Research Repository UCD

Title	Portability versus efficiency tradeoffs in MAC implementations for microsensor platforms
Authors(s)	Schoofs, Anthony, Stok, Peter van der, Stanley-Marbell, Phillip
Publication date	2009-05
Publication information	Schoofs, Anthony, Peter van der Stok, and Phillip Stanley-Marbell. "Portability versus Efficiency Tradeoffs in MAC Implementations for Microsensor Platforms" 1, no. 1 (May, 2009).
Publisher	IEEE
Item record/more information	http://hdl.handle.net/10197/1895
Publisher's version (DOI)	10.1109/LES.2009.2028040

Downloaded 2023-04-07T10:52:29Z

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)

© Some rights reserved. For more information

Portability Versus Efficiency Tradeoffs in MAC Implementations for Microsensor Platforms

Anthony Schoofs, Peter van der Stok, and Phillip Stanley-Marbell

Abstract—Medium access control (MAC) implementations control access of network devices to a transmission medium. For emerging communication protocols, the MAC is typically implemented in software, to enable adaptation to evolving de-facto or industry standards. Software MAC implementations are typically realized as state machines, executing code related to successive MAC states within periodic interrupts. This software construct yields minimal memory footprint and energy efficiency, but the resulting implementations are often tightly coupled to the platform's system software, and are thus nonportable across hardware and system platforms. This article presents an architecture that decouples MAC and system software, enabling portability, while preserving software efficiency.

Index Terms—Embedded systems, medium access control (MAC), personal area networks, sensor networks.

I. INTRODUCTION

MICROSENSOR platforms have scant resources, making the development of embedded software a cumbersome task. Many of the challenges these systems pose are however mutually contradictory: **Portability and modularity**—the requirements of hardware platforms in emerging embedded systems domains evolve rapidly, prior to the achievement of industry standards. This requires software *modularity* and *portability* to facilitate quick adaptation; **Efficiency**—software *efficiency* in terms of memory footprint, network and computation latency, and energy, is required as systems are almost always under severe constraints for these resources; **Flexibility**—low-level system software (e.g., device drivers) need to be ported to the hardware platforms of interest. Integration of components is very dependent on the software's *flexibility* (versus fragility); small changes in the implementation of one component should not lead to needs for significant reorganization of other components.

Medium access control (MAC) implementations control access of network devices to a transmission medium, and are one

of the major system components in wireless microsensor platforms. This article presents an implementation architecture that decouples MAC and system software, enabling portability, and flexibility, while preserving software efficiency.

II. BACKGROUND

For emerging communication protocols, the MAC is typically implemented in software, to enable adaptation to evolving de-facto or industry standards. Software MAC implementations are often realized as state machines, executing code related to successive MAC states within periodic network event or timer interrupts. This software organization yields minimal memory footprint and good energy efficiency, but the resulting implementations are often tightly coupled to the platform's system software. They are thus not easily portable across processor architectures and system platforms. MAC software requires accurate timing behavior to function correctly. When MAC software is tightly coupled with system software, and, for example, embedded within interrupt handlers, the timing behavior of other time-critical system components may be adversely affected.

An alternative design methodology, is to implement the MAC state machine as an application task, whose execution is not embedded within timer and network peripheral interrupts. Leaving an interrupt-driven software architecture creates several opportunities for improving portability, modularity, and flexibility. Furthermore, as previously observed in a related context [1], this can be achieved without sacrificing efficiency.

Appropriate software organizations may actually improve energy efficiency. For example, in wireless low-power microsensor platforms, a *radio transceiver* integrated circuit implements the interfacing between software and the wireless channel, and is typically the largest source of power dissipation. The energy consumed by the transceiver is related to the size and generation frequency of data packets/frames. Data processing on the node (e.g., signal processing, data aggregation, etc.) can be used to intelligently reduce the number of packets to be transmitted and thus potentially lower the energy consumption of the whole system [2] while improving timing performance.

A. Processing of Interrupts in Low-Power Control Processors

The event-driven nature of computing systems is particularly pronounced in microsensor platforms, where the primary activity of systems is to react to the occurrence of temporal or sensing events. In virtually all systems, the change of normal program control flow in response to such events is through an *interrupt mechanism*, in which program execution (or idling) is halted, and the processor jumps to a pre-determined address to execute a new stream of instructions. The ability to return to its

Manuscript received June 05, 2009. First published July 21, 2009; current version published September 23, 2009. This manuscript was recommended for publication by T. Abdelzaher. This work is partially financed by the European Commission under the Framework 6 IST Project Wirelessly Accessible Sensor Populations (WASP) and supported by Science Foundation Ireland under grant 07/CE/I1147.

A. Schoofs is with CLARITY: Centre for Sensor Web Technologies, University College Dublin, Dublin, Ireland (e-mail: anthony.schoofs@ucdconnect.ie).

P. van der Stok is with Philips Research, Eindhoven, The Netherlands (e-mail: peter.van.der.stok@philips.com).

P. Stanley-Marbell is with IBM Research GmbH, Zurich Research Lab, Switzerland (e-mail: pst@zurich.ibm.com).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/LES.2009.2028040

Fig. 2. Execution of MAC functions as requested by MAC primitives.

Fig. 3. Triggers and relationships between software components for processing packet transmission and reception.

request from the highest priority non-empty queue, and executes the corresponding function, starting on a backoff slot boundary.

As long as the priority queues are filled, the MAC timer is active and generates interrupts every 320 μ s. Upon interrupts from the CC2420 transceiver, the FreeRTOS MAC task executes the MAC functions according to the contents of a *task information structure*, which contains information on the currently active functions and the associated packets. Fig. 3 summarizes the successive steps executed for processing packet transmission and reception. The different software triggers are highlighted to emphasize the software constructs and the relationship between the components of the software system.

IV. PERFORMANCE AND PORTABILITY EVALUATION

A *hardware abstraction layer* was used to decouple the MAC's hardware needs from the facilities provided on a specific platform; additional details are available in a previous report [9]. The original implementation was performed for the CoolFlux DSP, and this was ported in three months to a TelosB-like node (having a TI MSP430 microcontroller and a TI CC2420 transceiver). The majority of the effort in the MSP430 port involved adapting to a new compiler and dealing with the new architecture's timer peripheral. Due to the lower clock frequency of the MSP430, a large part of the period was used to optimize the medium access mechanisms under heavy transmission load, which led to significantly different runtime behavior compared to the CoolFlux DSP.

A. Integration and Reuse

The proposed dual scheduling design is currently used for a wide range of applications. The benefits of the design allowed for fast prototyping of new applications with different requirements such as timing and process control, via the reuse and integration of software components. For example, a stroke rehabilitation application [10] required synchronized data sampling on different nodes to meaningfully correlate data at the monitoring station. Time synchronization and distributed task synchronization as described in [12] and [13] have also been successfully implemented within the presented software environment. An emotion sensing application [11] used these facilities gratefully. Finally, a lightweight IP stack and a 6LoWPAN adaptation layer have been implemented on top of the software system [14].

B. Memory Usage

The current version of the MAC software includes support for both beacon and non-beacon networks. Two 128 B buffers for outgoing and incoming packets are used. The code size is approximately 11 KB and the data usage is approximately 2.3 KB. In comparison, the source code base from which the implementation was derived had a data memory footprint of 21 KB for code and 2 KB for data, with an additional three buffers of 200 B each (albeit on a different processor architecture).

Part of the reason for the code and data memory usage reduction is the Target Compiler [15], which was especially designed for the CoolFlux architecture.

C. Latency and Throughput

The best useful throughput (MAC payload throughput) achieved on the SAND platform is 86 Kb/s. Each packet transmission included an 11B MAC header, a 2B MAC footer, and a 60B MAC payload. Including the MAC header in the calculation, the throughput reached 105 Kb/s. This result is far below the theoretical upper bound of 220 Kb/s as given in [16]. Our results are lower than the throughput of 140 Kb/s achieved by [16] with a Tmote Sky node running the Contiki Operating System in similar conditions. As another comparison, a data rate of 127 Kb/s with Jennic JN5121 wireless modules was reported elsewhere [17]. To understand the origin of the lower throughput, we analyzed our software constructs to determine the mechanism responsible for the poor throughput, or whether the results are intrinsic to the proposed dual scheduling design or to its implementation.

The authors of [16] claim that the bottleneck in single-hop transfer on microsensor platforms is neither the operating system nor clear-channel assessment, but rather, *serial peripheral interface (SPI)* packet copying. SPI data transactions are required between the microprocessor and the transceiver to load the TX-FIFO with data, and can also be triggered in the receive path by interrupts, or by sampling the status of the interface (i.e., polling). Fig. 4 shows a timing measurement realized on the SAND platform, with an interrupt-triggered SPI transaction. The duration of the SPI transaction is approximately 2 ms and represents one fourth of the total transmission duration. In an interrupt-triggered SPI transaction process, overhead comes from the context switch happening on each SPI interrupt and from the access to memory to fetch the next byte to transmit.

Fig. 4. Measured energy and time duration involved in the different steps of a packet transmission realized with the presented task-based MAC design.

Often, the overhead introduced by this method is longer than the transfer time. Consequently, for throughput purposes it is significantly faster to use polling.

The advantage of using an interrupt driven approach is the decoupling of the application code from the communication code, because calls to hardware are done asynchronously. The consequence is an interleaved execution of transmission and application thus increasing the processor utilization as less processor cycles are wasted on active waiting cycles during polling. Because of the above-mentioned consequences we made both interrupt-triggered and polling SPI control available to the application developer, for an optimization on either latency/energy/throughput or process multitasking according to application requirements. In our implementation we had chosen to optimize processor efficiency at the expense of communication latency and throughput. If available on a given microcontroller, DMA facilities could be used to further reduce CPU overhead, thereby increasing the SPI throughput.

The FreeRTOS context switch latency on the CoolFlux DSP was measured to be 17 μ s. During transmission using SPI polling, FreeRTOS context switches happen at each MAC tick interrupt, every 320 μ s, and at each FreeRTOS tick interrupt, every 1 ms. With approximately four ticks per millisecond, a FreeRTOS context switch of 17 μ s represents an overhead of about 7% on the available processing power, whenever the network stack is active; when the MAC queues are empty, and no MAC ticks are required, this is reduced to less than 2%. Overall, the SPI throughput performance depends on the processor architecture and SPI interface mechanism.

V. CONCLUSION

This article presented the design and implementation of a MAC architecture for LR-WPANs, in which MAC tasks are activated by a microkernel scheduler. This approach is in contrast to the typical interrupt-triggered MAC implementations,

and cleanly decouples the MAC implementation from that of the system software. Such a design enables portability and flexibility, while preserving software efficiency.

ACKNOWLEDGMENT

The authors would like to thank Albert Rietema and Peter van der Meer for their work on the software implementation of the MAC, and Elmahi Adam for his efforts on porting the MAC to the MSP430. Sheau Pei Chong performed the latency and throughput measurements.

REFERENCES

- [1] R. Min and A. Chandrakasan, "Mobicom poster: Top five myths about the energy consumption of wireless communication," in *Proc. SIGMOBILE Mob. Comput. Commun. Rev.* 7, San Diego, CA, 2003, no. 1, pp. 65–67.
- [2] B. Krishnamachari, D. Estrin, and S. B. Wicker, "The impact of data aggregation in wireless sensor networks," in *Proc. 22nd Int. Conf. Distrib. Comput. Syst.*, Vienna, Austria, 2002.
- [3] N. Jones, "Minimize your ISR overhead," *Embedded Systems Programming*, Jan. 2007.
- [4] *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for LR-WPANs*, IEEE 802.15.4 Standard-2003, 2003.
- [5] H. Roeven, J. Coninx, and M. Ade, "CoolFlux DSP—The embedded ultra low power C-programmable DSP core," in *Proc. GSPx 2004*, Santa Clara, CA, 2004.
- [6] R. Barry, FreeRTOS [Online]. Available: <http://www.freertos.org/>
- [7] M. Ouwerkerk, W. F. Pasveer, and N. Engin, "SAND: A modular application development platform for miniature wireless sensors," in *Proc. BSN 2006*, Boston, MA.
- [8] Texas Instruments, CC2420 2.4 GHz IEEE 802.15.4/ZigBee-Ready RF Transceiver.
- [9] M. Andree, Ed. *et al.*, Core Hardware Abstraction and Programming Model WASP European IST-034963 project public report, 2008.
- [10] R. D. Willmann *et al.*, "Home stroke rehabilitation for the upper limbs," in *Proc. 29th Annu. Int. Conf. IEEE EMBS Cite Internationale*, Lyon, France, Aug. 23–26, 2007.
- [11] J. H. D. M. Westerink *et al.*, *Probing Experience—From Assessment of User Emotions and Behaviour to Development of Products*, ser. Philips Research Book Series. New York: Springer, 2008, vol. 8.
- [12] M. Aoun, A. Schoofs, and P. van der Stok, "Efficient time synchronization for wireless sensor networks in an industrial setting," in *Proc. ACM Sensys 2008*, Raleigh, NC, Nov. 2008.
- [13] M. Aoun *et al.*, "Distributed task synchronization in wireless sensor networks," in *Proc. EWSN'09*, Cork, Ireland, Feb. 2009.
- [14] A. Schoofs *et al.*, "IP-based testbed for herd monitoring," in *Proc. IPSN'09*, San Francisco, CA, 2009.
- [15] "CoolFlux DSP C Programmer's Manual PLR-15542 Version 5.0, Target Compiler Distribution," Sep. 2005.
- [16] F. Osterlind and A. Dunkels, "Approaching the maximum 802.15.4 multi-hop throughput," in *Proc. HotEmNets 2008*, Charlottesville, VA, 2008.
- [17] Jennic, Calculating 802.15.4 Data Rates Application Note: JN-AN-1035, 2006.