Investigating Power-Reduction for a Reconfigurable Sensor Interface

Johann Glaser, Jan Haase, Markus Damm, Christoph Grimm
Vienna University of Technology. Institute of Computer Technology
{glaser,haase,damm,grimm}@ict.tuwien.ac.at

Abstract

The power consumption of different implementations of a sensor interface utilized for wireless sensor network nodes are investigated. Most sensor network nodes use a CPU to implement the main functionality in software. This maintains flexibility compared with a pure ASIC implementation. Unfortunately, to utilize the CPU for simple tasks as periodically controlling the data acquisition proved expensive in terms of energy consumption due to the wakeup-overhead and several waiting periods.

As an alternative a special reconfigurable peripheral block is introduced into the WSN SoC. It is responsible for these measurements while the CPU stays in an inactive "sleep mode". It is only activated if further processing is required (e.g. transmission of a notification packet via the wireless network if the measurement value has changed). A huge reduction of power consumption by a factor of 30 was achieved. The objective is to reduce power consumption while maintaining a high degree of flexibility for the implemented applications.[1]

1 Introduction

The emerging technology of wireless sensor networks (WSN) is applied in numerous applications, including building automation, automotive systems, container tracking, and geological surveillance. Typical implementations of sensor nodes comprise a central processing unit, memory, an RF transceiver, sensors, power supply and a software stack for both sensor control and network processing [7] as shown in Fig. 1 (except for the shaded "Sensor Interface" block).

Wireless sensor nodes obtain their energy either by energy scavenging (e.g., solar cells) or from a battery. In both cases, the power consumption of the node must be extremely low. Current energy scavengers only supply a low amount of power, while the capacity of batteries is limited, directly affecting the life time of the whole node.

So, a common challenge in virtually all application fields is the requirement for low energy consumption. This is especially difficult due to the high power consumption of the RF transceiver, sensors and CPU. Several techniques were proposed to tackle this. One option is to limit the activity of the node to a portion of the time (duty-cycling, [11]). Another approach is to integrate most components into a single chip (ASIC), because sensor nodes built with multiple individual commercially available components waste a lot of energy in voltage level adoptions (voltage matching problem, [7]).

This work examines a different problem: A typical task of a sensor node is the periodic wakeup for measurement of the sensor values. Therefore the CPU is activated from its sleep mode. Then the power supply of an external sensor is turned on. After some settling time the sensor value is ascertained by an analog/digital conversion (ADC). The resulting value is finally compared to the previous value and only if the difference is greater than a certain threshold, further processing is conducted, like sending the value via the wireless network. In all other cases the CPU immediately switches back to an inactive low-power mode.

This cycle consists of several simple tasks (set and read digital IOs, compare two integers) and delays (sensor settling time, ADC conversion). Even these are performed by the CPU, which is overqualified and oversized for these tasks. It consumes power during the whole active state as well as for the wakeup procedure.

Many commercial microcontrollers strive to reduce the overhead for waking up from the inactive low-power mode. However, our investigations revealed that even the energy consumption of the wakeup alone is still in the same order as realizing the whole sensor measurement with a specialized ASIC core.

We propose to avoid this waste of energy for some of these periodic tasks by introducing novel reconfigurable hardware blocks into the WSN node SoC (system-on-chip) (see Fig. 1, note the shaded "Sensor Interface"

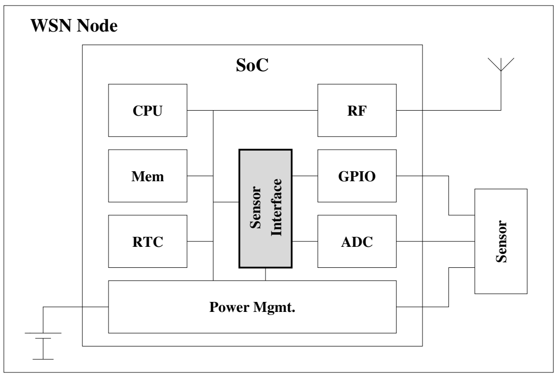

Figure 1. WSN node with reconfigurable hardware block as Sensor Interface

[1]This work is conducted as part of the Sensor Network Optimization through 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).

block). These take over some of the simple and periodic tasks from the firmware. This allows the CPU to stay in an inactive sleep mode for extended periods and only wakes up for more complex tasks.

The contribution of this work is to examine several ultra-low-power microcontrollers for the application of sensor measurements of a WSN node. The main goal is to compare these with a novel reconfigurable hardware architecture which is specifically designed to relieve the CPU from the periodic wakeups, thus reducing energy consumption.

In the next Section several approaches to assist the CPU in a SoC are compared. This is followed by a description of the proposed approach. Then the actual implementations of a sensor interface for a WSN node are described, followed by the power consumption measurement results. This paper is finished by a conclusion and outlook to future work.

2 Related Work

A processor specifically for WSN nodes with an asynchronous design was presented in [4]. It implements a task queue and is seconded by a timer and a message co-processor where the latter communicate with external sensors and the radio. While this design has an extremely low energy consumption ($\approx$24 pJ per instruction), the main processor core is still active during sensor acquisitions or wireless communication.

The reconfigurable analog and digital blocks introduced by [9] are designed to implement peripheral functions like timers, serial communication, PWMs and so on. These are not specifically for power reduction or to relieve the CPU from simple tasks but even require frequent CPU intervention, so this approach does not consider power reduction as proposed here.

As a true assistant to the CPU the commercially sold microcontroller ADuC70xx [2] includes programmable logic arrays with 16 PLA elements. These have a two-input look-up table, a flip-flop and numerous input multiplexers. This allows to implement flexible logic operations, but is limited in cell count.

Even more flexibility is provided by the embedded FPGA in [5] for the control of IO ports and protocols. The RISC CPU core of the high-performance SoC can be extended by the “Pipelined Configurable Gate Array (PiCoGA)” where application-specific functions can be mapped ([5, p. 87]). While this approach is very flexible, it is clearly designed for high-performance embedded systems and does not concentrate on low-power design.

The standalone approach for soft embedded programmable logic cores presented by [13] increased the efficiency by using a non-regular architecture, but synthesis tools get into difficulties then. In later work the overhead of standard cells was reduced by the introduction of architecture specific tactical cells [1]. This work is a perfect starting point for a logic core inserted between a sensor interface and the CPU. We propose to improve their approach by designing the inserted logic cores in an application specific manner for more power optimization potential.

3 Reconfigurable Architecture

The reason for energy gaps on the node level can be partially attributed to the architecture. Typical sensor node architectures require that the MCU is active for almost every single task. This approach is sensible in terms of versatility, such that all kind of applications, with their own sensing functions and communication protocols, are supported. However, in terms of energy efficiency, waking up the MCU and the associated subsystems is very costly. While a specialized chip would provide higher energy efficiency, the lack of flexibility would increase production costs since less chips could be produced.

The most widely used energy-saving concept at node system level is to switch components to an inactive low-power state when possible. In the course of the PAWiS project [8] it was revealed that many simple control and processing tasks are still accomplished by microprocessors as software programs, which is a main source of energy waste.

To tackle this energy waste problem, we propose novel *reconfigurable hardware blocks* introduced to a WSN SoC which independently conduct simple sub-tasks instead of the CPU [6]. The CPU is only activated if any further (more complex) processing is required (see Fig. 1). Therefore these logic blocks act as a “filter” for these events.

3.1 Using Reconfiguration

Shifting the border in a software/hardware partitioning process towards hardware reduces the flexibility of the final application. While software can be modified by reprogramming the code memory, synthesized logic cores require a redesign of the chip.

There are three important reasons for flexibility:

1. Covering multiple different applications for a higher market potential.
2. Adopting to different external components across PCB design cycles.
3. Fixing bugs without a chip redesign.

Therefore we propose to make the introduced dedicated hardware blocks *reconfigurable*.

In contrast to FPGA cores (e.g., [13]), which consist of fine-grained structures optimal for control dominated functions, we propose to also support multi-bit logic blocks and a multi-bit routing architecture for computational functions. To further reduce the power consumption and area requirement, the proposed approach requires the user to define an *application class* for which the reconfigurable block is inserted. This class describes the field of planned actual applications. Then the reconfigurable logic block is developed to be tailored to provide exactly those structures which are required to implement any of the desired applications. After the manufacturing of the SoC, the actual application is specified and implemented by configuring the reconfigurable block accordingly.

4 Implementation

The sensor interface application mentioned above was implemented for several microcontrollers, as a hard-coded hardware block, and with the novel reconfigurable hardware architecture.

4.1 Microcontroller Implementation

The power consumption of five different microcontrollers with 8- or 16-bit RISC CPUs was investigated (see Tab. 1). The selection criterion was that the respective manufacturers highlight the low power consumption of these chips.

The sensor control is implemented as a C function. The same C function was used and compiled for every chip using an appropriate compiler (see Tab. 1), with the only difference being an `#include` statement to define the appropriate special function registers and in the IO port handling. The execution time was separated into wait periods (sensor settling and ADC conversion), wakeup delay, interrupt latency, context save and restore, the actual function and return from interrupt. The PIC16LF72x also requires to determine the interrupt source because it has only one common interrupt routine.

For all implementations an RC oscillator was assumed due to its fast startup time. A crystal oscillator or a PLL would require several hundred to thousands of cycles to stabilize its frequency and is therefore not appropriate for the periodic wakeup tasks. Every CPU executed the instructions with $f = 4\,\text{MHz}$.

The execution times were calculated from counting the assembler instruction execution cycles and dividing by the operating frequency. The waiting period until the sensor output has settled was assumed as $10\,\mu$s. The ADC conversion time was assumed as $4.5\,\mu$s for all MSP430, $10\,\mu$s for the PIC16LF72x and $25\,\mu$s for the ATmega88PA. The power consumption was estimated by using typical data sheet values at $V_{\text{CC}} = 3\,\text{V}$ and $T = 25\,°\text{C}$.

4.2 Hardware Implementation

In this Section the final SoC architecture is sketched. The modifications for the actual implementations are described in the next Section (4.3). The functionality of the sensor interface was already sketched in Sec. 1. It is embedded in an SoC together with a CPU, ADC, memory and so on (see Fig. 1). It has interfaces to the sensor (via pins), ADC (on-chip) and the CPU (on-chip). Additionally, there are clock and reset signals, and the configuration and parameterization interfaces (see Fig. 2).

The data path consists of a register to store the ADC value, an arithmetic cell which calculates the absolute difference of the ADC value A and the previously stored value B, and a numerical comparator. Additionally, there is a counter with a pre-settable start value for the periodic wakeup cycle. Everything is controlled by a finite state machine (FSM) which generates two output signals to the sensor, the start signal for the ADC, the enable signal for the value register, and an interrupt signal for the CPU. Its inputs are the sensor's and the ADC's ready signal, the comparator, the counter and an enable-signal from the CPU.

The counter start value and the comparison value for the sensor value difference are parameterized as memory mapped registers from the CPU. The FSM is implemented as a look-up table with a register to store the current state. The look-up table is implemented as a RAM. Its content is setup via the serial configuration interface, which requires a wrapper to parallelize the data and generate the address and write signals.

4.3 FPGA Test Environment

Instead of the production of a chip, the circuit was realized with an FPGA. Note the stack of reconfigurability, where the underlying technology (FPGA) is reconfigurable and the implemented circuit itself is reconfigurable too. Due to the usage of a commercially available FPGA instead of a full- or semi-custom chip design, no optimized multi-bit cells were implemented.

The tests were performed with a Xilinx Virtex 4 XC4VFX20 FFG672 (speed grade 11) FPGA on a Xilinx ML405/6 RevC evaluation board. The current was measured as voltage drop across the $3\,\text{m}\Omega$ shunt resistors of the supply regulators for $1.2\,\text{V}$ (R182), $2.5\,\text{V}$ (R181) and

Table 1. Evaluated Microcontrollers.

Manufacturer	Microcontroller	Architecture	Compiler	Data Sheet
TI	MSP430F1232	16 Bit RISC	`msp-gcc` 3.2.3[a]	[12]
TI	MSP430F2232	16 Bit RISC	`msp-gcc`	[12]
TI	MSP430F5418	16 Bit RISC	`msp-gcc`	[12]
Microchip	PIC16LF72x	8 Bit RISC	`sdcc` 2.8.0[b]	[10]
Atmel	ATmega88PA	8 Bit RISC	`avr-gcc` 4.3.3[c]	[3]

[a] `http://mspgcc.sourceforge.net/`
[b] `http://sdcc.sourceforge.net/`
[c] `http://gcc.gnu.org/`

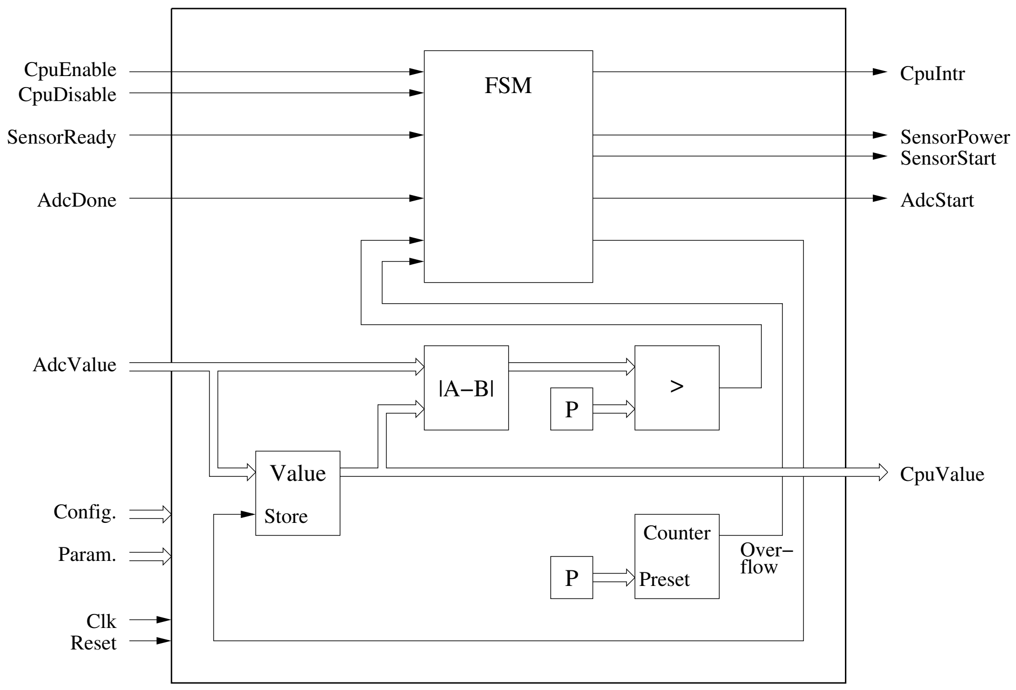

Figure 2. Sensor interface implementation: finite state machine with inputs and outputs and connections to the data path. The small boxes labeled with "P" are parameterization values.

3.3 V (R180). Only the core voltage (1.2 V) showed significant values. Therefore the other two supplies were not considered for the energy budget.

Some simplifications were introduced for the execution. Instead of implementing a whole sensor node including a microcontroller, only a test bench was instantiated to setup the parameterization and configuration. The only reconfigurable parts of the sensor example are the state machine and the parameterization values. Due to the early stage of this research project it does not have reconfigurable routing resources. In the hard-coded state-machine implementation, no reconfigurable parts were realized. Sensor and ADC are modeled with a digital signal providing a 16-bit number which increases and decreases periodically and non-linearly to simulate small and large changes of the measurement value.

Measuring the current of a specific circuit inside an FPGA is disturbed by the other circuits and the static current of the FPGA. Therefore the sensor interface was instantiated twice. One instance, the reference instance, was permanently operated to generate the proper control signals for the peripherals (sensor, ADC). The second instance, the device under test (DUT), was switched on or off. More precisely, its clock and its inputs were active or set to "0". The current consumption is then calculated as the difference of the current consumption with active and with inactive DUT. Actually the DUT was instantiated multiple times in parallel to achieve a higher difference in current consumption, which eases the measurement. The used clock frequency of 100 MHz is much higher than in any actual sensor node, but was also used to gain a larger increment in current consumption. For these measurements no division into parts was performed, because the hardware implementation only performs the pure function and does not need to wakeup or save and restore the context.

5 Experimental Results

The energy consumption values of a single sensor measurement performed by the different implementations are summarized in Tab. 2 and Fig. 3. While the microcontroller implementations only differ slightly (189.15 nJ to 266.22 nJ with an average of 219.08 nJ and a range of 35.2 %), the difference to the hardware implementations is tremendous (2.09 nJ and 5.61 nJ).

Implementation	Active	Sleep	Energy
MSP430F1232	1.21 mA	1.60 μA	191.48 nJ
MSP430F2232	1.53 mA	0.60 μA	222.63 nJ
MSP430F5418	1.30 mA	2.60 μA	189.15 nJ
PIC16LF72x	1.36 mA	1.80 μA	266.22 nJ
ATmega88PA	1.20 mA	0.90 μA	225.90 nJ
FPGA: Hardc.	n.a.	n.a.	2.09 nJ
FPGA: Reconf.	n.a.	n.a.	5.61 nJ

Table 2. Energy consumption of a sensor measurement performed by different implementations.

Compared to the lowest power microcontroller MSP430F5418, the hardcoded state machine implementation in the FPGA requires 90.5 times less energy. The two hardware implementations differ by a factor of ≈2.7 with the reconfigurable implementation requiring

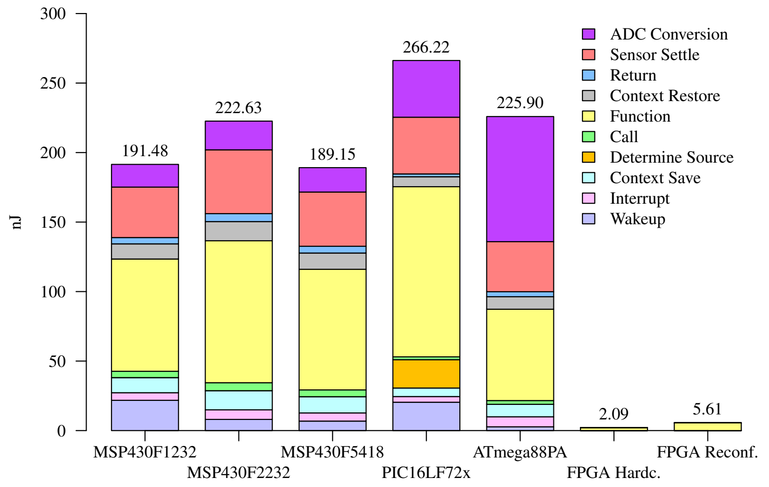

Figure 3. Energy consumption of one sensor measurement performed by different implementations.

more energy. But even this requires 33.7 times less energy than the MSP430F5418.

The two FPGA implementations only differ in the state machine type. In the reconfigurable implementation a Block RAM is used while the other one uses a hard-coded state machine. The difference in current consumption is therefore mostly caused by the Block RAM which offers several different configurations, layouts, and wrappers which adds overhead in power consumption. For a custom SoC the lookup table must be designed specifically to provide the required features, so we expect a lower difference for an implementation in real hardware.

Besides the current consumption overhead of the Block RAM both FPGA implementations comprise overhead due to the fine granularity of the logic functions for the data-path oriented tasks (absolute difference, comparison, counters) as well as for the routing on the FPGA chip.

The major difficulty of the comparison is the difference in semiconductor process by the FPGA and microcontroller. While the FPGA is a high-performance chip produced in a 90 nm CMOS process [14], the microcontrollers are produced in an ultra-low-power semiconductor process. A fair comparison would require to implement both, a low-power microcontroller core and the sensor interface with the proposed approach, on a common chip. However, we expect a large decrease of the total energy per sensor measurement for the hardware implementation because of the high overhead with the FPGAs. On the other hand, the firmware implementation will not experience such a large reduction because we already used optimized microcontrollers in this investigation. So the difference between these two implementations will further increase, which strengthens the importance of the proposed approach.

For example the MSP430F1232 draws 1.2 mA when active, even when waiting for the completion of the ADC conversion or until the sensor output has settle. These waiting periods alone consume a total 52 nJ of energy, which is nearly ten times the energy required by the reconfigurable FPGA implementation for the whole measurement cycle.

Finally, we investigated the power consumption of the MSP430F1232 depending on its operating frequency (see Fig. 4). For frequencies below 1 MHz the total energy consumption for a single measurement cycle is ≈ 110 nJ and does not depend on the frequency. Decreasing the frequency reduces the active current nearly proportionally, while the execution time of the task increases proportionally. Therefore both effects cancel each other out. Indeed, the energy budget of the function itself is nearly constant even at higher frequencies (≈ 80 nJ) due to this effect. Only the contributors with constant duration (Wakeup, Sensor Settle and ADC Conversion) are proportional to the operating current and thus to the frequency. This also shows that reducing the operating frequency below 1 MHz does not improve the total energy consumption. Using an operating frequency of 32 kHz even requires 192 nJ while 4 kHz increase the energy consumption to 864 nJ since the quiescent current dominates.

6 Conclusion

This paper presented a survey of the power consumption of a sensor interface of a wireless sensor network node in seven different implementations. Each assumes a microcontroller as main unit of the node. Five of the variations used firmware code on different microcontrollers to control the sensor measurements. Two other implementations relieved the microcontroller from this task by introducing special peripheral control blocks, one as hard-coded hardware block, the other one with a novel reconfigurable hardware architecture.

While the energy consumption of the firmware implementations only varies by 35.2 %, they differ by a factor of 90.5 from the hard-coded hardware block. The novel reconfigurable architecture requires ≈ 2.7 times more energy than this, but this is still 33.7 times less than the most efficient microcontroller implementation.

This proves the positive effect of the approach for power

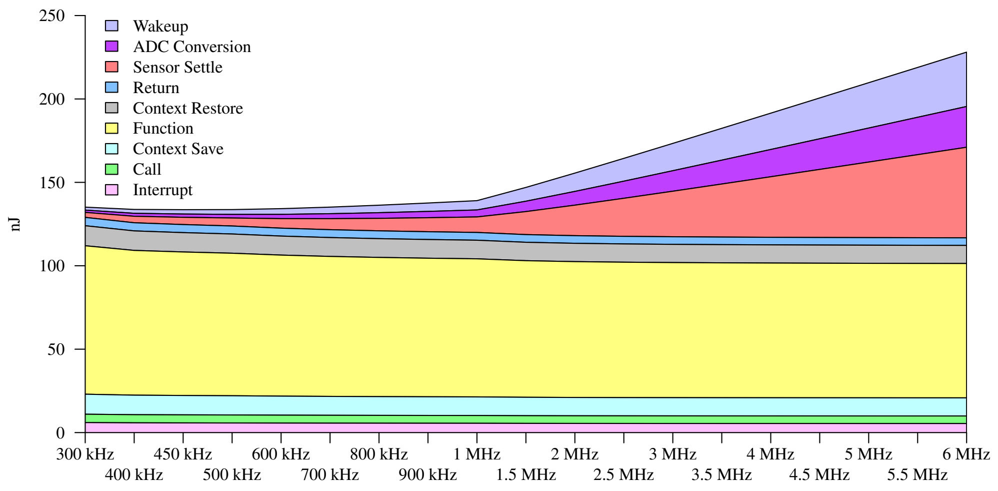

Figure 4. Frequency dependence of the energy consumption of one sensor measurement performed with the MSP430F1232.

saving in wireless sensor network nodes, namely accompanying the CPU with additional reconfigurable hardware blocks that take over simple tasks from the CPU to release it from frequent wakeups.

Future work includes research on reconfigurable routing, development of multi-bit cells and a tool chain to ease the implementation of a reconfigurable hardware block.

References

[1] V. Aken'Ova, G. Lemieux, and R. Saleh. An Improved "Soft" eFPGA Design and Implementation Strategy. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*, 2005.

[2] Analog Devices. *ADuC70xx Precision Analog Microcontroller*, 2006. Rev. A.

[3] Atmel Corporation. *8-bit AVR Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash: ATmega48PA, ATmega88PA, ATmega168PA, ATmega328P*, May 2009.

[4] V. Ekanayake, C. Kelly, and R. Manohar. An Ultra Low-Power Processor For Sensor Networks. In *ACM SIGOPS Operating Systems Review*, volume 38, pages 27–36, 2004.

[5] A. Lodi et.al. XiSystem: A XiRisc-Based SoC With Reconfigurable IO Module. *IEEE Journal of Solid-State Circuits*, 41(1), January 2006.

[6] J. Haase, M. Damm, J. Glaser, J. Moreno, and C. Grimm. SystemC-based Power Simulation of Wireless Sensor Networks. In *Proceedings of the Forum of Design Languages (FDL)*, Sophia Antipolis, 2009.

[7] 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. November 2005.

[8] S. Mahlknecht and M. Rötzer. Energy Supply Considerations for Self-sustaining Wireless Sensor Networks. In *IEEE Proceedings of the Second European Workshop on Wireless Sensor Networks*, pages 397 – 399, Istanbul, Turkey, 31. January– 2. February 2005.

[9] M. Mar, B. Sullam, and E. Blom. An Architecture for a Programmable Mixed-Signal Device. In *CICC*, pages 55–58, 2002.

[10] Microchip Technology Inc. *PIC16F72X/PIC16LF-72X Data Sheet*, March 2009.

[11] J. Rabaey, J. Ammer, B. Otis, F. Burghardt, Y.H. Chee, N. Pletcher, M. Sheets, and H Qin. Ultra-Low-Power Design – The Roadmap to Disappearing Electronics and Ambient Intelligence. *IEEE Circuits and Devices Magazine*, 22(4):23–29, July-August 2006.

[12] Texas Instruments. *MSP430x11x2, MSP430x12x2 Mixed Signal Microcontroller*, March 2003. SLAS361B.

[13] S. Wilton, N. Kafa, J. Wu, K. Bozman, V. Aken'Ova, and R. Saleh. Design Considerations for Soft Embedded Programmable Logic Cores. *IEEE Journal of Solid-State Circuits*, 40(2):485–497, February 2005.

[14] Xilinx, Inc. *Virtex-4 Family Overview*, v2.0 edition, 23 January 2007.