Multilevel Sensor Node Simulation within a TLM-like Network Simulation Framework.

Joseph Wenninger, Markus Damm, Javier Moreno, Jan Haase, Christoph Grimm
Institute of Computer Technology
Vienna University of Technology, Austria
{wenninger|damm|moreno|haase|grimm}@ict.tuwien.ac.at

Abstract

Simulating Wireless Sensor Networks is a complex task, since the interplay of node level and network level with respect to power consumption requires a holistic modeling and simulation approach. Regarding the sensor nodes, there are many modeling approaches and levels available, but a corresponding network level approach might not exist. In this paper, we present the integration of a multilevel simulator for the node level into a TLM-based network simulation approach. [1]

1 Introduction

The research on Wireless Sensor Network (WSN) has three main aspects. For one, there is the design of the single nodes itself, which is in principal not different from the design of any other embedded system. Another aspect is the network perspective, involving protocol design and routing strategies. The third main aspect is non-functional and mainly driven by the typical application scenarios of WSNs: power consumption. Since WSN-nodes often have limited power resources, a low power consumption is important.
While node-level and network-level design are almost orthogonal to each other, they are linked in a subtle way when power awareness comes into play. Power-optimal solutions on the node level might not yield power-optimal solutions on the network level, and vice versa. With respect to simulation, this means that both levels have to be taken into account. Since the node level can be very detailed, and the network might contain many nodes, the simulation becomes very costly.
A special focus of this work lies on automotive WSNs, like Tire Pressure Monitoring Systems (TPMS). Apart from these obvious wireless applications, WSN is also a possibility to control the complexity and size of the cable harness within the car, which currently amounts to cable lengths of up to 2 km.
Figure 1 gives an example of a WSN within a car. We can assume that the sensor data is collected and processed by a central unit (CU). Also, a field bus (e.g. CAN) will be present, since certain safety critical applications (e.g. brakes, steering) will never be implemented via radio. Therefore, we actually face a mixed wireless / wired sensor network, with some sensors attached to the bus.

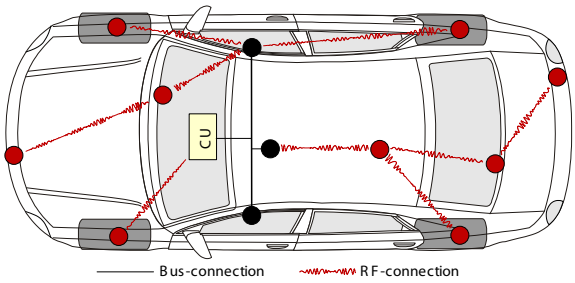

Figure 1: A mixed wireless/wired automotive sensor network

The sensor nodes in the WSN can differ largely regarding duty cycle, power consumption, operation distance, and power supply. Part of the nodes might be attached to the car's electrical power system, thus are provided with a virtually infinite power supply. While most of the nodes might be only designed to transmit data, some nodes will be also able to receive and forward messages within a multi-hop network.
The work we describe here is done as a part of the SNOPS (Sensor Network Optimizations by Power Simulation) project.
The remainder of this paper is organized as follows: We first give an overview of related work, esopecially the PAWiS (Power Aware Wireless Sensors) project, which is a predecessor to the SNOPS, project, and including an introduction to TLM. In Section 3, we discuss briefly how the TLM approach can be applied to WSN simulation. After that, we introduce the used simulator for the node level,

[1] This work is conducted as part of the Sensor Network Optimization by Power Simulation (SNOPS) project which is funded by the Austrian government via FIT-IT (grant number 815069/13511) within the European ITEA2 project GEODES (grant number 07013).

and outline in Section 5 how to integrate it into the TLM based simulation approach. We conclude in Section 6.

2 Related work

In [11] SystemC AMS was used to simulate nodes of a WSN at architecture level, including analog components. In [3] SystemC was successfully used to co-simulate the hardware and software parts of a wireless sensor network. Some preliminary tests were made with a pure SystemC model with excellent results. However, the lack of support for modeling upper layer network communication in SystemC finally led to the usage of NS-2 [7] to model the network.

In [13], a protocol for cooperative MIMO mobile sensor networks was proposed. For the modeling, TLM and SystemC was used, resulting in a very fast and efficient simulator that permitted to evaluate the network performance. However, no details were given on how TLM was used exactly.

2.1 The PAWiS framework

During the PAWiS (Power Aware Wireless Sensors) project [5], a simulation framework for wireless sensor networks was developed. It is based on the OMNeT++ Discrete Event Simulation System [10]. The PAWiS framework models the internal structure of sensor nodes and the communication among them. Each node is built as a virtual prototype divided into functional blocks, which are mapped into modules. The PAWiS Framework provides the module interfaces, as well as the messages used to communicate between them.

Due to the dedicated focus on power optimization, the framework provides a Power Meter where reporters, associated to modules, log the power consumption values. In addition to modules and the message infrastructure, which are an extension of those provided by OMNeT++, the PAWiS framework provides also a model of the environment, called the Air. The Air model includes calculating signal power received, the bit error ratio and modeling collisions.

As it is in OMNeT++, every communication between modules is done through messages. The PAWiS framework implements its own messages, inheriting from the OMNeT++ basic message class. Each communication layer can define their own message kind, with the required fields, and encapsulate messages received from the upper layer.

In the PAWiS project [5] a wakeup receiver (WuR) as an actual part of a WSN node was designed and implemented [9, 6]. As a second module of the WSN node an energy management unit including a harvester was designed [4].

2.2 Transaction Level Modeling

Transaction Level Modeling (TLM) is a modeling paradigm mainly targeting bus-based systems. The low-level signal events like setting and resetting acknowledge or writing on address or data lines are bundled into one single object called *transaction*. A transaction incorporates the whole communication process of a data transfer, possibly together with more or less accurate timing estimates for the transfers.
TLM is a technique mainly employed in C-based design, and especially within SystemC [8]. With TLM 2.0 [2], there exists an OSCI standard with a focus on model interoperability with standardized transaction objects and abstract protocol phases. The standard transaction class in TLM 2.0 is called *generic payload*, and has attributes like a command (read or write), a target address and a data section, which basically sets a focus on memory mapped busses.
Transactions are passed by reference via method calls among the different components of the system model. For example, a read request from a CPU to a Memory module together with the resulting data transfer can be captured by one transaction. In TLM 2.0, every component in the system model basically assumes one of three roles:

- *Initiators* (e.g. CPU models) create new transactions.
- *Interconnects* (e.g. busses and routers) do forward and possibly modify transactions.
- *Targets* (e.g. memories and I/O components) are the final destinations of the transactions.

The TLM 2.0 standard foresees several method interfaces for passing transactions, with the most important ones being the blocking and the nonblocking interface. The blocking interface has a method implemented only by targets, namely `b_transport(`*`transaction`*`,` *`time-offset`*`)`. The name "blocking" results from the fact that an implementation of this method is allowed to yield to the SystemC simulation kernel by calling `wait(`*`time/event`*`)`, therefore blocking the calling process. The time offset parameter passed with the transaction indicates when the transaction is valid with respect to the current simulation time.
The lifetime of a transaction regarding simulation time is from calling `b_transport(...)` until the return. Therefore, it is a single time span, which can also be zero. This approach disregards any intermediate steps during the data transfer (like address- and data-phase), and only allows to account for a time period regarding the whole transaction. Therefore, TLM 2.0 modeling using the blocking interface is usually referred to as the *loosely timed coding style*. This coding also has the additional feature that *temporal decoupling* can be used, which means that initiators can run ahead of the global simulation time

by maintaining a local timing offset, and synchronize with the global simulation time either periodically or depending on certain circumstances. This technique allows for a further simulation speedup, since it reduces context switches. The nonblocking interface has two methods: `nb_transport_fw(`*`transaction`*`, `*`phase`*`, `*`time-offset`*`)` is implemented by targets, and `nb_transport_bw(`*`transaction`*`, `*`phase`*`, `*`time-offset`*`)` is implemented by initiators. With the additional phase parameter, the lifetime of a transaction can be subdivided into several phases. The standard phases foreseen in TLM 2.0 are `BEGIN_REQ`, `END_REQ`, `BEGIN_RESP` and `END_RESP`, which roughly captures address- and data phase, but it is also possible to define custom phases. Also, the transaction can now be sent actively by the target, such that it can be passed back and forth at different points in simulation time between initiator and target. Since the timing information associated to the transaction here has in general more than two timing points, modeling with the nonblocking interface is called the *approximately timed coding style*. For details, consult [2].

3 Using TLM for WSN simulation

In [1], we proposed a simulation approach for WSN based on TLM, since from a simulation point of view, an environment model (e.g. as it is implemented in the PAWiS framework [12]) behaves not much different from a bus model in TLM 2.0 (see Figure 2): it gets transmitted data packets (which correspond to transactions), and forwards them.

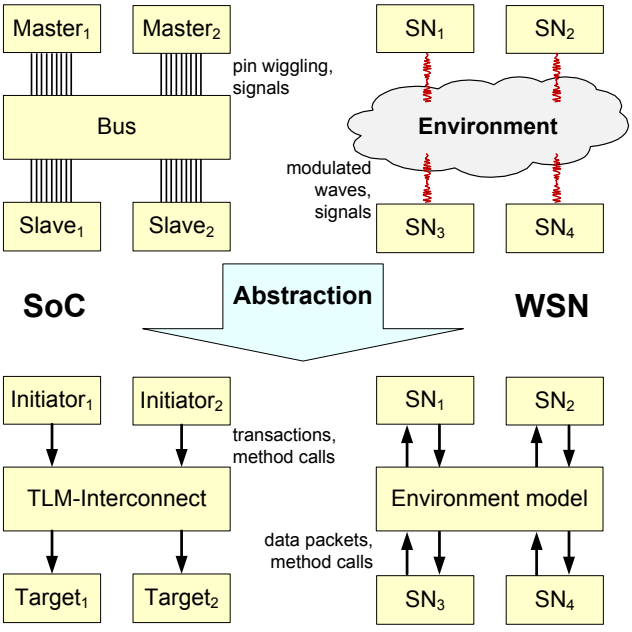

Figure 2: Similar abstractions for SoCs and WSNs (SN: Sensor Node)

However, regarding the objects involved in one transaction / data packet, there are some differences (see Figure 3). A data packet in a multi hop WSN can take different routes, where forks might occur. Also, nodes might receive a data packet, but might do nothing with it, since they are not involved in the routing to the given target. Still, the simulation has to account for this, since it influences the power consumption.

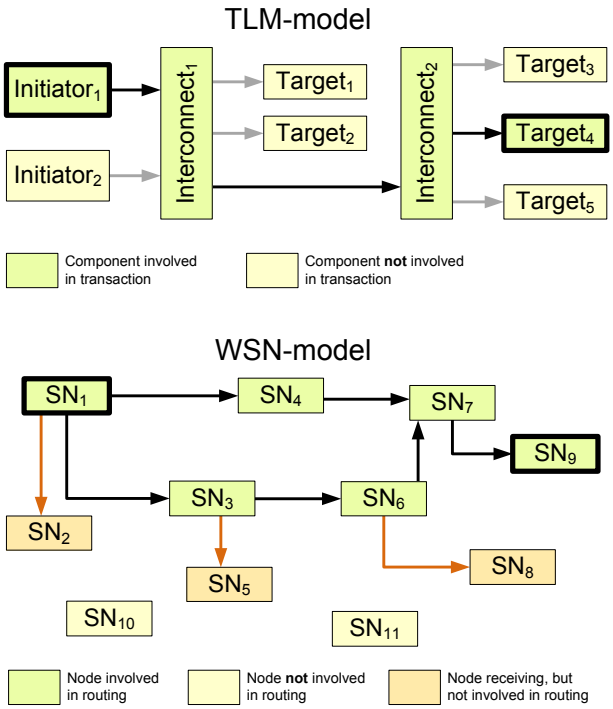

Figure 3: TLM Transaction path versus a WSN packet path

In [1], we presented an approach using the TLM 2.0 generic payload extension mechanism to deal with this differences. This extension mechanism is a way to add additional attributes of arbitrary kind to a generic payload object, and can be thought of as attaching a "Post-it" note to the transaction.

The potential benefits of this approach are manifold: With respect to the automotive scenario, TLM 2.0 offers a way to model the field bus in a straightforward manner. Also, TLM simulations are faster, as a first evaluation in [1] showed a 100% speedup. Regarding power simulation, we hope to get additional insight on the influence of network protocols on the power consumption by keeping the correlation of the transaction to the consumed power of the nodes propagating it through the network.

While the data representation in the transactions will be on a baseband-equivalent level like in PAWiS framework (see

Section 2.1), it is also possible to switch to a true baseband level or even an HF level data representation with this approach, depending on the modeling level of the sensor nodes.

Also, TLM is a very general approach, and therefore very flexible, especially when the need to couple with other C-based simulation environments arises. We will exploit this in Section 5.

4 A DLL based multilevel simulation framework

For the node modeling, a proprietary multi-level simulation framework is used, which is loosely based on the SystemC kernel. The framework abstracts above the low level SystemC codebase and basically only uses the discrete-event (DE) system, but with its own scheduler, which has its own time scale independent from the SystemC simulation time. The framework doesn't allow the use the wait function with time parameters at all.

Since the framework is a confidential development of our industry partner, we can't elaborate in detail on its multi-level modeling aspects, which is based on defining the communication means associated to the respective level by means of protocols, which are basically port-interface pairs.

The basic approach when building models with this framework is to write modules much like SystemC modules, but using the additional facilities provided by the framework library. This code is then compiled and linked against the libraries into a Dynamic Link Library (DLL). The final model is then represented by a proprietary configuration file, where the module instances used and their connectivity is specified. This configuration file is read by a simulator, which uses the DLL to instantiate and connect the modules, and finally runs the simulation (see Figure 4).

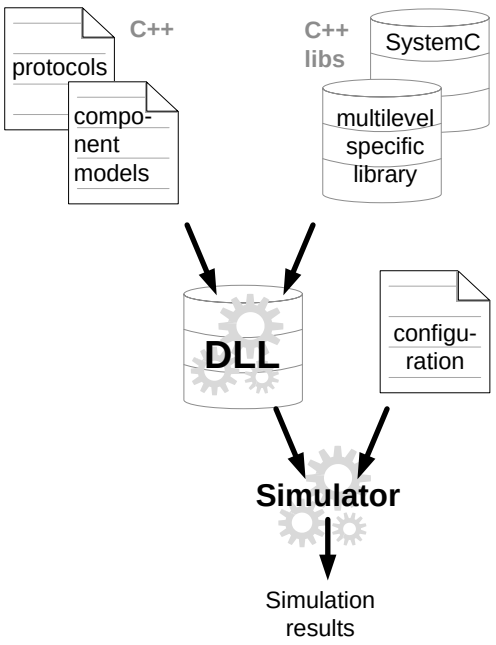

Figure 4: DLL-based simulation framework

The advantage of this framework is that systems can be built without the need to recompile everything and linking everything statically together. Every module is located in its own shared object (windows DLL). Instead of using signals and ports, the framework itself has its own connection mechanisms, with the advantage of a very high flexibility, which is needed especially for multi-level simulations, where in future e.g. modules in VHDL or Verilog could be connected directly to high level SystemC based code.
Each module can implement various instances of a protocol and even both sides of a protocol. The target purpose to be achieved within the SNOPS project is high simulation speed for power simulation while still being able to simulate some parts down to instruction set level or maybe even register transfer level.

5 Coupling with TLM

For the network simulation approach outlined in Section 3, it is necessary to integrate the TLM methodology into this multi-level simulation framework. Since the framework is already loosely based on SystemC, we decided to use the official OSCI TLM 2.0 library for our endeavors. We especially wanted to reuse the generic payload class, because it offers a high level of flexibility.
We faced some challenges tough, since the high level of abstraction above SystemC inhibits the usage of the `wait()` function with time parameters, and the usage of the `wait()` function taking an event object as trigger proved not to work well either. Another difficulty was that creating subprocesses or subthreads with `sc_spawn` is not possible either, since this is used by some of the TLM 2.0 utilities. The framework doesn't do the standard Sys-

temC elaboration phase for classic sc_modules, so we had to work around that, too.
The tasks for the integration therefore were:

- encapsulating the TLM sockets in standard SystemC modules and wrapping those modules with framework modules
- removing `wait`s and moving to the usage of framework based asynchronously scheduled method callbacks
- removing `sc_spawn` usage
- replacement of code needing the SystemC simulation time

For our purpose, we ported our implementation of wireless transceivers and the "air" as channel, which is heavily based on TLM 2.0. Regarding the framework, we kept an eye on not having to touch the existing framework itself, so we just used the public interface headers and the precompiled libraries and the runnerstub only. Since the framework itself uses dynamic binding for the purpose of runtime configurability, some SystemC features cannot be employed.
We had to rewrite the payload event queue, a TLM 2.0 utility needed for the approximately timed coding style which stores transactions and their phases and trigger a callback function according to the timing annotation. The reason is that it uses `sc_spawn` and an additional thread for its asynchronous callbacks. We replaced the process with time triggered scheduled methods provided by the framework and wrote a wrapper class around the scheduled method, which stores the payload data and the pointer to the real callback to make it possible to attach data to a scheduled method, because the scheduled methods do not have function parameters themselves. This has probably a performance impact, since for each deferred function invocation in a payload event queue an object has to be dynamically created and deleted.
If the `wait()` functionality is needed within a module, this can be achieved by scheduled methods using the framework simulation core. However, the core function with an ordinary `wait()` has to be split up into more iterations of the function, instead of just one iteration.
The endless loop in the various nodes, which did their task and waited in the end for a certain time and reiterated to do the next task, has been removed. The initialization part has been moved to the node constructor and the loop itself has been replaced to scheduled reinvocation of the mainloop function.
The first iteration of the code base created in our project group used the `simple_target_socket` and `simple_initiator_socket` sockets of the TLM library, which also use `sc_spawn`. We replaced them with the `multi_passthrough_sockets`, which on the other hand relied on the elaboration phase, especially on the `before_end_of_elaboration` and `end_of_elaboration` calls. Since this partly uses protected and/or private member functions, it first seemed to be a problem, because the framework does not use the elaboration phase at all. However, because the framework itself does not use ports, signals or sockets, we could just initiate the elaboration phase after the whole system has been created from the configuration file and the first call into our new subframework occured. From the semantic this is not optimal, but it works and the end result is the same as with the usual way.
Our binding just works that way: We have a singleton object, the air, which can be configured via the system configuration file, and the so called AirClient objects, which are now implemented as twins, one object for the binding to the framework and its uni directional call interfaces and one object for doing the SystemC related communication. If a framework client is created, it creates a tlm/air object too and registers it to the air environment. All data, including the payload data is published to the framework via the method call protocol interfaces, so nothing of the flexibility of our transmission environment simulation is missing when being combined with the overall multi-level framework (see Figure 5).

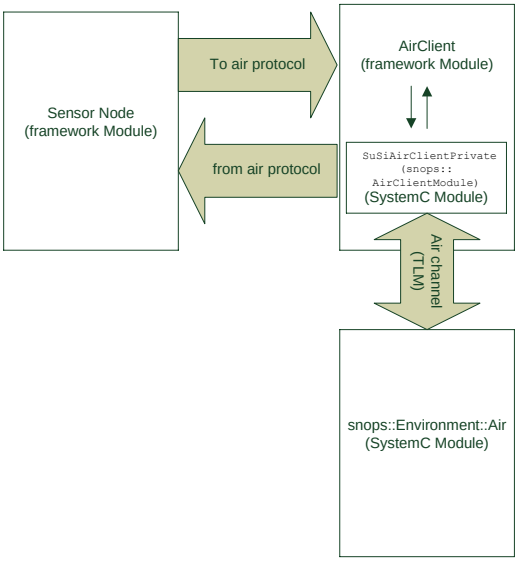

Figure 5: Module structure of TLM integration

6 Conclusion

In this paper, we propagated a WSN modeling and simulation approach taking into account the node level as well as the network level, with power simulation being the main driver. To this end, we proposed a simulation approach using a multilevel simulator for the sensor nodes, which is integrated into a TLM based network simulation. This approach promises a simulation speedup as well as new insights into the interaction of node level and network level design regarding power consumption.

References

[1]

[2] J. Aynsley. OSCI TLM-2.0 Language Reference Manual. Technical report, Open SystemC Initiative, 2009.

[3] F. Fummi, D. Quaglia, F. Ricciato, and M. Turolla. Modeling and simulation of mobile gateways interacting with wireless sensor networks. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 106–111, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[4] C. Hambeck, S. Mahlknecht, T. Herndl, and E. Halvorsen:. An Energy Harvesting System for In-tire TPMS. In *1st International Workshop on Power Supply On Chip*, Cork, Irland, 22.-29. Sept. 2008.

[5] S. Mahlknecht, J. Glaser, and T. Herndl. PAWiS: Towards a Power Aware System Architecture for a SoC/SiP Wireless Sensor and Actor Node Implementation. In *Proceedings of 6th IFAC International Conference on Fieldbus Systems and their Applications*, pages 129 – 134, Puebla, Mexiko, 14.-15. Nov. 2005.

[6] S. Mahlknecht and M. Spinola Durante. WUR-MAC: Energy efficient Wakeup Receiver based MAC Protocol. In *Proceedings of the 8th IFAC International Conference on Fieldbuses & Networks in Industrial & Embedded Systems, FET 2009*, Hanyang University, Ansan, Korea, 20.-22. May 2009.

[7] S. McCanne and S. Floyd. NS Network Simulator - version 2. Website, 1996. http://www.isi.edu/nsnam/ns.

[8] Open SystemC Initiative. *SystemC*™. http://www.systemc.org.

[9] M. Spinola Durante and S. Mahlknecht. An ultra low power Wakeup Receiver for Wireless Sensor Nodes. In *Proceeduings of the Third International Conference on Sensor Technologies and Applications, SENSORCOMM 2009*, Athens/Glyfada, Greece, 18.-23. June 2009.

[10] A. Varga. The OMNeT++ discrete event simulation system. In *European Simulation Multiconference (ESM'2001)*, Prague, Czech Republic, 2001.

[11] M. Vasilevski, F. Pecheux, N. Beilleau, H. Aboushady, and K. Einwich. Modeling and refining heterogeneous systems with systemc-ams: Application to WSN. *Design, Automation and Test in Europe Conference and Exhibition*, 0:134–139, 2008.

[12] D. Weber, J. Glaser, and S. Mahlknecht. Discrete Event Simulation Framework for Power Aware Wireless Sensor Networks. In *Proceedings of the 5th International Conference on Industrial Informatics, INDIN 2007*, volume 1, pages 335–340, Vienna, Austria, 23-26 July 2007.

[13] Q. Zhang, W. Cho, G. E. Sobelman, L. Yang, and R. Voyles. *Ubiquitous Intelligence and Computing*, pages 508–516. Springer Berlin/Heidelberg, 2006. ISBN 978-3-540-38091-7.