POLITECNICO DI TORINO
Repository ISTITUZIONALE

Network-aware design-space exploration of a power-efficient embedded application

Original
Network-aware design-space exploration of a power-efficient embedded application / Sayyah, P., Lazarescu, M.T., Quaglia, D., Ebeid, E., Bocchio, S., Rosti, A.. - ELETTRONICO. - (2012), pp. 567-574. (CODES+ISSS '12 Tampere, Finland October, 2012) [10.1145/2380445.2380531].

Availability:
This version is available at: 11583/2507479 since: 2020-10-22T12:06:56Z

Publisher:
Association for Computing Machinery (ACM)

Published
DOI:10.1145/2380445.2380531

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
ACM postprint/Author's Accepted Manuscript, con Copyr. autore

(Article begins on next page)

16 October 2026

Network-aware Design-Space Exploration of a Power-Efficient Embedded Application

Parinaz Sayyah
Politecnico di Torino, Italy
parinaz.sayyah@polito.it

Mihai T. Lazarescu
Politecnico di Torino, Italy
mihai.lazarescu@polito.it

Davide Quaglia
EDALab, Italy
davide.quaglia@edalab.it

Emad Ebeid
University of Verona, Italy
emad.ebeid@univr.it

Sara Bocchio
STMicroelectronics, Italy
sara.bocchio@st.com

Alberto Rosti
STMicroelectronics, Italy
alberto.rosti@st.com

ABSTRACT

The paper presents the design and multi-parameter optimization of a networked embedded application for the healthcare domain. Several hardware, software, and application parameters, such as clock frequency, sensor sampling rate, data packet rate, are tuned at design- and run-time according to application specifications and operating conditions to optimize hardware requirements, packet loss, power consumption. Experimental results show that further power efficiency can be achieved by considering also communication aspects during design space exploration.

Categories and Subject Descriptors

B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Design, Measurement, Performance

Keywords

HW/SW Timing and Power Simulation, Virtual Platform, Network Simulation, Wireless Sensor Network, Body Sensor Network

1. INTRODUCTION

The design of embedded systems is becoming increasingly complex in case of distributed applications made of several nodes consisting of hardware and software components interacting over a network. A realistic example of this kind of application is the health care wireless sensor network presented in Figure 1.

Each monitored person wears a number of wireless nodes that capture and process different kinds of body-related information; they exchange data and commands with a base station over a radio channel which is shared among different unrelated data flows.

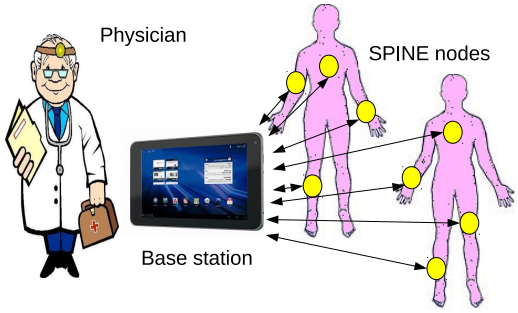

Figure 1: Example of distributed application based on networked embedded systems.

The wireless nodes are based on the open source Signal Processing in Node Environment (SPINE [8]) framework specification. It supports flexible and distributed signal processing for Wireless Body Sensor Network applications through a set of customizable functions for data acquisition, processing and communication.

The European project COMPLEX [3] uses SPINE-based software running on a 32-bit system-on-chip architecture from ST-Microelectronics as a case study for HW/SW design-space exploration (DSE) of the sensor node. Power consumption is key aspect in a mobile health care application to ensure long-time unattended operation of the sensor node. As described in Section 3.3, the ReISC SoC is highly configurable to save power when some components are not used. For each component, the definition of low-power or power-off periods usually depends on application timing constraints such as system promptness to asynchronous events or sampling rate of continuous values. In the specific case of wireless nodes transmitting on a shared channel, the network behavior plays a crucial role in determining the time behavior of the nodes with potential impact on their power consumption. Such network behavior depends on several causes and mainly by the number of concurrent data sources; an analytical approach to this study can become complex as the number of nodes increases. Therefore, network simulation is the most viable approach to make this timing analysis. Hence simulation must take into account the following aspects:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CODES+ISSS'12, October 7-12, 2012, Tampere, Finland.
Copyright 2012 ACM 978-1-4503-1426-8/12/09 ...$15.00.

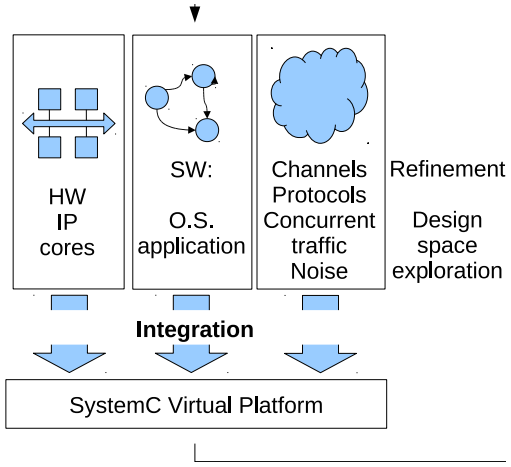

Figure 2: Generation of SystemC models from different domains: hardware, software and network.

- *software components*, such as the operating system and the application;
- *hardware components*, such as the CPU, memory, and peripherals;
- *networking scenarios* that, for instance, consist of channel behavior (e.g., path loss, collisions), communication protocols (e.g., IEEE 802.15.4), concurrent traffic and noise.

One approach for this kind of holistic simulation is the co-simulation [1] that combines and synchronizes the execution of several simulation tools, such as an instruction set simulator for software components, a VHDL simulator for hardware components, and a network simulator (e.g., NS-2 [5]) for the network. In [6] a co-simulation approach has been used to show that power optimization can be achieved by scaling CPU frequency according to the number of packets waiting to be transmitted because of network congestion.

With respect to past literature, the novel contributions of this work are:

- an industrial-level virtual platform written in SystemC [7] by STMicroelectronics is extended to simulate the wireless node in a full network scenario without using co-simulation;
- the hardware components and the network are modeled in detail, in order to enable the tuning of the power management strategy based on network timing constraints.

The resulting simulation flow is shown in Figure 2. The hardware blocks of the wireless node are modeled in SystemC at the Transaction Level (TLM) to capture their functional behavior at a high level. One of these blocks represents the CPU that executes the actual software components with machine-instruction accuracy. The hardware blocks are included in the SystemC model, together with network elements (protocol and channel). The resulting simulation models a full network scenario in which several wireless nodes interact in a realistic way.

The rest of the paper is organized as follows. Section 2 introduces the health care application. Section 3 describes the hardware platform of the wireless sensor node to be designed. Section 4 explains how all the aspects of the designed system and its operating environment can be modeled in SystemC. Section 5 describes experimentally how timing constraints derived from network simulation can be used to refine some system parameters to save power. Finally, conclusions are presented in Section 6.

2. APPLICATION

The COMPLEX project proposes a use case based on a complete and realistic Wireless Sensor Network (WSN) scenario. Its architecture, presented in Figure 1, consists of several nodes, namely *SPINE Nodes* and the *Base Station* (BS). It couples all major aspects of WSNs (sensing, processing, radio data transmission) with a non-trivial amount of local computation on the node. The raw sensed data are processed directly on the node, in order to reduce the energy consumed by the radio, which is the major limiting factor for the battery lifetime of WSN nodes.

The application is a virtual machine for health monitoring that conforms to the open specification of the SPINE [8] virtual machine for Wireless Body Sensor Network (WBSN) applications. It provides the user with a customizable set of functions that includes:

1. acquisition of various types of data (e.g., temperature, acceleration, blood oxygenation level, hearth pulse rate) and their storage in local circular buffers;
2. signal processing, including a variety of filtering, threshold detection, and other mathematical functions over the data in buffers;
3. communication via data packets with a base station either regularly or when events of interest are detected (e.g., emergency events).

The detailed application functionality is defined by the Base Station (BS) node at run time by means of configuration packets sent to each SPINE Node. The configuration can include the sampling period of individual sensors, buffer sizes, processing functions (e.g., max, mean, median) to be applied on raw sensed data, window size (i.e., the number of samples needed to compute a feature for the first time), shift size (i.e., the number of samples needed to compute the feature after the first time). The various configured tasks (e.g., sampling, feature computation, radio packet dispatch) can be selectively activated or deactivated at run time, depending on application demands.

One prominent application in the WBSN domain is fall detection by monitoring the body movements of elderly patients. In this paper, this is achieved by configuring the SPINE Node to collect and filter 3-axis acceleration samples and transmit these data over the wireless channel to the Base Station. The Base Station further processes the sensor data and detects falls based on the changes in the acceleration data over a timed window. The operator (physician or other health care professional) can monitor the patient condition by examining on the server the processed acceleration data from several sensor nodes.

To this end, the application should be configured by the operator. The first step is to connect the base station with the SPINE sensor node of interest. The operator selects those SPINE capabilities which are needed to define the

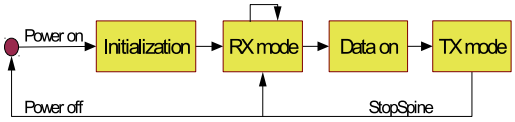

Figure 3: Behavior of the SPINE sensor node.

monitoring of interest, e.g., 3D acceleration. These decisions are then translated by the base station into configuration packets for the SPINE sensor node, which starts the requested measurements.

Specific menu items on the application user interface allow the operator to stop and resume the measurements at any time, or exit the sensor viewer altogether.

The base station can autonomously detect critical situations (e.g., the system detects that the monitored person falls when the average acceleration over a short interval of time exceeds a given threshold).

Figure 3 shows the behavior of the sensor node. After the *Initialization* phase it enters the *Receive mode* where it waits for configuration packets from the *Base Station* node.

The configuration packets are processed as received and the configuration sequence ends with a *StartSpine* command that changes the sensor node state to *Data On*. In this state, the tasks requested by the Base Station are executed.

Then the SPINE Node briefly enters the *TX mode* in which it samples and processes the sensor data and periodically sends packets with aggregated data to the Base Station.

Any incoming packets from the Base Station continue to be processed. These commands can change the set of tasks or their configuration, or can stop the processing altogether (*StopSpine* command).

3. HARDWARE PLATFORM

The hardware platform of the SPINE Node consists of:

- the *ReISC* System-on-Chip that performs sensing and data processing;
- the *RF Module* that provides wireless communications.

3.1 ReISC System-on-Chip

The ReISC SoC is the outcome of the design of a system on chip by STMicroelectronics, resulting in a real silicon chip which was taped-out at the end of 2009 using a 90 nm technology. The ReISC SoC is the first system on chip which drives a new family of ultra low power products.

The ReISC SoC encompasses the proprietary ReISC 3 core (Reduced energy Instruction Set Computer). It provides hardware support for 8/16/20/32 data sizes, variable 16 bit-based instruction length and secure data. ReISC 3 is targeted at ultra-low power applications. It operates up to 50 MHz frequency, contains embedded memories (1 Mbytes Flash memory and 32 Kbytes SRAM) and an extensive range of enhanced I/Os and peripherals.

It offers one 12-bit ADC, three general purpose 16-bit timers plus one internal timer, as well as standard and advanced communication interfaces: one I2C, two GPIOs, two SPIs, one USART, and one USB. A comprehensive set of power-saving modes allows the design of low-power applications.

The peripherals that are significant for the optimization of the SPINE Node are:

- the Serial Peripheral Interface (SPI), which handles the communication between the ReISC SoC and the RF Module;
- the General Purpose Input/Output (GPIO);
- the timers, which are used for synchronization purposes;
- the Analog-to-Digital Converter (ADC), which collects samples from the sensors;
- the Reset and Clock Control Unit (RCCU);
- the Power Manager, which is used to manage the power states of the ReISC SoC (CPU and peripherals).

The Serial Peripheral Interface (SPI) consists of a synchronous serial communication interface with a 4-pin protocol. It allows half/full-duplex, synchronous, serial communication with external devices. There are separate buffers for reception and transmission and the peripheral can operate in full-duplex mode. When the interface is configured as master it provides the communication clock to the external slave device. The interface is also capable of operating in multi-master configuration. It may be used for a variety of purposes, including simplex synchronous transfers on two lines with a possible bidirectional data line or reliable communication with CRC checking.

The General Purpose Input/Output (GPIO) is a set of pins whose behavior can be programmed through software.

There are up to four timers in the ReISC Soc platform, which may be used for a variety of purposes, including measuring the pulse length of input signals (input capture) or generating output waveforms or counting events. The internal timer is the simplest one, having only simple down-counting functionality.

The Analog-to-Digital Converter (ADC) converts a continuous signal (voltage or current) into a sequence of numbers proportional to the magnitude of the voltage or current. Its main feature is resolution, i.e., the number of discrete values it can produce over the range of analog values. ReISC ADC provides 16 multiplexed channels, with 12 bit resolution, interrupt generation at the end of conversion, and DMA request generation during conversion.

The Reset and Clock Control Unit (RCCU) manages the power-on reset for the ReISC SoC system, as well as generating the system clocks via PLLs. It allows also to enable/disable the peripherals and their clocks.

The Power Manager activates clock gating for each individual peripheral which is not used, selects the power-down state for analog hard macro when they are not required by the application, and selects the appropriate system clock frequency. In order to reduce power consumption to the minimum value, the core can also shut off the power domains for each ADC PLL and for the FLASH memory.

3.2 RF Module

The RF Module is connected through an SPI interface to the ReISC SoC, in order to provide its networking capability. The role and architecture of the RF Module inside the SPINE Node is presented in Figure 4. It consists of three components: the front-end, the network processor and the back-end.

The front-end component manages the SPI protocol to communicate with the ReISC SoC. It receives data from the

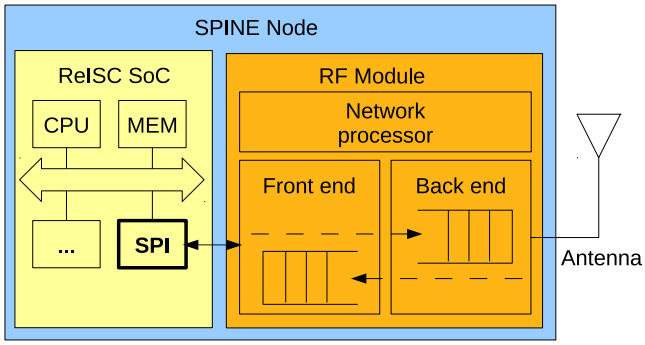

Figure 4: Architecture of the RF Module and interconnection with the ReISC SoC.

ReISC SoC to be sent to the network and those from the network, through the back-end component, to be sent to the ReISC SoC.

The SPI transmission uses simplex mode. The decision about setting the SPI controller in master or slave mode is made by the software running on the ReISC SoC. When the ReISC SoC SPI controller is in slave mode, it waits to receive data from the RF Module. When it enters the master mode it cannot be interrupted by the RF Module. Therefore, the RF module uses a local FIFO queue to temporally store the data to be transmitted to the ReISC SoC.

The network processor implements the medium access control according to the IEEE 802.15.4 [4] standard.

The back-end component addresses all the low-level details to send/receive bits on the radio channel. The transmission time of a packet is not constant since the 802.15.4 protocol provides a statistical access approach based on random wait with multiple re-transmissions in case of failure; therefore, the back-end component also contains a FIFO queue to store packets when the production rate of the ReISC SoC is higher than the transmission rate over the air.

3.3 Power consumption

To improve power consumption, the ReISC SoC can apply different power reduction techniques such as clock gating and power gating. The architecture is in fact hierarchically organized in power islands that can be switched off under the control of the power manager; finer control on the consumption can also be obtained by the RCCU that allows to set the enabling status of the peripherals and to enable or disable their clock.

The organization of the power islands is summarized here:

- An ALWAYS ON power island includes the ReISC core, the RCCU, the Power Manager, Timers, all the other components that must be kept always enabled.
- A FUNCTIONAL STATE power island (with retention flip-flops) contains the other peripherals that can be switched on/off, e.g., the SPI and the GPIOs.
- An ANALOG power island includes the ADC.

Table 1 reports the power consumption values of SPI and ADC peripherals as a function of the power mode.

Table 1: Power consumption values (in μW) of SPI and ADC peripherals as a function of the power mode.

Power mode	SPI	ADC
OFF	0	0
NO CLOCK	20	n/a
IDLE	40	30
WORKING	100	150

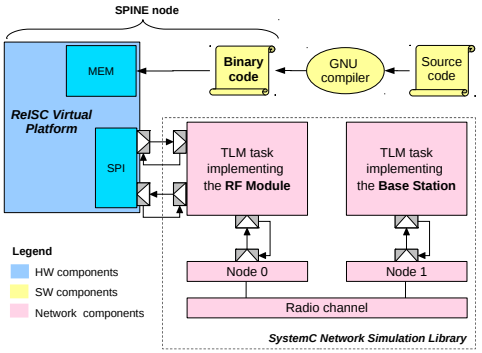

Figure 5: Modeling approach.

4. MODELING APPROACH

Figure 5 presents the whole simulation scenario, modeled completely in SystemC. The model of the SPINE Node consists of the ReISC SoC virtual platform provided by STMicroelectronics and the model of the RF Module. The former is an accurate model of the hardware components of the ReISC SoC, while the latter is modeled at the pure functional level by using the blocks provided by the SystemC Network Simulation Library (SCNSL) [2]. Software components (i.e., application and system code) are compiled for the target platform and then loaded into the memory module of the virtual platform to be executed by the CPU module. The network components, modeled by using SCNSL, consist of:

- an SCNSL task to represent the functional model of the RF Module of the SPINE Node;
- an SCNSL task to represent the functional model of the SPINE base station;
- an SCNSL task to represent the functional model of a source of concurrent traffic;
- the hosting nodes of the three previous tasks;
- the radio channel.

The simulation is performed at the TLM level; the ReISC SoC virtual platform is connected to the RF Module through two TLM sockets and each SCNSL task is connected to the corresponding SCNSL node by using a TLM socket.

It is worth remembering that we aim at optimizing the power consumption of the ReISC SoC and therefore we are

interested in modeling accurately its hardware components and the software components while transmission components are not part of the design goal.

4.1 ReISC Virtual Platform

The ReISC SoC platform is provided with a virtual platform simulation framework, shown in Figure 6. It consists of an ISS of the ReISC 3 processor which communicates with the hardware models of the peripherals through a bus model. A SystemC wrapper implements the interface among the instruction-set simulator (ISS) and the rest of the system, i.e. the peripherals, that are mostly modeled in SystemC. Only the components that are more closely linked to the ISS or to the memory, have been left under the direct control of the ISS, rather than in SystemC. In Figure 6 the SystemC peripherals are shown in orange, while the parts in yellow are modeled in C within the ISS.

4.2 Network simulation

The simulation of the health care application involves three network nodes:

- one Base Station (BS), with two main purposes:
  1. to broadcast configuration packets to the SPINE Node at initialization time;
  2. to listen for incoming data packets from SPINE Node;
- SPINE Node with three main purposes:
  1. receive and process the configuration packets from the BS;
  2. perform the configured computations;
  3. transmit the sample data to the BS;
- A traffic generation node which broadcasts packets on the shared channel by using the same channel access protocol of the first two nodes (i.e., IEEE 802.15.4). Since the channel models collisions, this node is used to evaluate the effect of concurrent traffic of the communication between the SPINE Node and the Base Station.

Figure 7 represents the network scenario described in SystemC by using SCNSL primitives to create the nodes (lines 12–14), the channel (line 17) and the tasks (lines 23–25). Node instances are also connected to the shared radio channel (lines 19–21).

In both BS and SPINE Node models, the interface to the channel is implemented by inheriting the SCNSL `Tlm-Task_if_t` interface and by using the corresponding transmit and receive methods, i.e., `Send()` and `b_transport()`, respectively.

Figure 8 shows the architecture of the `Base_Station_t` class. It includes two main member functions, namely `send-ConfigPacket` and `receiveDataPacket`. The model of the BS includes a SystemC `SC_THREAD` which periodically calls `sendConfigPacket` with an application-specific sequence of node configuration packets for the SPINE Node.

4.3 Simulation of the software components

The simulation of the application is performed by running the system software and the SPINE code on the ISS of the

```
01    #include <systemc>
02    #include <tlm.h>
03    #include <scnsl.hh>
04    #include "Spine_t.hh"
05    #include "Base_Station_t.hh"
06    int sc_main( int argc, char * argv[] )
07    {
08      ...
09      Scnsl::Setup::Scnsl_t * scnsl =
                    Scnsl::Setup::Scnsl_t::get_instance();
10
11      // Creation of nodes:
12      Scnsl::Core::Node_t * n0 = scnsl->createNode();
13      Scnsl::Core::Node_t * n1 = scnsl->createNode();
14
15      // Creation of the wireless channel:
16      ccs.channel_type = CoreChannelSetup_t::SHARED;
17      Scnsl::Core::Channel_if_t * ch =
                    scnsl->createChannel( ccs );
18
19      // node-channel binding:
20      scnsl->bind( n0, ch, bsb0 );
21      scnsl->bind( n1, ch, bsb1 );
22
23      // Mapping of task instances on to nodes:
24      Spine_t t0( "SPINE", n0);
25      Base_Station_t t1( "BS", n1);
26      ...
```

Figure 7: Network scenario described in SystemC by using SCNSL primitives.

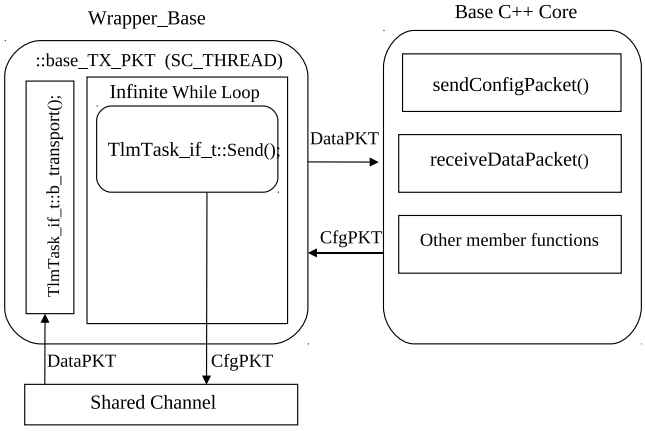

Figure 8: Architecture of the SystemC model of the Base Station.

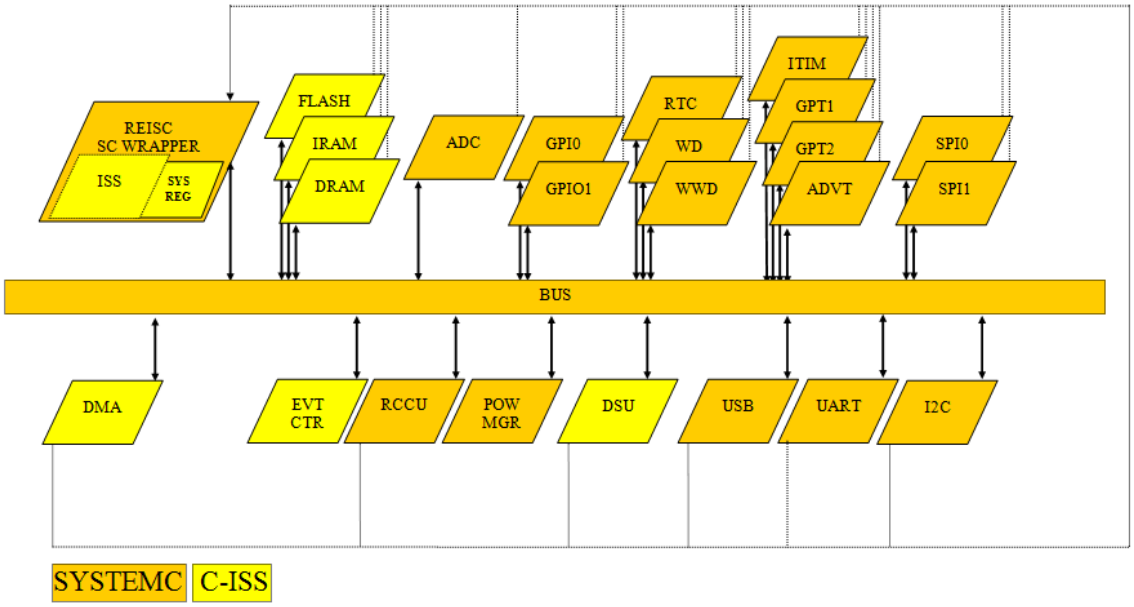

Figure 6: Architecture of the ReISC SoC virtual platform.

ReISC SoC virtual platform. In this experiment, the ReISC SoC is running at 50 MHz, with a timer configured to generate a periodic event for the SPINE application. After the initialization phase, the SPINE Node listens to receive four configuration 16-byte packets from the BS. The BS is configured to send packets every 200 ms. After receiving each packet, the radio sends it to the SPI controller which is configured to work in slave mode to raise an interrupt. The SPI interrupt handler copies the payload of the received packet into the packet buffer of the SPINE, application allowing the node to be customized and configured properly.

Upon receiving the *StartSpine* command from the BS, the SPINE Node engine starts by setting the SPI controller in master mode and initializing the timer to schedule a periodic task using the packet transmission interval defined at configuration time by the BS. Then the ADC is activated and configured to sample acceleration data over three channels. At every sampled value, the ADC interrupt is raised and the value is placed in the SPINE circular buffer. A window of samples is then processed and filtered and the result is sent to the SPI controller which sends it to the RF Module.

Note that the accelerometer sampling rate and the packet transmission rate are independent. The former can be much higher than the latter, in order to save radio power, possibly at the expense of the quality of fall detection.

5. CASE STUDY

We simulated 2 seconds of the application scenario:

- from 0 s to 0.4 s the Base Station sends four configuration packets;
- from 0.4 s to 2.0 s the SPINE Node sends sample packets to the Base Station;
- from 1.0 s to 2.0 s a third node transmits concurrent traffic which tries to saturate the channel (such concurrent traffic could represent the resulting effect of several SPINE Nodes of the WBSN).

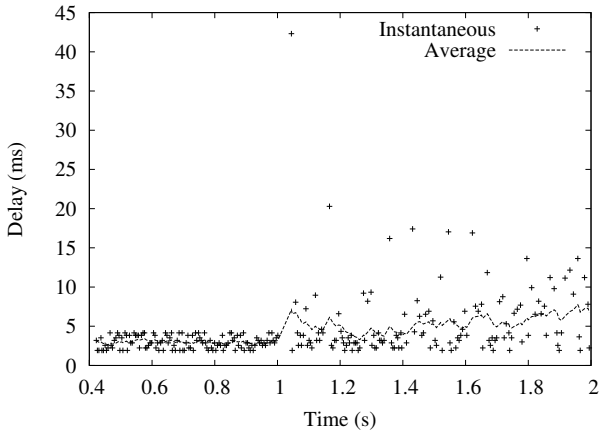

Figure 9: Instantaneous and average time spent by the SPINE Node to send a packet on the channel as a function of simulation time.

Figure 9 shows the instantaneous and average time spent by the SPINE Node to send sample packets on the channel as a function of simulation time. The value ranges from 1.9 ms to 42 ms according to the congestion level of the channel; the higher busy time during congestions is due to backoff retries and retransmissions. In this test case, the trasmission delay causes the loss of 76 packets during congested periods since the production rate of the ReISC SoC is higher that the transmission rate of the RF Module so that its output queue becomes full.

This result suggests that in some cases the optimal output rate is not determined only by application constraints but also by channel condition and the knowledge of channel

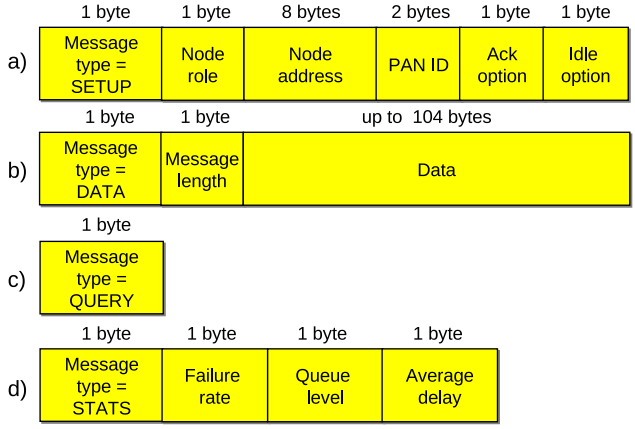

Figure 10: SPI message structure between the ReISC SoC and the RF Module: configuration command (a), data message (b), statistics request (c), and statistics response (d).

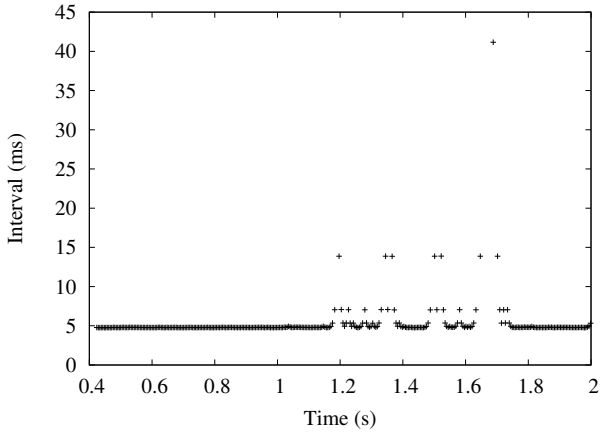

Figure 11: Time interval between two SPI data messages as a function of simulation time.

statistics can allow to improve the scheduling of node activities thus decreasing energy consumption. In this section we assume that the application requirements can be satisfied with a range of network transmission delays, and examine how an adaptive packet transmission scheduling strategy can reduce contention on the channel and reduce overall energy. In particular, we observed that transmission delays of up to 50 ms do not have a noticeable effect on the quality of fall detection (percentage of correct detection over percentage of false positives and false negatives). In order to use the knowledge of channel statistics to improve energy consumption, we need both a way to communicate network statistics to the ReISC SoC and an algorithm to adapt the scheduling of activities.

The SPI channel is not only used to send and receive data on the radio channel but also to configure the RF Module and to ask for transmission statistics. For this purpose, the data transmitted on the SPI channel are organized in messages structured as depicted in Figure 10. The first byte of the message determines its type. The first message is sent by the application to the RF Module to configure 802.15.4 parameters, i.e., node role (coordinator, router, end device), 64-bit MAC address, network identifier, acknowledge transmission, and behavior when it is not transmitting (idle or listening). The second message is used to carry packets to/from the network. The third message is sent by the application to the RF Module to ask for statistics about packets transmission. The fourth message is the reply which reports the number of transmission failures over 255 requests, the number of packets waiting in the output queue, and the average delay to complete a transmission (the metric shown in Figure 9. Failures happen when either the channel is found busy after a maximum number of attempts or when the maximum number of re-transmissions is reached. All these statistics can be used to estimate the channel condition.

Regarding the adaptive scheduling, in this work we assume to delay the transmission of a data message by the ReISC SoC according to the value of the variable named *Delay* which changes its value based on the following algorithm:

Table 2: Transmission performance and energy consumption of SPI and ADC peripherals as a function of the transmission policy.

Transmission policy	TX attempts	Received packets	PLR	Energy (μJ)
Non-adaptive	332	256	23%	400
k=1	300	233	22%	369
k=4	216	189	12%	297
k=5	205	182	11%	287
k=6	186	173	7%	271

1. Initialize *Delay* to 1 ms.
2. If the last SPI transmission failed for buffer overflow then $Delay = Delay * 2^k$ (up to a maximum, in order to preserve application functionality).
3. If the last SPI transmission was successful then $Delay = Delay/2$.

Figure 11 depicts the time interval between two SPI data messages as a function of simulation time by using the adaptive transmission scheduling with k=1. The plot shows that during channel congestion the SPINE Node sends less data.

Table 2 reports transmission performance and energy consumption of SPI and ADC peripherals as a function of the transmission policy. The second column reports the number of transmission attempts made by the SPI controller of the ReISC SoC. The third column reports the number of sample packets actually received by the Base Station. The fourth column reports the resulting packet loss rate (PLR). The fifth column reports the energy consumption of the ADC and SPI peripherals (the only components affected by the delay since the ReISC core is always running at 50 MHz). In fact, for each transmission attempt they are both working for 4.7 ms and idle until the next transmission attempt (power values are taken from Table 1). The results show that trasmission reliability increases by using a more aggressive adaptive policy while energy is saved by avoiding to send packets with a high loss probability.

Each scenario has been simulated in about 590 s on a 3 GHz Intel processor running Linus operating system.

6. CONCLUSIONS

We have presented a holistic approach to the simulation of an application based on networked embedded system. The simulation scenario has been written in SystemC/TLM and included a detailed representation of a system-on-chip running actual software components and connected to a functional representation of a wireless channel with interacting nodes. This virtual platform has been used to evaluate different application design solutions to optimize energy consumption of the SoC. In particular we adapted the transmission rate of a sensor node according to the congestion level of the channel due to the presence of concurrent traffic. The simulation approach allowed a fast exploration of different adaptation policies which increased the transmission reliability and reduced the energy consumption.

7. ACKNOWLEDGMENTS

This work has been partially supported by the COMPLEX FP7 European Integrated Project,funded by the European Commission under Grant Agreement 247999.

8. REFERENCES

[1] M. Chung and C.-M. Kyung. Enhancing performance of HW/SW cosimulation and coemulation by reducing communication overhead. *IEEE Transactions on Computers*, 55(2):125–136, Feb. 2006.

[2] SystemC Network Simulation Library – version 1, 2008. URL: http://sourceforge.net/projects/scnsl.

[3] European Commission. COdesign and power Management in PLatform-based design space EXploration - COMPLEX. *URL: https://complex.offis.de/*, (FP7-IST-247999), 2009.

[4] LAN/MAN Standards Committee of the IEEE Computer Society. IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs). Sept. 2006.

[5] S. McCanne and S. Floyd. NS Network Simulator – version 2. *URL: http://www.isi.edu/nsnam/ns*.

[6] F. Mulas, A. Acquaviva, S. Carta, G. Fenu, D. Quaglia, and F. F. Network-adaptive management of computation energy in wireless sensor networks. In *Proc. of ACM Symposium on Applied Computing (SAC)*, pages 756–763, Mar. 2010.

[7] OSCI and IEEE. IEEE Std 1666 - 2005 IEEE Standard SystemC Language Reference Manual. *IEEE Std 1666-2005*, pages 1–423, 2006.

[8] Signal Processing In Node Environment. SPINE home page. `http://spine.tilab.com/`.