A wireless embedded sensor architecture for system-level optimization

Jason Hill and David Culler
{jhill, culler}@cs.berkeley.edu

Abstract

Emerging low power, embedded, wireless sensor devices are useful for wide range of applications, yet have very limited processing, storage, and especially energy resources. Thus, a key design challenge is to support application-specific optimizations in a highly flexible manner. Power consumption and capabilities of the radio communication layer are the dominant factors in overall system performance. This paper presents a wireless sensor node architecture to achieve high communication bandwidth with the flexibility to efficiently implement novel communication protocols. The architecture is instantiated in an operational design using commercial microcontroller and radio technology. Its ability to optimize system performance by using unconventional protocols is demonstrated by four case studies involving power management, synchronization, localization, and wake-up.

1 Introduction

Emerging wireless embedded sensors combine sensing, computation, and communication in a single, tiny, resource constrained device. Their power lies in the ability to deploy a large number of nodes that are deeply integrated with a physical environment that automatically configure into distributed sensing platform. Usage scenarios range from the real-time tracking, to monitoring of environmental conditions, to ubiquitous computing environments, to *in situ* monitoring of the health of structures or equipment. While often referred to as networked sensors, they can also be actuators that extend the reach of cyberspace into the physical world.

The core design challenge for wireless embedded sensors lies in coping with their harsh resource constraints. Embedded processors with kilobytes of memory are used to implement complex, distributed, ad-hoc networking protocols. These constraints derive from the vision that these devices will be produced in vast quantities, must be small and inexpensive, and ideally will operate off of ambient power. As Moore's law marches on, these devices will get smaller, not just grow more powerful at a given size.

One of the most difficult resource constraints to meet in the context of wireless embedded sensors is power consumption. As physical size decreases, so does energy capacity. Because communication is often the single largest energy consumer, the optimization of wireless communication protocols is key to meeting energy constraints. Mature devices, such as cell phones and pagers, use specialized communication protocols on ASICs that provide ultra-low-power implementations of these protocols [1]. This efficiency is possible because they target a narrow set of well-defined application scenarios. The complex tradeoffs between power consumption, bandwidth, and latency can be explored and evaluated with respect to specific performance goals. The result can be seen in the standby battery life of about a week in cell phones while pagers run for months off of a AAA battery. Application specific pager protocols trade latency and bandwidth for decreased energy consumption. The power of networked embedded devices is their flexibility and universality. The wide range of applications being targeted by embedded wireless sensors makes it difficult to develop general-purpose protocols that are efficient for all applications.

This paper presents the architecture of a wireless embedded sensor node that enables application specific optimization of communication protocols. Much work has focused on the design of low-power radio circuits, reconfigurable logic, and datapath optimizations. Our focus is on architectural support for system-level optimization. By providing tight coupling between protocol and application level processing it not only allows for application-specific implementations of traditional protocols, but it also enables developers to experiment with radically different communication paradigms. Protocols can expose as much or as little information up into applications as they want. This flexibility leads to significant improvements in application performance. Moreover, we have been able to provide this flexibility while simultaneously supporting communication rates close to the peak radio performance.

Section 2 presents an overview of the system architecture, and Section 3 provides background into wireless communication. Section 4 explores design alternatives for the communication subsystem and presents our architecture. Section 5 evaluates our architecture by presenting application specific protocols that have been enabled. Section 6 overviews the related work and Section 7 concludes with future directions.

The main contributions of this work are (i) an architecture that supports application-specific optimization for novel communication protocols, (ii) a instantiation of the architecture using current microcontroller and low-power radio technology, and (iii) a demonstration of its flexibility on several unconventional protocols.

2 System architecture

The goal of our design is to provide an open platform for experimenting with wireless embedded networks. It must be compact, low power, and flexible in order to meet a wide range of experimental goals. The design is based on the architecture presented in [2]. It has a central microcontroller that performs all sensing, communication and computation. An event driven OS is used to multiplex the concurrent flows of information across this single controller, which is connected to an RF transceiver, a secondary storage device, a sensor oriented I/O system, and a power management subsystem. Secondary controllers are not used to abstract the raw characteristics of the I/O devices from the main processing unit. While this places higher demands on the central controller, it allows the controller to exploit the sensors and radio based on situational requirements. One of the key differences between our design and that described in [2] is that we have incorporated hardware support to increase radio bandwidth and time synchronization accuracy while maintaining the ability to implement a wide range of networking protocols. It also has greater capacity, a richer sensor interface, and better power management.

2.1 Overall block diagram

The node architecture consists of five major modules: processing, RF communication, power management, I/O expansion, and secondary storage. We will quickly go through the major modules to give a feel for the system as a whole.

In the MICA implementation of this architecture, shown in Figure 1, the main microcontroller is an ATMEGA103L running at 4 MHz [3]. It is an 8-bit microcontroller with 128 Kbytes of flash program memory and 4 Kbytes of system RAM. Additionally, it has an internal 8 channel, 10-bit ADC, 3 hardware timers, and 48 general-purpose I/O lines. It has one external UART and one SPI port. A coprocessor included to handle wireless reprogramming, is an AT90LS2343 [4] 8-pin flash-based microcontroller with an internal system clock and 5 general-purpose I/O pins. Additionally, in order to provide each node with a unique ID, we include a DS2401 silicon serial number [5].

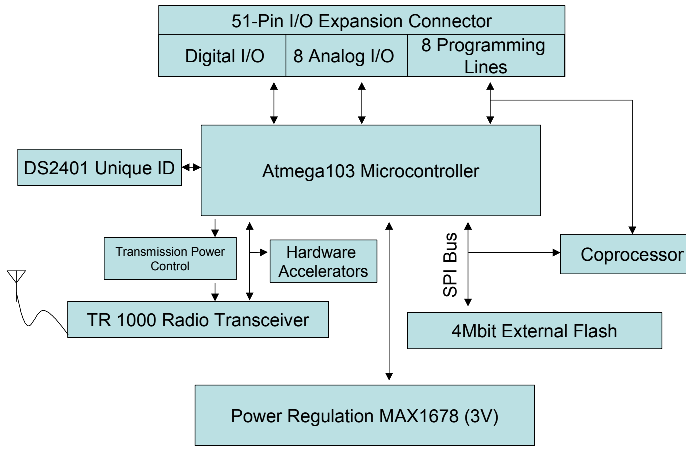

51-Pin I/O Expansion Connector
Digital I/O
8 Analog I/O
8 Programming Lines
DS2401 Unique ID
Atmega103 Microcontroller
Transmission Power Control
Hardware Accelerators
SPI Bus
Coprocessor
TR 1000 Radio Transceiver
4Mbit External Flash
Power Regulation MAX1678 (3V)

Figure 1:Mica node overview.

The RF module consists of an RF Monolithics TR1000 transceiver and the set of discrete components required to operate the radio[6]. It can be externally controlled to have a transmission radius ranging from inches to tens of meters and can operate at communication rates up to 115Kbps. Current into the TX modulation pin controls the transmission strength; to dynamically adjust the transmission strength of the radio we use a DS1804 digital potentiometer. The radio interface gives direct control over the transmitted signal allowing for the use arbitrary communication protocols.

Persistent data storage is provided by a 4Mbit external flash. It is an Atmel AT45DB041B serial flash chip[7]. It was selected because of its serial interface and its small 8-pin SOIC footprint. It is intended to store sensor data collected as well as program images that are to be programmed onto the main CPU. To hold any possible program destined for the microcontroller, the flash must be larger that the 128KB program memory on the main controller. This requirement eliminated the lower power EEprom based solutions from consideration because they are generally smaller than 32Kbytes.

The power subsystem is designed to regulate the supply voltage of the system; a Maxim1678 DC-DC converter provides a constant 3.0V supply [8]. The system is designed to operate off of an inexpensive battery that produces between 3.2V and 2.0V (e.g., pair of AA batteries). This chip was chosen because of its small form factor and its ultra high efficiency. The converter takes input voltage down to .8V and boosts it to 3.0V. This provides a clean, stable voltage source for the rest of the system. Additionally, it allows the system utilize a greater fraction of battery energy. In an alkaline battery more that 50% its energy lies below 1.2 V [9]. Without a boost converter, this energy is unusable. For ultra low power operation, the power system can be disabled allowing the system to run directly off the unregulated input voltage. A solid 3V supply is only required for proper radio operation, a lower voltage can be used to conserve energy when the radio is not in use.

The I/O subsystem interface consists of a 51-pin expansion connector designed to interface with a variety of sensing and programming boards. The expansion connector is divided into sections of 8 analog lines, 8 power control lines, 3 PWM lines, two analog compare lines, 4 external interrupt lines, an I2C bus, an SPI bus, a serial port, and a collection of lines dedicated to programming the microcontrollers. The expansion connector can also be used to program the device and to communicate with other devices, such as a PC serving as a gateway.

2.2 Operating system interaction

The Mica hardware platform has been designed to support the TinyOS execution model presented in [2]. TinyOS is an event-based operating system where all system functions are broken down into individual components that interact through narrow command and event interfaces. The component-based structure of TinyOS allows for an application designer to select from a variety of system components in order to meet application specific goals.

One of the key principles of TinyOS is that there are no long-running threads in the system. Each component acts as like a finite state machine using commands and events to transition from one state to the next. There is no blocking or waiting in system components. This is intended to project the natural interface of hardware into software. This projection allows for a migration of the hardware/software boundary.

A core part of the TinyOS runtime model is that commands and events execute quickly and run to completion, just as commands to hardware modules are accepted quickly. There is no mechanism for suspending and preempting a command. One of the consequences of this approach to system design is that commands and events cannot perform long running computation. If they did, time critical system events could not proceed. To allow for general computation inside a TinyOS component the concept of a task is introduced. A component can post tasks to a system scheduler for background processing. When executed, they run to completion and are not preempted by other tasks. At most one task is active in the system at any time. However, low-level system events are able to preempt tasks. This allows for complex computation to be performed in the background without interfering with the low-level event flow that may have real-time constraints. In this model, tasks simulate the parallelism that is inherent in many hardware components.

The use of the TinyOS programming model has a significant impact on the design of our communications subsystem. The fine-grain multithreading it provides allows us to interleave application-level and protocol-level processing on a single controller. Without a hardware boundary between application and protocol processing arbitrary communication interfaces can be constructed.

3 Communication basics

The focus of this paper is the design of a communication subsystem that allows for flexible, application specific optimization of communication protocols while simultaneously obtaining high bandwidth. To set the stage for the design, this section describes the basic operations associated with communication across a radio link.

The foundation of our communication subsystem is the TR1000. It is a simple amplitude shift keying (ASK) based radio transceiver. It only provides the basic modulation and sampling of the RF channel; all higher-level support must be provided by additional components. There is no pre-specified framing for packets or bytes and there is no maximum transmission length. The radio has three modes: transmit, receive and sleep. During transmission, the binary value placed onto a transmission pin is directly connected to an RF amplifier. The only requirement is that the bit width of the digital waveform must exceed a minimum pulse width and that the data coding on the channel must respect certain DC-balance requirements. The amplitude of the transmitted signal is proportional to the current at the transmit pin. Thus, virtually all aspects of the signal that is transmitted by the TR1000 are set externally. The TR1000's raw interface makes it possible to experiment with any number of different transmission protocols and media access control schemes.

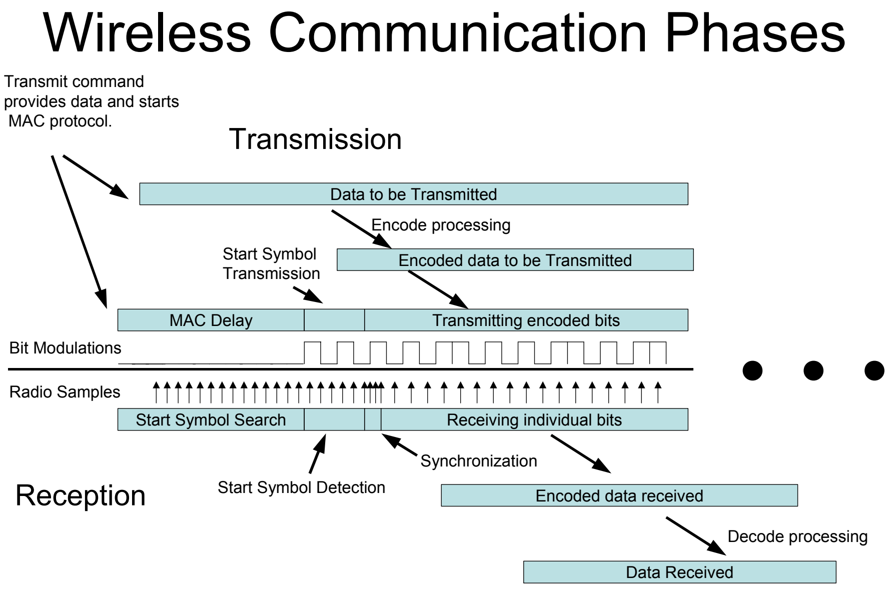


Figure 2: Phases of wireless communication for transmission and reception.

The receiving interface to the TR1000 is equally simple. There is an RX pin that is a digitized version of the base-band signal arriving at the radio. Simple thresholding of the base-band signal is used to determine if the radio is receiving a one or a zero. There is also external access to the raw base-band signal coming from the RF filters prior to digitization. This can be used to extract the incoming data transmission or to determine the strength of a transmission signal.

3.1 Wireless Data Transmission

To communicate over the radio, protocols must be built on top of it. Figure 2 illustrates the key phases of a packet-based wireless communication protocol and provides a framework for discussing architectural alternatives and the impact they have on application performance.

The first step in many wireless protocols is to encode the data for transmission. The coding schemes are designed to increase the probability of a successful transmission by preventing and correcting slight errors. The actual transmission begins with a start symbol that to signals to the receiver that a packet is coming. It is then followed by a synchronization signal and the encoded data. As the transmission proceeds, the transmitter must precisely control timing of each bit transition. Skewed bit transitions can cause the sender and receiver to get out of synch, resulting in an unsuccessful transmission.

Most protocols include a media access control (MAC) phase prior to the actual transmission. MAC protocols are designed to allow multiple transmitters to share a single communication channel. One of the simplest MAC protocols is collision avoidance (CSMA). Just prior to transmitting, the sender checks whether the channel is idle. If so, it proceeds, otherwise it waits for the channel to become idle.

For a receiver, the first part of data reception is to detect that a transmission has begun. The channel is monitored by sampling the receive pin. There is there is often a significant amount of noise on the communication channel that must be filtered and ignored while listening for the start symbol. The length and format of the start symbol can be optimized for the expected noise levels. In order to properly detect the start symbol, the receiver must sample the channel at least twice as fast as the transmission speed. Otherwise the relative phase of the sampling and the transmission may result in the receiver missing the start symbol.

Once detected, the receiver must then synchronize itself to the exact phase of the incoming transmission. This synchronization step is critical in allowing the receiver to determine the start and end of the bit windows being used by the transmitter. Synchronization requires the incoming transmission to be sampled repeatedly so the exact timing of the bit transitions can be determined.

Once synchronized, receiver then samples the value of the incoming signal at the center of each bit. Precise care must be taken to minimize skew in the sampling rate and timing. As the individual bits are extracted from the radio, they are assembled into blocks that are the encoded version of actual data messages. Finally, the blocks are decoded back into the original data and assembled into a packet. The decoding process can often correct bit errors in the received signal and reproduce the original data.

3.2 Driving the bits

One of the most delicate parts of the radio communication interface is driving and extracting the individual bits with precise timing. One of the most common mechanisms for performing this serialization/deserialization is to use programmed I/O on a dedicated microcontroller. With programmed I/O, the main data path of the microcontroller handles each bit transmission. The simplest mechanism for bit timing is to insert delay loops into code so that each pass through the code takes approximately the correct amount of time. While this mechanism is simple, it produces high variance in bit timings. As the data is processed, slightly different code paths may be taken resulting in skewed timing.

A better way to perform bit timing is to use the system timer to trigger periodic interrupts. Not only does this generate more accurate bit timing, but it also allows for the CPU to perform other work or to enter a low power state between each transition. However, the bit rate obtainable in this fashion is limited by the cost of saving and restoring state to execute the interrupt handler.

Many microcontrollers have hardware support for serialized transmission and reception with precise bit-timings, but it is intended for wired communication. The most common of these, the UART (Universal Asynchronous Receiver/Transmitter), uses a two-line communication bus (RX and TX). The UART performs the data serialization and de-serialization by accepting entire bytes and placing each bit onto the transmission line for the appropriate amount of time. While the UART works well for wired communication, it is not well suited to noisy, wireless communication. The UART automatically detects start of packet by capturing single bit; any transition on the RX line is seen as a start symbol. This causes slight noise on the communication channel to interfere with data reception.

Another popular chip-to-chip communication protocol is SPI. Unlike the UART, SPI has a no transmission detection mechanism. Instead, a third communication line provides the timing information for each bit. The receiver latches the data value of the RX line each time the clock line is pulsed. Unfortunately, the SPI protocol cannot be used directly for wireless communication because the synchronization signal is not present. The shortcomings of the hardware supported communication protocols force the use of programmed I/O to drive the radio in either a looping or interrupt-driven mode.

4 Communication Abstractions

The key architectural issue for low-power, wireless devices is the nature of the interface between the application processor and the radio. One option is to use secondary processor to implement the low-level communication protocols. This class of solution uses physical parallelism to create a clean partition of the workload. A specific interface is defined between the application-level processor and the protocol processor that is tailored to fit standard serializers. Unfortunately, these interfaces must abstract away radio specific details and limit an applications ability to directly interact with the radio. The physical separation forces the protocol-to-application interface to be narrow because it must be implemented over the low bandwidth, chip-to-chip protocols.

On the other hand exploiting fine-grained multi-threading to interleave protocol-level and application-level processing on a single controller creates the ability to have a rich and dynamic interface from the application down into the communication stack. Without a physical barrier to cross an arbitrary array of application specific protocols and protocol interfaces can be constructed. However, [2] showed that this resulted in CPU limited communication speeds. This is another example of the tradeoff between virtual and physical parallelism that has already seen in the debates over supercomputing interconnects [10].

In this section we present an architecture that maintains the flexibility enabled by multithreading while simultaneously providing high communication speeds. This is achieved through the use of hardware accelerators that facilitate the protocol processing yet do no abstract away the low-level details associated with radio communication. By positioned hardware support along side the application processor, we allows the application direct access to any aspect of the communication process yet provide a mechanism to offload some of the protocol processing to dedicated hardware.

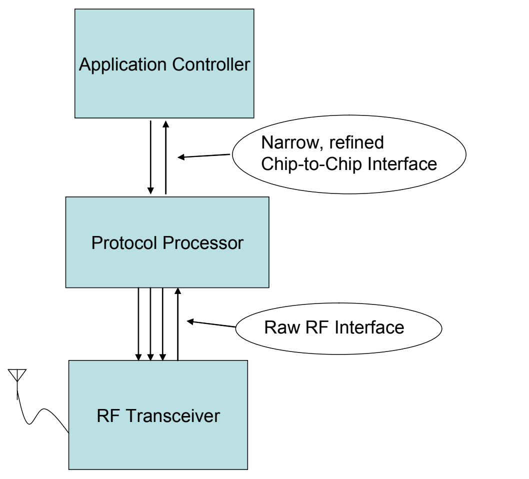


Figure 3: Dedicated radio controller supporting application level controller

4.1 Dedicated radio controller

In most wireless devices a dedicated radio controller handles all protocol processing. This is the approach used in many 802.11 cards, Bluetooth chipsets [11], cell phones[1], as well as in the RF Monolithics Virtual Wire [12] protocol kit design for our radio by the manufacturer. The use of a dedicated controller results in a significant reduction in the communication processing that must be handled on the central controller and creates a clean interface between applications and the communication subsystem. The separate coprocessor or protocol processor refines the raw data stream coming from the radio into a formatted packet interface. The host channel interface (HCI) of Bluetooth deals with high-level commands oriented at packet operations over a UART interface. The intricacies of packet synchronization, channel encoding and MAC protocols are hidden from the application.

While using specialized hardware potentially provides a more efficient, high-performance implementation for a particular communication protocol, the abstraction process often hides valuable information from applications and prevents the application from having fine-grained control over the radio. There are three key areas where the introduction of a secondary radio controller can result in a degradation of overall system performance. The first is that it forces applications to use a predetermined set of protocols. The broad range of intended applications being targeted by networked embedded sensors presents huge opportunities to exploit application specific communication protocols. Secondly, the addition of a secondary controller loosens the application's control over the radio. We show that precise control over the power modes of the radio and can be used to reduce overall energy consumption.

Finally, a secondary radio controller abstracts away the exact timing of packet transmission and arrival. Media access control protocols often involve the introduction of random delays for collision avoidance. The timing uncertainty introduced will directly impact the accuracy of time synchronization across nodes. In many applications, overall performance will be directly correlated to this accuracy.

4.2 Direct connection

In a PC several communication busses and a hierarchy of controllers separate the CPU from its I/O devices. As embedded computers developed, it was natural to maintain a separation between I/O and application control. Systems with dedicated protocol processor follow the pattern of hierarchal architectures. However, [2] showed that a fine-grained interleaving of application and protocol processing could be used to flatten the control hierarchy in wireless embedded sensors.

The handling of application and protocol processing on a single controller facilitates the development of applications specific communication protocols and protocol interfaces. Instead of using a preconceived communication protocols interfaces over chip-to-chip communication mechanism, a developer can tailor the communication protocols and interfaces meet application needs. Any information concerning the underlying communication channel can be exposed up into the application through arbitrary protocol interfaces.

For example, when implemented on the same CPU, the MAC layer can inform the application of exactly when the packet was actually transmitted based on shared system timers. For precise time synchronization, packets can be time stamped by applications after the MAC delays have been determined and packet transmission has begun. This leads to highly accurate time synchronization mechanism. A secondary benefit of utilizing a single CPU for application and protocol processing is that it also allows for a dynamic allocation of computational resources. During periods of low protocol overhead, applications can utilize unused CPU cycles.

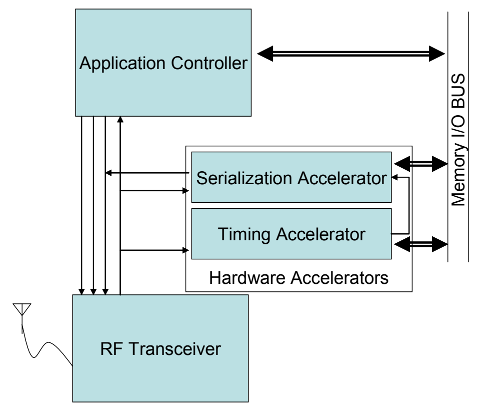


Figure 4: Direct connection with hardware acceleration.

Unfortunately, [2] resulted in an implementation that was CPU limited. It only was able to achieve a 10Kbps maximum bandwidth. The low-power processors used in wireless embedded sensors are not capable of directly handling high bandwidth communications using programmed I/O. Even when performing no application processing, our CPU can only transmit at 30Kbps when using programmed I/O alone. Many of high bandwidth chipsets use high-speed microcontroller cores to handle the processing overhead.

4.3 Mica's Hardware Accelerated Protocol Processing

In the Mica platform we develop an architecture that maintains the flexibility created by directly interfacing with the radio but then exploit dedicated hardware to improve efficiency. Instead of including additional hardware as an abstraction layer, we use it in the form of protocol accelerators. We place hardware accelerators are alongside the communication path between the controller and the radio. At any time, the controller can directly interact with the radio with exact precision. However, when appropriate the controller can enlist the help of the hardware accelerators. The accelerators are designed to automatically perform some of the CPU intensive portions of communication protocols. In our implementation, we use hardware accelerators for synchronization, bit timing, and bit sampling. While reducing CPU use, the hardware accelerators do not abstract away what operations are being performed. The architecture is designed to maintain a tight coupling between the application and the communication protocol while providing increased efficiency.

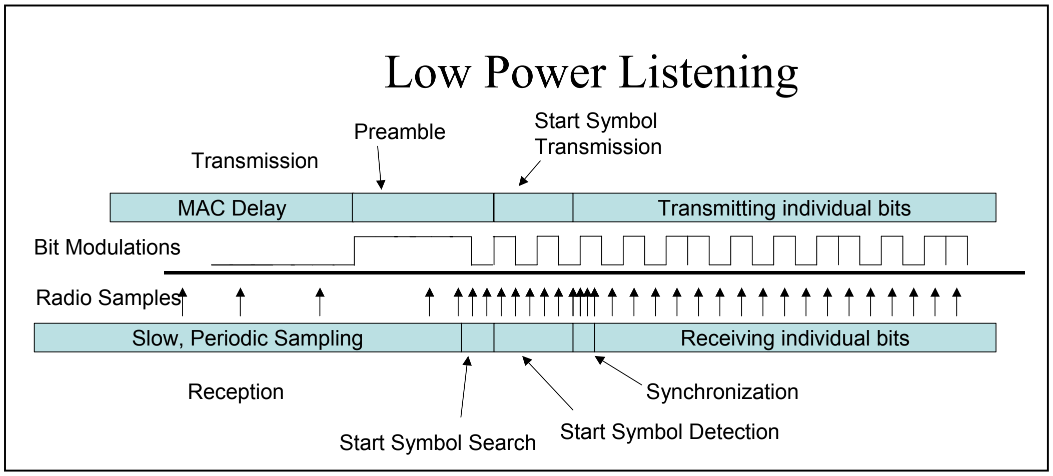
In our hardware-accelerated protocols, we continue to rely on programmed I/O to perform the start symbol detection. However, once detected we offload the overhead of synchronization to a timing hardware accelerator. The timing accelerator automatically captures the exact timing of the edge transition of the timing pulse. The incoming signal is automatically sampled every .25 us. By searching for the start symbol prior to engaging the timing hardware accelerator, we avoid the timing problems seen with the UART. Detection of the start symbol gives us an indication of when the timing pulse will arrive and keeps us from of assuming every pulse is the timing pulse. Additionally, unlike the UART our timing mechanism exposes the value captured instead of only keeping it for internal use. If desired this timing information can even be propagated up into an application

Once the timing information is captured, software then uses it to configure a serialization accelerator that automatically times and samples the individual bits. During the majority of a transmission and reception, the main data path only deals with the byte-oriented interface provided by the serializer. The individual bits are handled in hardware.

Each of these hardware accelerators has been built out of standard microcontroller functional units. They are implemented by combining the functionality of input capture registers, timer controlled output pins and the SPI communication hardware. The input capture register is used to automatically capture the timing pulse. The captured value can then be use to configure the timing register controlling an output pin. One of the control options is to have the hardware automatically toggle the output pin each time the counter expires. Once configured, this output pin becomes a clocking signal that times each arriving bit. Finally, SPI hardware is then used to capture the value of the radio signal each time the clock line is triggered by the counter. This is done by connecting the counter controlled output pin to the synchronous clock line of the SPI port.

The key difference between an accelerator based approach and a dedicated protocol controller is that the accelerators do not sever the connection between the application-level processor and the raw communication channel. They provide functional assistance but do not abstract away what the system is doing. The application still has the ability to take direct control over the radio.

By exploiting hardware accelerators, we are able to achieve significantly higher communication bandwidth than with programmed I/O alone. The implementation in [2] used programmed I/O exclusively and recorded a maximum bandwidth of 10Kbps. This was limited by the CPU overhead associated with taking an interrupt with each bit. With our exploitation of the SPI port and external timer as an accelerator, we have been able to reach speeds of 50 Kbps on the same processor while leaving CPU cycles for application processing. We have been able to demonstrate 115 Kbps transmissions on a TI MSP430 microcontroller that that also runs at 4 MHz but has



additional buffering in the SPI port. In addition to increasing the bandwidth, we also decrease the CPU overhead of the central controller by dealing with data in byte-sized chunks.

5 Evaluation

To demonstrate the functionality of our architecture we have used the Mica node to implement a traditional, packet-based protocol stack. It consists of an Active Messages [13] based application interface where tagged messages are automatically dispatched to application message handlers. Internal to the protocol implementation, there is a packet layer that can optionally perform CRC checks on each packet as well as a data-encoding layer that performs single error correction, dual error detection encoding on each byte allowing single-bit transmission errors to be fixed automatically.

In addition to delivering high bandwidth data communication using standard protocols, we have also been able to demonstrate how the flexibility we provided plays a critical role in enabling systems-level optimizations. We show this by implementing four application specific protocols that improve overall system performance for their intended application. The first of these is a protocol that explores the tradeoff between transmitter and receiver overhead.

5.1 Low Power Listening

In sensor networks, multi-hop routing topologies are built by having intermediate nodes act as relays for remote nodes. These routing nodes must listen for communication and propagate messages towards their destination. While listening, the radio consumes almost as much energy as when transmitting. Even when no communication is taking place, considerable amounts of energy is spent searching for the next packet. In many application scenarios, the energy spent while waiting for a transmission can represent more than 90% of a node's total energy budget. Embedded sensor researchers have claimed that high-level protocols must be used to reduce energy consumed of nodes when not actively transmitting [14]. One solution is to have windows of communication periods and windows of sleep periods[15]. However, it has been shown that in some situations poor interaction between high-level power saving techniques and low-level communication protocols can actually lead to an increase in energy consumption when using windowing mechanisms [16].

We have been able to show that alternative is to modify the lowest levels of the communication stack. Unlike windowing, which layer itself on top of standard low-level protocols, we change the underlying communication protocol to optimize for the receiver power consumption. Instead of having the sender simply transmit a start symbol followed by a packet, we can require that the sender first transmit a preamble to get the attention of any possible receiver. It can then follow with the start symbol and packet. This algorithm is depicted in Figure 5. The receiver then only has to listen often enough to pick up any portion of the attention signal. Precise control over the power state of the radio allows the protocol to turn off the radio between each sample. The duty cycle of the receiver becomes proportional to the length of this preamble. Once detected, the preamble will cause the receiver to search for the pending start symbol.

This protocol optimization trades power consumption on the sender for power consumption on the receiver. The sender must transmit longer, but the receiver can sample the radio channel less frequently. The optimal ratio is dependent on the communication patterns of the application. We have implemented a version of this scheme where the receiver has a 10% duty cycle and the sender must transmit a preamble of 5 bits. The 10% receiver duty cycle not only results in a 90% reduction of radio power consumption, it also produced a 33% reduction in the CPU overhead associated with monitoring the radio. On the other hand, the sender only incurs a slight increase in overhead – less than 1% overhead on a 30-byte packet. This slight overhead increase is because the sample rate of the receiver is 3000 times per second. Fine-grained control is used to power-on, sample, and power-off the radio in 30us. Application level windowing protocols generally have window sizes of seconds or more.

Depending on application specific goals, the receiver overhead can be reduced arbitrarily at the expense of bandwidth, latency, and transmission overhead. Moreover, an application can change the protocol at runtime based on network activity. This simple optimization demonstrates the benefits that are enabled by our architecture for flexible communication protocols by exploiting the ability to tailor protocols to application specific criteria.

5.2 Time Synchronization

. Many sensor applications need time correlated sensor readings and require an underlying time synchronization mechanism. It has been shown that the accuracy of distributed synchronization protocols is bounded by the unpredictable jitter on communication times[17]. Unlike in wide-area time synchronization protocols such as NTP, we can determine all sources of communication delay [18]. By exposing all sources of delay up to the application, we are able to minimize the unknown jitter. Additionally, by exploiting shared system timers, we are able to accurately assign precise time stamps to incoming packets that can be exposed to applications.

The Mica platform was designed with the intention of using the internal, 16-bit counter to act as the lower 16 bits of a continually running system time clock. This high accuracy system clock is directly linked to the synchronization accelerator that is used to capture the exact timing of the incoming packet. The synchronization accelerator automatically timestamps each packet with the value of the system timer. This time stamp can then be passed up to applications. The key is that the application and the protocol stack both have access to a shared timing mechanism that allows the application to have a reference to compare the time stamp to.

Additionally, during transmission the communication stack can timestamp a packet with this shared timer after all MAC delays have been determined. This allows the time synchronization to be independent of MAC delay and back off. The time stamp represents when the packet actually went over the radio and not when communication was initiated. During periods of high contention, the MAC delay may be hundreds of milliseconds. When hidden by external protocol engines, this unknown delay significantly reduces time synchronization accuracy.

With our implementation, we are able to synchronize a pair of nodes to within 2 microseconds of each other. Our skew of +/-2us can be directly attributed to several sources of jitter. The first is the raw RF transmission itself. When the sending there is a jitter of +/- 1us on the transmission propagation due to the internals of our radio. The arriving pulse is then captured by hardware with an accuracy of +/- .25us. Finally, we must synchronize its clock based on the captured value. This synchronization process introduces an additional +/- .625 us of jitter. This implementation is only possible because we have a rich interface between applications and protocols that allows us to exploit shared access to the high-accuracy system timer. This provides a common reference for exchanging timing information between the bottom of the network stack and the top of an application.

5.3 Localization

In addition to using the radio for data communication, nodes can also use the radio itself as a sensor. Direct access to the base-band signal being output by the TR1000's receiver chain can be exploited in several ways. One exciting used is in localization. The goal is to have a collection of nodes automatically determine the physical position of each member. Node location is a key enabler in many context aware and environmental applications. In many environmental applications it is essential to know the source of the data. It is not always possible to externally assign locations to each sensor.

Several groups have attempted perform localization in a sensor network by using RF signal strength[19-21]. The radio is used as an analog sensor to detect the strength of an incoming signal. RF propagation models are then applied to infer distance from a collection of strength readings.

By providing the processor direct access to the raw based-band signal our platform gives the application developer as much information as possible about the incoming signal. The central controller can look at the signal strength of each individual bit as well as the level of the background noise. Additionally, because the sender has direct control over the base band signal being transmitted, it can intentionally transmit a long duration pulses of variable strength to help the receiver determine the reception strength more accurately.

1) Cost of checking = (radio on time) * (radio power consumption)

2) Power consumption = (checking frequency) * (cost of checking)

3) Average wakeup time = ½ (checking period) = 1/(2 * checking frequency)

Figure 6: Power consumption equations for a sleeping sensor network waiting to get woken by a radio signal.

An alternative to RF localization is to use acoustic localization[22]. The propagation delay of acoustic pulses is measured and used to infer distance. For accurate results, it is essential to use the radio to achieve precise time synchronization between two nodes in order to correlate their senor readings in time. For accurate results, the transmission time of the acoustic pulse must be accurately communicated to the receiver and correlated with the receive time. The precise communication synchronization provided by the radio layer allows this to be done incredible accurately.

5.4 RF Wakeup

The raw interface to the radio can also be exploited to implement an ultra low power radio-based network wakeup signal. In many application scenarios it may be necessary to put a collection of nodes to sleep for a long period of time. The deployed network would be powered down to conserve energy. At a later time, a radio signal would be used to wake the nodes. For optimal performance, the network must consume as little energy as possible while asleep.

For any RF based wake-up protocol, you need to have each node periodically turn on the radio and check for wakeup signal. Figure 6 contains the equations necessary to determine the power consumption of a sleeping network. Each time a node checks for the wakeup signal, it will consume energy equal to the power consumption of the radio times the time the radios is on (1). The power consumed by the sleeping node will be the energy used each time it checks for the signal times the frequency of the check (2).

To minimize the energy consumption of the system while sleeping, you must minimize the time a radio must be turned on each time a node checks for the wakeup signal and minimize the checking frequency. However, frequency that they checks for a wake-up signal controls the amount of time that it takes for the network to wakeup. If each node were to check for a wake-up signal every minute, the average expected wake-up time for a single node would be 30 seconds (3). This means higher frequency checking yields faster wake-up times and better applications performance. Because of this, we focus on minimizing the time it takes to check for a wakeup signal.

Using a packet based radio protocol each node has to turn on the radio long enough to receive at least two packet times[1]. In our system, a packet transmission time is approximately 50ms, so each node would have to be awake for at 100ms each time it checks for a wake-up message. If a node needs to wake-up every minute, this yields a best-case radio duty cycle of .166%. This time window gets even larger when considering the effects of a multi-hop network and the contention associated with having a large number of nodes retransmit the wake-up signal simultaneously.

Instead of interacting with the radio over a high-level packet interface, our low power sleep mode implementation interacts directly with the analog base-band output. The wake-up signal is nothing more that a long RF pulse. Each time a node checks for the wake-up signal, it can determine that the wake-up signal is not present in 50us. The 50us signal detection time is a 2000x improvement over a packet based detection time. In our implementation, we choose to sample every 4 seconds which results in a .00125 % radio duty cycle, a 160x improvement over the packet-based protocol that was sampling once per minute. This ultra-low duty cycle implementation has been used to consistently wake-up multi-hop sensor network of more than 800 nodes. This radically different signaling scheme would not have been possible if a protocol processor was constraining applications use of the radio.

[1] If someone were sending a continual stream of wake-up packets, listening for two packet times would ensure that a complete packet was transmitted while the node was awake.

6 Related Work

There are several other groups looking into the architecture of wireless embedded devices. The Berkeley Wireless Research Center's PicoRadio project has also identified the importance of application specific protocols, however they attempt to build a flexible platform by exploiting reconfigurable hardware [23]. They add reconfigurable building blocks to their PicoRadio protocol processor. This gives it the flexibility to implement a large number of underlying protocols yet it still maintains a separation between protocol and application. It is not clear if the flexibility of the underlying hardware can be fully utilized by the application level processing. It will still be limited by the protocol processors external interface.

The WINS (Wireless Integrated Network Sensors) node developed by researchers from UCLA is also targeting this applications space [24]. Unfortunately, we have been unable to find out the internal architecture of their communication subsystem. However it is clear that there is a distinct partition between application and protocol processor. Their application interface is a WinCE based devices that interfaces with a separate communication and sensing platform. The loose coupling between sensing, communication, and applications would make it difficult to implement many of the algorithms we've demonstrated presented.

Researchers at UCLA have also demonstrated the benefit of exploiting application specific protocols by creating customized MAC layers adapted to sensor networks. Their customized sensor network communication protocols attempt to reduce energy consumption in order to increase application performance. They show 2-6x energy improvement when compared to standard 801.11-like wireless networking protocols [15].

The Radar localization work at Microsoft using RF signal strength of 802.11 networking cards has demonstrated the importance of exposing low-level protocol information up to applications [21]. All 802.11 cards must determine the signal strength of surrounding access points to determine which point to communicate through. However, many protocol stacks would not reveal the signal strength information to the applications. This inhibited their ability performing the localization analysis and demonstrates the importance of allowing applications access to underlying protocol information.

The Smart Dust[25, 26] project at UC Berkeley is investing the use of application specific hardware for the development of dust-sized sensor devices. In targeting extreme miniaturization and low-power consumption they are putting as much functionality as possible into special purpose hardware. They include a tiny microcontroller to perform highest-level application tasks. The communication protocols are implemented in silicon by special purpose hardware. While this design point meets their size and power goals, it is not clear that it creates an architecture that can be applied to a general class of applications.

7 Future directions

In this paper we have explored the tradeoff between using dedicated protocol processors versus interleaving the protocol processing and application level processing on a single chip. By exploiting virtual parallelism, we are able to have a rich, flexible interface between applications and their communication protocols. To increase efficiency we have introduced hardware accelerators that decrease overhead without compromising flexibility. By connecting hardware accelerators along side of the application level processor, valuable information is not abstracted away. The interposition of a dedicated protocol processor between the application processor and the raw communication device forces the application to use a preconceived set of abstractions possibly loosing valuable information.

Our hardware serializers have been built out of commodity microcontroller blocks. They are only one example of what is possible. We believe that a collection of hardware-based building blocks that can be robustly linked into application-level processing can have the potential to create flexible and efficient protocol implementations. One key step will be to determine the set of primitive operations that should be provided.

These primitives will likely include support for serialization, automatic periodic sampling, pattern matching, timing analysis and channel encoding. The use of FPGA cells is also intriguing. However, the key will be to provide a flexible interface into application-level processing. Even if the building blocks themselves are flexible, they may not be capable of providing a flexible interface into applications.

Another alternative is that sensor network protocols may eventually stabilize and result in a small set of optimizations can be directly implemented in hardware. While application specific optimizations will always be required, they may be constrained to a small set of options. The development of networked embedded senor protocols is just beginning. Anything is possible.

Bibliography

1. McMahan, M.L., *Evolving Cellular Handset Architectures but a Continuing, Insatiable Desire for DSP MIPS*. 2000, Texas Instruments Incorporated.
2. Hill, J., et al. *System architecture directions for networked sensors*. in *ASPLOS 2000*. 2000.
3. Atmel Corporation, *Atmega103(L) Datasheet*. 2001, Atmel Corporation: http://www.atmel.com/atmel/acrobat/doc0945.pdf.
4. Atmel Corporation, *AT90S2323/LS2323/S2343/LS2343 Datasheet*. 2001: http://www.atmel.com/atmel/acrobat/doc1004.pdf.
5. Dallas Semniconductor, *DS2401 Silicon Serial Number*: http://pdfserv.maxim-ic.com/arpdf/DS2401.pdf.
6. RF Monolithics, I., *TR1000 Data Sheet*. 1999: http://www.rfm.com/products/data/tr1000.pdf.
7. Atmel Corporation, *AT45DB041B Datasheet*. 2001: http://www.atmel.com/atmel/acrobat/doc1938.pdf.
8. Maxim, *Maxim 1-Cell to 2-Cell, Low-Noise, High-Efficiency, Step-Up DC-DC Converter, MAXIM1678*. 1998: http://pdfserv.maxim-ic.com/arpdf/MAX1678.pdf.
9. Energizer, *Alkaline AA battery data sheet*: www.energizer.com.
10. Homewood, M. and M. McLaren. *Meiko CS-2 interconnect. Elan-Elite design*. 1993.
11. *The Official Bluetooth Website*: http://www.bluetooth.com/.
12. RF Monolithics Inc, *RFM DR2000 Development Board*. 2000: http://www.rfm.com/products/data/dr2000manual.pdf.
13. von Eicken, T., et al. *Active messages: a mechanism for integrated communication and computation*. in *19th Annual International Symposium on Computer Architecture*. 1992.
14. Estrin, D., *Directed Diffusion*. 2000.
15. Wei Ye, J.H.a.D.E., *An Energy-Efficient MAC Protocol for Wireless Sensor Networks*. 2001: Submitted for review, July 2001.
16. M. Stemm, P.G., D. Harada, R. Katz. *Reducing power consumption of network interfaces in hand-held devices*. in *International Workshop on Mobile Multimedia Communications (MoMuc-3)*. 1996. Princeton, NJ.
17. Lamport, L., *Time, clocks, and the ordering of events in a distributed system.* Comm., 1978. **ACM 21**(7): p. 558-565.
18. Mills, D.L., *Internet time synchronization: the Network Time Protocol.* IEEE Trans. Communications, 1991. **COM-39**(10): p. 1482-1493.
19. Doherty, L., K.S.J. Pister, and L.E. Ghaoui. *Convex position estimation in wireless sensor networks*. in *IEEE Infocom*. 2001: IEEE Computer Society Press.
20. Borriello, J.H.a.G., *Location Systems of Ubiuitous Computing.* Computer, 2001. **34**(8): p. 57-66.
21. Bahl, P. and V. Padmanabhan, *RADAR: An in-building RF-based user location and tracking system.* IEEE Infocom, 2000. **2**: p. 775-784.
22. Priyantha, N.B., A. Chakraborty, and H. Balakrishnan. *The Circket Location-Support System*. in *MobiCom 2000*. 2000. Boston, Massachusetts.
23. J.L. Da Silva Jr., J.S., M. J. Ammer, C. Guo, S. Li, R. Shah, T. Tuan, M. Sheets, J.M. Ragaey, B. Nikolic, A Sangiovanni-Vincentelli, P. Wright., *Design Methodology for Pico Radio Networks*. 2001, Berkeley Wireless Research Center.
24. Pottie, G. and W. Kaiser, *Wireless Integrated Network Sensors (WINS): Principles and Approach.* Communications of the ACM, 2000. **43**.
25. B. Atwood, B.W., and K.S.J. Pister. *Preliminary circuits for smart dust.* in *Southwest Symposioum on Mixed-Signal Design*. 2000. San Diego, Ca.
26. K.S.J Pister, J.M.K., B.E. Boser, *Smart Dust: Wireless Networks of Millimeter-Scale Sensor Nodes.* Electronics Research Laboratory Research Summary, 1999.