11.974µs minimum required acquisition time [18]).

When data are transferred from microcontroller to radiofrequency transceiver, a trigger command enables transmission. At the right time (depending on network policy), the radiofrequency transceiver will transmit data to another node. Microcontroller that drives radiofrequency transceiver has different working states, detailed in Fig. 4 for a simple example.

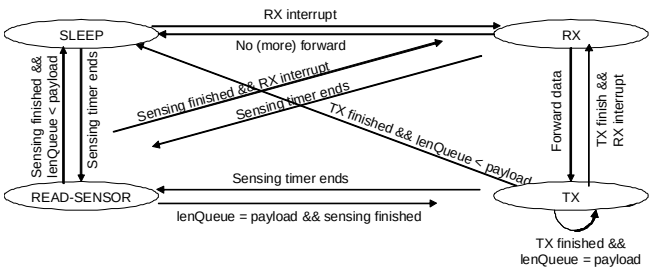

Figure 4. Simple capture and send program in microcontroller

This finite state machine has been implemented with realistic (from datasheet and measurement validated) parameters in terms of delays and power consumption. Actual hardware library with finite state machine models is detailed in table I. In our model, the microcontroller can configure some parameters of physical (PHY) and MAC layers in the radiofrequency transceiver registers (IEEE 802.15.4 - compliant).

Both IEEE 802.15.4 non-beacon and beacon modes are supported in our simulator. Non-beacon mode is based on a channel free access and packet-based philosophy. When a node wants to send data, it senses the channel, then sends them if the channel is free. If the channel is busy, it waits a random time (called back-off time) and then checks for free channel again. This method is called CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance). Beacon mode is a synchronized mode: the network coordinator (network head) sends synchronization packets to inform nodes when they can communicate. In this case, the mode is also channel-based, inspired from the well known TDMA (Time Division Multiple Access) method.

Time is organized according to a superframe that is defined by the network coordinator. Two beacon-mode algorithms exist: slotted CSMA-CA, and GTS communication non-predictive GTS and predictive GTS [2]. Slotted CSMA-CA is a CSMA-CA based communication, within a given slot time. In GTS algorithm, nodes that want to communicate send a GTS request to the coordinator during a first time slot (the Contention Access Period). Then, nodes are allowed to communicate during a following time slot (the Contention Free Period).

A power module has been implemented. It computes and monitors electrical power and energy consumed by sensor, microcontroller and radiofrequency transceiver. Different energy-saving (sleep) modes, data flow and global behavior can also be co-designed according to power constraints.

2) Language level

If user selects language input instead of finite state machine, several solutions are available. This step is currently being implemented in our framework. Input language can be in assembly, in C language, or in a high level language we have developed (MinTax). Software support of ATMEL ATMega128 is currently realized, we plan to support Texas Instruments MSP430 and Microchip PIC16LF88 later. The commercial platforms we are using for testcase and measurements are ATMEL AVRraven and Zolertia Z1, comprising for Texas Instruments MSP430 and ATMEL ATMega128 microcontrollers.

In order to meet the energy constraints in a Wireless Sensor Network, the processing and controlling unit is nearly all the time a microcontroller. Such devices often consume less than 5 mW. Meanwhile, they often have 8-bit or 16-bit datapaths that process less than 20 MIPS, and they embed less than 128 Kbytes of program memory (FLASH ROM), and less than 16 Kbytes of data memory (RAM). Such light architectures require specific lightweight solutions.

If assembly language is used, the code is analyzed in order to estimate process timing of microcontroller, and its associated power consumption.

If C language is selected, compilers are used to generate low level assembly and machine code. IAR Systems compiler is used for Texas Instruments MSP430, AVR-gcc is used for ATMEL ATMega128. C language compilation generates assembly code that is treated in the same way as direct assembly input. More precisely, lss output files from compilers are treated.

As a test-case, we have demonstrated that our simulator is moreover able to consider new languages that could better suit Wireless Sensor Network specificities. To prove this, the simulator supports a high level dedicated language we have developed, with an energy-aware syntax that allows to compactly write microcontroller tasks. That minimal syntax (MinTax, detailed in [19]), based on C language, has the advantage to require fewer characters, and also shorter radiofrequency communications for program exchanges. A MinTax and C comparison example is given in table II. We can clearly see that MinTax reduces the number of characters to code the example (pin toggle program). Data to send are also reduced by a 3 factor.

TABLE II. MINTAX – C COMPARIZON.

MinTax	**C**

f{
WT
$b%2
#
};
	void f()
	{
	while(true)
	{
	PORTB ^= (1<<2);
	}
	}

11 bytes	37 bytes

As in C language, this high level syntax permits designer to have hardware abstraction, and also to consider a single language on heterogeneous platforms. A functionality exchange in a heterogeneous network has been implemented as testcase. ATMEL AVRraven and Zolertia Z1 platforms were used, and functionality written in MinTax has been send from ATMEL ATMega to T.I. MSP430 through ATMEL AT86RF230 and T.I. CC2420 transceivers (IEEE 802.15.4 standard).

The compiler that is related to this language is based on classical compiler structure, as shown in Fig. 5. As it is embedded (compilation is done in-situ with low processing unit), all of its parts have been optimized for compactness. Two stages exist: an analysis stage then a synthesis stage. The analysis stage reads the high-level language, splits it into tokens and orders them. It recognizes for example variables names and functions calls. The synthesis stage generates the executable code (binary machine code). More information about classical compiler structure is detailed in [20].

 ISBN: 978-1-61208-166-3

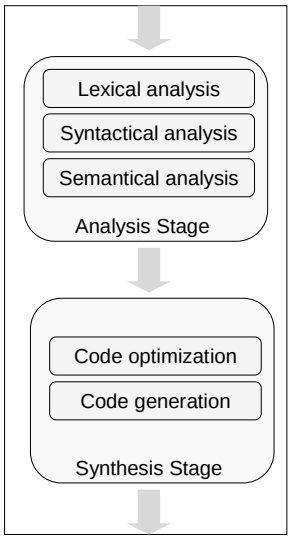

Figure 5. Classical compiler structure

Several variants of the compiler exist: it has been cross - developed on x86 personal computer and on several microcontroller architectures. The PC variant allows easier debugging because of its human-machine interface. It permits to deploy the code on the nodes by serial programming machine code (output files such as .hex). The microcontroller variant has been developed on several hardware architectures to prove the heterogeneity support. Software support, from low level (assembly) to novel high level specific languages (MinTax) has also been proved.

III. SIMULATOR INTERFACE AND RESULTS

User interface is shown in Fig. 6. It is composed of different sub-windows. The information appears graphically in the right window, to clearly display the network topology. Each node and coordinator is characterized by a spatial position. Lines between nodes represent possible communications routes according to position, transmit power and receive sensitivity. When parameters are changed, the graphical viewer refreshes the possible communications (lines). For this early version, free space communications are considered. Focus is set on communication capabilities and data rate, not on mechanical or electromagnetic environments. Hardware parameters of microcontrollers and radiofrequency devices are set. At higher level, many parameters of the IEEE 802.15.4 can be set. Sensors sampling rate and packets payload can also be changed.

By clicking on the launch button in the graphical interface, a SystemC simulation is launched in background. The simulation log is displayed in the bottom window of graphical interface, and a timing trace (VCD) viewer is opened. Output log files are also generated. From these results, we can explore design space in order to find the best-suited design solution.

As a test example, we simulated an 8 nodes network. We chose Microchip PIC16LF88 and MRF24J40 as target test hardware. As IEEE 802.15.4 data-rate is low (250 Kb/s), a systematic trade-off between payload (number of sent data bytes per packet), sampling rate (of ADC) and packet delivery rate has to be explored.

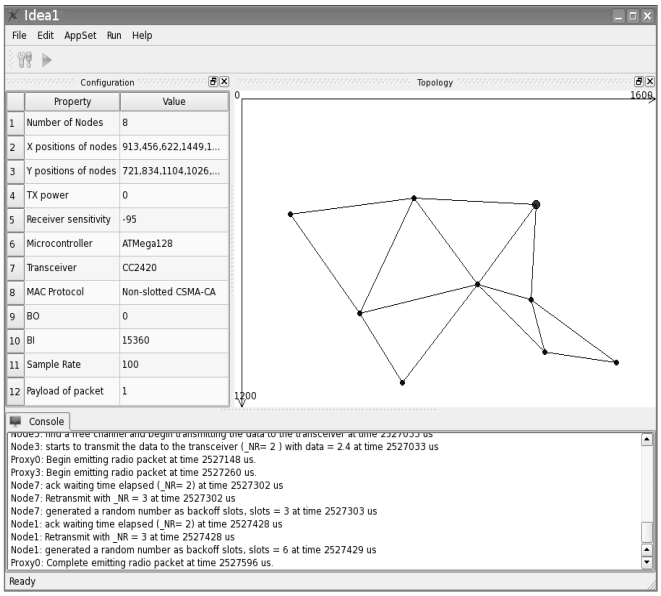

Figure 6. Simulator graphical interface

Simulator can output a transient VCD trace and text log. Log permits to process data in order to evaluate Packet Delivery Rate (PDR), Packet Latency (PL), Energy Per Packet (EPP), and average power consumption. Packet delivery rate is the ratio of number of successful packets over the total number of sent packets is measured. Packet latency is the time needed by a packet to go from one node to another one. Energy per packet is related to the product of sent packets by the sample period. Average power consumption is the one consumed by electronic devices. Global power consumption of hardware devices level is available; this result was shown in [21].

Moreover, it is now possible to detail the power consumption of each hardware part in microcontroller. Fig. 7 shows decomposition of power consumption in microcontroller according to analog to digital converter (ADC), serial communication (SPI), processing of CPU, and sleep state. This analysis is done for various frequencies of data sampling. Power consumption is high when sample rate is high, because nodes are always busy in these two cases. Moreover, as almost maximal usage is reached from 100Hz, power consumption difference is small for 100 and 1000 Hz. Most power consuming states are CPU processing and inter-chip communications. It clearly shows the impact of hardware support of radiofrequency device on power consumption of microcontroller: depending if IEEE 802.15.4 is hardware supported in radiofrequency device or not, power consumption distribution changes. Indeed, the bigger part of physical and MAC layers to be managed by the microcontroller, the more power consumption will be observed for this device. At the same time, radiofrequency

 ISBN: 978-1-61208-166-3

transceiver will have different idle and sleep timings according to this eventual IEEE 802.15.4 hardware support, so different power consumption is impacted too.

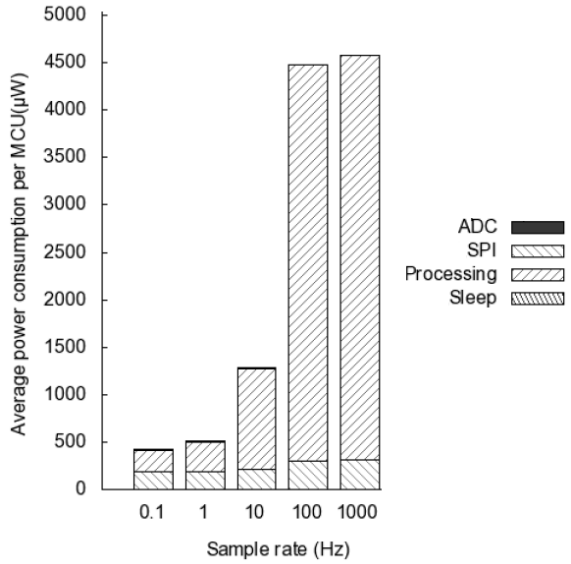

Figure 7. Microcontroller unit (MCU) detailed power consumption

IV. CONCLUSION

A Wireless Sensor Network design framework, based on SystemC, has been presented. It permits design space exploration, considering hardware and software details. Hardware is modeled as a finite state machine, characterized by timings and power consumption of each state. Software can be modeled as finite state machine in the same way. Current work is done in order to take real program inputs at different levels: assembly, C language using existing compilers, or high level minimalist language (MinTax) associated to its in-situ compiler we already have developed. This language permits to exchange functionalities between non-compatible (heterogeneous) processing units, with a small energy cost. A graphical user interface permits to easily simulate and compare several IEEE 802.15.4 configurations and programs on many interchangeable (and parameterized) hardware devices.

REFERENCES

[1] M. Horton and J. Suh, “A vision for wireless sensor networks”, IEEE Microwave Symposium Digest, USA, June 2005, pp. 361-364.

[2] IEEE 802.15 WPAN Task Grup 4, “IEEE 802.15 part 15.4-2006 wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (WPANs), http://www.ieee802.org/15/pub/TG4.html, last access date: July 2011.

[3] A. Salhieh, J. Weinmann, M. Kochhal, and L. Schwiebert, “Power efficient topologies for wireless sensor networks”, International Conference on Parallel Processing, Spain, Sept. 2001, pp. 256-163.

[4] W. Du, D. Navarro, F. Mieyeville, and F. Gaffiot, “Towards a taxonomy of simulation tools for wireless sensor networks” International Conference on Simulation Tools and Techniques, Spain, March 2010, pp. 1-7.

[5] P. Levis, N. Lee, M. Welsh, and D. Culler, “Tossim: accurate and scalable simulation of entire tinyos applications”, International Conference on Embedded Networked Sensor Systems, USA, Nov. 2003, pp. 126-137.

[6] M. Varshney, D. Xu, M. Srivastava, and R. Bagrodia, “sQualNet: an accurate and scalable evaluation framework for sensor networks”, International Symposium on Information Processing in Sensor Networks, USA, April 2007, pp. 196-205.

[7] S. McCanne and S. Floyd, “Network simulator NS-2”, http://www.isi.edu/nsnam/ns, last access date: July 2011.

[8] A. Varga, “The OMNeT++ discrete event simulation system”, European Simulation and Modeling Conferences, Czech Republic, June 2001, pp. 319-324.

[9] B. Titzer, D. Lee, and J. Palsberg, “Avrora: scalable sensor network simulation with precise timing”, Information Processing in Sensor Networks, USA, April 2005, pp. 477-482.

[10] F. Fummi, D. Quaglia, and F. Stefanni, “A systemC-based framework for modeling and simulation of networked embedded systems”, Forum on Specification and Design Languages, Germany, Sept. 2008, pp. 49-54.

[11] L. Mottola and G.P. Picco, “Programming wireless sensor networks: fundamental concepts and state of the art”, ACM Computing Surveys. Volume 43, Issue 4, Dec. 2010, pp. 1-19.

[12] A. Dunkels, B. Gronvall, and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors”, IEEE International Conference on Local Computer Networks., USA, Nov. 2004, pp. 455-462.

[13] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, “A dynamic operating system for sensor nodes”, International Conference on Mobile systems Applications and Services, USA, June 2005, pp. 163-176.

[14] P. Levis and D. Culler, “Mate: a tiny virtual machine for sensor networks”, International Conference on Architectural Support for Programming Languages and Operating Systems, USA, Oct. 2002, pp. 85-95.

[15] N. Brouwers, K. Langendoen, and P. Corke, “Darjeeling, a feature-rich VM for the resource poor”, ACM Conference on Embedded Networked Sensor Systems, USA, Nov.2009, pp. 169-182.

[16] J. Koshy and R. Pandey, “VMSTAR: synthesizing scalable runtime environments for sensor networks,” International Conference on Embedded Networked Sensor Systems, USA, Nov. 2005, pp. 243-254.

[17] A. Dunkels, “Programming Memory-Constrained Embedded Systems”, PhD Thesis, Swedish Institute of Computer Science, 2007.

[18] Microchip Inc., “PIC16F87/88 Enhanced Flash Microcontrollers with nanoWatt Technology Datasheet”, http://www.microchip.com, last access date: July 2011.

[19] M. Galos, F. Mieyeville and D. Navarro, “Dynamic reconfiguration in wireless sensor networks”, International Conference on Electronics Circuits and Systems, Greece, Dec. 2010, pp. 918-921.

[20] R. Mak, “Writing compilers and interpreters”, Ed Wiley Computer Publishing, ISBN 471-11353-0.

[21] W. Du, F. Mieyeville and D. Navarro, “Modeling energy consumption of wireless sensor networks by systemC”, International Conference on Systems and Networks Communications, France, Aug. 2010, pp. 94-98.

 ISBN: 978-1-61208-166-3