A Methodology for Power Consumption Evaluation of Wireless Sensor Networks

Andrey Somov, Ivan Minakov, Alena Simalatsar, Giorgio Fontana and Roberto Passerone
Dipartimento di Ingegneria e Scienza dell'Informazione
University of Trento, Italy
Emails: {somov, minakov, simalats}@disi.unitn.it, {giorgio.fontana, roberto.passerone}@unitn.it

Abstract

Energy consumption is one of the most constraining requirements for the design and implementation of wireless sensor networks. Simulation tools allow one to significantly decrease the effort and time spent to choose the right solution. Existing simulators provide varying degrees of analysis for communication, application and energy domains. However, they do not provide enough flexibility to estimate the consumed power for a wide range of wireless sensor network (WSN) hardware (HW) platforms. In this paper we present a flexible and extensible simulation framework to estimate power consumption of sensor network applications for arbitrary HW platforms. This framework allows designers of sensor networks to estimate power consumption of the explored HW platform which permits the selection of an optimal HW solution and software (SW) implementation for the desired projects.

1 Introduction

The design of WSN systems faces challenges specific to their domain of application. For instance, sensor nodes are commonly resource-constrained, battery-powered embedded devices. Replacing the batteries is, however, at best inconvenient, and sometimes downright impossible after the network deployment. Therefore, the efficiency of using the battery power determines the lifetime of the nodes and of the entire network. Due to these limitations, energy consumption is one of the most constraining requirements for design and implementation of wireless sensor networks. Thus, power analysis is a critical step in the sensor network development process. The choice of the application algorithms, operating systems, scheduling policies and program style may have a considerable impact on energy consumption. Power analysis provides a study of critical operations on limited power sources. It gives the designers a clear idea of how to adapt their applications in order to prolong the system lifetime. Reliable and accurate energy evaluation can be achieved by capturing all relevant low-level details and operating states of the studied HW platform. Rough approximation and high abstraction HW modeling fail due to the lack of such information. However, HW modeling should be "lightweight" enough and should act on the proper abstraction level in order to provide scalable simulation for numerous sensor nodes.

Several modeling approaches and simulation techniques have been introduced to perform power consumption evaluation from the network point of view [5, 6, 20, 25]. In these approached the system is modeled as a set of communicating concurrent processes, where each process represents the activity of a single node. Existing simulators [5, 6, 8, 11, 12, 13, 18, 23] provide various degrees of analysis for communication, application and energy domains. Many tools are focused either primarily on the network or software simulation, without proper care of detailed HW processing. Others exploit a cycle-accurate simulation strategy to examine operations with fine grained details [30]. Such tools normally include support only for a single HW architecture, without the ability to extend them. Neither provides enough flexibility to value energy for a wide range of HW WSN platforms.

In this paper we present a methodology for power consumption evaluation of the individual nodes of a wireless sensor network. Our methodology supports the Platform Based Design (PBD) paradigm [16, 28], providing power analysis for various sensor platforms by defining separate abstraction layers for application, services, hardware and power supply modules. It is implemented as a SystemC-based framework that combines the event-driven simulation engine and HW model composer, and allows a user to describe the application using a set of service calls and user functions described in C/C++ language. The presented methodology allows one to estimate the lifetime of a node composed of three main parts: transmission module, computational module and a battery.

2 Related Work

In this section we are going to present several system-level design (SLD) methodologies used for the design space exploration of Wireless Sensor Networks (WSN). We will also introduce several frameworks for networks

simulations. In addition we will talk about the software battery models and design tools used to create them.

2.1 SLD Methodologies

PBD is a methodology that combines the specification, validation and synthesis steps of the design flow, while maintaining a clear separation between the corresponding models [16, 28]. By doing so, the designer can operate separately on the distinctive steps and maintain a global view of the impact of his/her design decisions on the final implementation. The methodology includes hardware and embedded software design, where the design of the system starts at a high level of abstraction (initial design description) and proceeds to a detailed implementation by mapping the executable functional model onto progressively more detailed architectures under a set of constraints. The framework presented in this paper follows the design principles of PBD methodology.

There exist several SLD methodologies applying PBD principles developed in the area of WSN design space exploration. Bonivento et al. have presented a new methodology for the design of WSN [14], which is well correlated with the nature of our research work. They introduced a framework called Rialto that initially included two basic platforms, the application interface called Sensor Network Service Platform (SNSP) and the hardware platform layer called Sensor Network Implementation Platform (SNIP). Lately this framework was extended by a third intermediate layer called Sensor Network Adhoc Protocol Platform (SNAPP) that defines a library of communication protocols and the interfaces that these protocols offer to the SNSP. This framework allows the application description independently from the network architecture. The sensor network service platform is used to describe an application in a Rialto Model in terms of logical component queries and commands. This model is then translated into RialtoNet format that allows exploration of the possible sequence of queries and commands from the application. Later the functional description is mapped into an architecture platform instance. This work is focused mainly on high level operating system services, while we will focus on single node platform power consumption estimation.

The COmmunication Synthesis Infrastructure (COSY) [25] developed by Alessandro Pinto et al. is a framework for the design exploration and synthesis of interconnection networks, where "interconnection networks" refers to networks on-chip (NoC) as well as distributed embedded systems. They introduce a general methodology for the design space exploration of networks that allows studying of the optimization algorithms, communication protocols, partial designs, and models for interconnection design in terms of network performance and cost. We can apply this methodology in the future while extending our methodology for being used for a whole WSN power consumption estimation.

2.2 Design Tools

NS-2 simulator: NS-2 [5], perhaps, is the most popular general purpose network simulator. NS-2 supports simulation for widely used IP network protocols. These include TCP, routing and multicasting protocols for conventional wired and wireless networks. NS-2 has a highly extensible object-oriented architecture with discrete-event engine. Its object-oriented model allows extension of simulation functionality by adding customers components and libraries. The simulation in NS-2 environment is based on a combination of C++ and OTcl [7] languages where protocols are implemented in C++. OTcl is used as a scripting language to describe and control the simulation process. The complexity of NS-2 object-oriented model creates substantial dependencies and execution overheads. It makes impossible to scale simulation for a large number of network units, which is inherent to WSNs. While object-oriented model is advantageous in terms of extensibility, it is a restriction for scalability and performance. Besides, NS-2 does not provide representation for the HW network components.

OMNet++ simulator: Like NS-2, OMNet++ [6] provides deep analysis of network activities at the packet layer. Besides, OMNet++ provides a GUI front-end for simulation and debugging processes. It has a component-based architecture with a discrete-event simulation kernel. It exploits modules and channels to implement and connect simulation components, where components are connected in a hierarchical fashion via generic interfaces (gates). OMNet++ has extension for sensor network simulation, called SenSim [23]. It represents sensor node as modular hierarchical structure of simple OMNet++ components. This simulatorprovides more scalability and runs faster than NS-2. However, despite the apparent benefits of OMNet++ and SenSim, there is no precise and accurate HW model of sensor node. It, in turn, does not allow to study sensor networks from an energy perspective.

TOSSIM: TOSSIM simulation environment is included in the TinyOS [11] framework. TinyOS has gained general acceptance as a standard operating system for WSN applications. It has a component-oriented programming model, based on the nesC language [3]. A TinyOS program is presented as set of components, where each componnet is an independent computational entity. The TinyOS framework includes a simple FIFO task scheduler and hardware independent drivers for abstract HW components. The inter-component communications occur through command-event mechanism. By changing a small number of TinyOS components, TOSSIM simulates the behavior of the low-level hardware. It includes models for CPUs, analog-to-digital converters (ADCs), clocks, timers, flash memories and radio components. The network communication over the wireless channel is abstracted as a directed graph, where vertexes and edges represent nodes and links between them, respectively. TOSSIM simulation architecture provides high level of scalability and execution speed for the networks with large

2

number of sensor nodes. However, the abstract HW model of TOSSIM does not capture low-level details of timings and interrupts, which can be important for precise power analysis. In addition, simulation is supported only for the single HW platform (Micaz [2]). Obviously, it largely restricts the applying scope.

VIPTOS and VisualSense: VIPTOS (Visual Ptolemy and TinyOS) is a graphical development and simulation environment for TinyOSbased WSN applications [18]. VIPTOS bridges together the VisualSense [12] simulator and the TinyOS framework. VisualSense is a Ptolemy II [9] based graphical simulation environment designed for WSNs. It exploits the actor-oriented computational model of Ptolemy II, a general modeling framework for heterogeneous embedded systems. VisualSense defines actor-oriented models for sensor node subsystems and communication channels. However, VIPTOS does not provide accurate HW representation of sensor node. Substantially, it focuses more on algorithmic and application domains. Additionally, VIPTOS has been integrated only with the first version of TinyOS, which is not currently supported.

AVRORA: AVRORA [30], like TOSSIM, is one of the widely used WSN simulation tool. It exploits cycle accurate instruction-by-instruction manner to run code. AVRORA runs actual applications without the need to specially adapt it for simulation. AVRORA represents each HW component as corresponding object classes thus as classes of CPUs, Timers, flash memories, ADCs and off-chip components such as sensors. The HW model of a single sensor node is the combination of such objects in a hierarchical manner. The CPU object contains the simulation engine with the event queue for the entire node. This architecture allows node replication for network simulation, where each node is run as independent computational entity. However, AVRORA supports solely AVR MCU [1] cores and does not provide any extensions for others CPU architectures.

2.3 Battery Models

Several battery models have been designed recently. However, in most cases each type of the model makes an emphasis on specific battery purpose and has various levels of accuracy. Mathematical battery models [26, 27] apply empirical equations to address the battery charge/discharge behavior, efficiency, or simulate the electrochemical or thermal behavior of the battery. In fact, the mathematical models can not be adopted for circuit simulation since its accuracy lies in the range of 5-20% and do not provide one with the current-voltage characteristic.

Electrochemical models [29, 19] are complex and require much time for simulation since they typically present the battery at low level. However, the electrical models [22, 15, 21] are the most appropriate for circuit analysis, simulation and optimization. They are easy to handle, provide the electrical engineers with the current-voltage characteristics, and, finally, their accuracy lies within 5%.

The electrical battery model presented in [17] takes into consideration all dynamic characteristics of the battery. It consists of two parts: battery lifetime evaluation and voltage-current characteristics emulation. Battery lifetime evaluation part contains a resistor, which determines energy loss during long idle period, a capacity to characterize battery charge, and a current, which is applied to charge or discharge the battery. The voltage-current characteristics emulation part of the model contains open-circuit voltage, which is able to change according to the state of charge (SOC) of the battery, and an RC network. This model is both intuitive and capable to predict an accurate runtime and voltage-current parameters. We implemented this model in Spice Circuit Simulation tool [24] and used it to evaluate the life time of a single node.

3 Methodology

Our methodology is based on the PBD paradigm [16, 28]. In our methodology we distinguish four separate layers as depicted on Figure 1. We start our description from the *platform layer*, where we represent the node HW platform as the composition of several HW components. Here, the CPU plays the role of a central component that manages access to peripheral components, such as: transceiver, ADC, timers, a flash memory and an IRQ controller. Our approach to modeling HW platforms is based on separating the behavior of each component from its performance. The behavior is represented as a Finite State Machine (FSM), whose states correspond to the possible operating modes of the component (e.g., running, sleeping, hibernate, etc.). The behavior of a timer, for example, is essentially the same for different timer components. The states and transitions of the FSM are then given attributes that include information on the energy consumption and performance of the operation. These numbers depend on the particular component, and can be obtained by estimation or by characterization to improve accuracy. Each HW platform is therefore represented in our framework as the interconnection of a set of appropriate FSMs for its components. In practice, several standard platforms can be defined upfront in the form of libraries. The user is then able to choose the specific platform library by configuring the kind and number of components, and their associated performance parameters.

The upper layer, called *application layer*, is used to describe an application through a set of semantic primitives. This is an open layer, where the application description is left to the user. Each application is presented as a set of services calls and user functions described in C/C++ language. At this layer, users interact with the hardware by means of services provided by the *services layer* in a purely functional way. Our future work includes using the application layer to interface our framework to exist-
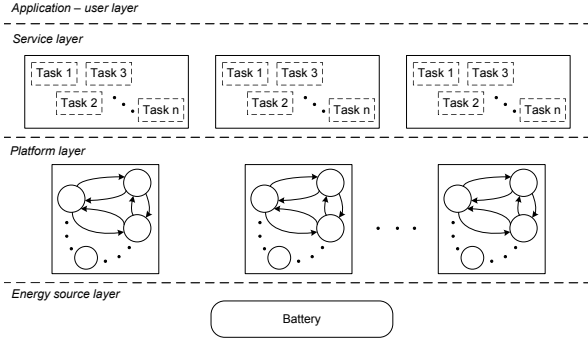
Figure 1. Methodology

ing software platforms, such as TinyOS 2.0. For that case, the idea is to interface the Hardware Presentation (HP) and the Hardware Adaptation (HA) layers of TinyOS directly with the service layer of our framework. This way, a native TinyOS application could be run on our simulator without change.

The *services layer* acts as an interface that defines a set of services available to the end user to specify an application. This layer abstracts details of the presented HW and provides a HW independent API for the application needs. The services themselves are commands to the HW modules to perform particular actions. Conceptually they are similar to HW independent drivers. By separating the platform services from the actual HW platform, we introduce additional separation between the system behavior and its cost/performance, thus providing flexibility to the design space exploration.

The total set of services provided by the *services layer* is composed of a number of subsets, where each subset is dedicated to a particular hardware component, and represents the maximum possible set of services that may be provided by all possible platform vendors. The choice of a particular platform defines the subset of the total set of services by making them available to the *application layer*. However, the number of provided services by a hardware component may vary from one supplier to another, which introduces constraints to the application interpretation. Thus, at the application layer validation of the tasks composition according to the chosen platform is taking place. For example, if the application requires a service that is not specified for a particular platform, the user will see a compilation error. Thus the user needs to be aware of the set of available services.

As an example here we present a set of basic services provided by a Timer component:

```
void RunTimer( void );
void RunTimerPeriodic( void );
void StopTimer( void );
void SetTimerUPeriod( u_time );
void SetTimerMPeriod( u_time );
void SetTimerParam( uint );
void EnableIrq( bool );
```

The execution of a service triggers the transitions of the FSMs representing the dedicated hardware component,

thus changing the power state of this component.

Besides the components services, there exist some global services to specify user defined functions as interrupt service routines (ISRs) for every component that may generate an interrupt (IRQ) event. Inside the ISR the user can define an arbitrary functionality (e.g., data processing, MAC layer etc.). Currently, we abstract this functionality which has to be annotated with the equivalent time needed to execute it on the target HW component. We assume that all the IRQ handlers are atomic and can not be preempted by other ISRs and that the power consumed to perform the context switch operation is negligibly small in comparison to the total power consumed by the interrupt handlers execution. In addition, the shift of the execution time frame of an IRQ handler within one operation cycle has no influence to the total power consumed during this cycle.

In addition to IRQ handlers the CPU may execute some background computation represented as a main CPU thread that have no real-time requirements (e.g., light up a LED). The execution of this main thread will be suspended by any IRQ coming from peripheral components until the IRQ event handler completes its execution. In our case study (below), the main thread includes only one command that sets the CPU to the *idle* mode.

Because we deal with power consumption, we introduce an *energy source layer*, which is currently represented with a battery. Each hardware component, being in a particular state, will consume a certain amount of power, which is accounted by the battery model. This layer may also include harvesting components or other sources of energy, which opens possibility to analyze power management techniques.

4 Case Study

In this section we describe the case study simulation of an abstract WSN application mapped to a TelosB platform. In particular, we have studied the influence of parameters of the real device and application behavior on the power consumption.

4.1 The Application

A typical application for WSN can be divided into several standard functional stages in which it first gathers some controlled quantity while sensing the environment, processes this data and forms the packets that are then sent to a base station. The case study presented in this paper follows this standard scenario. It performs a periodical sensing of the environmental temperature, with the period equal to one second. An on-board ADC is used to convert the analog value into its digital representation. When the converted data are ready, the application turns on the transceiver (*radio*), composes the network packet, waits till the *radio* is in the transmission mode and sends the data to an abstract base station. After the data transmission is over, the *radio* is switched to the Power Down state where it stays until it is reawaken in the following period

4

triggered by means of a timer. The interrupt handler of the timer used to trigger the ADC conversion is as follows:

```
void _interrupt Timer_ICR(void* ptr){
    SysTop* pSys = (SysTop*)ptr;
    pSys->Adc.RunConvertion();
}
```

Here, "pSys" represents a particular HW platform that includes the "ADC" component providing the "RunConvertion()" service.

The communication algorithm (a network protocol) is abstracted from the real MAC and routing protocol implementation by annotating the functionality of a network stack with time needed to perform abstract operations with data packets. Basically we focus on the HW state transitions during the execution process. The current consumption waveform we obtained is presented as a sequence of pulses. An example of a zoomed in single pulse is depicted in Figure 2.
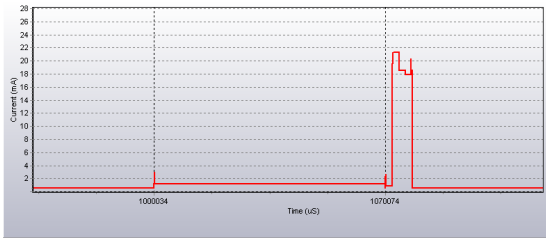

Figure 2. Example of a zoomed pulse

4.2 The Platform

As a target HW architecture we have chosen the TelosB platform [10], due to its popularity in WSN systems design. The TelosB mote is composed of the MSP430Fx1611 microcontroller, an external flash memory chip, the CC2420 transceiver, a set of the sensors and Leds. It is powered by two AA batteries, which should provide voltage in range from 2.1 to 3.6V.

The HW platforms in our framework are defined using the preprocessing functions:

```
#define _TELOSB_MOTE_
```

The composition of the HW components, energy and timing parameters for HW operations are included in a platform library (telosb.lib). On the platform layer, this library defines a set of the simulated HW components and provides platform specific information to the generic HW modules. The energy and timing parameters that we used in the simulation were partially received out of direct measurements and the rest was taken from the components specifications of the TelosB mote. Table 1 presents the possible energy conditions of the TelosB mote. The SystemC module of the TelosB mote is introduced as a SystemC CPU module that coordinates the work of the peripheral component, such as an ADC, a transceiver, timers and an interrupt controller included and initialized inside the platform SystemC module:

States	Current consumed	CPU	
Radio		*Active (8MHz)*	2 mA
Voltage regulator off (OFF)	0.02 uA	*Active (1MHz)*	0.6 mA
Power Down mode (PD)	20 uA	*ADC-12 conversion*	500 uA
Idle mode (IDLE)	400 uA	*SPI transmission*	700 uA
RX	18.8 mA	*LPM0*	300 uA
TX(-25 dBm)	8.5 mA	*LPM1*	80 uA
TX(-15 dBm)	9.9 mA	*LPM2*	20 uA
TX(-10 dBm)	11 mA	*LPM3*	10 uA
TX(-5 dBm)	14 mA	**LED (*each*)**	
TX(0 dBm)	17.4 mA	*On*	2.1 mA

Table 1. Energy states of the Telosb mote

```
          ...
SysTop(sc_module_name _name ): GCpu( _name ),
                  Adc("Adc"),
                  Radio("CC2420"),
                  Timer_1("Timer_A"),
                  Timer_2("Timer_B"),
                  IRQ_manager("IRQ")
          ...
```

When performing the coordination, the CPU operates in the *active* state, thus consuming more power compared to the *idle* mode, when no interrupts need to be handled. The CPU spends most of the time in the power safe idle (LPM0) mode. Thus only about 1% of the CPU computational/time resources is utilized by the case study application. The rest of the time, the application operates with 2 timers (Timer A and Timer B), ADC (ADC 12), USART (SPI mode) controller and a radio transceiver (CC2420).

4.3 The Energy Source

The energy layer is represented by a battery model, described in Section 5. This model is adjusted in accordance with 1.2 V type NiMH battery specification [4], which supports up to 1.6 V while recharging. The model provides the system with 3.1 voltage level that is equal to the power supply of two batteries connected in series of 100 mAh of total capacity. Low capacity is chosen to decrease the simulation time of the battery charge-discharge process. The minimal voltage level required to supply the TelosB platform is of 2.1V. Thus, in the power consumption estimation framework we use a voltage range from 2.1 to 3.1V.

5 Spice compatible NiMH battery model

In our work we have implemented the electrical battery model proposed by Chen [17] using PSpice [24]. Due to the absence of controllable components in the standard PSpice library, we started the model development from controllable resistor and capacitor modeling, so that their characteristics could be changed in accordance with seven behavior equations derived by Chen et al. [17]

The Spice model of the controllable resistor(see Figure 3) is designed in accordance with the derivative of *Ohm's law* ($V = IR$). This component has three pins, i.e., *controllable*, *input* and *output*. The *multiplier* component performs the multiplication of resistance provided by the *controllable* pin, and current from current-controlled voltage source of the *input* pin. The desirable voltage appears at the *output* pin of the controllable resistor.
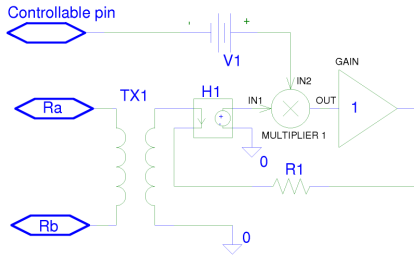
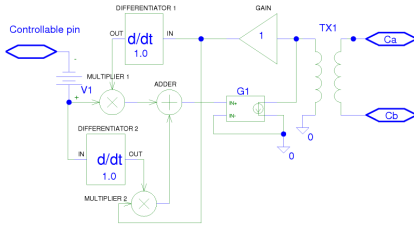5

Figure 3. Controllable resistor

Figure 4. Controllable capacitor

The controllable capacitor (see Figure 4) component contains the same three pins as the controllable resistor and is designed on the basis of the following equation:

$$i = C \cdot dV/dT + dC/dT \cdot V \qquad (1)$$

It contains two terms: the product of capacitance with the voltage derivative, and the product of voltage with the capacitance derivative. To get the first term the model multiplies the capacitance from the *controllable* pin and differentiates the voltage from the voltage-controlled current source which is connected directly to the *input* pin. The second term is derived by multiplying the voltage from the *input* pin with the differentiated capacitance from the *controllable* pin. Finally, the *adder* component accumulates both terms. In order to save space and avoid the undesirable interconnections during the model debugging, we created the sub circuits for controllable components.

In addition, in order to connect both components to nodes with a voltage other than ground, we applied a linear transformer with high inductance in parallel to the *input* and *output* pins of the controllable components.

To ensure that the controllable models of the resistor and the capacitor function properly, we simulated both using Spice. Next, we compared the results with a conventional resistor and capacitor respectively. The simulation curves of the controllable and conventional components are equal.

The PSpice battery model is depicted in Figure 5. The designations of the components and their interconnections are performed in accordance with the electrical battery model [17].

Pins V_{batt+} and V_{batt-} are positive and negative poles of the implemented PSpice model. Pins a-b in the created sub circuits are standard input-output pins for passive components. The *Control* pin establishes the controllable components behavior specified by behavior equa-
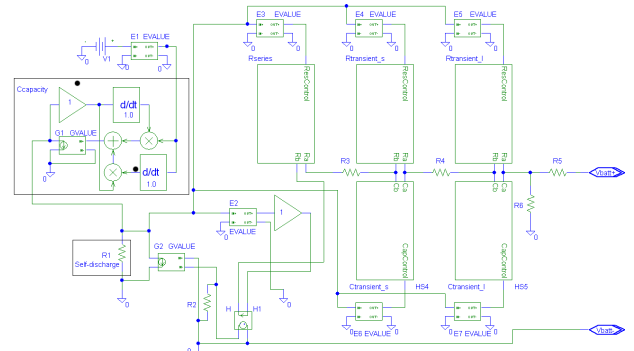

Figure 5. Spice model of a NiMH battery

tions [17] for each component. *Value form* (EVALUE) components E1, E3-E7 determine the value changing of controllable components. into $C_{capacity}$ (battery charge), R_{series} (instantaneous voltage drop of the step response), $R_{transient\_s}$, $R_{transient\_l}$, $C_{transient\_s}$ (short-time constants of the step response), $C_{transient\_l}$ (long-time constants of the step response) models correspondingly. The values of these components are changing according to seven behavior equations derived by Chen et al. [17]. Self-discharge resistor R1 represents self-discharge loss of power when battery is kept for a long time.

5.1 Battery Model Verification

To validate the NiMH battery model developed in PSpice we have applied a constant current charge and discharge, and two various pulse discharge experiments both for the model and a real battery. In our experiment, we used a 200-mAh, 8.4 V NiMH Coop rechargeable battery (9 V prismatic type). This battery type has the cut-off discharge voltage at 7.0 V level (the gap is 1.4 V) that is more convenient for battery model verification in comparison to 1.0 V cut-off discharge level of 1.2 V battery type (the gap is 0.2 V). The first experiment aimed to discharge the battery with a constant 23 mA current which is the typical current consumption of TelosB sensor nodes with the radio turned on. The end-of-charge battery voltage is 7 V. To record the battery voltage we used (in all cases) an Agilent 34411 digital multimeter. The discharge profile of the real battery and of battery model is depicted in Figure 6. The maximum voltage and runtime error for each charge/discharge profile are shown in Table 2.

The second case is a continuous battery charging with 0.1C current, where C is a nominal battery capacity (see Figure 7). Agilent E3631A DC power supply was applied to generate the charging current.

The last two cases refer to battery discharging with 0.1C and 0.2C current pulses (see Figure 8 and 9 respectively). 0.1C current pulses have a 1000 s pulse width for a 1200 s pulse period. 0.2C current pulses, in turn, have a 2000 s pulse width for a 2500 s pulse period. The pulses were generated by an HP 33220A programmable pulse generator which handled the Schrack RA 200006 relay to close and break the loop.
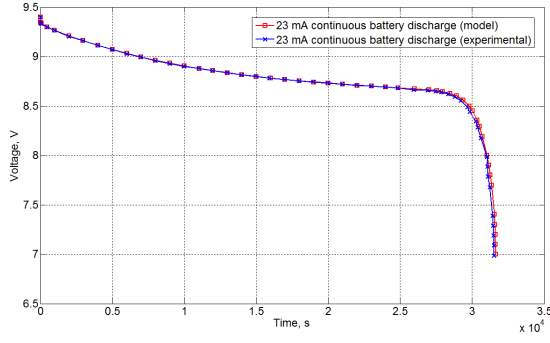
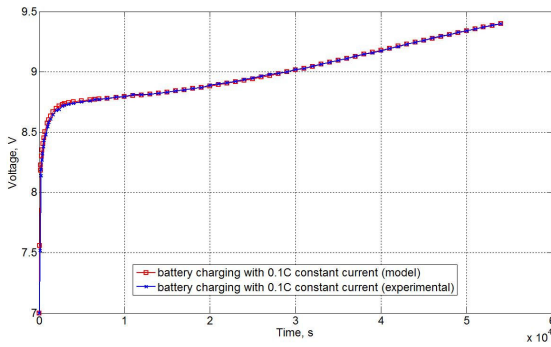**Figure 6. Battery discharging with continuous 23 mA current**

Figure 7. Continuous battery charging (0.1C)

The implemented PSpice battery model on the basis of electrical model proposed in [8] demonstrates accurate voltage response and runtime prediction.

Comparison type	Max error in voltage (mV)	Runtime error, %
0.1C continuous charging	43	0.278
23 mA continuous discharging	34	0.099
0.1C pulse discharging	24	0.352
0.2C pulse discharging	26	0.112

Table 2. Battery model validation results

6 Experimental Results

Figure 10 presents the graph that is composed of two curves, one dedicated to the battery charge period (during 80000s up to 3.1V) and the other one to its discharge (during 93962s down to 2.1 voltage level) using pulses generated by the SystemC framework simulating the work of a single node with the 100% duty cycle. The battery discharging is quite smooth at 3.1 to 2.4 voltage range. However, starting from 2.4 V the discharging curve slumps
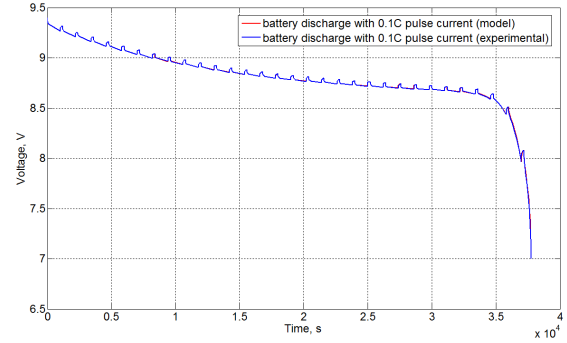

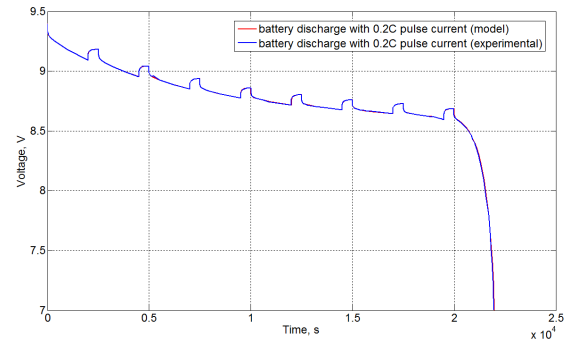**Figure 8. Pulse battery discharging (0.1C)**

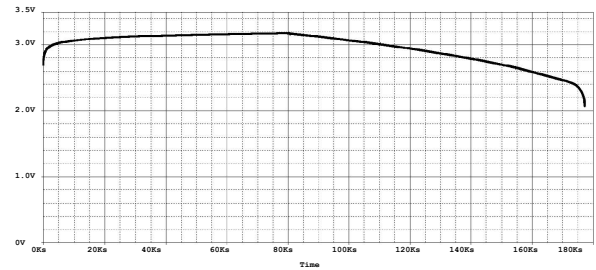**Figure 9. Pulse battery discharging (0.2C)**

quickly.

Figure 10. 100% duty cycle

To carry out the energy consumption experiments we were charging the battery model during 80000 seconds up to 3.1 V (we imply two cylindrical type NiMH batteries connected in series). The battery model capacity was defined as 100mAh. The input discharge pulses for the battery were generated by sensor node state simulator presented in this paper. The five discharge profiles were applied in order to estimate the system life time: 100%, 50%, 50% with doubled discharge and idle pulses, 10% and 1% duty cycles. Table 3 presents the battery run time under specified duty cycle. It is obvious that 1% duty cycle is the most economical mode for system use. Moreover, in comparison to both of 50% and 10% duty cycle experiments this mode is more energy efficient in terms of utilization of battery charge unit per second. Besides, two various experiments with 50% battery discharge re-

7

Duty cycle	Run time (at 2.1 V)
100%	177962 s
50%	347945 s
50% doubled*	346602 s
10%	1744743 s
1%	17449958 s
*time interval and pulse time are doubled	

Table 3. Battery run time

veal that short time pulses lead to the battery life time increase. This fact can be explained by the relaxation effect of the battery. The experimental results presented in Table 3 give a general idea how the discharge pulses rate has an influence on the system long-term operation.

7 Conclusion

We have presented a SystemC-based methodology for power consumption evaluation of an individual node of a wireless sensor network that supports the PBD paradigm. We first gave a general overview of the methodology. Then we have applied the methodology to perform power consumption estimation of a case study application running on a TelosB mote in terms of a battery lifetime. We show that with our framework it is possible to capture the influence of the relaxation effect to the battery lifetime.

As for our future work we plan to include simulation of heterogeneous networks by extending the presented methodology with an additional *network layer* and provide ability to model and simulate a network of nodes. Furthermore, a simpler (linear) customizable battery model will be implemented/added as a part of the framework. It will support quicker analysis/evaluation of power consumption and lifetime estimation. The other interesting area is the support of already available WSN frameworks and OSs. Our main focus is on TinyOS due to its popularity in the WSN community. Currently we are investigating ways to link TinyOS applications with our service layer.

Acknowledgment

References

[1] Atmel avr cpu. http://www.atmel.com/.
[2] Micaz, mica2, telosb. http://www.xbow.com/index.aspx.
[3] nesc. http://nescc.sourceforge.net/.
[4] NiMH battery specification. http://www.eemb.com/NI-MH_NI-CD_battery.html.
[5] Ns-2. http://www.isi.edu/nsnam/ns/.
[6] Omnet++. http://www.omnetpp.org/.
[7] Otcl. http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/.
[8] Powertossim. http://www.eecs.harvard.edu/ shnayder/ptossim/.
[9] Ptolemy ii. http://ptolemy.berkeley.edu/ptolemyII/.
[10] TelosB platform. http://www.xbow.com/.
[11] Tinyos. http://www.tinyos.net/.
[12] Visualsense. http://ptolemy.eecs.berkeley.edu/visualsense/.
[13] A. Acquaviva, F. Fummi, G. Perbellini, and D. Quaglia. A systemc-based framework for modeling and simulation of networked embedded systems. In *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pages 49–54, Stuttgart, Germany, 23-25 Sept 2008.
[14] A. Bonivento, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. Platform based design for wireless sensor networks. *MONET*, 11:469–485, 2006.
[15] S. Buller, M. Thele, R. W. D. Doncker, and E. Karden. Impedancebased simulation models of supercapacitors and li-ion batteries for power electronic applications. *IEEE transactions on Industrial Applications*, 41(3):742 – 747, May/June 2005.
[16] L. P. Carloni, F. D. Bernardinis, A. L. Sangiovanni-Vincentelli, and M. Sgroi. The art and science of integrated systems design. In *Proceedings of the 28th European Solid-State Circuits Conference, ESSCIRC 2002*, Firenze, Italy, September 2002.
[17] M. Chen and G. A. Rincon-Mora. Accurate electrical battery model capable of predicting runtime and iv performance. *IEEE transactions on energy conversion*, 21(2):504– 511, June 2006.
[18] E. Cheong, E. A. Lee, and Y. Zhao. Viptos: A graphical development and simulation environment fortinyos-based wireless sensor networks.
[19] D. Dennis, V. S. Battaglia, and A.Belanger. Electrochemical modeling of lithium polymer batteries. *J. Power Source*, 110(2):310–320, August 2002.
[20] F. Fummi, D. Quaglia, and F. Stefanni. A systemc-based framework for modeling and simulation of networked embedded systems. In *Proceedings of Forum on Specification, Verification and Design Languages, 2008. FDL 2008*, pages 375–378, Orlando, Florida, USA, 2008.
[21] L. Gao, S. Liu, and R. A. Dougal. Dynamic lithium-ion battery model for system simulation. *IEEE Transactions Compon. Packag. Technol.*, 25(3):495–505, Sep. 2002.
[22] S. C. Hageman. Simple pspice models let you simulate common battery types. *Electronics Design, Strategy, News*, pages 17–132, Oct 1993.
[23] D. S. Iyengar. LSU sensorsimulator user manual, February 2006.
[24] OrCAD. *OrCAD PSpice A/D*. Online Manual, USA, October 1998.
[25] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. Cosi: A framework for the design of interconnection networks. *Design & Test of Computers, IEEE*, 25(5):402–415, Sept.-Oct. 2008.
[26] D. Rakhmatov, S. Vrudhula, and D. A. Wallach. A model for battery lifetime analysis for organizing applications on a pocket computer. *IEEE transactions on VLSI Systems*, 11(6):1019–1030, Dec. 2003.
[27] P. Rong and M. Pedram. An analytical model for predicting the remaining battery capacity of lithium-ion batteries. In *Proceedings Design, Automation, and Test in Europe Conference and Exhibition*, pages 1148–1149, Munich, Germany, March 2003.
[28] A. L. Sangiovanni-Vincentelli. Defining platform-based design. *EEdesign*, February 2002.
[29] L. Song and J. W. Evans. Electrochemical-thermal model of lithium polymer batteries. *J. Electrochemical Society*, 147:2086–2095, 2000.
[30] B. L. Titzer and D. K. Lee. Avrora: Scalable sensor network simulation with precise timing.