IMPLICATIONS FOR PRACTICAL WIRELESS SENSOR NETWORKS

Unique Operating System Architecture and Transceiver Standards

David A. Border
College of Technology, Bowling Green State University, 1000 E. Wooster Avenue, Bowling Green, Ohio, U.S.A.

Keywords: Wireless sensor networks, Motes, Event driven, Multithreaded, Zigbee.

Abstract: Hallmarks of Wireless Sensor Networks (WSN) include their use in demanding environment, autonomous and untethered operation, low power requirements, miniaturization and low costs. Such hallmarks led to the understandable requirement that WSN sensors contain a specialized microprocessor unit; an innovative microcontroller unit which is strikingly dissimilar to general purpose CPUs found in the marketplace today. This paper examines two important aspects of current sensor node development work, that of microcontroller Operating System (OS) architectures and transceiver standards. The choices of OS designs are intended to match and complement the usefulness of the sensor node itself, while meeting hardware constraints (e.g. memory limitations). The paper details these design choices and how they are being met. Of equal interest, the paper discusses how the choice of transceiver standards for the WSN is determined by the overall design goal of device autonomy. One such "device autonomy discussion topic" relates to the reader how device power consumption levels are being reduced through the use of a newly developed transceiver standard.

1 INTRODUCTION

Consumer electronics, military and aerospace industries have driven a demand for increased miniaturization of electronic devices for many decades. The basic dynamics of the miniaturization push is understood by most. It is not difficult to cite examples in terms of improvements to the weight, size, and power consumption of new device families. More often than not, such improvements are often accompanied by significantly lower per unit cost-pricing. While electronics miniaturization is obviously pervasive throughout the industry and an obvious marketplace phenomenon, it is the prediction of and enumeration of new application concepts for these families, that is more difficult.

The miniaturization of sensors, computing and wireless (radio) networking and the ability of computers to analyze data from such devices in real time or near real time has opened a number of rather unique computing applications. Use of Wireless Sensor Networks built on miniaturized, often untethered, lower price devices have been actualized or suggested for a large number of application types (Zhao, F., et al., 2004). Here is a sampling of such types: conservation and habitat monitoring, airborne toxic chemical discharge monitoring, structural monitoring, and armament industry applications (e.g. intelligent minefields).

There are a number of design problems that need to be addressed when developing WSNs. Most stem from a single demand: device autonomy. From this condition power consumption limitations follow (Dutta and Culler, 2005). Power constraints affect the choice of processor families. Power constraints limit rate of sensor data block transmissions and block size. Power constraints dictate the need for special features, such as a processor and radio sleep states. Quite a number of design problems can be listed. This paper considers two design problems of the WSNs: the OS architecture and the transceiver standards.

2 PROCESSOR BACKGROUND

In 1971 Intel released the 4004, an IC typically acknowledged as the first commercial microprocessor. Matched to the microprocessor were a number of support chips. These formed a microprocessor development family. Since that time,

Border D. (2009).
IMPLICATIONS FOR PRACTICAL WIRELESS SENSOR NETWORKS - Unique Operating System Architecture and Transceiver Standards.
In *Proceedings of the International Conference on Wireless Information Networks and Systems*, pages 38-44
DOI: 10.5220/0002244700380044

Intel microprocessor families have gain complexity, functionality, and speed. The functionality-price ratio has also improved. This microprocessor maturation process has fit well to a market whose product scope consists of servers, workstations, desktop computers and laptops. Microcontrollers and Digital Signal Processors have served a separate computer market consisting of devices (typically classified as dedicated task processors or real time machines) with sufficient special I/O functionalities, instruction/bus speed and memory capacities to perform critical tasks. Some microcontroller families have followed a development path similar to that of Intel, i.e. development of larger and larger data bus widths, larger and larger address spaces, and more complex instruction sets, however this is not true of all microcontroller families.

An example of the class that has adopted a "non Intel" approach is illustrated by a number of microcontrollers that implement RISC architecture. The instruction set design is not the only difference, the accumulator/bus sizes also differ, with the RISC machines available in 4-bit, or 8-bit or 16-bit processor denominations. Further, some of these RISC microcontroller families have implemented low-power architectures that are suitable for non-AC, battery-only, applications. These families are reminiscent of the early Intel families, that is, they have modest bus sizes and reduced memory address spaces (such as 1K of memory space). However, unlike the old Intel processors, these devices have very small physical dimensions, operate at lower voltages and have much lower current draw. In addition, unlike the old Intel microprocessors these families contain features typical of some microcontrollers, such as built in analog-to-digital converters.

3 OS ARCHITECUTURE BACKGROUND

Over a number of decades, certain computer design practices or conditions have dominated for periods of time. With respect to operating systems, as manufacture of digital computers became more common, proprietary hardware/software systems enjoyed pre-eminence in the mainframe and minicomputer market. However, this practice has been eclipsed by other practices. With the advent of the mature desktop computer market, hardware has become more nearly a commodity piece for computer manufacturers. In addition, in many cases today operating systems kernel development is detached from the computer manufacturers themselves. For the most part, desktop and laptop operating systems are proprietary software enterprises of one company, Microsoft.

Microsoft has achieved a tremendous commercial presence and has sought enterprises with significant vertical and horizontal market integration. Microsoft has tuned a subset of its desktop OS and marketed it for small computing device applications, named Windows CE. This smaller footprint OS distribution has allowed Microsoft penetration of a segment of the wireless sensor network world, the Gateway WSN device. Other small computing device OS companies market proprietary OS software besides Microsoft. This includes Symbian with its mobile phone OS, SymbianOS.

Apart from Windows, a number of other OS systems vie for market share. Principally, these are found in the server market, although some are in the desktop, media and graphics markets. Among these are OSes are Unix, and Unix-like OSes. Examples are: Sun Microsystems and Apple. Sun uses a proprietary Unix version called Solaris. Apple uses the proprietary Unix version MacOS X. Linux, a Unix-like OS, is also found in this market. Interestingly, through virtualization software products (e.g. VMware) many of these OSes can jointly reside as guest OSes on the same server.

Concerning Linux, in a somewhat less than obvious OS creation mechanism, certain software development entities (foundations, groups, universities, commercial ventures and so forth) take its open source kernel and integrate a very needed software tool set (e.g. GNU) to it and produce a Linux distribution. This distribution base is made possible because of its kernel's public licensing scheme. Also, unlike Microsoft, Linux distributions have been ported to other CPU architectures (note, this is also true with a number of other Unix-like OSes). Thus, Linux is better poised to exploit a variety of hardware platforms than Microsoft. A number of Linux distributions are used in small computer and embedded computer applications.

While Linux has gained a widespread interest within the software development community in past years through public licensing, other OSes have also exploited this licensing, sometimes referred to as "free open source software" (FOSS). Some of these are real time operating systems, such as RTLinux and RTEMS.

4 PLATFORMS FOR OS ARCHITECTURE

Wireless Sensor Networks contain two basic computing device types: the sensor node and the gateway device. Since the gateway device may have relaxed power and size constraints when compared to the sensor node devices, its OS architecture will not be considered here. All attention will be given to the sensor nodes.

The sensor node by design consumes low power amounts, well under 1W. This restriction eliminates from consideration the Intel families (x86) found today within desktops, laptops, and servers. It also eliminates the microcontroller families that have grown in size and complexity from consideration for selection as the CPU of a sensor node. What remains are low power microcontrollers. Appropriate WSN low power microcontrollers include: Motorola AT90LS8535, ATMELS AVR series processors (specifically named: ATmega163 and ATmega128), and the Texas Instruments MSP430 (Polastre et al., 2005). The newer system-on-chip (SoC) architecture is stated to be preferable to non-SoC architectures for WSN work (Beck and Johnson, 2007). The SoC design is a practical way to achieve truly inexpensive, miniature, sensor nodes.

Some SoC computer architectures begin by realizing a CPU core based on well known non SoC chips. An example of such a device is: MC9RS08KA2 Series Microcontroller. This microcontroller is manufactured by Freescale and is based on their low power small dimension RS08 CPU core, a core modeled on their venerable HCS08 microcontroller family. Another example of such a device is the CC1010 Series Microcontroller. This microcontroller is manufactured by Chipcon (Texas Instruments) and is based on an 8051 compatible processor. An important point to consider for extreme miniaturization of CPUs for sensor applications is capability of the SoC to perform radio functions. A number of SoC computer architectures do integrate an on-chip radio transceiver. This section must be in one column.

5 OS ARCHITECTURE FOR WSN

The low wattage mandate coupled with the low cost and miniaturization goal for sensor WSN devices require that they have very low memory (RAM and ROM) capacities. This constrains OS design significantly (Gay et al., 2007). This disallows the use of a proprietary OSes such as Windows CE and non-proprietary OSes such as Linux for these devices since their OS features consume these finite resources. Instead the OSes found in the smaller market of embedded realtime or near realtime microcontrollers are considered the model to be followed when designing an OS to accomplish the tasks of program and memory management, hardware and data flow management.

To conserve resources, popular memory-constrained embedded systems OS architectures often rely on the “event driven” programming model rather than a multithreaded programming model (Dunkels et al., 2006). One expression of the event driven model is realized by implementing an OS program scheduler that posts “event specific tasks” to a program queue once an event happens. The program queue is emptied in a first in, first out fashion. They are executed in a single thread manner, with the thread itself being uninterruptible by other tasks or the scheduler (although tasks may include internal code to allow interrupts to programmatically affect their execution). This greatly simplifies memory architecture and memory demand. The scheduler, user applications and code components are all compiled into a single executable. Thus, all code (OS and user) shares the same memory addressable memory space. This single memory space design concept has its advantages, for instance, program debugging is much more straightforward. This OS architecture style has been adopted by the WSN OS “TinyOS” (Hill, et al., 2000). TinyOS (TinyOS v1 and TinyOS v2) is commonly cited as a standard for WSN OSes. To optimize OS performance, some embedded systems use specialized programming languages; for example, TinyOS is coded in nesC, a “C” style language extension. TinyOS first uses the nesC compiler to generate program code that is compatible to Gcc, the “C” compiler. A Gcc microcontroller compatible compiler then creates the output executable code.

The TinyOS architecture can be viewed as grouping of four software functionalities:

1. Scheduler and User Application
2. High Level Components
3. Synthetic Hardware Components
4. Hardware Abstraction Components

The components are the primary element in a TinyOS application. Each has interfaces that allow interactions with other components. Hence they are “wired” elements and can be put together to accomplish various sensor operations. They are named based on their function or device name.

Examples of TinyOS components by name include: Temperature, UART, I2C_bus, and RFM (RF Monolithics radio device).

One factor that separates WSN OS models from a number of other OS models in the embedded OS design community is the nature of the underlying hardware task. In many embedded OS applications, the OS seeks to service hardware that is primarily composed of non-communication intensive devices. In WSN, a primary service is to provide intensive data communication. This is reflected in TinyOS component structure, with a number of components devoted to transport of packets, bytes and bits within the OS. In general, sensor data can originate at the sensor or it can result from a transit operation (a "hop") from another WSN device. Data throughput is paramount to system performance. Blocking of/or waiting on I/O is not desirable. Loss of events because of scheduler program queue overflow is also not desirable (Decker et al., 2006). If an OS can justify a new native feature based on its benefits to I/O reliability, it will likely be accepted by the WSN community.

The consequence of the OS architecture using (generally) uninterruptible tasks, as in the case of TinyOS, can result in the un-expected blocking of subsequent event handling. A number of OS variants seek to remedy the lack of task prioritizing and task multithreading present in TinyOS. These include TinyMOS, Contiki, Mantis OS, t-kernel, and Nano-RK (Bhatti et al., 2005; Eswaran et al., 2005). Specifically, the OS features that these seek to achieve or improve on are:

1. Multithreading of user tasks
2. Assignable task priorities
3. Expanding task pre-emption abilities (implementation of time-slicing etc.)
4. Facilitating different operation on different hardware platforms (portability)

A negative consequence of coding multithread-ding capability into the OS is the adverse effect on memory and processor resources. In all cases, inclusion of multithreading will cause more overhead because of the need to implement OS thread switching actions. Additionally, dedicated stack memory assignments are needed for the thread management. Program debugging is also made more difficult in a multithreaded environment.

Besides the complexity associated with multithreading itself, the multithreaded approach creates unintentional deficits that must be corrected with new programming features (hence more OS complexity). An example of this is a critical energy savings features in event driven systems that are incidentally lost in multithreaded systems. Specifically, in TinyOS energy savings is assured because a task runs until completion and once all tasks are complete, the OS can invoke a simple hardware sleep state to conserve energy until the next event. It provides for a straightforward energy savings algorithm. In multithreading as program execution is switched from task to task, the trick is not to wait until all the tasks have ended to implement a sleep state. Rather to gain energy conservation in multithreading, it is desirable for the existing threads to invoke the sleep state programmatically, during, say, blocked I/O transactions. Thus the OS must be expanded to efficiently accommodate programmatic sleeps, as in the "usleep" C function.

Advantages can exist for the programmer who uses a more complex multithreaded WSN OSes as compared to simpler OS designs, such as the TinyOS. Consider the case of a user requiring a rather long-lived routine, say execution-to-completion in the 100 ms range in an event driven OS based system. To avoid overflows of bounded buffers, for operational reliability the programmer should recode the larger program into a cluster of smaller segments, each with shorter lifetimes; each whose design works against the ill effectives exhibited by long-lived routines. It can be argued that for this case the required programming skill level well exceeds the abilities of an average programmer. A more complex, interruptible, OS allows the programmer of average experience to code programs without worrying about whether he/she has written a long-lived program. If a long-lived program results from their work, the OS will seamlessly interrupt that code as needed to meet system needs and events thereby avoiding catastrophic outcomes. However, if long-lived tasks are not required in an application, then the usefulness of TinyOS as a sensor platform is undiminished. Further, development work performed using TinyOS benefits from its robust library of ready made program components.

Regardless of OS scheduling and threading designs, much of the OS development work is performed for the hardware dependency level. For WSN sensor devices typical hardware includes:

1. Clock (timers)
2. Serial Interface (e.g. I2C to temperature sensor)
3. ADC (to analog devices, e.g. light sensor)
4. Bitwise output (e.g. LEDs)

5. Radio Module
6. Interrupt(s)

While the amount of work is significant, its scope is adjustable. By choice, the OS can invest in a less expansive hardware interface and put the burden of device initialization, control and command with the programmer, or alternately it can perform these functions nearly transparently through well written supporting routines.

In WSN deployment, sensor devices network traffic is concentrated at a network edge in Gateway devices. Gateway devices are responsible for matching two different network standards, the transceiver wireless standard of the sensor nodes, and the network standard of the high level system (e.g. IEEE 802.3 or IEEE 802.11). If the sensors are deployed in a redundant manner, the gateways may be responsible for managing the sensor network (i.e. issuing startup-shutdown commands to the sensors), and managing data flow (Ilyas 2004).

Gateway devices, unlike sensor node devices, do not suffer the extreme constraints on size, cost and energy consumption.With x86 devices being offered in industrial form factors, such as the PC/104 form, its CPUs are suitable for use in WSN installations. In turn, this allows incorporation of a Windows or Linux OS into the device. Somewhat surprisingly however, commercial WSN gateway manufacturers have chosen to skip use of x86 devices, in favor of RISC machines, such as that of Advanced RISC Machine architecture (ARM, Xscale) machines. The choice of these RISC machines leaves the OS selection process open to a small set of Linux OS variants and Windows CE. Since the RISC machines have a long history of use in handheld battery powered computers, they are well suited for the WSN environment.

6 BACKGROUND WIRELESS RECEIVER STANDARDS

There are a number of ways wireless transceiver standards can be categorized. One way is to group them by spatial coverage, another by intended market. Classifying wireless into spatial coverage yields three distinct groupings by area of service:

1. Wide
2. Local
3. Personal

Classifying into market type yields three categories:

1. Mobile Telephony
2. Mobile Internet
3. Personal Devices

The within the categories standards will vary with respect to the frequency band used, the communication techniques and data rates employed, message syntax and data protocol. Between the categories, the energy consumption requirements are likely to differ. For example, in some desktop deployments schemes mobile internet (WiFi) is used so that the tasks of running cables and installing jacks are minimize; in this example energy consumption is not critical. This is in contrast to mobile telephony where minimization of energy consumption is always desirable.

7 WSN TRANSCEIVER STANDARDS

The same constraints that exist during the determination of a WSN OS architecture exist for determination of a WSN Transceiver Standard. A primary consideration is power; the transceiver standard must consume low levels of power. WSN sensors do not have the luxury of an AC-based recharge as do mobile cellular phones. A secondary consideration is the protocol complexity, where complexity is assumed to correspond directly to length of program code. The second consideration is interrelated to the first since long programs consume more energy than short programs. For these reasons, the ideal WSN transceiver standard is a low power, low complexity network protocol.

The power constraint and complexity constraint can be used to eliminate certain transceiver standards from consideration. Consider the IEEE 802.11b standard. Its power consumption would drain a few AA batteries in a matter of hours. Its protocol complexity would require a significant increase in program code and program memory. The rather quick data rate of the standard (11 Mbps max), while attractive, is overkill for sensor nodes recording temperatures, performing light sensing, and other low speed acquisition and reporting tasks.

The power constraint can be used to eliminate other existing transceiver standards as well. In discussions of digital communication systems, a primary design concern cited is the maximizing the Bits/s/Hz figure. With power constraints, the importance of the Bits/s/Hz figure is discarded in favor of J/Bits/s/Hz. When considering power issues

three regions of power consumption are considered, the transmitter, the power amplifier, and the receiver. A common mobile low power device, the cellular phone, uses transceiver standards that provide for transmission distances greater than 1000 meters. At these distances, much of the energy consumed is consumed by the analog power amplifier section. The power levels consumed in this section are well above those levels that are acceptable in WSN work. Therefore WSN applications restrict themselves to ranges under 200m or so.

While the energy per bit associated with cellular transmissions standards eliminate them for consideration in WSN applications, certain cellular phone standards are of value. For example, cellular phone units mitigate the power amplifier consumption by use of power management schemes. These schemes minimize "on" time in favor of low power idle states or equivalent states. Additionally, newer cellular devices seek to minimize the analog circuitry in favor of lower power CMOS digital devices. Energy-aware transceivers make use of these concepts (Schurges, C., 2002).

Finally, the choice of signaling scheme is also critical. Here, low power devices can choose to sacrifice spectrum for power efficiencies. Thus highly spectral efficient signaling schemes such as QAM and M-ary modulation are rejected in favor of power efficient schemes. Certain schemes that are constant envelope modulation (e.g. Frequency Shift Keying - FSK) or near constant envelope modulation (e.g. Offset Quadature Phase Shift Keying - OQPSK) are well matched for use with the power efficient direct modulation transmitter architecture (Otis, et al., 2004). Data rates for WSN sensors are intended to be modest, running well under 1 Mbps. It is assumed that higher rate devices such as realtime video would require specialized solutions outside the WSN norm.

8 TOWARDS A TRANSCEIVER STANDARD SOLUTION

The wireless communities' standards for wireless personal network (WPAN) are lead by the IEEE 802.15 standards. This is the standards group that is most likely to attract placement of specialized low power, low range, and low firmware/software complexity data communication standards. Indeed this is the case; IEEE 802.15.4 is one such new standard that has risen to meet these requirements. Interestingly, while being a standard, it is simultaneously a fee-based supported membership organization named the Zigbee Alliance. The practice of an alliance operating in conjunction with a standard is not uncommon.

Zigbee is a low cost device specification that operates in the industrial, scientific and medical (ISM) radio band. It implements energy conserving modulation schemes (e.g. FSK, OQPSK). It supports network topologies and network operation in a fashion that avoids the software and firmware complexities found in standards such as IEEE 802.3 and IEEE 802.11. It supports sleep states that are important to meeting WSN energy requirements. Its overall energy requirements are modest due to its reliance on CMOS digital circuits and minimization of analog components. This combined with programmatic use of sleep states (etc.) allows Zigbee transceiver devices to operate over extended periods using batteries.

Recently (summer 2007), through the work of Nokia, the existing Bluetooth standards, the IEEE 802.15.1 standard, has been expanded to include an ultra low power separate Bluetooth communication definition, called Wibree. One point of differentiation between the two standards is the device range. Zigbee indoor range is about 30m, while Wibree has a lower range value of 10m. Another point of difference is their method of their implementing network topologies. Also, Zigbee has already been incorporated into existing WSN nodes (e.g. MicaZ, Telos sensors), while Wilbree hasn't. Finally, despite Zigbee's status as a new device, it already exhibits a certain maturity as its radios have already been combined with microcontrollers to produce a SoC device suitable for WSN applications. An example of a SoC device used in academic studies of sensor node operation is TI's CC2430 (Leopold, et al., 2007). The future of Wibree's use in WSNs is not clear at this time.

9 SUMMARY

The needs of the wireless sensor network community are being met through innovative developments in microcontroller operating systems and wireless personal networks. In each case the existing models and architectures have been adjusted to meet energy constraints, size constraints, and cost constraints unique to WSN work. For the transceiver, modulation schemes and transmitter architectures have been selected to minimize power consumption. For designers from other backgrounds, some of these selections are somewhat surprising

since they reject commonly sought after goals, such as maximum bit throughput, and radio range.

Considering the operating system, an OS designer from another background might be surprised by the issues at hand. Gone are the incentives for large scale memory addressing capabilities, incorporation of parallel processing for dual core machines and other such features. In contrast, the WSN sensor node OS and application task(s) is accomplished by TinyOS in a single address space and as a single piece of code. Programmers with a background of embedded systems are more likely to be comfortable with the size constraints of the WSN development than others. But as discussed for WSN sensor nodes, both power constraints and good data throughput are critical to OS success. Not all embedded applications has this set of constraints.

From what has been identified in this discussion it is evident that the existing WSN solutions will be challenged to improve as time progresses. In terms of OS development, developers will continue to test and improve the "event driven" model and the "multithreaded" model approaches. Additionally, developers will continue to work towards determining the best host programming language for their particular OS (e.g. nesC, C). Developers will also continue to port WSN OS solutions from one hardware platform to another. However, regardless of future improvements it can be stated that to date the unique architectural challenges for both the OS and transceiver of wireless sensor networks have been successfully met.

REFERENCES

Zhao, F., Liu, J., Cheong, E., Dutta, P., and Whitehouse, P., 2004. Wireless Sensor Networks: Seamless Computing across the Physical and PC Worlds. In *Talk at Microsoft Research Faculty Summit 2004, Redmond, WA*, http://research.microsoft.com/~zhao/talks/Zhao_FacultySummit04_p.pdf.

Dutta, P. K., and Culler, D. E., 2005. System Software Techniques for Low-power Operation in Wireless Sensor Networks. In *Proceedings of the 2005 International Conference on Computer-Aided Design (ICCAD'05) 2005, pp. 925-932.* ICCAD'05.

Polastre, J., Szewczyk, R., and Culler, D., 2005. Telos: Enabling Ultra-low Power Wireless Research. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks: Special Track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS), April 2005.* IPSN/SPOTS.

Beck, N., and Johnson, I., 2007. Shaping TinyOS to Deal with Evolving Device Architectures: Experiences Porting TinyOS-2.0 to the Chipcon CC2430. In *Proceedings of the 4th workshop on Embedded Network Sensors (EmNets '07), June 2007, pp. 83-87, ISBN 978-1-59593-694-3.* EmNets '07.

Gay, D., Levis, P., and Culler, D., 2007. Software Design Patterns for TinyOS. In *ACM Transactions on Embedded Comput. Syst. 6, 4, Article 22, September 2007, 39 pages.* ACM.

Dunkels, A., Schmidt, O., Voigt, T., and Ali, M., 2006. Protothreads: Simplifying Event-driven Programming of Memory-constrained Embedded Systems. In *SenSys '06: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, 2006, ISBN 1-59593-343-3, pp. 29-42, Boulder, Colorado,* USA. ACM, New York, NY, USA.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K., 2000. System Architecture Directions for Networked Sensors. In *SIGPLAN Not. 35, 11, Nov. 2000, pp. 93-104.* SIGPLAN.

Decker, C., Riedel, T., Peev, E., and Beigl, M., 2006. Adaptation of On-line Scheduling Strategies for Sensor Network Platforms. In *2006 IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS), Issue, Oct. 2006, pp. 534–537.* IEEE.

Bhatti, S., Carlson, J., Dai, H., Deng, J., Rose, J., Sheth, A., Shucker, B., Gruenwald, C., Torgerson, A. and Han, R., 2005. Mantis OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms. In *ACM/Kluwer Mobile Networks and Applications (MONET), Special Issue on Wireless Sensor Networks, 2005.* ACM.

Eswaran, A., Rowe, A., and Rajkumar, R., 2005. Nano-RK: An Energy-Aware Resource-Centric RTOS for Sensor Networks. *In Proceedings of the 26th IEEE International Real-Time Systems Symposium December 5-8, 2005, RTSS. IEEE Computer Society, Washington, DC, pp. 256-265.* IEEE Computer Society.

Ilyas, M., 2004. *Handbook of Sensor Networks: CompactWireless and Wired Sensing Systems.* CRC Press.

Schurges, C., 2002. Energy-aware Wireless Communications, In *Ph.D. thesis.* University of California Los Angeles.

Otis, B.P., Chee, Y.H., Lu, R., Pletcher, N.M., Rabaey R.M., 2004. An Ultra-Low Power MEMS-Based Two-Channel Transceiver for Wireless Sensor Networks. In *Symposium on VLSI Circuits, 2004. Digest of Technical Papers, June 17-19, 2004, pp. 20- 23. ISBN: 0-7803-8287-0.* IEEE.

Leopold, M., Chang, M., Bonnet, P., 2007. Characterizing Mote Performance: A Vector-Based Methodology. In *Technical Report no. 07/06 Dept. of Computer Science University of Copenhagen Universitetsparken 1 DK-2100 Copenhagen, Denmark, ISSN: 0107-8283, 2007.* University of Copenhagen.