A hardware accelerated implementation of the IEEE 802.15.3 MAC protocol

Daniel Dietterle, Jean-Pierre Ebert, and Rolf Kraemer

IHP microelectronics GmbH, Wireless Communication Systems, PO Box 1466, 15204 Frankfurt (Oder), Germany
{dietterle, ebert, kraemer}@ihp-microelectronics.com

Abstract. We present a hardware/software implementation of the IEEE 802.15.3 MAC protocol. Processing-intensive and time-critical protocol tasks are handled by a protocol accelerator that is integrated on-chip with a 32-bit general-purpose processor in order to achieve a moderate (20–40 MHz) system clock frequency. This enables low-power wireless devices compliant with this standard, providing high data rate, multimedia communication.

One of the main tasks of the protocol accelerator is to analyze received or transmitted beacons. Based on the channel time allocations broadcast in the beacon and frame information stored in a hardware transmission queue, frames are transmitted without immediate control of the processor. Other features of the protocol accelerator include CRC generation, handling of immediate acknowledgment frames, and direct memory access.

Keywords: Personal area networks, protocol implementation, hardware accelerator.

1 Introduction

Wireless communication systems have attracted tremendous research work in industry and academia in the past decade. As the outcome of this research, new communication protocol standards, applications and products have been developed. In the future, we will see new applications in, for instance, personal health care, multimedia entertainment, or industrial automation making use of wireless body area network (BAN), personal area network (PAN) or mesh networking technology. These applications are characterized by a demand for high data rates, quality-of-service (QoS) provisioning, as well as energy efficiency. Low energy consumption is critical for networks of battery-powered devices with targeted network lifetimes of months and years.

In 2003, the IEEE has standardized a medium access control (MAC) and physical layer specification for high data rate wireless PANs, known as IEEE 802.15.3 [1]. Work on improvements of the MAC layer, alternative physical layers as well as a mesh networking extension has continued. We see this standard as a good candidate for future wireless applications and an enabling technology for low-power, wireless multimedia communications for a number of reasons:

Please use the following format when citing this chapter:

Dietterle, D., Ebert, J.-P., Kraemer, R., 2007, in IFIP International Federation for Information Processing, Volume 248, Wireless Sensor and Actor Networks, eds. L. Orozco-Barbosa, Olivares, T., Casado, R., Bermudez, A., (Boston: Springer), pp. 215-226.

The IEEE 802.15.3 MAC protocol offers an isochronous data service, supporting multimedia as well as industrial applications with requirements for guaranteed transmission opportunities. To save energy, devices may go into power-save mode. Synchronized power-save sets ensure that all devices in the set wake up at the same time. The used channel access scheme is time-division multiple access (TDMA). This way, the protocol can easily be used with ultra wide band (UWB) transceivers, that do not provide a channel sensing capability. Contention access based on channel sensing is optional. Furthermore, the specified high data rates from 11–55 Mbit/s reduce the transmission duration of frames. While high-rate transceivers typically consume more *power* than low-rate transceivers, still the amount of consumed *energy* per bit can be less than for low-rate transmissions. High-rate transmissions become more energy-efficient when messages can be aggregated to larger packets and the overhead for switching the radio transceiver from power-down to active mode is reduced.

High data rates on the physical layer and complex protocol functions, such as scheduling allocated time slots and managing power-save sets, require more processing power on the devices, especially for devices that are capable of acting as piconet coordinator (PNC). However, in many applications all networked devices must be battery-powered, like in a wireless network of sensors located on the human body. We advocate the use of protocol accelerators for processing-intensive and time-critical protocol tasks in order to achieve a moderate (20–40 MHz) system clock frequency. With this approach, the protocol functionality is partitioned into software executed on a general-purpose micro-controller and hardware integrated on-chip with the processor.

The remainder of this paper is organized as follows. Section 2 briefly introduces the IEEE 802.15.3 MAC protocol with a stress on timing-critical functionality. In Sect. 3, our protocol design flow—from an abstract SDL model until hardware/software partitioning—is presented. In the main part of the paper, Sect. 4, the design of the protocol accelerator for the IEEE 802.15.3 MAC protocol is explained. Finally, we present related work and conclusions.

2 IEEE 802.15.3 MAC Protocol

In a wireless network operating according to the IEEE 802.15.3 standard, there is one piconet coordinator (PNC) and a number of associated devices. The PNC broadcasts beacon frames at regular time intervals. These beacons contain, among others, information about the time of the next beacon and when other devices may access the channel in the same superframe, i.e. in the time until the next beacon.

A channel time allocation list, which is part of the beacon frame, announces the time slots that are reserved exclusively for the devices. Additionally, the PNC may define a contention access period (CAP) following immediately after the beacon. In this CAP, all devices can access the channel using a random backoff procedure. Data communication among the associated devices is peer-to-peer. An example of a piconet and a beacon frame is shown in Fig. 1.

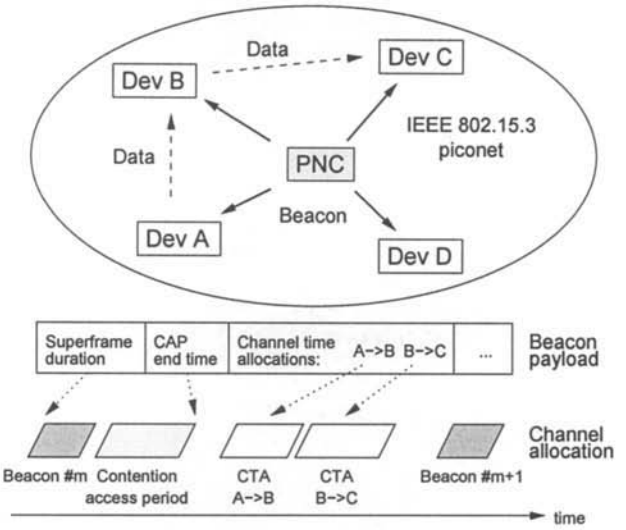

Fig. 1. IEEE 802.15.3 network topology and superframe structure (example)

IEEE 802.15.3 MAC frames consist of a 10-byte header followed by an arbitrary length payload field. Furthermore, a 32-bit frame check sequence (FCS) is calculated over the payload and added to all transmitted frames. The CRC-32 algorithm is used for this purpose. The frame receiver performs the same calculation and compares the final CRC value to determine if any bit error has occurred during the transmission.

In most cases, especially in the case of command frames, the sender expects the receiver to acknowledge the correctly received frame immediately, i.e. exactly 10 microseconds after the end of the frame. If an acknowledgement frame is not received, the sender will retransmit the frame. The basic MAC frame format and the immediate acknowledgment policy is shown schematically in Fig. 2.

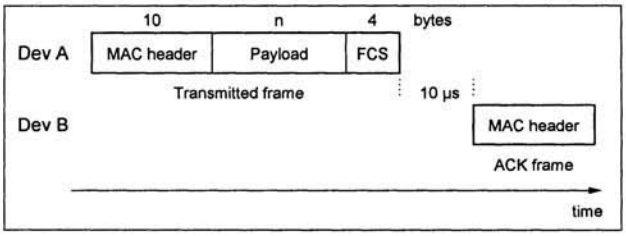

Fig. 2. IEEE 802.15.3 MAC frame format and immediate acknowledgment policy

The specified data rates in the standard range from 11 to 55 Mbit/s, however the same MAC protocol can be used with much higher or lower data rates, as well.

3 Design Flow

The starting point of our design flow was the IEEE 802.15.3 MAC protocol specification [1]. The protocol implementation is targeted for a system-on-chip based on the LEON2 processor. LEON2 is a synthesisable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture [4].

3.1 Protocol Modeling

We have modeled all the necessary functionality in SDL [2] using Telelogic TAU SDL suite [3]. SDL (Specification and Description Language) is a formal description technique that allows to model behavior by means of communicating extended finite state machines. By extensively simulating our IEEE 802.15.3 MAC protocol model we could verify its correct functionality. Details of this model can be found in [10].

The validated SDL model is the basis for the MAC protocol implementation by an automatic transformation. The effort of re-implementing the protocol in C/C++ would be too high and error-prone compared to an optimization approach where inefficient SDL concepts in the model are replaced by equivalent functions with less overhead. Additionally, the time to achieve a fully tested implementation is considerably shortened.

3.2 Operating System Integration

The next step was to target the SDL model to an operating system (OS), in our case the Reflex OS [8]. For this purpose, we developed a so-called Tight Integration model [11]. This replaces the SDL run-time environment with a tailored, very efficient OS integration layer.

Reflex is a tiny, event-flow oriented OS for deeply embedded systems [7]. Although quite similar to TinyOS [9], the operating system most often used for wireless sensor nodes, we believe it is better tailored for our system because of its earliest-deadline-first process scheduling strategy. Whereas TinyOS tasks run to completion before any other task is scheduled, time-critical tasks (activities) will interrupt lower-priority activities in Reflex. Such a behavior is difficult to achieve in TinyOS.

The required memory space for the operating system Reflex, the integration layer, and a simple SDL system was measured to be about 20 kbytes for a system targeted for the LEON2 processor.

3.3 Hardware/Software Co-Design

Some of the MAC protocol functionality underlies tight timing constraints, for instance acknowledgment (ACK) frame transmission has to start exactly 10 microseconds after the end of a received frame. To identify bottlenecks in the pure software implementation and to estimate the required clock frequency to meet all timing constraints, we performed a profiling of the software. For that purpose, the software was simulated using the LEON2 instruction set simulator (ISS) TSIM [5].

Furthermore, with TSIM it is also possible to model the behavior of hardware components that are connected to the LEON2 processor via the on-chip bus. This enables the simulation of the system with protocol functionality mapped to hardware. To achieve this, the corresponding functions were removed from the software model and put into a hardware component. This allowed us to study the new timing behavior of the protocol implementation and optimize the hardware/software partitioning until all timing constraints were met and the required clock frequency was acceptable.

As a result of the hardware/software partitioning we identified the frame reception and transmission procedure, superframe timing control, immediate acknowledgment handling, and parts of the transmission queue to be designed in hardware. In other words, all the low-level, timing-critical and processing-intensive tasks of the channel access mechanism have been mapped to the hardware partition. This corresponds well to the lowest service layer in our SDL model, which is called Transport Engine (see Fig. 3).

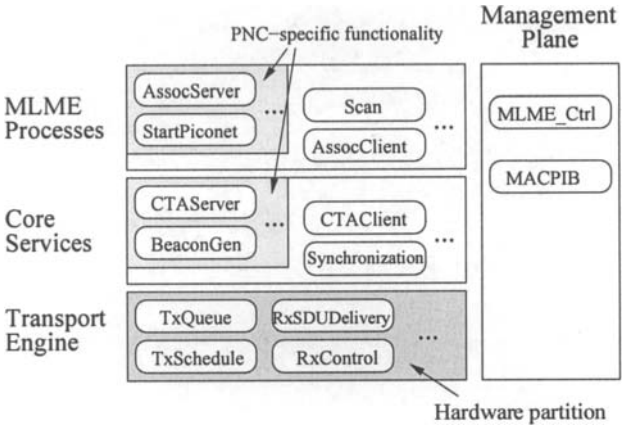

Fig. 3. Functional layering of the SDL processes of the MAC protocol model, partition mapped to hardware highlighted.

The remaining protocol functionality is handled by the LEON2 processor. Interrupts are used to signal protocol-related events from the hardware to the

software. Conversely, the software interacts with the hardware block by writing to and reading from a number of control registers.

4 Protocol Accelerator Design

In this section we first introduce the target hardware platform for the protocol accelerator. Then, we present the overall architecture of the accelerator and how its components work together to provide the desired functionality. This is followed by a detailed presentation of the transmission queue component. We show how our design can be used for different data rates and non-standard physical layer timing parameters and how the protocol software and hardware accelerator interact with each other. Finally, the section is concluded with results from an FPGA implementation of the complete system.

4.1 Target Hardware Platform

The MAC protocol accelerator is one important component of our wireless communication platform. The heart of this hardware platform is the LEON2 processor [4], that runs protocol and application software. We pursue a single-chip integration of the MAC protocol, digital baseband processing and the RF frontend, as shown in Fig. 4.

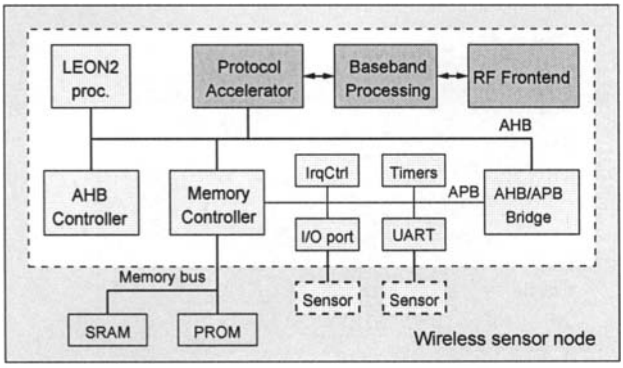

Fig. 4. Sensor node implementation based on our wireless communication platform (white box).

The protocol accelerator is connected to the system bus, the AMBA AHB bus (see Fig. 6). Via an AHB master interface it is possible for the accelerator to directly access the system memory, for instance to store and retrieve frame data without involving the LEON2 processor.

For data transfer to/from the baseband processor as well as status indications from the physical layer, the MAC-PHY interface was designed. It is of

master/slave type with the MAC protocol accelerator acting as master. The accelerator sends commands to control the start of transmission or reception and to exchange frame data. The baseband processor contains a 32-byte data buffer for storing frame data in both, RX and TX, direction in order to decouple the two protocol layers with respect to timing (see Fig. 5). There are some signals from the physical layer to the MAC hardware accelerator that indicate the status of these buffers and can be used for flow control.

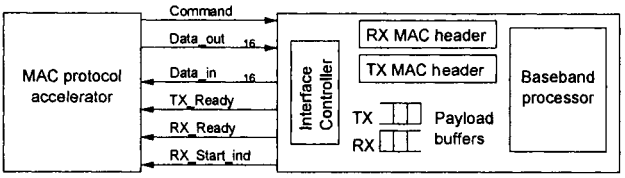

Fig. 5. Interface between MAC protocol accelerator and the physical layer. *TX_Ready* when TX buffer not full, *RX_Start_ind* when reception of new frame has started, *RX_Ready* when RX buffer not empty.

4.2 Architecture

Figure 6 shows the main components of the protocol accelerator. The tasks performed by each of the main components are listed below. They reflect the protocol functions that have been identified to be designed in hardware in Sect. 3.

- In receive direction, to retrieve frame data from the physical layer byte by byte, perform filtering and CRC check, and to store the data at a given memory location by means of direct memory access (components *Rx controller*, *CRC*, and *DMA*).
- In transmit direction, to retrieve frame data from a memory location, calculate and append the check sum, and to push the data to the physical layer (components *Tx controller*, *CRC*, and *DMA*).
- To signal a successful reception or transmission of a frame to the processor by an interrupt (component *Interrupts*).
- To analyze received and transmitted beacon frames and extract information on channel time allocations (component *Beacon parser*).
- To manage a queue of frames that are to be transmitted and to select an appropriate frame for transmission (component *Transmission queue*).
- At the start of a time slot or following a frame transmission, to query a new frame from the queue and, in the case that the frame must be acknowledged by the receiver, wait for the acknowledgment frame (components *Scheduler* and *Timers*).
- To perform the backoff procedure in the contention access period (components *Scheduler* and *Timers*).

- To send an acknowledgment at the right time upon reception of a frame that needs to be acknowledged (components *Scheduler*, *Timers*, and *Tx controller*).

An additional component (*CalcDuration*), that is not shown in Fig. 6 for simplicity, calculates the actual duration of a frame transmission based on its payload length and data rate. This component is used to determine if a frame transmission fits into an available time slot and when a transmission initiated by the protocol accelerator will be completed by the physical layer.

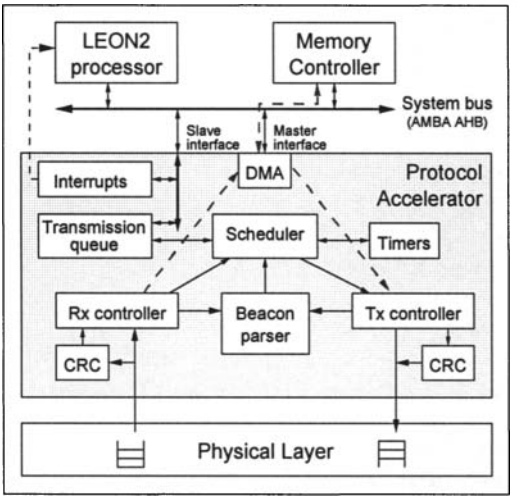

Fig. 6. Hardware architecture of the protocol accelerator (direct memory access data path highlighted).

4.3 Transmission Queue

The *Scheduler*, *Transmission queue*, and *DMA* components in the protocol accelerator facilitate a frame transmission operation that is not directly controlled by the processor. This allows to reduce the clock frequency of the processor and, hence, leads to possible energy savings. The *Transmission queue* component, that shall be discussed in this section, has a key role in this operation.

The transmission queue designed for the IEEE 802.15.3 MAC protocol accelerator contains and manages a table with information on frames to be transmitted—not the frame data itself, which is stored in application memory. The protocol software running on the LEON2 processor fills the table according to the generated frames and their transmission order. An interrupt is signaled to the processor as soon as a frame from the queue has been transmitted successfully or

its maximum retransmission limit has been reached. This indicates to the software that the entry in the table is free and can be reused by another frame. An update, however, does not have to happen immediately as there are still enough frames in the hardware transmission queue.

The current design contains 8 table entries, but there might be many more, e.g. 32 or 64. In order to find the right frame upon request from the *Scheduler* quickly, there are ordered lists for different frame types, for instance for beacon frames or frames that can be transmitted in the contention access period. The table index of the first list item is kept in a register. From this item the complete list can be traversed by following the table index of the next list item, that is stored in the table. Furthermore, there is a reference to the previous list item to support delete operations efficiently. In essence, this forms a double-linked list. These lists are also used to preserve the right order of data frames where the MAC protocol provides this service to the higher layers.

4.4 Support for Flexible Timing

The protocol accelerator has been designed such that it is not limited to a fixed data rate or time intervals between consecutive frame transmissions (interframe spaces). Instead, a number of software programmable registers have been introduced that completely determine the timing behavior of the protocol. These registers are read by the *Scheduler* to calculate the point in time when the next frame can be transmitted. There are registers for SIFS, MIFS, and backoff slot duration, each given in microseconds.

Additionally, there is one programmable register which contains a rate factor and is used by the above mentioned *CalcDuration* component to calculate the duration of frame transmissions.

4.5 Software Interface

The protocol software interacts with the hardware accelerator through a set of registers that are accessible from the processor via memory mapped I/O. A special memory region is reserved for the accelerator. Additionally, interrupts are used to signal events from the protocol accelerator to the processor.

The protocol accelerator contains a number of configuration registers that store MAC protocol information like the piconet or device identifier. It is possible to enable or disable the scheduler with another register. When the scheduler is enabled, the protocol accelerator will analyze beacon frames and seize transmission opportunities, otherwise it is just in scanning mode and delivers received frames.

The protocol accelerator features four maskable interrupts: superframe start, MAC header received, MAC frame received, and transmission queue interrupt. The latter is triggered when a queued frame has been sent or was discarded. Two registers indicate the first free queue index and the first index that contains a transmitted or discarded frame. This way, the software does not have to browse the complete queue.

While the software is modifying the transmission queue, the hardware may not access its contents in order to avoid inconsistencies. Similarly, when the hardware is browsing the queue for a suitable frame to transmit, the software may not alter its lists. Therefore, a lock must be acquired by the software before an operation on the queue can be performed and released when it is done. When the queue is locked, the hardware cannot access the queue.

To control the receive operation, there are registers for the address where the next payload can be stored and for the size of this buffer. When the accelerator has written data to this buffer, no other payload can be received unless a new payload buffer address is provided by the software. An acknowledgment frame will be generated only if the header and payload of the received frame have been stored.

4.6 Results

In a first step, we have implemented the LEON2 system and protocol accelerator on a Xilinx Virtex-II FPGA. We have successfully tested the complete MAC protocol implementation, i.e. the protocol software running on the LEON2 processor and the protocol accelerator, by connecting two such boards with wires. This emulates a network of two devices. The wires couple the boards below the MAC layer, data symbols are transferred serially at a rate of 20 Mbit/s. Table 1 shows the usage of FPGA resources of the same LEON2-based system with and without the protocol accelerator.

Table 1. FPGA resources used by the MAC protocol system.

Resources	LEON2 system		Difference
	Original	With prot. acc.	
4 input LUTs	11,582	24,034	12,452
Occupied slices	6,828	14,365	7,537
Block RAMs	20	22	2
Equivalent gate count	1,427,060	1,681,651	254,591

After the successful test on the FPGA, the complete LEON2-based MAC processor including Flash memory and peripherals has been designed and taped out as an ASIC in 0.25 μm CMOS technology. The chip occupies an area of 31.9 mm^2 and consumes 15 mW/MHz. Based on our synthesis results, the silicon area of the protocol accelerator is about 1.8 mm^2.

5 Related Work

The design of dedicated hardware to speed up data processing and save energy has long been exploited in digital system design. MAC protocol accelerators for wireless communication systems have been reported, e.g. by Meng *et al* [13] and Fujisawa *et al* [14], for the IEEE 802.11a standard, and by Haroud *et al* [15] also for the IEEE 802.15.3 standard. In the latter publication, little is said about the mechanism to initiate frame transmission and the interface between hardware and software.

The use of hardware queues for autonomous transmission of packets is well-known practice and has also been described in [13] and [14]. However, hardware transmission queues are not limited to the design of wireless communication systems, but are used also in high-performance network switching devices such as ATM routers.

6 Conclusion

We have presented a hardware/software implementation of the IEEE 802.15.3 MAC protocol. A prototypical system has been developed for an FPGA board and shows correct and timely protocol behavior at a system clock frequency of 40 MHz with 20 Mbit/s physical layer data rate.

While details on our software design have been published elsewhere, we have focused in this paper on the design of a protocol accelerator and on the communication between the hardware and software part. Moderate clock frequencies of the general-purpose microprocessor—in our case the LEON2 processor—could be achieved mainly because of using direct memory access to payload data in RAM and a hardware transmission queue.

We plan to integrate hardware cores for encryption and decryption in the protocol accelerator, since security algorithms are data-processing intensive and also time-critical tasks. Our current hardware accelerator architecture allows this extension without much design effort.

The proposed protocol accelerator can be used as an IP core for future wireless multimedia communication systems that need to consume low power. The hardware provides programmable registers to adapt to the intended physical layer data rate and timing parameters. Furthermore, its standard AMBA AHB bus interface and the interrupt mechanism make it easy to be integrated with different hardware platforms. The software interface to the protocol accelerator is simple but effective.

Though the presented protocol accelerator is designed specifically to support the IEEE 802.15.3 MAC protocol, its architecture and many of its components can be reused with slight modifications for other protocols. In fact, we plan to use it for a low-power wireless sensor node based on the IEEE 802.15.4 standard, where the channel access mechanism is quite similar.

Acknowledgment

This work was partly funded by the Federal Ministry of Economics and Technology (BMWi) of Germany under grant no. 01 MT 306.

References

1. IEEE Standard 802, "Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks," 2003.
2. ITU-T, "ITU-T Recommendation Z.100. SDL: Specification and Description Language," 1999.
3. Telelogic AB. (2004). Telelogic Tau SDL Suite [Online] Available: http://www.telelogic.com/products/tau/sdl
4. Gaisler Research AB. (2006). LEON2 Processor [Online]. Available: http://www.gaisler.com
5. Gaisler Research AB, "TSIM Simulator User's Manual," 2006.
6. Pender Electronic Design GmbH, "GR-CPCI-XC2V Development Board User Manual," 2005.
7. K. Walther, R. Hemmerling, and J. Nolte, "Generic Trigger Variables and Event Flow Wrappers in Reflex," in *ECOOP — Workshop on Programming Languages and Operating Systems*, 2004
8. J. Nolte, "Reflex - Realtime Event FLow EXecutive," Available from http://www-bs.informatik.tu-cottbus.de/38.html?&L=2, 2006.
9. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Networked Sensors," in *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93–104.
10. D. Dietterle, I. Bababanskaja, K. Dombrowski, and R. Kraemer, "High-Level Behavioral SDL Model for the IEEE 802.15.3 MAC Protocol," in *Proc. of the 2nd International Conference on Wired/Wireless Internet Communications (WWIC)*, P. Langendörfer, M. Liu, I. Matta, and V. Tsaoussidis Ed. Lecture Notes in Computer Science, Vol. 2957. Springer-Verlag, Berlin Heidelberg New York, 2004, pp. 165–176.
11. G. Wagenknecht, D. Dietterle, J.-P. Ebert, and R. Kraemer, "Transforming Protocol Specifications for Wireless Sensor Networks into Efficient Embedded System Implementations," in *Proc. Third European Workshop on Wireless Sensor Networks (EWSN 2006)*, Lecture Notes in Computer Science, Vol. 3868. Springer-Verlag, Berlin Heidelberg New York, 2006, pp. 228–243.
12. D. Dietterle, J.-P. Ebert, G. Wagenknecht, and R. Kraemer, "A Wireless Communication Platform for Long-Term Health Monitoring," in *Proc. PerCom Workshops 2006*, 2006, pp. 474–478.
13. T. H. Meng, B. McFarland, D. Su, and J. Thomson, "Design and implementation of an all-CMOS 802.11a wireless LAN chipset," IEEE Commun. Mag., vol. 41, no. 8, Aug. 2003, pp. 160–168.
14. T. Fujisawa, J. Hasegawa, K. Tsuchie, T. Shiozawa, T. Fujita, T. Saito, and Y. Unekawa, "A single-chip 802.11a MAC/PHY with a 32-b RISC processor," IEEE Journal of Solid-State Circuits, vol. 38, no. 11, Nov. 2003, pp. 2001–2009.
15. M. Haroud, L. Blazević, and A. Biere, "HW accelerated ultra wide band MAC protocol using SDL and SystemC," in *Proc. IEEE Radio and Wireless Conference (RAWCON'04)*, IEEE, 2004.