

## Sensor Node Hardware

### The various sensor node hardware.

- ❖ Sensor node hardware can be grouped into three categories, each of which entails a different set of trade-offs in the design choices.
- **Augmented general-purpose computers:**
  - ❖ These nodes typically run off-the-shelf (OTS) operating systems such as Win CE, Linux, or realtime operating systems and use standard wireless communication protocols such as Bluetooth or IEEE 802.11.
  - ❖ Because of their relatively higher processing capability, they can accommodate a wide variety of sensors, ranging from simple microphones to more sophisticated video cameras.
  - ❖ Compared with dedicated sensor nodes, PC-like platforms are more power hungry.
  - ❖ However, when power is not an issue, these platforms have the advantage that they can leverage the availability of fully supported networking protocols, popular programming languages, middleware, and other off-the-shelf(OTS) software.
  - ❖ Examples- Low power PCs, Embedded PCs (eg., PC104), Custom designed PCs(e.g., Sensoria WINS NG nodes, PDAs)
- **Dedicated embedded sensor nodes:**
  - ❖ These platforms typically use commercial OTS (COTS) chip sets with emphasis on small form factor, low power processing and communication, and simple sensor interfaces.
  - ❖ Because of their COTS CPU, these platforms typically support at least one programming language, such as C.
  - ❖ However, in order to keep the program footprint small to accommodate their small memory size, programmers of these platforms are given full access to hardware but barely any operating system support.
  - ❖ A classical example is the TinyOS platform and its companion programming language, nesC.
  - ❖ Example- Berkeley mote family, UCLA Medusa family, Ember nodes, MIT iAMP.
- **System-on-chip (SoC) nodes:**
  - ❖ Designers of these platforms try to push the hardware limits by fundamentally rethinking the hardware architecture trade-offs for a sensor node at the chip design level.
  - ❖ The goal is to find new ways of integrating CMOS, MEMS, and RF technologies to build extremely low power and small footprint sensor nodes that still provide certain sensing, computation, and communication capabilities.

- ❖ Since most of these platforms are currently in the research pipeline with no predefined instruction set, there is no software platform support available.
- ❖ Example- Smart dust, BWRC picoradio node, PASTA node.

### **BERKELEY NOTES**

- ❖ The Berkeley notes are a family of embedded sensor nodes sharing roughly the same architecture. Figure shows a comparison of a subset of mote types.







Mote type		WeC	Rene	Rene2	Mica	Mica2	Mica2Dot
Example picture							
MCU	Chip	AT90LS8535	ATmega163L	ATmega103L	ATmega128L		
	Type	4 MHz, 8 bit	4 MHz, 8 bit	4 MHz, 8 bit	8 MHz, 8 bit		
	Program memory (KB)	8	16	128	128		
	RAM (KB)	0.5	1	4	4		
External nonvolatile storage	Chip	24LC256			AT45DB014B		
	Connection type	I2C			SPI		
	Size (KB)	32			512		
Default power source	Type	Coin cell	2xAA			Coin cell	
	Typical capacity (mAh)	575	2850			1000	
RF	Chip	TR1000			CC1000		
	Radio frequency	868/916 MHz			868/916 MHz, 433, or 315 MHz		
	Raw speed (kbps)	10			40	38.4	
	Modulation type	On/Off key			Amplitude shift key	Frequency shift key	

FIGURE :A comparison of Berkeley notes.

#### ***Features of MICA mote:***

- ❖ The MICA notes have a two-CPU design.
- ❖ The main microcontroller (MCU), an Atmel ATmega103L, takes care of regular processing.
- ❖ A separate and much less capable coprocessor is only active when the MCU is being reprogrammed.
- ❖ The ATmega103L MCU has integrated 512 KB flash memory and 4 KB of data memory. Given these small memory sizes, writing software for notes is challenging.
- ❖ Ideally, programmers should be relieved from optimizing code at assembly level to keep code footprint small. However, high-level support and software services are not free.
- ❖ Being able to mix and match only necessary software components to support a particular application is essential to achieving a small footprint.

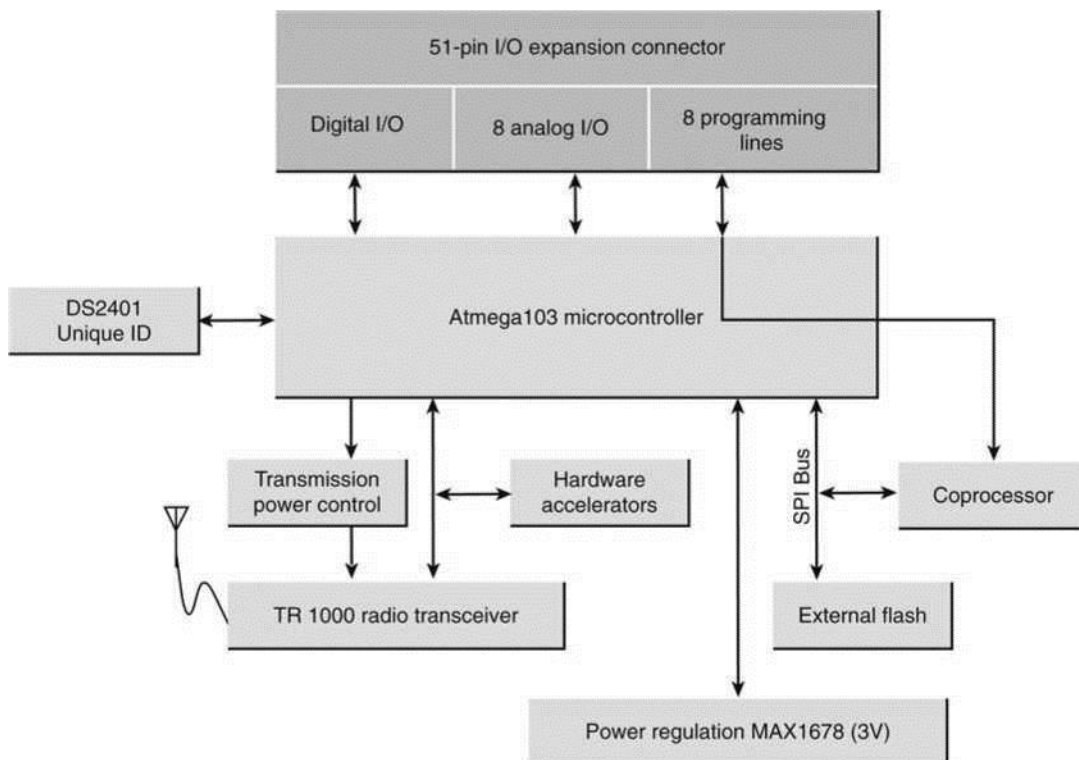


FIGURE :MICA mote architecture.

In addition to the memory inside the MCU, a MICA mote also has a separate 512 KB flash memory unit that can hold data.

- ❖ Since the connection between the MCU and this external memory is via a low-speed serial peripheral interface (SPI) protocol, the external memory is more suited for storing data for later batch processing than for storing programs.
- ❖ The RF communication on MICA motes uses the TR1000 chip set (from RF Monolithics, Inc.) operating at 916 MHz band.
- ❖ With hardware accelerators, it can achieve a maximum of 50 kbps raw data rate. MICA motes implement a 40 kbps transmission rate.
- ❖ The transmission power can be digitally adjusted by software through a potentiometer (Maxim DS1804). The maximum transmission range is about 300 feet in open space.
- ❖ A sensor/actuator board can host a temperature sensor, a light sensor, an accelerometer, a magnetometer, a microphone, and a beeper.
- ❖ The serial I/O (UART) connection allows the mote to communicate with a PC in real time.
- ❖ The parallel connection is primarily for downloading programs to the mote.

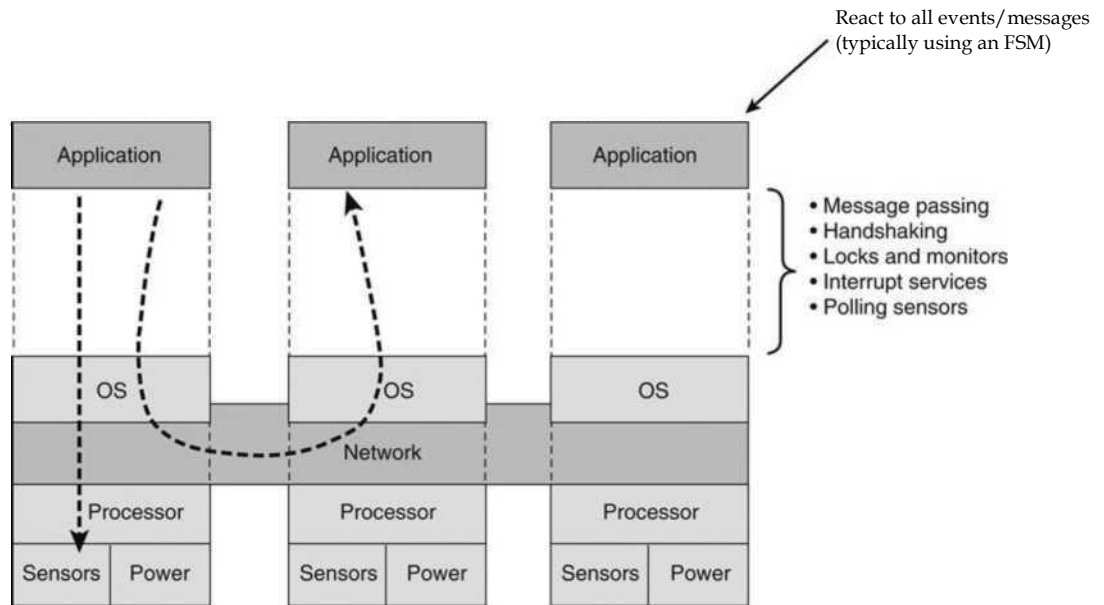
Component	Rate	Startup time	Current consumption
MCU active	4 MHz	N/A	5.5 mA
MCU idle	4 MHz	1 $\mu$ s	1.6 mA
MCU suspend	32 kHz	4 ms	<20 $\mu$ A
Radio transmit	40 kHz	30 ms	12 mA
Radio receive	40 kHz	30 ms	1.8 mA
Photoresister	2000 Hz	10 ms	1.235 mA
Accelerometer	100 Hz	10 ms	5 mA/axis
Temperature	2 Hz	500 ms	0.150 mA

FIGURE - Power consumption of MICA motes.

## Sensor Network Programming Challenges

### The challenges in Sensor network programming.

- ❖ Traditional programming technologies rely on operating systems to provide abstraction for processing, I/O, networking, and user interaction hardware, as illustrated in Figure below.
- ❖ When applying such a model to programming networked embedded systems, such as sensor networks, the application programmers need to explicitly deal with message passing, event synchronization, interrupt handling, and sensor reading.
- ❖ As a result, an application is typically implemented as a finite state machine (FSM) that covers all extreme cases: unreliable communication channels, long delays, irregular arrival of messages, simultaneous events, and so on.
- ❖ In a target tracking application implemented on a Linux operating system and with directed diffusion routing, roughly 40% of the code implements the FSM and the glue logic of interfacing computation and communication.
- ❖ For resource-constrained embedded systems with real-time requirements, several mechanisms are used in embedded operating systems to reduce code size, improve response time, and reduce energy consumption.



**FIGURE-** Traditional embedded system programming interface.

Real-time scheduling allocates resources to more urgent tasks so that they can be finished early. Event-driven execution allows the system to fall into low-power sleep mode when no interesting events need to be processed.

- ❖ At the extreme, embedded operating systems tend to expose more hardware controls to the programmers, who now have to directly face device drivers and scheduling algorithms, and optimize code at the assembly level.
- ❖ Although these techniques may work well for small, stand-alone embedded systems, they do not scale up for the programming of sensor networks for two reasons.
- ❖ Sensor networks are large-scale distributed systems, where global properties are derivable from program execution in a massive number of distributed nodes.
- ❖ Distributed algorithms themselves are hard to implement, especially when infrastructure support

is

limited due to the ad hoc formation of the system and constrained power, memory, and bandwidth resources.

- ❖ As sensor nodes deeply embed into the physical world, a sensor network should be able to respond to multiple concurrent stimuli at the speed of changes of the physical phenomena of interest.
- ❖ TinyOS and TinyGALS are two representative examples of node-level programming tools.
- ❖ Other related software platforms include Mate , a virtual machine for the Berkeley motes.

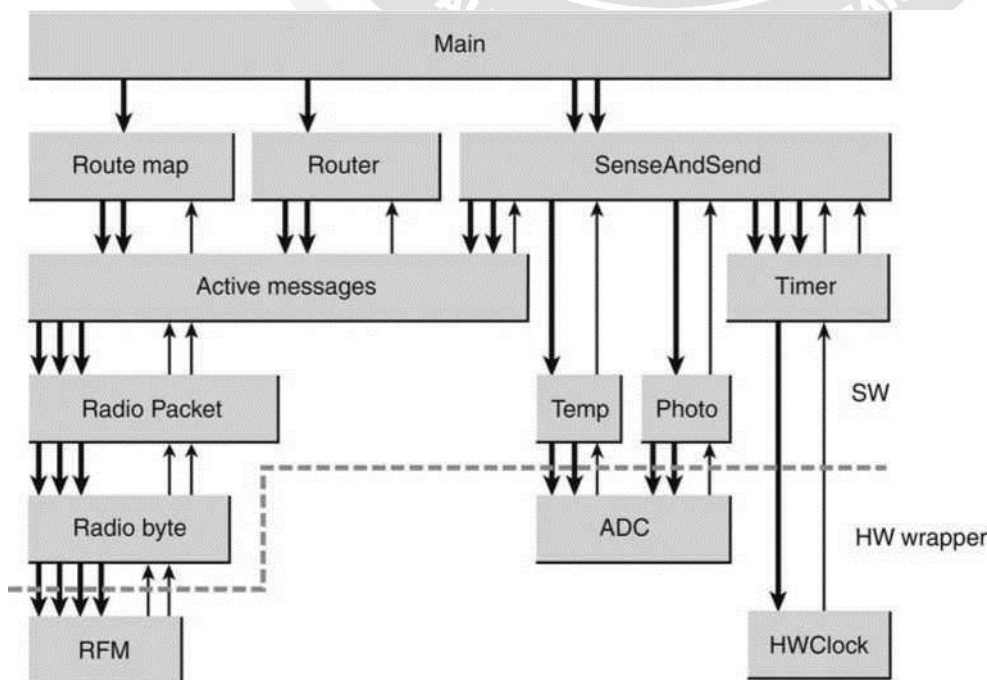
- ❖ Observing that operations such as polling sensors and accessing internal states are common to all sensor network application, Mate defines virtual machine instructions to abstract those operations.

### Node-Level Software Platforms

**The node level software platforms for Sensor networks.**

#### Operating System: TinyOS

- ❖ TinyOS aims at supporting sensor network applications on resource-constrained hardware platforms, such as the Berkeley motes.
- ❖ To ensure that an application code has an extremely small footprint.
- ❖ TinyOS chooses to have no file system, supports only static memory allocation, implements a simple task model, and provides minimal device and networking abstractions.
- ❖ To a certain extent, each TinyOS application is built into the operating system.
- ❖ Like many operating systems, TinyOS organizes components into layers.
- ❖ Intuitively, the lower a layer is, the “closer” it is to the hardware; the higher a layer is, the “closer” it is to the application.
- ❖ In addition to the layers, TinyOS has a unique component architecture and provides as a library a set of system software components.
- ❖ A component specification is independent of the component implementation.
- ❖ Although most components encapsulate software functionalities, some are just thin wrappers around hardware.

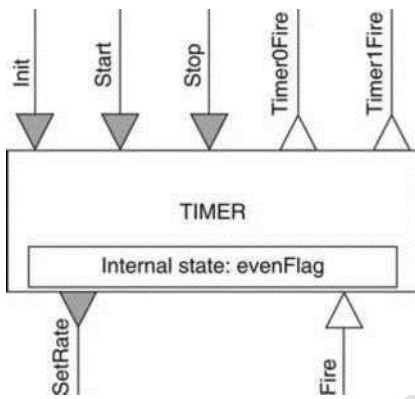


**FIGURE** The FieldMonitor application for sensing and sending measurements.

- ❖ An application, typically developed in the nesC language covered in the next section, *wires* these components together with other application-specific components.
- ❖ Let us consider a TinyOS application example—FieldMonitor, where all nodes in a sensor field periodically send their temperature and photosensor readings to a base station via an ad hoc routing mechanism.
- ❖ A diagram of the FieldMonitor application is shown in above Figure, where blocks represent TinyOS components and arrows represent function calls among them.
- ❖ The directions of the arrows are from callers to callees.

**Timer component of the Field Monitor application**

- ❖ This component is designed to work with a clock, which is a software wrapper around a hardware clock that generates periodic interrupts.
- ❖ The method calls of the Timer component are shown in the figure as arrowheads.
- ❖ An arrowhead pointing into the component is a method of the component that other components can call.
- ❖ An arrowhead pointing outward is a method that this component requires another layer component to provide.
- ❖ The absolute directions of the arrows, up or down, illustrate this component's relationship with other layers.
- ❖ *For example*, the Timer depends on a lower layer HWClock component.
- ❖ The Timer can set the rate of the clock, and in response to each clock interrupt it toggles an internal Boolean flag, evenFlag, between true (or 1) and false (or 0). If the flag is 0, the Timer produces a timer0Fire event to trigger other components;
- ❖ Otherwise, it produces a timer1Fire event.
- ❖ The Timer has an init() method that initializes its internal flag, and it can be enabled and disabled via the start and stop calls.



**FIGURE :** The Timer component and its interfaces.

- ❖ A program executed in TinyOS has two contexts, *tasks* and *events*, which provide two sources of concurrency.
- ❖ Tasks are created (also called *posted*) by components to a task scheduler.
- ❖ The default implementation of the TinyOS scheduler maintains a task queue and invokes tasks according to the order in which they were posted.
- ❖ Thus tasks are deferred computation mechanisms. Tasks always run to completion without preempting or being preempted by other tasks. Thus tasks are nonpreemptive.
- ❖ The scheduler invokes a new task from the task queue only when the current task has completed.
- ❖ When no tasks are available in the task queue, the scheduler puts the CPU into the sleep mode to save energy.
- ❖ The ultimate sources of triggered execution are events from hardware: clock, digital inputs, or other kinds of interrupts. The execution of an interrupt handler is called an *event context*.
- ❖ The processing of events also runs to completion, but it preempts tasks and can be preempted by other events.
- ❖ Another trade-off between nonpreemptive task execution and program reactivity is the design of split-phase operations in TinyOS.
- ❖ Similar to the notion of asynchronous method calls in distributed computing, a split-phase operation separates the initiation of a method call from the return of the call.
- ❖ A call to a split-phase operation returns immediately, without actually performing the body of the operation.
- ❖ The true execution of the operation is scheduled later; when the execution of the body finishes, the operation notifies the original caller through a separate method call.