Implementation Analysis of the IEEE 802.15.4 MAC for Wireless Sensor Networks

Thomas Basmer, Henry Schomann, Steffen Peter
IHP, Im Technologiepark 25, D-15236 Frankfurt (Oder), Germany
Email: {basmer, schomann, peter}@ihp-microelectronics.com

***Abstract*—IEEE 802.15.4 is the dominant Medium Access Control protocol in wireless sensor networks. To implement this protocol on such severely resource constraint devices, several trade-offs have to be considered. This paper presents an analysis of IEEE 802.15.4 software implementations available for typical sensor node systems, such as the MSP430 from Texas Instruments. We discuss the available implementations concerning the supported features, flexibility issues and the efficiency of the implementations. The latter is based on a static code analysis that allows to measure and compare the number of clock cycles, needed to execute specific MAC functionalities. The result of the study is that the performance of the implementations does not correlate with the number of supported features but rather inversely with the implementation flexibility. The results do not only support integrators of 802.15.4-based networks but help identifying key aspects of future implementations of the protocol.**

I. Introduction

Wireless sensor networks (WSN) consist of resource constraint devices - nodes- which typically send data measured by sensors attached to the nodes towards a sink. These networks find application, for instance, in ubiquitous environmental monitoring, medical surveillance, and industrial automation systems. Typically the nodes are small, battery-powered, have limited computation capabilities and a small amount of memory. This is why reduction of computation efforts and memory consumption is a primary goal when integrating such systems. This is particularly true for the wireless network protocols, since sending and receiving data is an integral task of the nodes.

The most often used Medium Access Control (MAC) protocol used in WSNs is the standard IEEE 802.15.4 [1]. The MAC controls the access to shared communication medium in networks, which in WSNs is the air. The MAC has to guarantee a fair access distribution and collision free data transfer for all participants of the network. This is a complex task with severe timing constraints. Implementations of the protocol trade off these requirements with functional or flexibility limitations.

To evaluate the trade-offs, in this paper we analyze IEEE 802.15.4 software implementations available for the Texas Instruments MSP430 microcontroller [2], which is widely used as processing unit in WSNs, e.g. Tmote Sky [3], TinyNode [4], IHPNode [5]. We discuss available MAC implementations concerning the supported features defined in the standard and the efficiency of the implementations. The latter is based on a static code analysis. As part of this analysis we measure and compare the microcontroller clock cycles, needed to execute specific MAC functionalities.

This paper is structured as follows: After a brief introduction of the IEEE 802.15.4 standard and the MSP430 microcontroller, we present the related work done on analyzing this MAC protocol. This is followed by a presentation of available software implementations of the standard. Section III. presents our analysis approach. The results of the analysis are presented and discussed in Section IV. The paper closes with an outlook and the conclusion.

TABLE I
Main features of IEEE 802.15.4 separated into full function devices (FFD) and reduced function devices (RFD). The features are classified as mandatory(m) or optional(o).

Feature	FFD	RFD
Transmit/receive data packets	m	m
Activate/deactivate radio transceiver	m	m
Energy detection within the current channel	m	o
Link quality indication	m	m
Channel selection and clear channel assessment(CCA)	m	m
Beacon management and beacon reception	m	m
Beacon sending	m	o
Guaranteed time slot management	o	o
CRC16	m	m
Acknowledgment frame delivery	m	m
Association and disassociation	m	m
Security	m	m
Beacon Request	m	o
Orphan scan	m	o
Listening for beacons	m	m

A. IEEE 802.15.4

In this section we describe properties of the IEEE 802.15.4 MAC protocol standard. The properties are important to understand differences in the implementations compared later in this paper. The standard IEEE 802.15.4 was designed to cope with the requirements of low-rate Wireless Personal Area Networks (LR-WPAN). Therefore it should guarantee a reliable access to the wireless medium in applications with low duty cycles, low data rates ($< 250kiB/s$) and a limited amount of energy. IEEE 802.15.4 supports star as well as peer-to-peer network topologies. Due to this flexibility with reasonable footprint, it became the de-facto standard used in the field of wireless sensor networks.

Another benefit is the separation of full function devices (FFD) and reduced function devices (RFD) which can work together in the same network. A full function device is used as network coordinator, cluster head or normal device. It provides extensive features to control and supervise its cluster or the whole network. The RFDs are used as simple network nodes, e.g. to collect data. They can only communicate over their assigned coordinator.

Features of the devices are divided into mandatory ones, which must be implemented and optional ones, which could be implemented to be compliant to the standard. Naturally, the set of supported functions influences the complexity of the implementation as we will see later in this paper. The mandatory and optional features of both device types (FFD and RFD) are listed in Table I. This overview gives an impression on the scope of the standard and is used to discover the range of the different software implementations.

B. *MSP430*

As mentioned above our work is focused on MSP430 microcontroller systems. This is caused by the fact that this microcontroller family is well suited for low power systems and widely used in the field of wireless sensor network devices. The MSP430 is a 16 bit reduced instruction set cycle (RISC) system in von Neumann architecture (i.e. one address space used for program code and data). Its address bus supports a width of 16 or 20 bit, depending on the configuration. The 20-bit configuration (called x-architecture) is applied in the *TIMAC* implementation, which is introduced later in this paper.
The instruction set contains 27 core instructions and several emulated instructions. A core instructions used on registers only needs one clock cycle for execution. Emulation combines different core instructions to an emulated one, which reduces the code size but needs more time. Generally, there are three different instructions types available: single operand instructions, double operand instructions and jump instructions. The needed clock cycles per instruction depends on the type and the addressing mode for source and destination of the instruction.

C. *Related Work*

Related work in the context of the analysis of light-weight implementations of IEEE 802.15.4 is relatively scarce. There are some evaluations of IEEE 802.15.4 implementations with focus on network performance [6], [7], [8], [9], [10], [11], [12], e.g., reliability, throughput or separate parts of the standard, e.g., AES, CRC or auto-acknowledgment. The implementations themselves are not evaluated. Work on profiling tools for software for microcontrollers has also been rarely published. In [13] existing profiling tools like msp430-gprof, OProfile and Vtune are evaluated. In this work a good overview about software profiling issues and requirements is given. Those tools are developed for profiling high end processors and not usable on microcontroller devices. This result of the survey motivated the author to implement the novel tool msp430-eprof for the MSP430 microcontroller family, which, however, is currently not publicly available.

II. IEEE 802.15.4 Software Implementations

In this section the five different software implementations currently available for the MSP430 microcontroller are introduced. The implementations are *TKN15.4*, *open-ZB*, *TIMAC*, *IHP_MAC_SDL*, and *IHP_MAC_cb*.

A. *TKN15.4*

The *TKN15.4* implementation has been developed at the Technical University of Berlin [14]. It is component-based and platform-independent, which allows reusability in many systems.
The software is written in nesC to be used with the operating system for wireless sensor nodes TinyOS [15]. The current version is implemented with focus on the TMote sky sensor node (one of the first commercial available sensor node) [3]. The required software components can be configured at compile time. To fulfill the timing constraints several functions had to be shifted from the software MAC layer to the hardware PHY layer of the TMote sky and its CC2420 radio module [16]. For example the CSMA/CA procedure, the acknowledgment procedure and the CRC16 calculation are performed in the radio front-end. This improves the performance but leads to increased hardware dependencies.
Functions that are still missing in this implementation are the Guaranteed Time Slot (GTS) functionality and the security options. Notable is that *TKN15.4* assigns the role of the device (FFD or RFD) at runtime as part of the software. This improves the flexibility since the role of the device can be changed during its lifetime. However, it also requires the software to contain all functionality even if it is not needed. This may be disadvantage in very memory constrained devices such as sensor nodes.

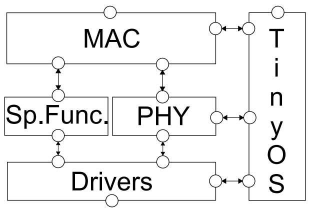

Fig. 1. Software architecture of the *TKN15.4* implementation

From implementation view, the *TKN15.4* software can be divided into five parts (see Figure 1). The first part (TinyOS) contains the operating system dependencies. It has interfaces to all other parts. The upper part (MAC) contains the functionalities of the IEEE 802.15.4 standard. It has an interface to the network layer above and to the physical layer (PHY) and special function layer (Sp.Func) below. These two layers provide a hardware abstraction, which allows the MAC operate without knowledge of the underlying hardware. For a new hardware platform only this hardware abstraction layer (HAL) has to be adjusted.
The PHY component provides interfaces to the physical layer functionality of the radio device. Functions like auto-acknowledgment and hardware CRC are accessed via the special function component. As mentioned above, these special functions are applied to improve the performance and to save execution time and energy.

B. *open-ZB*

The *open-ZB* IEEE 802.15.4 MAC implementation has been developed at the Polytechnic Institute of Porto [17]. As *TKN15.4* it is written in nesC for TinyOS. The software implements nearly the full standard except extended address field names and security functionalities. *open-ZB* is available for MICAZ [18] and Tmote sky sensor node. Contrary to *TKN15.4*, the scope of *open-ZB* is limited to clients, which means that devices with the role as network coordinator are not supported.
The *open-ZB* implementation can be divided into four parts: the MAC implementation containing all main functionalities, the physical layer part containing the general functions to access the hardware, the drivers and the operating system part (see figure 2). The TinyOS block applies internal functionalities of the operating system and provides interfaces to the three other parts. The MAC has interfaces to the network layer above and the physical layer below. It contains the main functionalities of the implementation. The HAL is implemented within the physical layer part. It provides an abstract interface to the MAC part.The physical layer uses the hardware dependent drivers. To adapt *open-ZB* to other platforms, TinyOS and the hardware drivers must be available for this hardware. The physical layer part must also be adapted to the new drivers.

C. *TIMAC*

The *TIMAC* [19], [20] implementation is offered by Texas Instruments for use with MSP430 microcontrollers in x-architectures and Chipcon radio modules like the CC2420. It can be found at [21]. Additionally to the 20-bit address bus, the MSP430 x-architecture also provides additional instructions like pushm. Using the pushm

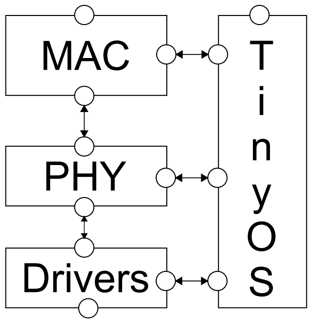

Fig. 2. Software architecture of the *open-ZB* implementation

instruction, it is possible to push more then one register onto stack using a single command. The instruction saves two clock cycles per register push compared to the original push instruction. The implementation is certified to be compliant to the IEEE 802.15.4 standard. Unfortunately it was not possible to analyze this software because it is only available as compiled library. Also, no detailed information of the internal software structure is available. The source code can be bought from Texas Instruments for 10.000$. Due to the background of the TI as manufacturer of the chipsets it can be assumed that the implementation is very efficient but comes with a lack of portability. However, we could not evaluate this assumption.

D. IHP_MAC_SDL

This software is implemented at IHP for use on Tmote sky sensor nodes. The implementation is derived from a Specification and Description Language (SDL) model of the IEEE 802.15.4. The generated code is ported to the Reflex [22] operating system. The implementation is event-driven. This allows to divide the MAC into several functional tasks which are activated by events, e.g., interrupts. The implementation consists of the beacon-enabled and non-beacon mode. Security services and GTS management have not been implemented yet.
The *IHP_MAC_SDL* implementation comprises five parts (see figure 3). The Upper MAC provides an interface to the network layer above and the core MAC implementation, containing the main functionalities, below. Below the MAC part the Transport Engine is located. It represents the hardware abstraction layer of the implementation, providing an interface to the hardware drivers. The Reflex operating system has interfaces to all parts of the implementation. To adapt the implementation to other platforms Reflex and the drivers must be available for that hardware. Additionally the Transport Engine must be adapted to the new hardware drivers.

E. IHP_MAC_cb

The *IHP_MAC_cb* implementation of the IEEE 802.15.4 has also been developed at IHP. It claims to be as hardware and operating system independent as possible. It is a component-based (cb) implementation in C++ programming language. The different functionalities are encapsulated into different components with well-defined interfaces. The component-based design needs only changes in few components to adjust the implementation to new hardware or operating systems. It is possible to configure the MAC functionalities to the requirements of the implementation. Software-only blocks can easily be replaced by blocks using hardware accelerators. The implementation consists of all mandatory functionalities of the standard. From the discussed implementations it is the only one executing the CRC checksum calculation in software, which increases the portability. Security

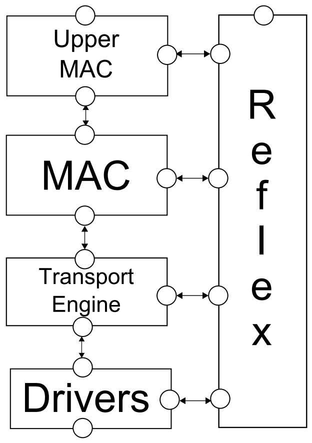

Fig. 3. Software architecture of the IHP_MAC_SDL implementation

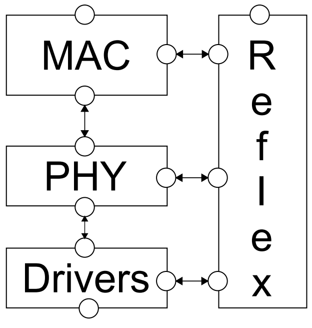

Fig. 4. Software architecture of the IHP_MAC_cb implementation

options and beacon-enabled mode are not implemented, yet. Like *open-ZB* the software only addresses network clients. Setup of a network coordinator is not supported. The implementation uses the same architecture as *open-ZB* but it uses the Reflex operating system instead of TinyOS (see figure 4). The physical layer part (PHY) represents the HAL providing an abstract interface to the underlying drivers for the hardware components. To use this implementation on other platforms the drivers and the operating system Reflex must be available for this hardware. Also the PHY part must be adapted to the new drivers.

F. Summary

This section introduced the properties of five implementations IEEE 802.15.4 for the MSP430. We presented the functional range of these implementations and summarized it in a Table for a better comparison.
One important distinction between the implementations is the amount of functionality realized in software and hardware. Each implementation has a slightly different focus on what has to be done in software and what has to be done in hardware. Table II gives a short overview on this.
It must also be noticed, that none of the five implementations provides a complete implementation of the standard. The standard is very complex (more than 600 pages) and lavish functions like security are not implemented yet.
In the next section the properties of the implementations are analyzed in detail.

TABLE II
IEEE 802.15.4 FUNCTIONS IN THE IMPLEMENTATIONS (HW: HARDWARE, SW: SOFTWARE IMPLEMENTATION, N.I. NOT IMPLEMENTED).

Function	***TKN15.4***	***open-ZB***	***TIMAC***	***IHP_MAC_SDL***	***IHP_MAC_cb***
Frame construction	SW	SW	SW	SW	SW
Security	n.i.	n.i.	n.i.	n.i.	n.i.
CRC16	HW	HW	HW	HW	HW/SW
GTS	n.i.	SW	n.i	SW	n.i.
Beacon mode	SW	SW	SW	SW	n.i.
Channel scan					
Beacon request	SW	SW	SW	SW	n.i.
Listening for beacons	SW	SW	SW	SW	n.i.
CSMA/CA					
slotted	HW	SW			n.i.
unslotted	HW	SW	SW		SW
Acknowledgment	HW	HW	HW	HW	SW
Address filtering	HW	SW		SW	HW/SW

III. Our Analysis Approach

In this paper we want to analyze the runtime of several IEEE 802.15.4 functions in different implementations for the MSP430 microcontroller. In this section typical approaches to measure the implementation properties are briefly discussed. This section concludes with a presentation of our applied analysis flow
To analyze the implementation parameters generally, there are four options to measure the runtime:

- **Time Stamp Counter Register**
  In high-end processors , such as the Intel Pentium 4 or AMD K8, a special counter register, incrementing with every clock cycle, is available to generate time stamps. That could be used for such purpose. In microcontrollers like MSP430 such register is absent.
- **Microcontroller Internal Timer**
  In general a microcontroller comes with at least one internal timer component. These timers can also be used to measure runtime. But in most cases they are used by the application it self and they are not usable for profiling. Further, it is not possible to run internal timers with the real system clock and the resolution is not sufficient for clock cycle count.
- **Simulator**
  Simulators could also be a good choice for runtime measurements. We had a look at a simulator for the MSP430 microcontroller family. It provides more capabilities to get additional information about the system and the running software. The inspected simulator for this microcontroller family is MSPsim from SICS [23]. This Java based tool simulates on instruction level but not clock cycle accurate. It also does not implement the behavior of all MSP430 instructions.
- **Profiling Tools**
  The usage of profiling tools included in IDEs for the software development, e.g. Code Composer Studio (Texas Instruments) and the IAR Workbench, can also be feasible way to analyze software runtime. The work of [13] deals also with software profiling on microcontroller systems and examines several other profiling tools. Those tools are developed for profiling high end processors and so the author develops its own profiling tool called msp430-eprof that is currently not publicly available.

Unfortunately the first three approaches cannot be applied for our analysis. The time stamp counter register cannot be applied because it is absent in the MSP430. The internal timers of the microcontroller can not be used for analysis because typically they are used for the application itself and changes to the software must be done. More promising are simulators. The MSPsim, however, is not clock cycle accurate and it does not support the full instructions set. The fourth approach, using profiling tools is the most promising one. However, with those tool we encounter several problems:

- strong dependencies of the software to the operating systems
- strong dependencies of the operating systems to special compilers
- Code Composer Studio does not support a good clock cycle profiling
- Reflex is based on C++, but IAR does only support embedded C++ for MSP430 microcontrollers

As answer to these issues we applied a novel analysis flow: As first step the MAC implementation is compiled and an assembler listing is generated. This listing is loaded into our tool as a second step. It calculates the clock cycles needed for every instruction line and sums the values for the different subroutines. After this we identify the position of the different MAC functionalities and which subroutines are involved to calculate its runtime. This method may become exhaustive especially if the code is not well structured. Thus, such a manual flow actually requires significant time. To accelerate the flow we wrote a simple script-based profiling tool for the static code analysis. This tool is not subject of this paper.
We focused our analysis on the main IEEE 802.15.4 functions and did the examination on all five implementations presented above, except *TIMAC* because no sources are available. The results of that analysis are presented in the next section.

IV. Analysis Results

The results of our examination are summarized in Table 3. As discussed before, *TIMAC* could not be analyzed, due to the absence of the sources. For the other four implementations we analyzed the different implementations of important MAC functions:

A. Analysis

1) Channel Scan: The first examined function is the active channels scan. It is used in the beacon-enabled mode to scan for a network coordinator. Therefore, the device broadcasts a request command that triggers the coordinator to send a beacon the device can scan for. There is also a passive scan available, in which the device scans for beacons without sending a request. In *TKN15.4* the active scan takes 719 clock cycles and in *open-ZB* 1052 clock cycles. *IHP_MAC_SDL* needs 2593 cycles, caused by the extensive use of procedure calls (worst case: 5 clock cycles for call instruction + 16 [registers] * (4 clock cycles for register push + 3 clock cycles for register pop instruction)) and buffers. The platform-independent *IHP_MAC_cb* implementation does not support active scans.

2) Association Request and Response: The next functionality is the association that is separated into request and response. The request is a special data frame that must be generated and sent by every node that wants to participate in a network. The response is sent by the coordinator to confirm the association of the requesting node. The effort for an association request lies in the range of 486 to 790 cycles. Only the *IHP_MAC_SDL* implementation needs 1630 clock cycles, caused by expensive procedure calls.

TABLE III
RUNTIME OF DIFFERENT MAC OPERATIONS IN CLOCK CYCLES (N.I. MEANS NOT IMPLEMENTED, HW MEANS REALIZED IN HARDWARE).

Function	***TKN15.4***	***open-ZB***	***IHP_MAC_SDL***	***IHP_MAC_cb***
Beacon request	719	1052	2593	n.i.
Orphan scan				
Association				
request	763	486	1630	731
response	790	n.i.	1527	n.i.
Create/send 127 byte package	809	971	1520	2561
Receive/analyze 127 byte package	240	1425	962	2676
Build acknowledgment frame	HW	HW	HW	161
Synchronization	520	1052	2641	n.i.
CRC16	HW	HW	HW	2060

3) Package Sending: Clearly, sending and receiving are the most used MAC functions in a sensor network. We examined the sending effort for the maximum package length of 127 byte. In this task the package header must be generated and the data bytes as well as the checksum have to be appended. After this the package is handed over to the physical layer. The package generation and hand over takes between 809 clock cycles in *TKN15.4* and 971 clock cycles in *open-ZB*. The *IHP_MAC_SDL* implementation consumes 1520 clock cycles, mainly caused by the reasons mentioned before. The package generation and sending procedure contains 21 procedure calls, in contrast *TKN15.4* needs 6 procedure calls. The *IHP_MAC_cb* implementation requires significantly more time with 2561 clock cycles, mainly due to checksum calculation implemented in software which alone consumes 2060 clock cycles.

4) Package Reception: For reception of a 127 byte package it must be analyzed and the payload must be passed to the network layer. Analysis contains checksum calculation and verification and header extraction. It consumes 240 clock cycles for *TKN15.4* and 962 clock cycles for *IHP_MAC_SDL*. *IHP_MAC_cb* needs 2676 clock cycles because of its software implemented checksum calculation. The *open-ZB* needs 1425 clock cycles to analyze a received package and forward it to the layer above. During this process the package date is exhaustively copied, regardless if it is necessary or not.

5) Package Reception Acknowledgment: In some application it is necessary to acknowledge the reception of a package to the sender. The period to acknowledge the reception of a package is limited to $192\mu s$ in the standard. Within this time the package must be analyzed, filtered and an acknowledgment frame must be built. To reach this timing the acknowledgment is performed by every implementation using hardware accelerators in the connected radio modules on the platforms. The amount of clock cycles needed for the hardware acknowledgment is not presented in the datasheets of the radio modules. Only *IHP_MAC_cb* realizes this functionality in software to guarantee its hardware independence. It takes 161 clock cycles to generate an acknowledgment frame. On a sensor node platform running with a 1 MHz clock this takes $161\mu s$, hence a software only acknowledgment is not possible.

6) Synchronization: For low duty cycle application with long periods of inactivities between the data transfers, the communication link can run out of synchronization, caused by the clock drift of the low-cost oscillators used on the sensor network devices. Therefore a resynchronization procedure is needed. Synchronization needs 520 clock cycles in *TKN15.4* and 1052 in *open-ZB*. In the *IHP_MAC_SDL* implementation it takes 2641 clock cycles. The synchronization is here realized using an active scan. In *IHP_MAC_cb* synchronization is not implemented yet.

TABLE IV
MAIN FEATURES OF THE IEEE 802.15.4 AND IF THEY ARE REALIZED (✓) OR NOT (❍) IN THE DIFFERENT SOFTWARE IMPLEMENTATIONS

Feature	***TKN15.4***	***open-ZB***	***TIMAC***	***IHP_MAC_SDL***	***IHP_MAC_cb***
Transmit/receive data packets	✓	✓	✓	✓	✓
Activate/deactivate radio transceiver	✓	❍	✓	❍	✓
Energy detection within the current channel	✓	✓	✓	✓	✓
Link quality indication	✓	✓	✓	❍	❍
Channel selection and CCA	✓	✓	✓	✓	✓
Beacon management and beacon reception	✓	✓	✓	✓	❍
Beacon sending	✓	✓	✓	✓	❍
Guaranteed time slot management	❍	✓	❍	✓	❍
CRC16	✓	✓	✓	✓	✓
Acknowledgment frame delivery	✓	✓	✓	✓	✓
Association and disassociation	✓	✓	✓	✓	✓
Security	❍	❍	❍	❍	❍
Beacon request	✓	❍	✓	✓	❍
Orphan scan	✓	✓	✓	❍	❍
Listening for beacons	✓	✓	✓	❍	❍

B. Conclusion of the Analysis Results

The measured results described above are summarized in Table III and Table IV. Table III shows the consumed clock cycles for the different MAC functions for the IEEE 802.15.4 implementations except *TIMAC*. Table IV shows the presence of the mandatory and optional features, which were initially introduced in Table I.

1) TKN15.4: *TKN15.4* is the most efficient implementation in case of runtime. It comes with the shortest runtime except in the association request. It can switch its role (coordinator or client) at runtime. Therefore a lot of additional code must be compiled and stored in the small microcontroller memory. To the best of our knowledge *TKN15.4* has not been certified if it is compliant to the IEEE 802.15.4 standard. If we compare the entries of Table I to the *TKN15.4* column of Table IV it can be seen, that *TKN15.4* fulfills all mandatory and optional features needed for FFD and RFD devices. So it seems the software is theoretically compliant to the standard and can be used for FFD and RFD. The Checksum calculation, acknowledgment, CSMA/CA and address filtering are shifted into the radio module. That speeds up their execution but makes this software more dependent on the underlying hardware.

2) open-ZB: The *open-ZB* implementation only addresses network clients. This reduces its functional range and complexity. It has already passed compliant test to guarantee it is IEEE 802.15.4 standard conform. It is the fastest implementation of the association request. The software also uses the underlying hardware for checksum calculation and acknowledgment to speed up this time critical functions. This leads to more hardware dependencies. The comparison of the standard its features and the implemented features in *open-ZB* shows that it not fully implements all RFD and FFD features. For RFD the activation and deactivation of the radio module is missing. For FFD also the active scanning (beacon request) has to be implemented.

3) TIMAC: The analysis of the *TIMAC* implementation is focused on its functional description. It has also passed compliant tests to guarantee its IEEE 802.15.4 conformity. If we compare Table I with the *TIMAC* column of IV it can be seen, that *TIMAC* also fulfills the mandatory and optional features for FFD and FFD.

4) IHP_MAC_SDL: IHP_MAC_SDL needs significantly more time than the other hardware-accelerated implementations. For synchronization it needs five time more clock cycles then *TKN15.4*. The implementation language SDL, which is a very fast way to implement protocols graphically, generates code that uses many expensive subroutine calls. That increases the load on the microcontroller since it has to store and restore the registers content for the additional task switches. The checksum calculation and acknowledgment are using the hardware of the platforms radio module. This improves the execution time at the expense of hardware dependence.
The comparison of Table I and Table IV shows that this implementation does not fulfill the features for FFD and RFD. To fulfill RFD requirements the activation and deactivation of the radio, the link quality indication, security and listening for beacons must be implemented. For FFD further the orphan scan is needed.

5) IHP_MAC_cb: This platform-independent implementation only addresses network clients. That reduces the complexity and range of the implementation. As the only implementation *IHP_MAC_cb* realizes all functions in software to be really hardware independent. However, the modular architecture allows to adopt the software to use the hardware accelerators in the future. This is required for the acknowledgment because in software the timing constraints could not be met. If we compare Table I to the corresponding column in Table IV it can be seen that the implementation does not fulfill the requirements for FFD or RFD. The forwarding of link quality indication to higher layers, all the beacon functionality either scanning for beacons still has to be implemented.

V. Conclusion and Future Work

In this paper we presented an overview on existing IEEE 802.15.4 Medium Access Control implementations for the MSP430 microcontroller family. Therefore, we performed a static code analysis to assess the software runtime of the different MAC implementations. This analysis determines the number of clock cycles required for specific operations of the MAC protocol. This measure, which is a good indication for the actual energy consumption of sensor node devices, was also applied in the trade-off analysis in which we compared the implementation flexibility and the supported features of implementations. It turned out that, while no implementation supports all features, at least *TIMAC* and *open-ZB* are proved to be compliant to the standard. Nevertheless, security functions have not been integrated in any of the implementation. While the run time of some functions of the implementation varies by factor of up to five, we could not identify speed advantages of implementations with reduced feature set. The performance rather correlates with the number of functions supported by hardware. The hardware dependencies as we see them in *TIMAC* and *TKN15.4* reduce the portability but deliver good performance, while the *IHP_MAC_cb*, which is all software, needs the most clock cycles for the operations.
In future this examination result can be basis for directed optimizations of available implementations of the standard IEEE 802.15.4. It appears hardware accelerators are necessary to cope with the identified performance bottlenecks. However, such accelerator functions may be integrated into the microcontroller or as part of a system on chip, rather than be unloaded to a fixed radio. Such an implementation could be easily adapted to achieve an efficient IEEE 802.15.4 compliant hardware/ software co-design. Another future activity is the extension of our profiling tool chain to support other microcontroller architectures. It will extend the presented analysis to other platforms.

References

[1] *IEEE Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, IEEE Std., Rev. IEEE Std 802.15.4-2006 and IEEE Std 802.15.4-2007a, 2007.
[2] *CMSP430F15x, MSP430F16x, MSP430F161x MIXED SIGNAL MICROCONTROLLER*, Texas Instruments, 2009.
[3] Moteiv Corporation, "Tmote Sky Ultra low power IEEE 802.15.4 compliant wireless sensor module http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf," 2006.
[4] H. Dubois-Ferrière, L. Fabre, R. Meier, and P. Metrailler, "Tinynode: a comprehensive platform for wireless sensor network applications," in *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*, 2006.
[5] K. Piotrowski, A. Sojka, and P. Langendoerfer, "Body area network for first responders-a case study," in *Proceedings of the Fifth International Conference on Body Area Networks*, ser. BodyNets '10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.
[6] IEEE, Ed., *Performance Evaluation of the IEEE 802.15.4 MAC for Low-Rate Low-Power Wireless Networks*. Los Angeles: Department of Electrical Engineering, University of Southern California, 2004.
[7] B. Bougard, F. Catthoor, D. C. Daly, A. Chandrakasan, and W. Dehaene, "Energy Efficiency of the IEEE 802.15.4 Standard in DenseWireless-Microsensor Networks: Modeling and Improvement Perspectives," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, IEEE, Ed., 2005.
[8] M. Petrova, J. Riihijärvi, P. Mähönen, and S. Labella, "Performance Study of IEEE 802.15.4 Using Measurements and Simulations," in *WCNC 2006 proceedings*, IEEE, Ed., 2006.
[9] F. Vater and P. Langendörfer, "An Area Efficient Realisation of AES for Wireless Devices," *Information Technology 03/2007*, pp. 188–193, 2007.
[10] W. I. C. 6th International Conference, WWIC 2008, Ed., *An Encryption-Enabled Network Protocol Accelerator*, ser. LNCS5031, vol. 6. Springer, 2008.
[11] E. Lenchak, "CRC Implementation With MSP430," Texas Instruments, Application Report, 2004.
[12] J. Flora and P. Bonnet, "Never Mind the Standard Here is the TinyOS 802.15.4 Stack," University of Copenhagen, Technical Report 06/10, 2006.
[13] R. Hildebrandt, "Softwaremethoden zur Senkung der Verlustenergie in Microcontrollersystemen," Ph.D. dissertation, Technische Universitat Dresden, July 2007.
[14] J. H. Hauer, "TKN15.4: An IEEE 802.15.4MAC Implementation for TinyOS 2," Telecommunication Networks Group, Technical University Berlin, Technical Report TKN-08-003, 2009.
[15] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "Tinyos: An operating system for sensor networks," 2005. [Online]. Available: http://dx.doi.org/10.1007/3-540-27139-2_7
[16] *Chipcon AS SmartRF CC2420 Preliminary Datasheet*, Chipcon, 2004.
[17] A. Cunha, M. Alves, and A. Kouba, "An IEEE 802.15.4 protocol implementation (in nesC/TinyOS): Reference Guide v1.2," Polytechnic Institute of Porto (ISEP-IPP), Porto, Technical Report 1.2, 2007.
[18] *MicaZ WIRELESS MEASUREMENT SYSTEM*, crossbow, 2004.
[19] "802.15.4 MAC Application Programming Interface," Texas Instruments, Tech. Rep., 2009.
[20] "MAC Sample Application Software Design," Texas Instruments, Tech. Rep., 2007.
[21] [Online]. Available: http://focus.ti.com/docs/toolsw/folders/print/timac.html
[22] K. Walther and A. Sieber, "The Reflex Whitepaper," Lehrstuhl Verteilte Systeme / Betriebssysteme BTU Cottbus, Tech. Rep., 2010.
[23] J. Eriksson, A. Dunkels, N. Finne, F. Österlind, and T. Voigt, "MSPsim an Extensible Simulator for MSP430-equipped Sensor Boards."