System-level Modeling of Wireless Integrated Sensor Networks

Virk, Kashif M.; Hansen, Knud; Madsen, Jan

Published in:
International Symposium on System-on-Chip (SoC)

Link to article, DOI:
10.1109/ISSOC.2005.1595672

Publication date:
2005

Document Version
Publisher's PDF, also known as Version of record

Citation (APA):
Virk, K. M., Hansen, K., & Madsen, J. (2005). System-level Modeling of Wireless Integrated Sensor Networks. In *International Symposium on System-on-Chip (SoC)* IEEE. https://doi.org/10.1109/ISSOC.2005.1595672

System-level Modeling of Wireless Integrated Sensor Networks

Kashif Virk Knud Hansen Jan Madsen

Computer Science & Engineering Section
Department of Informatics & Mathematical Modeling
Technical University of Denmark, Lyngby 2800, Denmark
email: {virk, jan}@imm.dtu.dk

Abstract— **Wireless integrated sensor networks have emerged as a promising infrastructure for a new generation of monitoring and tracking applications. In order to efficiently utilize the extremely limted resources of wireless sensor nodes, accurate modeling of the key aspects of wireless sensor networks is necessary so that system-level design decisions can be made about the hardware and the software (applications and real-time operating system) architecture of sensor nodes. In this paper, we present a SystemC-based abstract modeling framework that enables system-level modeling of sensor network behavior by modeling the applications, real-time operating system, sensors, processor, and radio transceiver at the sensor node level and environmental phenomena, including radio signal propagation, at the sensor network level. We demonstrate the potential of our modeling framework by simulating and analyzing a small sensor network configuration.**

I. Introduction

Wireless sensor networks have emerged as a promising infrastructure for a new generation of monitoring applications. Owing to their small form-factors, ad-hoc deployment, and extended periods of unattended operation requirements, these wireless sensor networks form an extremely resource- and energy-constrained sensing, computing, and communication environment which makes the design and optimization of these systems a challenging task. In particular, the design of the sensor nodes requires a deep understanding of their various constituent components, their underlying technologies and the interactions between those components. Figure 1 shows the elements of a wireless sensor node and its hardware and software partitioning.

In order to be able to explore the design space at very early stages in the design process, it is important to have an accurate system-level model of the sensor network capturing all the inter-relationships among the diverse processors, software processes and radio- and sensor interfaces. In this paper, we present an extension of our earlier work on SystemC-based multiprocessor SoC modeling framework [1] which can provide the wireless sensor network designers a system-level abstraction of the sensor network for system-level design-space exploration to meet the requirements mentioned above[1].

Numerous sensor network simulators implemented in software exist, either in the open source or as commercial products, which can be broadly categorized into *improvised* sensor network simulators - based on existing network simulators or discrete-event simulation frameworks - and *custom* sensor network simulators. Typical examples of *improvised* sensor network simulators are: ns-2 [2], Opnet Wireless Module [3], and OMNeT++ [4] while common examples of *custom* sensor network simulators include: TOSSIM [5] and its extension PowerTOSSIM, Avrora [6] and its extension AEON, and Atemu [7]. Most of the *improvised* sensor network simulators emphasize sensor network level simulations (concentrating on the simulation of wireless communication protocol stacks) while a majority of the *custom* sensor network simulators focus mainly on sensor node level simulations (mostly code or processor simulations) and are either specific to certain sensor network research projects or support a limited number of sensor node platforms. A unified sensor node level as well as sensor network level simulator does not exist so far despite such attempts [8]. Moreover, to the best of our knowledge, none of the sensor network modeling approaches, reported so far, addresses the issue of designing sensor network systems from a hardware/software codesign perspective.

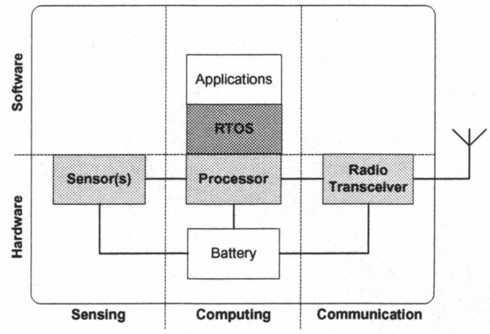

Fig. 1. *Sensor Node*

The main contribution of this work is to apply a HW/SW Codesign approach for the system-level modeling of a generic sensor node platform embedded in a generic sensor network environment model forming a system-level sensor network model which is fairly detailed as well as sufficiently efficient.

The rest of this paper is organized as follows: Section II

[1] A part of this work was funded by the ARTIST and the Hogthrob Projects.

0-7803-9294-9/05/$20.00 ©2005 IEEE

provides the methodology and implementation details for our sensor network model. The results of our implementation and a simulation example elaborating our modeling framework are presented in Section III. Section IV, finally, provides conclusions and the future directions of our work.

II. Sensor Network Model

In our SystemC-based modeling framework, a sensor network model is designed following the principle of composition. We model a sensor network at two levels: the sensor network level (Figure 2) and the sensor node level (Figure 3). This section describes the details of each of these levels and their inter-relationships and interactions.

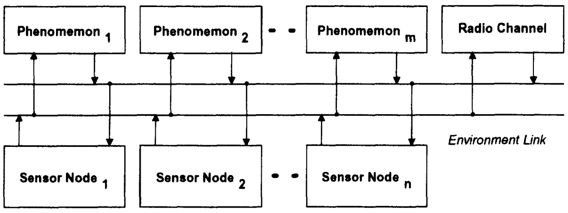

Fig. 2. *Sensor Network Level Model*

A. Sensor Node Level Model

At the sensor node level, a sensor node platform model is split into two sections: the software section - for functional simultion of the sensor node platform and the hardware section - to enable estimation of the energy consumption of the sensor node platform.

The software section of the sensor node platform model consists of the application model, comprising a set of task models, and the RTOS model, composed of a set of RTOS services [1].

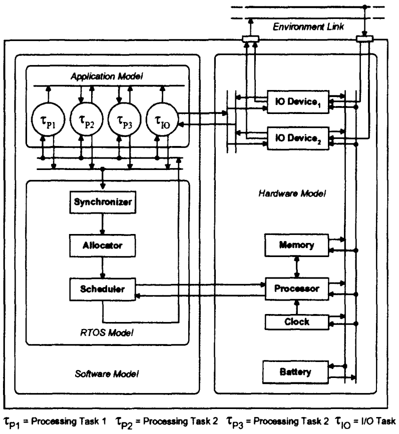

Fig. 3. *Sensor Node Level Model*

1) Application Model: The sensor node application software is modeled as a set of task models which are executed on the sensor node processor(s) under the control of RTOS(s). To accurately model the sensor node application, it is important to handle both the tasks and their possible inter-dependencies. The dependencies among the tasks are resolved by the synchronizer which is a component of the RTOS model.

The task models are the abstract building blocks from which the sensor node application model is composed. From the point of view of their activation mechanism, task models can be either *time-triggered (periodic)* or *event-triggered (sporadic)*. While periodic task models represent repetitive tasks, sporadic task models handle the response of the application model to the events that are generated either by the environment model or by other task models. In addition, from the point of view of their function or behavior, task models are organized into two groups:

- *processing task models* (τ_P) model the usage of a sensor node processor and are controlled by the RTOS model.
- *I/O task models* (τ_{IO}) model the usage of the I/O devices on a sensor node platform, e.g., the sensors and the radio transceiver. These task models form, a link between the RTOS model and the environment model with which they are interfaced using specific interface protocols (e.g., poll-based/interrupt-based, serial/parallel, etc.). There are two separate I/O tasks to model the radio transceiver behavior. The send task models radio transmission and the receive task models radio reception.

The function or behavior of a task is modeled as a finite-state machine (FSM) with five states as indicated in Figure 4: *idle*, *ready*, *running*, *preempted*, and *self-preempted*. Each task model is characterized by a set of parameters, such as the worst- and the best-case execution time, context-switching overhead, deadline, period (for a periodic task), offset, resource requirements, and precedence relations. Upon initialization, each task starts in the *idle* state and, if its offset value is zero, it transits to the *ready* state. The task remains in the *ready* state until it receives a `run` command from the RTOS scheduler upon which it transits to the *running* state. When the task has finished its execution, it issues a `finished` message to the scheduler and transits back to the *idle* state. At any time during its execution, a task may be preempted by the scheduler and it then enters into the *preempted* state where it waits till it receives a `resume` command from the scheduler which enables it to reenter the *running* state. The *self-preempted* state models the ability of an application task to release processor control to some other applicaion task requesting it, while it is waiting for an interrupt from an I/O device. Note that the *self-preempted* state is different from the *preempted* state in that the task itself controls its transition to and from it, while the transition to and from the *preempted* state is controlled exclusively by the scheduler.

The occurence of an interrupt is modeled by the `self-resume` message from a task in the self-preempted state. To service the interrupt, the priority of the self-preempted task is updated to the maximum level when it self-resumes. Thus, an interrupt is handled by the RTOS scheduler by interrupting the execution of whatever task is running at the time of its occurrence to service the interrupt and the portion of the application task running after self-resumption represents interrupt servicing. The only difference between running a

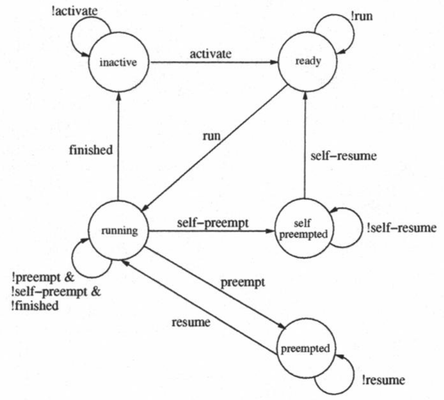

Fig. 4. *Task Model*

high-priority task and interrupt servicing is that a high-priority task may not preempt a running task if it has the same priority, while interrupt servicing does preempt a running task, even the interrupt servicing of another, previously-occured interrupt (e.g., in case of nested interrupts).

2) RTOS Model: The RTOS model is composed of three independent modules that model the basic RTOS services. The model is designed such that any of the RTOS services can be changed in a simple and straight-forward manner. Each module handles its relevant data independently of the other to preserve composability. A scheduler models a real-time scheduling algorithm. A synchronizer models dependencies among tasks. An allocator models the mechanism of resource sharing among tasks. For details on the model and how it is implemented in SystemC, we refer to [1].

All the task models are connected to the RTOS model through a pair of SystemC master/slave ports. In addition to that, the I/O task models are connected to the master/slave ports of the sensor node platform model which, in turn, are connected, in a similar way, to the components of the environment model. The receive and the sense task models also have activation ports (see Figure 3).

3) Battery Model: The hardware section of the sensor node platform model contains energy macro models[2] for the processor, memory, clock, and I/O devices alongwith a battery model. The battery model handles the energy consumption of a sensor node. It is connected to each of the hardware component models of the sensor node and decreases its energy resources depending on their power draw. At each clock cycle, the battery model updates it energy resources according to a certain specified function depending on the selected battery model (simplistic linear battery discharge models as well as more advanced battery models, which take the hysteresis phenomenon into account, can be selected). The link between the hardware component models is bidirectional which enables modeling the demise of a sensor node when its battery runs out of energy. The battery model can also inform the hardware component models when its energy resources go below predefined thresholds.

B. Sensor Network Level Model

At the sensor network level, a sensor node platform model is embedded in an environment model that models the environmental phenomena to be sensed by the sensor network application.

1) Environment Model: The environment model represents an abstraction of the environment as observed at the outputs of the sensors on the sensor nodes. It is composed of different component models each of which corresponds to the phenomenon monitored by the sensor network application. The environment model connects all the instantiations of the sensor node model – any of which can request it for data pertaining to a certain phenomenon. The environment model can also generate events for any instantiation of the sensor node model.

To model sensing, an I/O task model requests or gets events from the environment model component corresponding to the phenomenon (temperature, movement, etc.) according to a certain interface protocol (poll-based/event-based, serial/parallel, etc.). The receiver part of the radio transceiver is treated as a special kind of sensor and the transmitter part as a special kind of actuator. Thus, the radio signal propagation through the environment is treated as a special kind of phenomenon. The radio channel model, therefore, forms a special component of the environment model.

III. Example

This section describes an example illustrating the capabilities of our sensor network model to capture the mechanism of radio communication among the sensor nodes. The example configuration consist of 5 sensor nodes, two of which are transmitting a message while the rest are receiving it (see Figure 5).

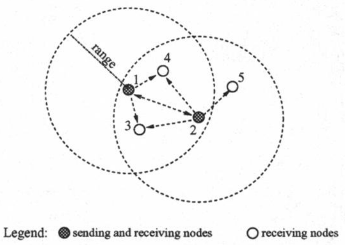

Fig. 5. *Example Topolgy*

On each sensor node, the processor runs the I/O tasks modeling the communication protocol. The transmission component of the communication protocol is described in Figure 6 and the reception component of the communication protocol is described in Figure 7. Two I/O task models have been instantiated for this example. The send task is a low-priority task, i.e., it does not preempt a running task when it initially starts. Once it has started, it periodically self-preempts and self-resumes. Everytime the send task enters its 'running' state, it steps through the states of the send protocol, either

[2] The energy macro modeling approach refers to the pre-characterization of a hardware or a software macro-block in terms of its energy consumption using empirical, simulation, or analytical models. A macro-block comprising a system can be defined at any level of abstraction by trading-off accuracy with efficiency or vice vera, e.g., a hardware macro-block can be defined at the RT-level or a software macro-block can be defined at the instruction-level.

causing transition(s) to the next state(s) of the send protocol or retaining its existing state. The receive task is a high-priority task (its activation is based on the timer interrupts). Similar to the send task, the receive task executes the receive protocol in its 'running' state.

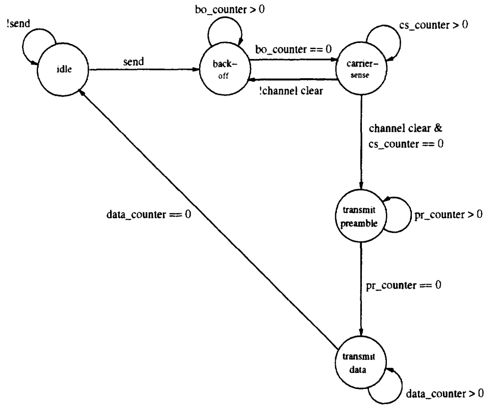

Fig. 6. *Send Protocol running on Send Task Model*

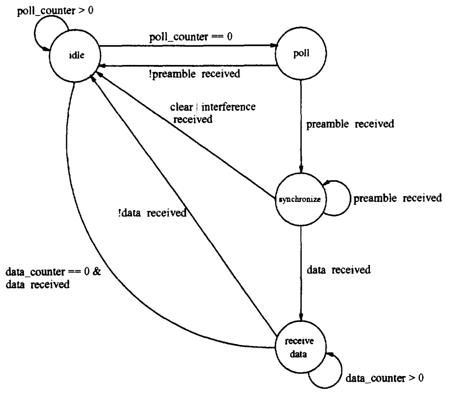

Fig. 7. *Receive Protocol running on Receive Task Model*

The simulation output waveforms corresponding to the example are presented in Figure 8. This figure represents the state of each task in terms of processor occupation as well as in terms of the communication protocol state for the send and the receive tasks. The example illustrates the behavior of the MAC (CSMA protocol) in the case of channel contention. The send task of the sensor node 2 fails to obtain channel access at its first attempt (because it detects that the sensor node 1 is transmitting). It, therefore, backs-off for a random period of time before reattempting to gain access to the radio channel. On its second attempt, the transmission of the sensor node 1 has finished and the radio channel is clear, so the sensor node 2 can send. The reason why the sensor node 1 gains access to the radio channel first is because its initial back-off time was smaller than that of the sensor node 2. Furthermore, notice that once the send task of the sensor node 1 has finished transmitting, the receive task of the sensor node 1 polls the radio channel, detects the preamble from the sensor node 2 and receives the packet sent by it.

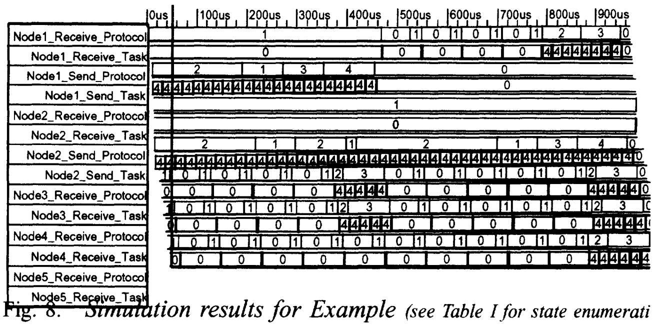

Fig. 8. *Simulation results for Example (see Table I for state enumerati*

TABLE I

State No.	Send/Receive Task	Send Protocol	Receive Protocol
0	inactive	idle	idle
1	ready	back-off	poll
2	running	carrier-sense	synchronize
3	preempted	transmit preemble	receive data
4	self-preempted	transmit data	

IV. Conclusions

We have presented a system-level wireless sensor netw modeling framework based on SystemC. The aim of our m eling framework is to provide designers of sensor netwc with a simple modeling and simulation framework in wh one can experiment with different application task mappir RTOS policies and communication protocols in order to ficiently utilize the limited resources available. Using framework, one can also study the consequences of des decisions taken at the sensor node-level on the behavior performance of the sensor network. We are currently work on extending our modeling framework to incorporate m accurate power modeling. This will enable us to estimate h different power management strategies can improve the ser network lifetime.

References

[1] J. Madsen, K. Virk, and M. Gonzalez, "Abstract RTOS Modelling Multiprocessor System-on-Chip," in *International Symposium on Sys on-Chip*, November 2003, pp. 147–150.

[2] University of California, Berkeley, "Network Simulato http://www.isi.edu/nsnam/ns/.

[3] OPNET Technologies Inc., "OPNET Wireless Modu http://www.opnet.com/products/wirelessmodule.

[4] OMNet++, "OMNet++: Discrete-Event Simulation Syst http://www.omnetpp.org/.

[5] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate Scalable Simulation of Entire TinyOS Applications," in *Proceeding 1st International Conference on Embedded Networked Sensor Sys (SenSys 2003)*. ACM Press, 2003, pp. 126–137.

[6] B. Titzer, D. K. Lee, and J. Palsberg, "Avrora: Scalable Sensor Netv Simulation with Precise Timing," in *Proceedings of 4th Internati Conference on Information Processing in Sensor Networks (IFIP 2C* April 25-27, 2005.

[7] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, and M. K "ATEMU: A Fine-Grained Sensor Network Simulator," in *Proceedin First International Conference on Sensor and Ad Hoc Communicat and Networks (SECON 2004)*, October 4-7, 2004.

[8] H. Park, W. Liao, K. H. Tam, M. B. Srivastava, and L. He, "A Un Network and Node Level Simulation Framework for Wireless Se Networks," Center for Embedded Networked Sensing, UCLA, Tech. F Nr. 25, September 2003.