An Ultra Low Power System Architecture for Sensor Network Applications

Mark Hempstead, Nikhil Tripathi, Patrick Mauro, Gu-Yeon Wei, David Brooks
Division of Engineering and Applied Sciences
Harvard University
{mhempste, nikhil, mauro, guyeon, dbrooks}@eecs.harvard.edu

Abstract

Recent years have seen a burgeoning interest in embedded wireless sensor networks with applications ranging from habitat monitoring to medical applications. Wireless sensor networks have several important attributes that require special attention to device design. These include the need for inexpensive, long-lasting, highly reliable devices coupled with very low performance requirements. Ultimately, the "holy grail" of this design space is a truly untethered device that operates off of energy scavenged from the ambient environment. In this paper, we describe an application-driven approach to the architectural design and implementation of a wireless sensor device that recognizes the event-driven nature of many sensor-network workloads. We have developed a full-system simulator for our sensor node design to verify and explore our architecture. Our simulation results suggest one to two orders of magnitude reduction in power dissipation over existing commodity-based systems for an important class of sensor network applications. We are currently in the implementation stage of design, and plan to tape out the first version of our system within the next year.

1. Introduction

Wireless sensor networks are poised to transform the way society interacts with the physical world, driven by an explosion of systems research in sensor networks. Sensor networks have been proposed and deployed for a wide variety of applications such as habitat monitoring [13, 23], structural monitoring, and emergency medical response [7, 12]. While the application space seems limitless, it is actually limited by the operating lifetime of the battery-operated wireless sensor nodes. Current deployments rely on commercially available wireless sensor network devices (e.g., Mica2 [4]). Such devices typically consist of a basic microcontroller, a radio, and a variety of (often MEMS-based) sensors. One of the main limitations of these platforms is that they are built using commodity chips, which themselves are not specifically designed for wireless sensor networks. As a result, they suffer several inefficiencies that lead to high power consumption and limited operational lifetimes. To address this limitation, this paper presents the design and analysis of an ultra low power device specifically for sensor network applications. In these systems, the CPU, radio, and sensor devices are responsible for the majority of the total system power and we show that the general-purpose nature of commodity microcontrollers results in inefficient power usage, presenting an opportunity to significantly reduce its power.

This paper outlines our design approach, which studies all levels of the system design from the applications down to the circuits and even the choice of process technology. This holistic approach enables us to uncover architectural and circuit-level design tradeoffs that guide design decisions in order to meet our low power goals and long lifetime requirements. The power target of the system architecture is 100 μW for normal workloads. We chose this power level with the ultimate objective of implementing a truly untethered device that can operate indefinitely off of energy scavenged from the environment.

The intermittent, event-driven nature of sensor network application workloads motivates several architectural design features. We optimize the architecture for frequent, repetitive behavior that is characteristic of sensor network applications. These optimizations include hardware acceleration and offloading immediate event handling from the general purpose computing component. Features such as event-driven computation improve performance (thus permitting a slower system clock) and reduce power consumption by eliminating unnecessary operating system overhead. In order to meet the long-lifetime demands of many wireless sensor network deployments, our architecture enables fine-grain power management to minimize extraneous dynamic and static power consumption. Efforts to minimize idle (or leakage) power has led us to investigate tradeoffs between process technology generations. Given the relatively low performance requirements of the sensor nodes,

Device/Mode	**Current**	**Device/Mode**	**Current**
CPU		*Radio*	
Active	8.0mA	Rx	7.0 mA
Idle	3.2mA	Tx (-20 dBm)	3.7mA
ADC Acquire	1.0mA	Tx (-8dBm)	6.5mA
Extended Standby	0.223mA	Tx (0 dBm)	8.5mA
Standby	0.216mA	Tx (10 dBm)	21.5mA
Power-save	0.110mA	*Sensors*	
Power-down	0.103mA	Typical Board	0.7 mA

Table 1. Mica2 platform current draw measured with a 3V power supply.

we argue that the most advanced process technology is not necessarily the lowest power solution.

The rest of the paper is organized as follows: Section 2 presents a brief background of wireless sensor network devices and related work. Then, Section 3 presents our study of wireless sensor network applications. Section 4 details our system architecture and explains several architectural optimizations made to reduce power consumption. Subsequently, our study of process technologies and circuit design optimizations are presented in Section 5. Section 6 presents system-level simulation results (supported by power data extracted from key circuit-level simulations) that validate our architecture.

2. Background and Related Work

We leverage active systems research in wireless sensor networks to provide insights and details of power consumption in currently available hardware platforms. We first set out to determine whether efforts to reduce power in the computational component of a sensor node is warranted, dictated by Amdahl's law. Power consumed for radio communication is generally recognized to be significant and is the focus of much research effort to reduce power consumed by the circuitry itself and to minimize radio usage [16, 26]. While radio power is indeed significant, power required for computation can also be appreciable. The PowerTOSSIM project [21] studied the power consumption of the widely used Mica2 mote available from Crossbow [4]. A summary of power consumed by the CPU, sensors, and radio is presented in Table 1. The table shows that active CPU and radio power numbers are comparable. Given the ability to operate the CPU in both active and idle (or lower power) modes, these numbers do not present the complete picture. The computational demands of the application determine the CPU's actual activity and radio usage. The PowerTOSSIM paper provides a detailed breakdown of energy consumed by different components for a variety of applications. In these results, the CPU power ranges from 28% to 86% of the total power consumed and roughly 50% on average. Furthermore, data filtering and more efficient communication protocols can shift activity from the radio to the CPU. Although there may be ways to reduce CPU power in existing hardware platforms, there is an innate inefficiency associated with using general purpose CPUs for sensor network workloads. Simulation results in Section 6 reveal the potential for significantly reducing power consumption when the computational unit uses an architecture designed to leverage the event-driven nature of sensor networks.

Several organizations are actively involved in designing hardware for sensor network devices. The devices that have been used widely for research and in some commercial deployments, such as the Mica2 [4] and Telos [17] motes, employ general-purpose microcontrollers that do not efficiently handle interrupt processing. However, the primary task of a sensor network device is to handle timer and external interrupts since their applications are inherently event driven [9]. Therefore, these devices must run an event-driven operating system (TinyOS [10]) to mask the deficiencies of the hardware platforms that have not been designed specifically for sensor networks. The first custom device for sensor networks is the Spec architecture [9], which includes hardware accelerators for tasks such as message start-symbol detection. In fact, the newer generation radio chips incorporate some of these features [3]. In our architecture, we intend to use accelerators not merely to improve performance, but also as a power-saving measure. The SNAP architecture, which is an asynchronous design initiative described in [5], is the only example of an event-driven architecture for sensor network devices that we have come across in literature. However, the SNAP architecture does not exploit the powerful event-driven paradigm apart from getting rid of the TinyOS overhead. In other words, its primary computing engine is still a general purpose microcontroller that must remain powered on all the time, even when events occur rarely, thereby incurring leakage power. The Smart Dust project out of UC Berkeley developed a general-purpose microcontroller with low-power design techniques for use in a sensor network device [25]. All of the existing architectures for wireless sensor network devices fail to optimize common-case behavior of applications, because they all suffer from the overly general-purpose nature of the primary computing engines. Our design approach seeks to fully leverage the event-driven nature of applications as we pursue the design of next-generation low-power sensor network nodes.

Scavenging energy from the environment and using this energy to power the sensor network device would greatly increase the effective lifetime of a wireless sensor node. There are many sources of energy available in the environment such as solar, vibration, and electro-magnetic radiation, and researchers have developed techniques to harness this energy [20]. For example, vibrational energy can be translated into electrical energy through piezoelectric materials that induce an open circuit voltage when placed under mechanical stress. While using vibration as an energy source is promising, the power output is limited to the order of a hundred μW (for mote-size devices). The PicoRadio project out of UC Berkeley built a proof-of-concept transmitter that operates at very low duty cycles while powered off of solar and vibrational energy [19]. Based on these demonstrations and the belief that energy-harvesting technology will improve, we have set the design target of our device at 100μW.

3. Sensor Network Applications

During the initial phase of our design process, we studied the wireless sensor network application space to understand the computational needs of sensor network workloads. Hardware requirements vary widely depending on the projected lifetime, computational complexity, and communication needs of the deployment. We found that the monitoring class of applications, characterized by low duty cycles, long deployment lifetimes, and regularity of operation provide well-defined and interesting constraints for sensor node design.

Typical monitoring applications can be broken down into a clear set of regular tasks. Nodes typically complete several data generation tasks that include taking sensor samples, preparing messages containing data, and sending radio messages. Nodes also complete ad-hoc routing tasks such as receiving messages, looking up routing information, and sending radio messages.

The interval of sensor readings depends on the phenomenon being measured and these rates are typically very low. UC Berkeley's Great Duck Island (GDI) application measured all sensors every 70 seconds, then transmitted a packet [13, 23]. Harvard's deployment of sensor nodes to measure infrasound on the Tungurahua volcano measured samples at 100 Hz and sent 4 radio messages a second with 25 samples per packet [8]. While both of these applications transmitted to a base station that was one hop from the sensor nodes, other deployments may require nodes to also serve as communication relays due to large physical separation between nodes.

The ultimate goal of a monitoring deployment is to provide continuous sensing for years to decades without being touched. Past deployments of sensor networks for environmental monitoring had limited lifetimes (a few weeks or months) due to the relatively high power consumption of commodity hardware. Therefore, an ultra low power system is required to achieve these deployment goals. The next section describes the goals and implementation of our architecture, which is designed specifically for this application class.

4. System Design and Architecture

4.1. Motivation and Goals

Our architecture replaces most of the functionality of a general purpose microcontroller with an event-driven system specifically optimized for monitoring applications. A summary of our design goals for the system architecture are presented below and detailed discussions of goals and how the architecture meets these goals follows in the next subsection.

1. *Event-Driven Computation:* Eliminate unnecessary event-processing overhead with an event-driven hardware platform.
2. *Hardware Acceleration to Improve Performance and Power:* Build a system composed of several components that are optimized for specific tasks.
3. *Exploiting Regularity of Operations within an Application:* Optimize the common-case behavior within an application.
4. *Optimization for a Particular Class of Applications:* Optimize the common-case behavior of monitoring applications to reduce power, while still providing general-purpose processing capability to enable broad functionality.
5. *Modularity:* Provide an easily extensible system architecture that allows different sets of hardware components to be combined into a larger system targeting a particular application.
6. *Fine-grain Power Management Based on Computational Requirements:* Provide explicit programmer-accessible commands for fine-grain resource and power control.

4.2. Architecture Description

To fulfill our design goals, we seek to replace the basic functionality of a general-purpose microcontroller with a modularized, event-driven system described in this section. The system architecture is illustrated in Figure 1. There are two distinct divisions within the system in terms of the positions of the components with respect to the system bus. We refer to the components to the right of the bus as *slave* components and those to the left as *master* components (except the memory, which is a slave). The system bus has three divisions – data, interrupt, and power control. The slaves compete for the interrupt bus using centralized arbitration if more than one slave has an interrupt to signal. The slaves also respond to read or write requests from the master side on the data bus, thus allowing the masters to read their information content and control their execution. Power control lines are explained later.

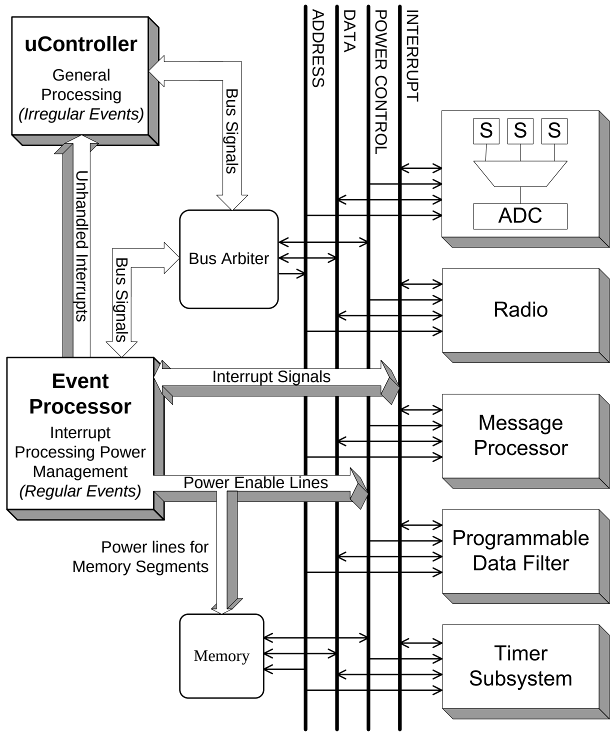


Figure 1. Block Diagram of System Architecture

The features of the architecture are best understood in the context of design goals listed in the previous subsection.

4.2.1. Event-Driven System We propose an event-driven system in which all of the master components are involved with event handling, and the slaves assist the master components in their tasks and signal the occurrence of events to trigger the master components. All external events, such as the beginning of radio packet reception, are expressed as interrupts by an appropriate slave component. The slave components also raise interrupts for their internal events, such as completion of an assigned task. To the master components, there is no distinction between external and internal events. Also, since the occurrence of all events is signaled by interrupts, we will use the terms *event* and *interrupt* interchangeably. The system idles until one of the slaves signals the presence of an event, and when all outstanding events have been processed, the system returns to its idle mode. Since all the system does is respond to events, there is no software overhead for interrupt handling.

4.2.2. Hardware Acceleration to Improve Performance and Power There is a general-purpose microcontroller in our system. However, unlike other sensor network device architectures, the intent of our design is for the microcontroller to be the last resort for any computation, i.e. the microcontroller should be called upon to perform a task only if the rest of the system does not have the requisite functionality. Specific tasks that are considered common to a wide variety of application are offloaded to hardware accelerators, which can be more power and cycle efficient than the general-purpose microcontroller. Hence, the microcontroller can usually be powered down by gating the supply voltage. This not only reduces active power but also leakage, which can be a very significant source of power consumption for low duty cycle operation. Some questions that arise are: How are the hardware accelerators configured for their tasks? How are interrupts handled while the microcontroller is asleep? The answers to these questions are deferred to the discussion of how the architecture exploits regularity of operations.

All of the hardware accelerators in the system are slave components. There is a timer block that sets alarm events, which may be used to sample data from the sensors, in a Time-Division Multiple Access (TDMA) radio scheme, or for any tasks to be performed at regular intervals. In the absence of a hardware timer, a software timer would have to be implemented in the microcontroller, requiring the microcontroller to always be active. There is a generic filter slave for basic data processing. In our architecture, this block is a simple threshold filter with a programmable threshold. We also implement a message processor block to offload packet processing and avoid waking up the microcontroller for common events such as packet forwarding and transmitting packets of collected samples. The slave components also include essential sensor network device components such as the radio, and a block of sensors and Analog-to-Digital Converters (ADCs).

4.2.3. Exploiting Regularity of Operations within an Application All immediate interrupt handling is offloaded to the *event processor* block while the microcontroller is powered down. The event processor is a simple state machine that can be programmed to handle an interrupt by transferring data blocks between the slave devices and setting up control information for these devices to complete their tasks.

The event processor can also be programmed to wake up the microcontroller if the requisite functionality for processing the interrupt is not otherwise available. To some extent, the event processor can be perceived as an intelligent DMA controller. Thus, there are two levels of interrupt service routines (ISRs) to handle an interrupt: at the event processor level and at the microcontroller level. ISRs for both the event processor and the microcontroller are stored in the main memory, which is a unified instruction and data memory connected to the bus.

We now elaborate further on the notion of *regular* and *irregular* events. A regular event is one that can be processed wholly by the event processor and the slave components. An irregular event is one that requires the microcontroller. One of the tasks involved in mapping an application to our system is to determine the partitioning of events into regular and irregular events. The regularity of an event is determined by the functionality present in the slaves and the event processor. For a typical application, events such as sampling, transmitting samples, and forwarding packets would ideally be regular while application or network reconfigurations would often be classified as irregular.

4.2.4. Optimization for Monitoring Applications The system does not seek to satisfy any real-time requirements and only one outstanding interrupt is possible. As a result, slave devices may continue to write their interrupts to the interrupt bus. However, if the system begins to be overloaded, events will simply be dropped. Outstanding events cannot preempt either the microcontroller or the event processor. The system bus, the microcontroller, and the event processor are all non-pipelined. All of these simplifications give rise to a light-weight system that is well suited to handle monitoring applications while consuming very little power.

4.2.5. Modularity All of the slave devices are attached to the system bus and are memory-mapped. Both control and data are communicated to and from the slaves by simply reading from and writing to appropriate addresses in the memory. Thus, the event processor is not aware that data is being transferred between separate slaves, or that control information is being written to the slaves. This memory-mapped interface allows the system design to be extremely modular and new components (and hence new functionality) can be added on to the system bus without modification of the event processor or the microcontroller.

4.2.6. Fine-grain Power Management Based on Computational Requirements Since the master components are triggered by interrupts, the ISRs for each interrupt can configure the system according to its computational requirements for handling the interrupt. To sufficiently curb leakage power, special instructions within the event processor are used to gate the supply voltages of system components. Note that the system does not infer the resource usage for an event; rather, the ISR programmer selects the components to turn on depending on the needs of the application. Individual power enable lines are required for each component under direct control. Vdd-gating and power down implementations will vary depending on the circuit-level design of the individual slave components. Such power control may not only be exercised over the slave components, but also over segments within the main memory that contain temporary data, such as application scratch space. We believe that event-driven programmable resource usage is one of our most significant innovations. It allows configuration of system power consumption with very little logic overhead, as opposed to a technique that attempts to infer resource usage. Also, it allows the addition of several specific components to the system as slaves that can be used in varying combinations to provide the functionality required by an application. Any component unused in an application can be turned off (i.e., supply voltage gated) and is nearly invisible during the entire lifetime of the application. Therefore, the system can satisfy the general-purpose requirements of applications by providing a broad range of slave components, enabling an on-demand functionality that imposes negligible overhead when a component is not required.

4.3. System Components

We now discuss some of the more interesting system components in greater detail.

4.3.1. System Bus As discussed earlier, the system bus comprises the data bus, the interrupt bus, and power control lines. The data bus has address, data, and control signals indicating read and write operations. In our current implementation the address bus has 16 lines, the data bus has 8 lines, and there is one control signal each for read and write operations. The address space for our memory-mapped architecture is therefore 64K. The address and control lines can be driven only by the event processor and the microcontroller in mutual exclusion as determined by the bus arbiter, which is currently just a mux. The data lines are driven by the slave that determines that the current request lies in its address range, and are demultiplexed to the originator of the request, i.e., the event processor or the microcontroller.

The interrupt bus has 6 address lines and control signals for arbitrating the writing of interrupts by slaves. The system is therefore capable of handling 64 interrupts in the current model. The event processor has control signals in the interrupt bus to indicate when it has read the current interrupt address.

The power control lines are handshake pairs for each slave or memory segment controlled. The handshake is relevant only when a component is turned on, to determine the time when the component can be used. The system currently makes no assumptions about the time taken to wake up for the components over which explicit power control is exercised.

4.3.2. Microcontroller The microcontroller is used to handle irregular events, as discussed in previous sections, such as system initialization and reprogramming. The microcontroller is a simple non-pipelined microcontroller. It implements an 8-bit Instruction Set Architecture (ISA). We plan to leverage currently available computational cores with necessary modifications for our low-power features.

4.3.3. Event Processor The event processor is essentially a programmable state machine designed to perform the repetitive task of interrupt handling. Figure 2 illustrates a

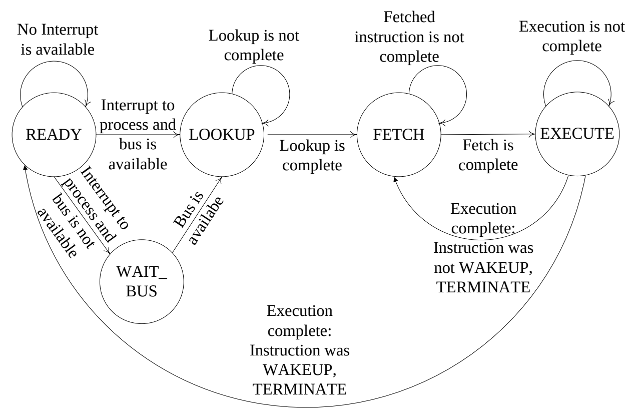

No Interrupt is available
Lookup is not complete
Fetched instruction is not complete
Execution is not complete
READY
Interrupt to process and bus is available
LOOKUP
Lookup is complete
FETCH
Fetch is complete
EXECUTE
Interrupt to process and bus is not available
Bus is availabe
WAIT_BUS
Execution complete: Instruction was not WAKEUP, TERMINATE
Execution complete: Instruction was WAKEUP, TERMINATE

Figure 2. Event Processor State Machine

simplified version of the actual state machine within the event processor. Because the event processor is an important component of our architecture we now explain its functionality in detail. The event processor idles in the READY state until there is an interrupt to process. When an interrupt is signaled, the event processor transitions to the LOOKUP state if the data bus is available, i.e., the microcontroller is not awake. If not, the event processor transitions to the WAIT_BUS state and waits until the microcontroller relinquishes the data bus. In the LOOKUP state, the event processor looks up the ISR address corresponding to the interrupt. The lookup table is stored in memory, and the starting location of the table, offset by an amount proportional to the interrupt address, contains the address of the event processor ISR. When the lookup is complete, the event processor transitions to the FETCH state, in which the first instruction at the ISR address discovered in the LOOKUP state is fetched. The event processor stays in the FETCH state until all the words of the current instruction have been fetched, and then it transitions to the EXECUTE state.

The instructions within an event processor ISR can be one of the following – SWITCHON, SWITCHOFF, READ, WRITE, WRITEI, TRANSFER, TERMINATE, or WAKEUP. Table 2 provides a summary of the operations corresponding to the instructions. The event processor has one register used to store temporary data. The opcodes are each 3 bits and the instructions vary in the number of words they span.

The EXECUTE state holds until the instruction has been completely executed, e.g., the complete transfer has been completed for a TRANSFER instruction. A component is completely powered on for the SWITCHON instruction. If the instruction is not a WAKEUP or TERMINATE instruction, the event processor returns to the FETCH state and fetches the next instruction in the ISR for execution. For WAKEUP or TERMINATE instructions, the event processor returns to the READY state and waits for the next interrupt.

4.3.4. Timer Subsystem The timer subsystem consists of a set of four 16-bit timers in our current implementation. Each timer is essentially a counter that counts down to zero from a pre-configured value, and then generates an alarm event. The timers can be chained to allow alarm events to be generated for larger intervals of time. Each timer can be paused, disabled, and reconfigured.

4.3.5. Message Processor Our architecture enables hardware accelerators designed for specific tasks. For example, our architecture uses a message processor block to handle regular message processing tasks, including message preparation and routing. Simple tasks such as table lookup and check-sum calculations can be sped up using hardware implementations (with low power overhead).

Currently, the message processor interface has two memory blocks for each message as well as memory-mapped control words. Data is transfered to the message processor from sensor devices and once the message has been prepared the message processor fires an interrupt and the message is sent to the radio. All incoming messages are transfered from the radio to the message processor. If the message is a regular message, the message processor looks up whether the message should be forwarded. If the message is an irregular message, then an interrupt is fired and the event processor wakes up the microcontroller. Our message processor model handles standard 802.15.4 packets [28].

4.3.6. Radio Like the new Telos mote, our architecture interfaces with the low-power CC2420 802.15.4 radio from ChipCon [3]. This radio provides hardware support for tasks such as start-symbol detection, error detection, etc., and we plan to take advantage of these features as they are consistent with our system design approach. A simple radio model enables us to fully test our system architecture concepts without having to explicitly build a transceiver.

5. Process Technology and Circuit Techniques

In addition to architectural innovations, circuit techniques and the choice of process technology can significantly impact the power consumed by the sensor nodes. This section presents the results of a process technology simulation study and the architecture and circuit design of a low-power SRAM. Traditionally, the choice of process technology has been straight forward. To push the envelope of performance, the most advanced technology with the smallest feature size and smallest parasitic capacitance should be used. However, subthreshold leakage current is becoming a significant fraction of the total power in designs that use advanced deep-submicron process technologies [1]. When choosing a process technology for sensor network hardware one must choose the technol-

Instruction	Size	Description
SWITCHON	One word	Turn on a component and wait for acknowledgment that the component is ready to proceed.
SWITCHOFF	One word	Turn off a component
READ	Three words	Read a location in the address space and store to the register
WRITE	Three words	Write a location in the address space from the register
WRITEI	Three words	Write an immediate value to a location in the address space
TRANSFER	Five words	Transfer a block of data within the address space
TERMINATE	One words	Terminate the ISR without waking up the microcontroller
WAKEUP	Two words	Terminate the ISR and wake up the microcontroller at a microcontroller ISR address

sor Instruction Set

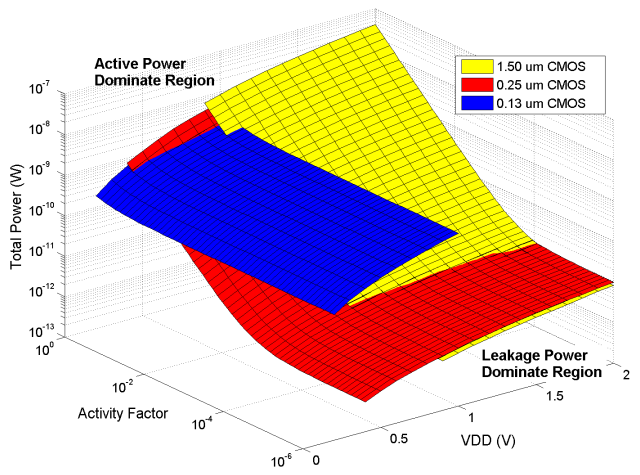


Figure 3. Analysis of process technology for low-duty cycle sensor node applications

technologies consume less active power. We assume a synchronous design that operates off of a globally distributed clock.[1] To account for both sources of power, we used Equation 1 to model the active and leakage power tradeoff,

$$P_{total} = \alpha(T/T_{target})P_{active} + (1-\alpha(T/T_{target}))P_{leakage} \quad (1)$$

where α is the activity factor and T_{target} is the maximum expected cycle time required to accommodate all applications. We chose $30\mu s$, which is the time a typical 802.15.4 radio takes to transmit one byte of data [28]. T, P_{active}, and $P_{leakage}$ are the measured period of oscillations, active power, and leakage power, respectively, for each technology node, temperature, device, and voltage simulated. Figure 3 presents a 3D illustration of Equation 1, which compare total power across Vdd and activity for different technology nodes. Supply voltage was scaled to the lowest value that was still less than T_{target}. Notice that more advanced deep sub-micron technologies consume much less power for high activity factors compared to older technologies. However, for low activity factors, expected for sensor network applications, leakage power dominates. Hence, the most advanced technologies are less desirable due to their higher leakage characteristics. The process decision should balance active energy and leakage power for the activity factor ranges of the applications detailed in Section 3.

ogy that considers both active and leakage power in the context of low duty cycle operation.

5.1. Simulation Study

To study the power and performance tradeoffs of different technologies, we ran a comprehensive set of HSPICE simulations for several eleven-stage ring oscillators comprised of various static CMOS gates. Simulations were run across a wide range of temperatures, supply voltages, and process technologies. Transient simulation results of the oscillators generated active power data. Leakage power was simulated by disabling the feedback in the ring.

Given the characteristically low workload requirements of sensor network applications, leakage power is a major concern. Several researchers have studied and modeled leakage power, but they do not compare different process technologies [2, 6, 14, 18]. Our simulation results show that even with aggressive voltage scaling, deep sub-micron technologies incur higher leakage current penalties, which discourage their use for sensor network applications.

Older technologies with higher threshold voltages exhibit lower leakage current than newer, faster technologies that utilize lower threshold voltages to enable aggressive voltage scaling. However, advanced deep sub-micron

5.2. Low Power Memory Design

In order to optimize the power consumed by our architecture, we designed a 2-kilobyte custom on-chip SRAM. The architecture's overarching paradigm of switching off unneeded circuit elements is present in the memory. For example, the SRAM is divided into banks of 256 bytes to allow unused portions of memory to be Vdd-gated. The SRAM architecture is illustrated in Figure 4. Both leakage power and active power can be reduced through this banked architecture. By partitioning the memory, the voltage supply to

1 While an asynchronous implementation may benefit from the event-driven nature of sensor network applications, we do not feel the potential to reduce active power is justified in light of higher design complexity and circuit overhead required for asynchronous operation. Furthermore, asynchronous designs do not reduce leakage power.

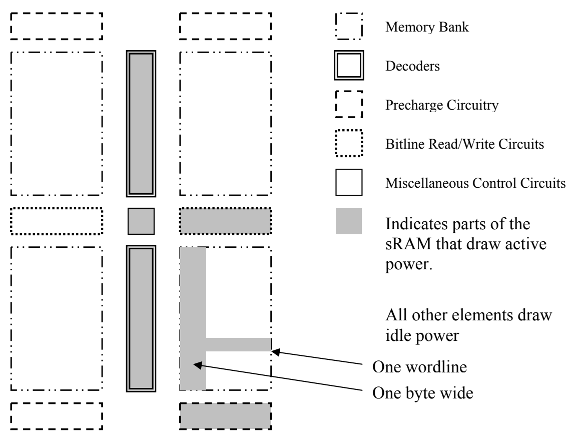

Memory Bank
Decoders
Precharge Circuitry
Bitline Read/Write Circuits
Miscellaneous Control Circuits
Indicates parts of the sRAM that draw active power.
All other elements draw idle power
One wordline
One byte wide
Power Analysis of a 1KB Block of Memory
Note: All banks are being used, i.e. none are gated.

Figure 4. Power usage characteristics of a 1KB SRAM block.

Active Power	Idle Power	Gated Power
1.93 μW	409 pW	342 pW

Table 3. Power for a Single 256B Bank and All Associated Control Circuitry (V_{supply} = 1.2V)

unused banks can be shut off through Vdd-gating, resulting in over a 98% reduction in power drawn by the memory bank – when not Vdd-gated the bank draws 66.5 pW of power, compared to less than 1 pW when gated. It takes 950ns (or less than one clock cycle) to power up a bank after it has been gated.

The SRAM was layed out in a 0.25μm technology and the extracted netlist (with parasitics) was simulated using Nanosim[22]. The power characteristics for a single bank of memory with all associated control circuitry are summarized in Table 3. The 2-kilobyte SRAM design consumes 2.07 μW while operating at 100kHz and 1.2V.

Future revisions of the memory will also include an intelligent precharging scheme. Precharging each bitline consumes the most power when a bank is active, so we envision reducing this power by only precharging the bitlines of the cells that will be accessed. In order to do this we will have to include additional decoder and precharge control circuitry. However, we believe this cost will be offset by a 35% reduction in total active power when this scheme is implemented.

6. Proof of Concept Results

6.1. Performance Methodology and Estimates

While the performance requirements of wireless sensor nodes are very low, the performance of our architecture relative to general-purpose microcontrollers is an important issue, because it determines the minimum required clock rate of our system. In this section, we present our performance modeling methodology and cycle-level comparisons between our architecture and the Atmel Atmega128 microcontroller used in the Mica2 sensor node.

6.1.1. Performance Modeling - SystemC Simulator We used a cycle-accurate simulator written in SystemC to characterize the cycle-level behavior of our architecture. SystemC is a set of C/C++ libraries that is used to model high-level architectural behavior [15]. Currently, the simulator has 8000 lines of code (excluding SystemC library code). We implemented a modular design to which models of slave components can be easily attached. The SystemC model allowed us to explore several design choices rapidly until we arrived at the current version of our architecture. We also utilized this model to provide component utilization statistics that allowed us to perform power analysis for various workloads.

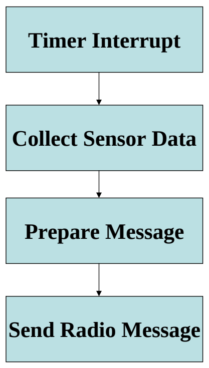
The simulator can currently take in assembly code for both the microcontroller and the event processor, as well as other simulation data required such as received data packets and data sampled by the sensor block. Thus, it is possible to specify complete applications. A few applications were mapped to the simulator by hand (we are considering compilation from higher-level languages). The same applications were also ported to a simulator for the MICA platform, and the cycle counts for both platforms were collected and compared.

6.1.2. Test Application In order to evaluate our architecture, we began with the simplest application that is representative of existing real-world applications such as habitat monitoring [23]. We then added complexity to this application in stages and the final application is fairly complex, including standard sensing and transmission of data, multi-hop routing, and remote application reconfiguration.

We describe the four application versions according to the complexity added in each stage:

1. Periodically collect samples and transmit packets containing the samples.
2. Periodically collect samples and transmit packets containing the samples if it is above a certain threshold.
3. Receive and forward incoming messages from other sensor nodes.
4. Receive and handle incoming reconfiguration messages. (These messages include changes to the sampling period and the sensor threshold value.)

The base application collects samples and transmits the packets. For our architecture, the processing of a sample is initiated by the timer firing an interrupt. The event processor responds to this interrupt by sampling the ADC and transferring the value to the message processor. The message processor prepares the message and signals an event that causes the event processor to transfer the packet to the radio



```
<timer intaddr>:
SWITCHON <sensor>
READ <sensor:mem>
SWITCHOFF <sensor>
SWITCHONN <message proc>
WRITE <message proc>
WRITEI <ctrl_wrd> <message proc>
TERMINATE;

<message proc mesg. ready intaddr>:
SWITCHON <radio>
TRANSFER <msg size> <message proc> <radio>
SWITCHOFF <message proc>
WRITEI <ctrl_wrd> <radio>
TERMINATE;

<radio, message sent intaddr>
SWITCHOFF <radio>
TERMINATE;
```

Figure 5. Diagram and Code of Monitoring Application. *The code displayed are the ISR routines for the event processor. Actual address values have been omitted to make the code easy to read.*

block and setup the radio for transmission. The pseudo-code for the program that runs on the event processor for this application is shown in Figure 5. Similarly, the second version of the application includes a very simple threshold filtering operation.

In a multi-hop routing environment, message forwarding is expected to be a fairly frequent activity and we, therefore, map it as a regular event in our architecture. When a message arrives, an interrupt is fired by the radio block to indicate that a packet has been received. The event processor responds by transferring the packet to the message processor, which signals whether the message has been previously received (this is performed by searching for the packet ID in the routing table). If the message has been previously received, the packet is dropped, otherwise the event processor sets up the radio to forward the packet.

The last version of the test application contains two irregular events that require intervention from our general-purpose microcontroller. In this case, message handling is the same as in the preceding case until the message processor receives the packet. If the message processor determines that the message is not a simple forwarding request, then it signals an interrupt indicating that intervention by the microcontroller is required. The event processor wakes up the microcontroller in response to an irregular event signaled by the message processor. The microcontroller decodes the message to determine whether the timer needs to be reconfigured or whether the filter threshold needs to be modified.

6.1.3. Cycle Performance Estimates Cycle count results using our SystemC simulator and the Mica2 cycle simulator, Atemu [11], for each application task are shown in Table 4. Each row represents the measurement of a particular segment of code. The first two rows provide measurements of the send path of our application as described in the previous section. The next two rows display cycle comparisons of the receive path for both regular and irregular messages. As explained in Section 4.3.6, our architecture assumes the use of 802.15.4 compatible radios like the CC2420 from ChipCon [3], which implements the radio stack in hardware. Therefore, to ensure an accurate comparison we did not count the cycles for the instructions in the TinyOS radio stack run on the Mica2.

Measurement	**Mica2**	**Our System**	**Speedup**
Total send path w/out filter	1522	102	14.9
Total send path w/ filter	1532	127	12.1
Process regular message	429	165	2.6
Process irregular message			
Timer change	234	136	1.7
Threshold change	11	114	0.096
Units	Cycles	Cycles	×

Table 4. Comparison of cycle count for the test application written on our architecture and on TinyOS for the Mica Platform.

For the Mica2 platform, processing a sample includes the software-equivalent implementation of our test applications, in addition to the overhead of TinyOS, required for context switching and task scheduling. Our architecture handles task scheduling natively in the design and, therefore, we see a large difference in cycle counts. Because our architecture is optimized for regular events, it does not show improvements for irregular tasks that require the general-purpose microcontroller.

It is clear that the emphasis of our proposed architecture, for typical events seen within a sensor node, has significant cycle-count advantages over commodity systems. These advantages enable our architecture to operate at significantly lower clock rates while maintaining sufficient performance to keep up with the 802.15.4 radio standard and process sensor data requests at a level required by typical applications.

For the Mica2 platform, the applications were written using the TinyOS component library. Because the test applications can be created using typical TinyOS components, programming these applications is straight forward. However, the code size of the final application incorporating all components was 11558 bytes for instructions when ported to the Mica2 platform. This is significant compared to the 180 byte memory footprint required for our system.

Ideally, one would like to compare our results with other designs specifically tailored for sensor networks such as the SNAP architecture [5]. Unfortunately, this is complicated by the fact that the SNAP paper and results assume the older radio chip and software stack, and we do not have access to their simulation environment. Hence, we can only compare two relatively simple applications that were reported in the paper: *blink*, which sets a timer to periodically interrupt the processor to blink an LED; and *sense*, which periodically samples data from the ADC and computes a running average. From the published results, SNAP takes 41 cycles and 261 cycles respectively, while our architecture can complete these operations within 12 cycles and 24 cycles. For comparison, the Mica2 requires 523 cycles to run *blink* and 1118 cycles for *sense*.

As can be seen from Table 4, the number of cycles taken to process one timer event for the sample, filter, and transmit application takes 127 cycles. The cycle count at 100 KHz gives us a maximum sample rate of roughly 800 samples/second. This maximum rate seems very reasonable considering the fact that most documented sensor network applications have sample rates less than a 100 samples/second. It should also be noted that the clock rate was chosen to accommodate the radio communication data rate of the 802.15.4 standard, 250 Kbits/second [28].

6.2. Power Estimation Methodology and Results

6.2.1. Methodology Since the power consumption of our system can be accurately characterized only after a fabricated chip has been measured, we restrict ourselves to obtaining a conservative estimate based on the active power consumption of the system for its most frequent activities, i.e. the processing of regular events. Again, since we expect to use commodity radio and sensor components, we do not consider these components in our estimates. Because we have not completed the floorplan for our system, we also do not currently include power estimates for global routing signals, buses, and clocks, although we do consider local clock driver circuitry.

The event processor is the largest power consumer in the system since this is a component that must always be powered. Moreover, this block has the most complex microarchitecture of all of the components involved with regular event processing. Hence, for this block, we have obtained conservative estimates by going through the complete process of synthesizing a VHDL model, performing placement, routing, and simulating the final netlist. For the other components, we have broken them down into common substructures such as incrementers, comparators, buffers, etc., and estimated the power consumption numbers for all of these components by simulating netlists synthesized for these sub-structures and combining the results. For all of the memories used (including the main memory block), we use the estimates described in Section 5.2.

6.2.2. Power Estimates The power estimates for the main components of the system are presented in Table 5. The power numbers are shown for active and idle modes (gated clock) of each component at a supply voltage of 1.2V and a clock frequency of 100KHz. In the subsection on workload analysis (Section 6.3), the power estimates are correlated with duty cycle values for sample application workloads to provide a better understanding of the actual power consumption of the system operating under practical situations.

	Idle/Active	Vdd 1.2(V)
Event Processor	Active	14.25 μW
	Idle	0.018 μW
Timer	Active	5.68 μW
	Idle	0.024 μW
Message Processor	Active	2.57 μW
	Idle	0.025 μW
Threshold Filter	Active	0.42 μW
	Idle	$\sim$0.0 μW
Memory	Active	2.07 μW
	Idle	0.003 μW
System	Active	24.99 μW
	Idle	0.070 μW

Table 5. Power Estimates for Regular Event Processing in the System

The active power consumption corresponds to a situation in which the event processor always has an outstanding interrupt to handle. Therefore, the event processor is always switching in this mode because it begins to process a new interrupt the moment it gets done with the current one. The idle power corresponds to a duration in which the event processor is not provided an interrupt to handle. Both of these situations are extreme cases that we do not expect in normal situations.

Implementation details of each block were required to provide accurate power estimates. The message processor block is comprised of a CAM (Content Addressable Memory) structure for the routing table, a counter for keeping track of the packets transmitted, and two 32-byte buffers to allow packet processing and transfer to/from the message processor to take place in parallel. The timer block consists of four decrementers with registers to store the current count, zero-detect logic to fire interrupts, and a small buffer to store current timer configuration. The threshold filter consists mainly of a comparator and a register to store the threshold value. Active power estimates are obtained for cases in which the relevant sub-structures within a slave component are always switching, and idle power estimates are obtained for settings in which none of the sub-structures are switching. Finally, the system active and idle power estimates were obtained by summing up the active and idle powers for the components. It is important to note that the idle power numbers do not reflect Vdd-gating as this feature has not been fully characterized for these components.

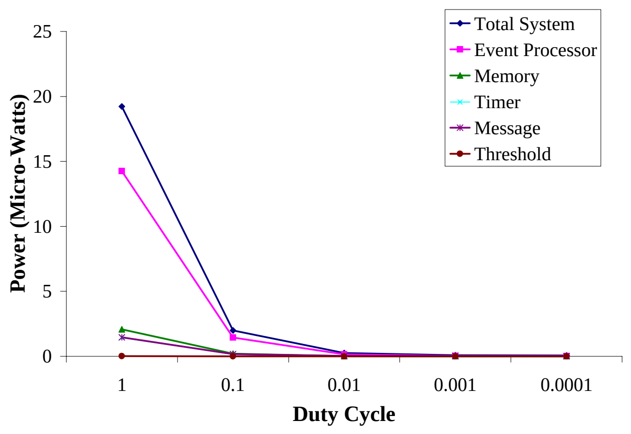


Figure 6. Estimated power varying node duty cycle sample application *A duty cycle of 1.0 is roughly 800 tasks per second.*

6.3. Workload Analysis

The active power estimates presented in the previous section are conservative for practical applications since component activity patterns due to application workload were not taken into account. We now correlate the performance and power estimates obtained in Sections 6.1.3 and 6.2.2 for a real application to obtain power numbers that take into account the utilization of each component of the system. The application we consider for this analysis is the second application described in Section 6.1.2, i.e. a sample is collected periodically, filtered, and a packet is transmitted containing the sample if it passes filtering. For the sake of simplicity, we assume that the sample is always transmitted, i.e. the sample passes the threshold check. This case is the more conservative one, because it is when the system has to do more work and all components are active sometime during the processing of one sample.

An upper bound on the sample rate is obtained by assigning a utilization ratio of 1 to the event processor, i.e. the event processor is always active. At this utilization ratio for the event processor, the system is processing the maximum rate of 800 samples per second as described in Section 6.1.3. The power consumption of each component (and the system) is calculated considering the utilization of the component within the system for several event processor utilization ratios, beginning from the limiting ratio of 1. As a point of reference, the volcano [8] deployment has a duty cycle of 0.12 and the GDI experiment [23] has a duty cycle of approximately 0.0001.

The resulting curves for each component and the system total are shown in Figure 6. For this application, one of the 4 timers in the timer subsystem is always on while the rest are idle. Also, the threshold filter is used for 3 cycles out of the total system 127 cycles per sample, and the message processor is used for 70 cycles per sample. The system power consumption drops to below 2 μW for even reasonably high sample rates.

For the purpose of comparison, the power consumption of the Atmel microcontroller is also estimated for the sample rates corresponding to the utilization points of the event processor. The idea is to compare the power numbers for the same work done for both systems, with the utilization of the Atmel microcontroller normalized to those of the event processor in our system. Again, we use the cycle counts presented in Table 4 for the sample-filter-transmit application. For the active and idle power estimates, we use the measured current values for the Atmel microcontroller in Table 1. We found that the trends are similar to Figure 6 but with a power consumption of a little over two orders of magnitude higher than our architecture.

With the advent of the TI-MSP430 next-generation general-purpose microcontroller used in the Telos mote, the energy consumption difference will likely shrink [17, 24]. For example, the TI-MSP430 reports an active mode power dissipation of between 616μW and 693μW at 1MHz and 2.2V. It has been reported for other sensor network applications that the 32KHz idle mode, which dissipates power between 44μW and 123μW, is the most practical low-power mode for the TI-MSP430 because of the ability to manage peripherals and service interrupts in this mode [27]. If we assume equal cycle-level performance with the Atmel processor, with a sampling rate corresponding to the 0.1 utilization point for our system, the MSP430 will consume between 113μW and 192μW. At this time we are unable to compare power results with the SNAP processor because the published results do not include enough data for us to make an accurate comparison.

7. Conclusion and Future Directions

This work describes a holistic approach to the design of a wireless sensor network device. Employing an application-driven design philosophy, this work describes the selection of process technology, circuit design considerations, and a novel system architecture for sensor devices. In order to provide efficient operation and enable fine-grain power control, our architecture provides explicit support for the event-driven nature of sensor network applications and provides key functionality in separate hardware blocks. Our estimates for the key components of the system include a total active power of $\sim$25μW and idle power is $\sim$70nW. With a duty cycle of 0.1 or less, the average power drops to less than 2μW. These results represent a substantial savings over existing systems.

This project is currently in the circuit-level implementation phase of the architecture described in this paper, and we expect to tape out a chip implementing key components of our architecture to validate design decisions within the next year. We plan to utilize the lessons learned from this implementation work to guide the development of future optimizations to our base architecture, and we plan to consider additional slave devices to expand the space of well-optimized applications.

Acknowledgments

We would like to thank the Motes group at Harvard, specifically Matt Welsh, Victor Shnayder, Bor-rong Chen, and Geoff Werner-Allen for their insight into sensor network applications.

This material is based upon work supported by the National Science Foundation under grant number 0330244.

References

[1] International technology roadmap for semiconductors. *Semiconductor Industry Association*, 2004.

[2] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, pages 23–29, July-Aug 1999.

[3] Chipcon AS. CC2420 2.4GHz IEEE 802.15.4/ZigBee-ready RF Transceiver. `http://www.chipcon.com`.

[4] Crossbow Technology Inc. Mica2 sensor node. `http://www.xbow.com`.

[5] V. Ekanayake, I. Clinton Kelly, and R. Manohar. An ultra low-power processor for sensor networks. In *Proc. ASPLOS*, Oct 2004.

[6] D. J. Frank, R. H. Dennard, E. Dowak, P. M. Solomon, Y. Taur, and H.-S. P. Wong. Device Scaling Limits of Si MOSFETs and Their Application Dependencies. *Proceedings of the IEEE*, 89(3):259–288, March 2001.

[7] T. R. F. Fulford-Jones, G.-Y. Wei, and M. Welsh. A Portable, Low-Power, Wireless Two-Lead EKG System. In *In Proceedings of the 26th IEEE EMBS Annual International Conference*, San Francisco, CA, Sept 2004.

[8] Geoffrey Werner-Allen and Matt Welsh. Monitoring volcanic eruptions with a wireless sensor network. `http://www.eecs.harvard.edu/~werner/projects/volcano/`.

[9] J. Hill. *System Architecture for Wireless Sensor Networks*. PhD thesis, UC Berkeley, May 2003.

[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[11] M. Karir, J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. S. Baras. ATEMU: A Fine-grained Sensor Network Simulator. In *Proceedings of First IEEE International Conference on Sensor and Ad Hoc Communication Networks (SECON'04)*, Santa Clara, CA, Oct 2004.

[12] K. Lorincz, D. Malan, T. R. F. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, S. Moulton, and M. Welsh. Sensor networks for emergency response: Challenges and opportunities. *IEEE Pervasive Computing*, Oct-Dec 2004.

[13] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, GA, Sept. 2002.

[14] S. Narendra, V. De, S. Borkar, D. Antoniadis, and A. P. Chandrakasan. Full-chip sub-threshold leakage power prediction model for sub-0.18 μm cmos. In *Proc. ISLPED*, Aug 2002.

[15] Open SystemC Initiative. SystemC. `http://www.systemc.org`.

[16] J. Polastre, J. Hill, and D. Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, Nov 2004.

[17] J. Polastre, R. Szewczyk, C. Sharp, and D. Culler. The Mote Revolution: Low Power Wireless Sensor Network Devices. In *In Hot Chips 16: A Symposium on High Performance Chips*, Aug 2004.

[18] R. Rao, A. Strivastava, D. Blaauw, and D. Sylvester. Statistical analysis of subthreshold leakage current for vlsi circuits. *IEEE Transactions On VLSI Systems*, 12(2):131–139, Feb 2004.

[19] S. Roundy, B. P. Otis, Y.-H. Chee, J. M. Rabaey, and P. Wright. A 1.9GHz RF Transmit Beacon using Environmentally Scavenged Energy. In *Proc. ISLPED*, Aug 2003.

[20] S. Roundy, P. K. Wright, and J. Rabaey. A study of low level vibrations as a power source for wireless sensor nodes. *Computer Communications*, 26(1):1131–1144, July 2003.

[21] V. Shnayder, M. Hempstead, B.-R. Chen, G. W. Allen, and M. Welsh. Simulating the Power Consumption of LargeScale Sensor Network Applications. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, Nov 2004.

[22] Synopsys Corporation. NanoSim - High capacity and high performance circuit simulation. `http://www.synopsys.com/products/mixedsignal/nanosim/nanosim.html`.

[23] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *Proc. the First European Workshop on Wireless Sensor Networks (EWSN)*, January 2004.

[24] Texas Instruments. TI MSP430F149 Ultra-Low Power Microcontroller. `http://www.ti.com`.

[25] B. A. Warneke and K. S. Pister. An ultra-low energy microcontroller for smart dust wireless sensor networks. In *Proc. ISSCC*, Jan 2004.

[26] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed Energy Conservation for Ad Hoc Routing. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (ACM MobiCom)*, Rome, Italy, July 2001.

[27] P. Zhang, C. Sadler, S. Lyon, and M. Martonosi. Hardware Design Experiences in ZebraNet. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, Nov 2004.

[28] ZigBee Alliance, http://www.zigbee.org. *IEEE 802.15.4 Standard*.