

sensors

Electronics
& ComputersMachine
ManufacturingProcess
Industries

Automotive

Aerospace/Military
Homeland SecuritySpecialty
MarketsWireless
& M2M

MICA: The Commercialization of Microsensor Motes

Miniaturization, integration, and customization make it possible to combine sensing, processing, and communications to produce a smart, network-enabled wireless sensor. Here's how it works.

Apr 1, 2002

By: [David Culler, U.C. Berkeley](#), [Jason Hill, U.C. Berkeley](#), [Mike Horton, Crossbow Technology, Inc.](#), [Kris Pister, U.C. Berkeley](#), [Robert Szewczyk, U.C. Berkeley](#), [Alec Woo, U.C. Berkeley](#)

sensors

Sensors

New technology is changing the nature of sensors and the way they interface with data acquisition and control systems. Researchers at U.C. Berkeley have developed an open-source hardware and software platform that combines sensing, communications, and computing into a complete architecture. The first commercial generation of this platform was dubbed the Rene Mote, and several thousand of these sensors have been deployed at commercial and research institutions worldwide to promote the development and application of wireless sensor networks.

The platform's development community is based on the open-source model, which has become well known with the increasingly popular Linux operating system. Most development work is done in the public domain, and it includes the hardware design and software source code. Users of the technology contribute their developments back to the community so that the base of code and hardware design grows rapidly. Although there's no official consortium, the current community includes U.C. Berkeley, U.C. Los Angeles, Intel Research Labs, Robert Bosch Corp., U.S. Air Force Research Labs, Crossbow Technology, and others.

To implement improvements based on feedback on the first few thousand wireless sensors deployed, U.C. Berkeley and the collaborating researchers devised a second-generation platform. The platform is named MICA because its final electronic implementation resembles its silicate relative, which separates into thin mineral leaves. Likewise, the MICA electronic hardware is a series of thin processor/radio and sensor circuit cards sandwiched together to form an integrated wireless smart sensor. In the past, this integration wasn't possible, but advances in low-power CMOS wireless communications devices and MEMS sensors makes this possible.

Hardware Design of Wireless Sensors

The basic MICA hardware now uses a fraction of a watt of power and consists of commercial components a square inch in size. But MICA's developers expect end users and OEMs to create many flavors of hardware to meet the needs of a variety of applications. With advances in MEMS and low-power wireless technology, the plan is to more deeply integrate and customize versions of the hardware so that they can outperform current designs based on commercial components. Researchers have completed the basic hardware design, and Crossbow Technology has built several thousand units and distributed them to developers.

The hardware design consists of a small, low-power radio and processor board (known as a mote processor/radio, or *MPR*, board) and one or more sensor boards (known as a mote sensor, or *MTS*, board). The combination of the two types of boards form a networkable wireless sensor.

The MPR board includes a processor, radio, A/D converter, and battery. The processor is an ATMEL ATMEGA, but there are other processors that would meet the power and cost targets. The processor has 128 KB of flash memory and 4 KB of SRAM. In a given network, thousands of sensors could be continuously reporting data, creating heavy data flow. Thus, the overall system is memory constrained, but this characteristic is a common design challenge in any wireless sensor network.

Dealing with the tight memory constraint is given special consideration in the development of a software framework or operating system for MICA's MPR modules. The processor has three sleep modes: idle, which just shuts the processor off; power down, which shuts everything off except the watch-dog; and power save, which is similar to power-down, but leaves an asynchronous timer running. Power is provided by any 3 V power source, typically two AA batteries. Photo 1 shows a picture of the MICA hardware.



Photo 1. The MICA processor and radio board includes an Atmel chip, which serves as the processor and runs the TinyOS. The board's 51-pin connector interfaces with the sensor boards. On the back side of the processor/radio board is a second 51-pin connector and the radio.

The radio is the most important component of the MPR module because it represents the real-world communication conduit. The hard real-time constraints encountered in dealing with the radio (and with the sensors) form a second challenge for the software running on the MICA. The radio consists of a basic 916 MHz ISM band transceiver, antenna, and collection of discrete components to configure the physical layer characteristics, such as signal strength and sensitivity. It operates in an ON/OFF key mode at speeds up to 50 Kbps. Control signals configure the radio to operate in either transmit, receive, or power-off mode. The radio contains no buffering, so each bit must be serviced by the processor in time.

The MPR modules contain various sensor interfaces, which are available through a small 51-pin connector that links the MPR and MTS modules. The interface includes an 8-channel, 10-bit A/D converter; a serial UART port; and an I²C serial port. This allows the MPR module to connect to a variety of MTS sensor modules, including MTS modules that use analog sensors as well as digital smart sensors. The MPR module has a guaranteed unique, hard-coded 64-bit address, which is the Digital ID from Dallas Semiconductor. Figure 1 shows a block diagram of the MPR module.

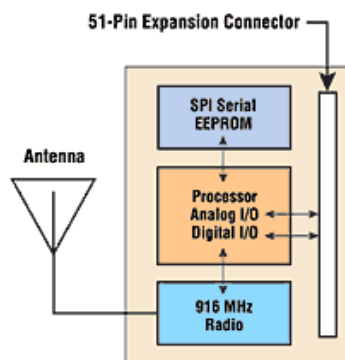


Figure 1. The MICA processor/radio board has all the necessary electronic components to interface with a wide variety of sensors.

Power consumption equates to battery life. Long battery life is desired, and in some applications, one to five years is required. The processors, radio, and a typical sensor load consumes about 100 mW. This figure should be compared with the 30 μ W draw when all components are in sleep mode. The overall system must embrace the philosophy of getting the work done as quickly as possible and then going into sleep mode. This is a third key constraint on the software design for wireless networked sensors.

The MTS sensor boards are easily designed and configurable. The only requirement is the use of the standard 51-pin connector and one of the three hardware interfaces (i.e., analog, UART digital, and I²C digital). The MTS boards currently include light/temperature, two-axis acceleration, and magnetic sensors and 4–20 mA transmitters. Researchers are also developing a GPS board and a multisensor board that incorporates a small speaker and light, temperature, magnetic, acceleration, and acoustic (microphone) sensing devices. The MICA developer's community welcomes additional sensor board designs.

Software and the TinyOS

A considerable portion of the challenge faced by the developers of MICA devices is in the software embedded in the sensors. The software runs the hardware and network—making sensor measurements, routing measurement data, and controlling power dissipation. In effect, it is the key ingredient that makes the wireless sensor network produce useful information.

To this end, a lot of effort has gone into the design of a software environment that supports wireless sensors. The result is a very small operating system named TinyOS, or Tiny Microthreading Operating System, which allows the networking, power management, and sensor measurement details to be abstracted from the core application development. The operating system also creates a standard method of developing applications and extending the hardware. Jason Hill of U.C. Berkeley authored the original open source operating system.

A good way to understand the TinyOS is to familiarize yourself with the design considerations and constraints that led to its creation and architecture. All of these considerations and complexities make it clear why an operating system like TinyOS is so valuable to the users and OEMs of wireless sensors.

Low-Power Modes and Small Physical Size. Long-term operation of wireless sensors places a premium on power. Battery size is the greatest single size constraint for the sensor in many situations. Most applications require three to five years of battery life. To achieve this level of performance, the software must execute all necessary functions quickly and then turn off the hardware.

Self-Configuration. Long, complex installation procedures destroy the benefit of wireless sensors. Installing a sensor should be as easy as gluing the unit to the point of measurement. This creates a software need for a self-configuring network.

Real-Time Requirements. A sensor network's primary mode of operation is to flow sensor information from place to place with some processing in between. There's little storage or buffer capacity on a wireless sensor because of size and cost considerations. Therefore, there's a lot of inbound and outbound traffic. Hard real-time constraints, especially in controlling radio communications, create the need for efficient multi-threading.

Robust and Reliable Performance. Most wireless sensor networks will consist of numerous devices that are largely unattended. The attending engineer will expect them to be operational most of the time. To that end, the operating system on a single node or sensor should not only be robust but also able to continue functioning when other devices on the network fail. This will ensure that if one sensor or device should fail, the network or application is not jeopardized (see Figure 2).

Diversity in Design and Use. Networked wireless sensors will tend to be application specific rather than general purpose, and because of cost and size considerations, they'll carry only hardware and software actually needed for the application. With the wide range of potential applications, the variation in physical devices will likely be great. Therefore, the software components will require an

Apps

There are many applications for wireless sensor networks. Some are new; others are traditional sensor applications that can be improved using wireless sensors. The overall list of applications includes:

- Physical security for military operations
- Environmental data collection
- Seismic monitoring
- Industrial automation
- Future consumer applications, including smart homes.

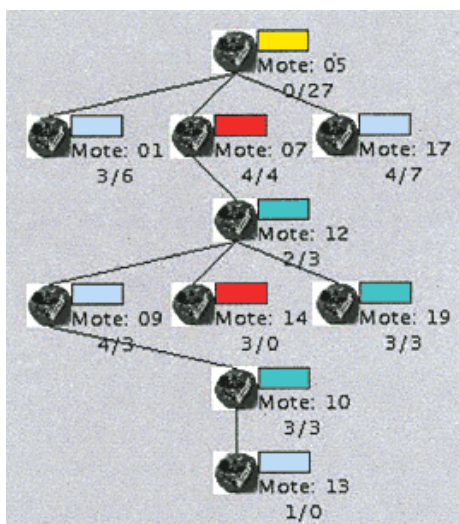


Figure 2. This is an example of a small network of wireless mote sensors, communicating light and temperature in a multi-hop route.

unusual degree of efficiency and modularity. Those who use and manage these networks will need a generic development environment that allows them to construct specialized applications from a spectrum of devices without heavyweight interfaces.

The TinyOS provides a base framework and development environment that functions well under these extreme constraints of power, size, and cost. Additional software development can be done using the application developer to customize the distributed measurement application.

Bringing It All Together

MICA developers and U.S. Air Force Research Labs used the new technology to create a wireless sensor network at the 29 Palms Marine base in southern California. Using the TinyOS, the Rene Mote hardware, and an MTS magnetic sensor board, they deployed a real-world vehicle tracking application. An unmanned aircraft dropped about 30 wireless magnetic sensors along a road. The sensors were packaged in a thin layer of foam to protect them from the hard landing on the desert floor. Once safely on the ground, the sensors formed a wireless network and began looking for magnetic anomalies. As a vehicle passed by the sensors, they would detect the vehicle from its magnetic signature. As the vehicle continued along the network, the engineers were able to estimate the vehicle's speed and direction. The unmanned aircraft returned overhead to collect the data from the network and transmit them to the remote operation command headquarters. The entire development of the application, including the demonstration, took fewer than 60 days.

For Further Information

The MICA community welcomes new application developers and sensor manufacturers. Additional information can be found at:

- [MICA hardware](#)
- [TinyOS source code](#)
- [TinyOS and MICA sensors](#)

The Tiny OS

This operating system consists of a tiny scheduler and a graph of components. A component has four interrelated parts: a set of command handlers, a set of event handlers, an encapsulated fixed-size frame, and a bundle of simple tasks (see Figure 3). Tasks, commands, and handlers execute in the context of the frame and operate on its state.

To facilitate modularity, each component also declares the commands it uses and the events it signals. Commands are nonblocking requests made to lower-level components. Event handlers are invoked to deal with hardware events. Tasks perform the primary work for the application, but they can be pre-empted by events.

The task scheduler is a simple FIFO scheduler using a bounded size scheduling data structure. It's crucial that the scheduler is power aware. The TinyOS scheduler puts the processor to sleep when the tasks are complete.

Here is a pictorial example of a simple component and its function declarations:

Code Declarations:

```
/* Messaging Component Declarations */
//ACCEPTS:
char TOS_COMMAND(AM_send_msg)(int addr, int type, char* data);
void TOS_COMMAND(AM_power)(char mode);
char TOS_COMMAND(AM_init)();
//SIGNALS:
char AM_msg_rec(int type, char* data);
char AM_msg_send_done(char success);
//HANDLES:
```

Tiny OS Key Facts

Software Footprint 3.4 KB

Power Consumption on Rene Platform

Transmission Cost: 1 μ J/bit
Inactive State: 5 μ A
Peak Load: 20 mA

Efficient Concurrency Support

At peak load 50% CPU sleep

Efficient Modularity

Events propagate through stack <40 μ S

```

char AM_TX_packet_done(char success);
char AM_RX_packet_done(char* packet);
//USES:
char TOS_COMMAND(AM_SUB_TX_packet)(char* data);
void TOS_COMMAND(AM_SUB_power)(char mode);
char TOS_COMMAND(AM_SUB_init)();

```

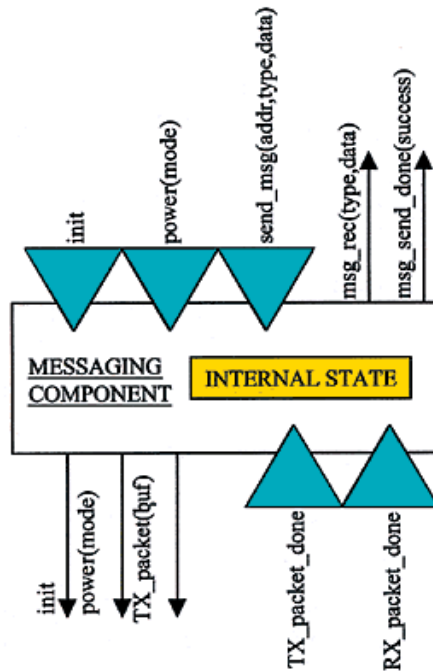


Figure 3. TinyOS structures the run-time software into components. Each component has state (component frame) and a bundle of tasks, commands, and handlers.