Introduction to Motes

Portions adapted from:

Crossbow, Inc.

Bill Maurer
maurer@dsplabs.com
DSP Labs, Livermore Ca., USA

www.intel-research.net/berkeley/

Why Smart Dust?

• Advances in low power wireless communication technology and micro-electromechanical sensors (MEMS) transducers

• “Digital Nervous System”
• “Physical Internet”
• “Ubiquitous Computing”
NEST Technology

• How do you combine sensing, communication and computation into a complete architecture?
• What are the requirements of the software?
• Networked-Embedded-Systems-Technology

Ad hoc sensing

• Autonomous nodes self assembling into a network of sensors
• Sensor information propagated to central collection point
• Intermediate nodes assist distant nodes to reach the base station
Today’s Hardware

• Assembled from off-the-shelf components
• 8bit MCU
• Low-Power Radio
• Sensors
• I/O

Key Software Requirements

• Capable of fine grained concurrency
• Small physical size
• Efficient resource utilization
• Highly modular
What is TinyOS?

• An Open-Source Development Environment

• A Simple Operating System

• A Programming Language and Model

• A Set of Services

TinyOS – Development Environment

• Windows and Linux
• Multiple Hardware Platforms
  – Not only Crossbow
• Multiple Sensors
  – Not only Crossbow
• Debugging Tools
• Reference Applications
TinyOS – A Simple Operating System

- Scheduler
- Concurrency Intensive
- Limited Resources – SW Components for Efficient Modularity

TinyOS – A Programming Language and Model

- Separation of construction and composition:
  - programs are built out of components
- Specification of component behavior in terms of set of interfaces
- Components are statically wired to each other via their interfaces.
  - This increases runtime efficiency
TinyOS - Services

- Radio, MAC, Messaging, Routing
- Sensor Interfaces
- Power Management
- Security
- Debug
- Time

Why TinyOS?

- Unix Analog (aka 1969)
  - A Uniform Programming Language
    - C
  - A Uniform Abstraction
    - E.g., device abstraction
  - Open Source
    - Many Different Developers
    - Many Different Needs
  - Many Tools
Who Controls TinyOS?

- UC Berkeley Invented
  - 1st written by Jason Hill in 2000
  - Large portion of development changed to Intel-Berkeley Research Lab

- Intel-Berkeley Research Lab has largest role today in core ‘OS’ components
  - www.intel-research.net/berkeley/

Real-World Deployments

Great Duck Island
http://www.greatduckisland.net/

Center for Embedded Network Sensing
http://www.cens.ucla.edu/
Introduction to TinyOS and nesC Programming

- TinyOS Kernel Design and Implementation
- nesC Software Concepts and Basic Syntax
- nesC Code Lab
- TinyOS Packet Networking and PC Base Station Lab

TinyOS Design Goals

- Support Networked Embedded Systems
  - asleep but remain vigilant to stimuli
  - bursts of events and operations
- Support Mica Hardware
  - power, sensing, computation, communication
- Support Technological Advances
  - keep scaling down
  - smaller, cheaper, lower power
TinyOS Design Options

- Can’t Use Existing RTOS’s
  - Microkernel Architecture
    - VxWorks, QNX, WinCE, PalmOS
  - Execution Similar to Desktop Systems
    - PDA’s, Cell Phones, Embedded PC’s
  - More Than a Order of Magnitude Too Heavy & Slow
  - Energy Hog

TinyOS Design Conclusion

- Similar to Building Networking Interfaces
  - Data Driven Execution
  - Manage Large # of Concurrent Data Flows
  - Manage Large # of Outstanding Events
- Add: Managing Application Data Processing
- Conclusion: Need a Multi Threading Engine
  - Extremely Efficient
  - Extremely Simple
TinyOS Kernel Design

• TinyOS Kernel: 2 Level Scheduling Structure
  – Events
    • Small Amount of Processing
    • E.g. Timer, ADC Interrupts
    • Can Interrupt Longer Running Tasks
  – Tasks
    • Not Time Critical
    • Tasks - Larger Amount of Processing
    • E.g. Computing an Average on an Array
    • Run to Completion WRT other Tasks
      – Implies Only Need a Single Stack

TinyOS Applications Under The Hood

• Application is created in the nesC Language
  – Details of nesC Forthcoming
• nesC Programming Language Supports the TinyOS Kernel Design (Events and Tasks)
bool TOSH_run_next_task(void) {
    uint8_t old_full;
    void (*func)(void);  // Assuming func is a pointer to a function
    if (TOSH_sched_full == TOSH_sched_free) {
        return 0;
    } else {
        old_full = TOSH_sched_full;
        TOSH_sched_full++;
        TOSH_sched_full &= TOSH_TASK_BITMASK;
        func = TOSH_queue[(int)old_full].tp;
        TOSH_queue[(int)old_full].tp = 0;
        func();
        return 1;
    }
}

int main(void) {
    RealMain$hardwareInit();
    TOSH_sched_init();
    RealMain$StdControl$init();
    RealMain$StdControl$start();
    RealMain$Interrupt$enable();
    while (1) {
        TOSH_run_task();
    }
}

static inline void TOSH_run_task(void) {
    while (TOSH_run_next_task())
    TOSH_sleep();
    TOSH_wait();
}

Hardware and Kernel Initialization
Application Initialization
Infinite Loop

1. First Run All Tasks in the Task Queue (Strictly a FIFO)
2. Then Sleep (In Low Power Mode)
3. And Wait for an Interrupt

Task Runs To Completion (But May Be Interrupted By An Event)

---

Overhead of TinyOS Primitive Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cost(cycles)</th>
<th>Time(μSecs)</th>
<th>Normalized to Byte Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte Copy</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Signal an Event</td>
<td>10</td>
<td>2.5</td>
<td>1.25</td>
</tr>
<tr>
<td>Call a Command</td>
<td>10</td>
<td>2.5</td>
<td>1.25</td>
</tr>
<tr>
<td>Schedule a Task</td>
<td>46</td>
<td>11.5</td>
<td>6</td>
</tr>
<tr>
<td>Context Switch</td>
<td>51</td>
<td>12.75</td>
<td>6</td>
</tr>
<tr>
<td>Hardware Interrupt (hw)</td>
<td>9</td>
<td>2.25</td>
<td>1</td>
</tr>
<tr>
<td>Hardware Interrupt (sw)</td>
<td>71</td>
<td>17.75</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code Size(bytes)</th>
<th>Data Size(bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Init</td>
<td>172</td>
</tr>
<tr>
<td>Scheduler</td>
<td>178</td>
</tr>
<tr>
<td>C runtime</td>
<td>82</td>
</tr>
<tr>
<td>Total</td>
<td>432</td>
</tr>
</tbody>
</table>
TinyOS/nesC Application Notes

- Everything is Static
  - No Dynamic Memory (no malloc)
  - No Function Pointers
  - No Heap
- nesC Compiler Analysis
  - Data Race Conditions
  - Function Inlining
  - Development Made Easier
  - Robustness Improved

Application Memory Map

- Text/code - Executable Code
  - In the 128K Program Flash
- data – Program Constants
  - In the 128K Program Flash
- bss - Variables
  - In the 4K SRAM
- Free Space - Fixed (No Dynamic Memory)
- stack - Grows Down in the Free Space
TinyOS Concepts Embodied by nesC
– Tasks, Events, Commands

• Tasks
  – Background computation, non-time critical

• Events
  – Time critical
  – External Interrupts
  – Originator gives a ‘Signal’
  – Receiver gets/accepts an ‘Event’

• Command
  – Function call to another Component
  – Cannot Signal

Concepts of SW Components

- **Interfaces** (xxx.nc)
  - Specifies functionality to outside world
  - Tell outside world
    - what commands can be called
    - what events need handling

- **Software Components**
  - **Module** (xxxM.nc)
    - Code file, code implementation
    - It codes the Interface
  - **Configuration** (xxxC.nc)
    - Linking/wiring of components
    - When top level app, drop C from filename xxx.nc
    - optional Module

TinyOS app Blink – Blinks the Red LED
BlinkM.nc
Blink.nc

Ad infinitum...
Blink.nc Application - A top level configuration
SW component used to form an executable

What the executable does:
1. Main initializes and starts the application.
2. BlinkM initializes ClockC's rate at 1 Hz.
3. ClockC continuously signals BlinkM at a rate of 1 Hz.
4. BlinkM commands LedsC red led to toggle each time it receives a signal from ClockC.

Note: The StdControl interface is similar to state machines (init, start, stop); used extensively throughout TinyOS apps & libs

Mote Hardware
Family of Motes

<table>
<thead>
<tr>
<th>Mote Type</th>
<th>WeC</th>
<th>Renee</th>
<th>Mica</th>
<th>Mica2</th>
<th>Mica2Dot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>AT90L88855</td>
<td>Atmega163</td>
<td>Atmega128</td>
<td>Atmega28</td>
<td>Atmega128</td>
</tr>
<tr>
<td>CPU Clock (MHz)</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>7.3827</td>
<td>4</td>
</tr>
<tr>
<td>Program Memory (KB)</td>
<td>8</td>
<td>16</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Ram (KB)</td>
<td>0.5</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>UARTs</td>
<td>1</td>
<td>1</td>
<td>2 (only 1 used)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SPI</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>I2C</td>
<td>Software</td>
<td>Software</td>
<td>Software</td>
<td>Hardware</td>
<td>Hardware</td>
</tr>
<tr>
<td>Nonvolatile storage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chip</td>
<td>24LC256</td>
<td>AT45DB041B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size (KB)</td>
<td>32</td>
<td>512</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio</td>
<td>RFM TR1000</td>
<td>Chipcon CC1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>916 (single freq)</td>
<td>916/433 (multiple channels)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio speed (kbps)</td>
<td>OOK</td>
<td>ASK</td>
<td>FSK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmit Power Control</td>
<td>Programmable resistor potentiometer</td>
<td>Programmable via CC1000 registers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td>SecMode (software)</td>
<td>Manchester (hardware)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mica2 and Mica2Dot

- ATmega128 CPU
  - Self-programming
- Chipcon CC1000
  - FSK
  - Manchester encoding
  - Tunable frequency
- Lower power consumption
Common Platform Architecture

- Atmega uP
  - 32Khz crystal and 4Mhz (7.3728Mhz Mica2) crystal.
- 10 bit ADC
- UART (Mica2/Mica2Dot have 2)
- SPI bus
- I2C bus (hardware for mica2/mica2dot)
- Radio (RFM or Chipcon 1000)
- External serial flash memory (512K byte)
- Connectors for interfacing to sensor and programming boards
- 3 programmable leds (1 for Mica2Dot)
- JTAG port (Mica, Mica2, Mica2Dot)

The CC1000 Radio Interface

- Dedicated cpu bus (lines) to configure radio registers for radio frequency, power,.....
- Dedicated SPI bus for data transfer. CC1000 is bus master.
- Radio generates one interrupt every 8 bits when in receive mode.
- Runs usually at 38K or 19K bit rate (default) Manchester (2x bit)
- More in-depth radio discussion later in session.

<table>
<thead>
<tr>
<th>Baud Rate</th>
<th>Xmt or Rcv Time(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>19K</td>
<td>~40msec</td>
</tr>
<tr>
<td>38K</td>
<td>~20msec</td>
</tr>
</tbody>
</table>

(*) Does not include random delay
The Flash Memory Interface

- 512 K bytes of flash (non-volatile) storage
- Useful for data logging.
- Used by GSK (Generic Sensor Kit) and TinyDB for data logging.
- Used by XNP for code download.
- Serial interface to Atmega uP
- TinyOS driver (Logger..) bit bangs interface
- Attached to 2nd uart port on Mica2. Another driver (UCB) uses synchronous uart for high speed data transfer (5KB/Sec driver)
- Beware, device consumes 15 ma when storing to memory

The ADC Interface

- Eight channels of 10 bit ADC, multiplexed.
- Dedicated channels (Mica2):
  - ADC0 : Radio’s RSSI
- Shared Mica2 Channels
  - ADC7 : Battery monitor (can be shared with another channel but will have ~10K ohm impedance.
  - ADC4..ADC7: JTAG. If using JTAG debug these channels won’t work as ADC inputs.
- Shared Mica2Dot Channels
  - ADC1: Shared for both thermistor and battery voltage
  - ADC4..ADC7: JTAG. If using JTAG debug these channels won’t work as ADC inputs
<table>
<thead>
<tr>
<th>Pin</th>
<th>Name</th>
<th>Description</th>
<th>Pin</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
<td>Ground</td>
<td>27</td>
<td>UART_TXDO</td>
<td>Uart0 Tx</td>
</tr>
<tr>
<td>2</td>
<td>VSNR</td>
<td>Voltage (battery)</td>
<td>28</td>
<td>UART_RXDO</td>
<td>Uart0 Rx</td>
</tr>
<tr>
<td>3</td>
<td>INT0</td>
<td>GPIO</td>
<td>29</td>
<td>PW1</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>4</td>
<td>INT1</td>
<td>GPIO</td>
<td>30</td>
<td>PW2</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>5</td>
<td>INT2</td>
<td>GPIO</td>
<td>31</td>
<td>PW3</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>6</td>
<td>INT3</td>
<td>GPIO</td>
<td>32</td>
<td>PW4</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>7</td>
<td>BAT_MON</td>
<td>Battery Monitor Voltage</td>
<td>33</td>
<td>PW5</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>8</td>
<td>LED1</td>
<td>Green Led</td>
<td>34</td>
<td>PW6</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>9</td>
<td>LED2</td>
<td>Yellow Led</td>
<td>35</td>
<td>PW7</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>10</td>
<td>LED3</td>
<td>Red Led</td>
<td>36</td>
<td>PW8</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>11</td>
<td>RD</td>
<td>GPIO</td>
<td>37</td>
<td>PW9</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>12</td>
<td>WR</td>
<td>GPIO</td>
<td>38</td>
<td>PW10</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>13</td>
<td>ALE</td>
<td>GPIO</td>
<td>39</td>
<td>PW11</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>14</td>
<td>PROG_MOSI</td>
<td>Programmer Pin</td>
<td>40</td>
<td>PW12</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>15</td>
<td>PROG_MISO</td>
<td>Programmer Pin</td>
<td>41</td>
<td>PW13</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>16</td>
<td>SPI_CLK</td>
<td>Radio Clock</td>
<td>42</td>
<td>PW14</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>17</td>
<td>USART1_CLK</td>
<td>Usart clock</td>
<td>43</td>
<td>PW15</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>18</td>
<td>USART1_RXD</td>
<td>Usart receive</td>
<td>44</td>
<td>PW16</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>19</td>
<td>USART1_TXD</td>
<td>Usart xmit</td>
<td>45</td>
<td>PW17</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>20</td>
<td>I2C_CLK</td>
<td>I2C bus clock</td>
<td>46</td>
<td>PW18</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>21</td>
<td>I2C_DATA</td>
<td>I2C bus data</td>
<td>47</td>
<td>PW19</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>22</td>
<td>ADC2</td>
<td>GPIO/ADC CH2</td>
<td>48</td>
<td>PW20</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>23</td>
<td>ADC1</td>
<td>GPIO/ADC CH1</td>
<td>49</td>
<td>PW21</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>24</td>
<td>ADC0</td>
<td>GPIO/ADC CH0</td>
<td>50</td>
<td>PW22</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>25</td>
<td>GND</td>
<td>Ground</td>
<td>51</td>
<td>VCC</td>
<td>Voltage (battery)</td>
</tr>
<tr>
<td>26</td>
<td>AC+</td>
<td>Blue: OK to use</td>
<td>52</td>
<td>GND</td>
<td>ground</td>
</tr>
<tr>
<td>27</td>
<td>AC-</td>
<td>Yellow: OK to use but has shared functionality</td>
<td>53</td>
<td>THRU1</td>
<td>Thru user connect</td>
</tr>
<tr>
<td>28</td>
<td>BAT_MON</td>
<td>Red: Do not use</td>
<td>54</td>
<td>THRU2</td>
<td>Thru user connect</td>
</tr>
<tr>
<td>29</td>
<td>LED2</td>
<td>Blue: OK to use</td>
<td>55</td>
<td>THRU3</td>
<td>Thru user connect</td>
</tr>
<tr>
<td>30</td>
<td>LED3</td>
<td>Blue: OK to use</td>
<td>56</td>
<td>PW23</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>31</td>
<td>LED4</td>
<td>Blue: OK to use</td>
<td>57</td>
<td>PW24</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>32</td>
<td>LED5</td>
<td>Blue: OK to use</td>
<td>58</td>
<td>PW25</td>
<td>GPIO/PWM</td>
</tr>
<tr>
<td>33</td>
<td>LED6</td>
<td>Blue: OK to use</td>
<td>59</td>
<td>PW26</td>
<td>GPIO/PWM</td>
</tr>
</tbody>
</table>

See Atmega128 specification for more information regarding signal functionality.
Mica2Dot Sensor Interface

- 6 ADC Channels
- 6 I/O Channels

Power Budgets

<table>
<thead>
<tr>
<th>SYSTEM SPECIFICATIONS</th>
<th>Currents value</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micro Processor (Atmega128L)</td>
<td>current (full operation)</td>
<td>6 ma</td>
</tr>
<tr>
<td></td>
<td>current sleep</td>
<td>8 ua</td>
</tr>
<tr>
<td>Radio (Chipcon 1000)</td>
<td>current in receive</td>
<td>8 ma</td>
</tr>
<tr>
<td></td>
<td>current xmit</td>
<td>12 ma</td>
</tr>
<tr>
<td></td>
<td>current sleep</td>
<td>2 ua</td>
</tr>
<tr>
<td>Flash Serial Memory (AT45DB041)</td>
<td>write</td>
<td>15 ma</td>
</tr>
<tr>
<td></td>
<td>read</td>
<td>4 ma</td>
</tr>
<tr>
<td></td>
<td>sleep</td>
<td>2 ua</td>
</tr>
<tr>
<td>Sensor Board</td>
<td>current (full operation)</td>
<td>5 ma</td>
</tr>
</tbody>
</table>

Average, full operation, current: ~15 ma

AA Batteries are ~1800ma which mean ~ 120hrs (5 days)
**Batteries**

- Lithium – 3.6, fast decay, more expensive.
- Beware of low battery voltage (adc, flash programming….)
- DC Booster may/may not help
- UCB Mica2Dot NiMH 3AH, single cell, with booster and recharge.

---

**Crossbow Sensor Boards**

<table>
<thead>
<tr>
<th>Part #</th>
<th>Mote Support</th>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTS101CA</td>
<td>MICA,MICA2</td>
<td>Light (photo resistor) Temperature (Thermistor) Prototyping area</td>
</tr>
<tr>
<td>MDA300CA</td>
<td>MICA2DOT</td>
<td>Prototyping</td>
</tr>
<tr>
<td>MTS300CA</td>
<td>MICA, MICA2</td>
<td>Light, Temperature, Acoustic, Sounder, 2-Axis Accelerometer (ADXL202), and 2-Axis Magnetometer</td>
</tr>
<tr>
<td>MTS500CA</td>
<td>Mica2Dot</td>
<td>Prototyping</td>
</tr>
<tr>
<td>MDA300CA</td>
<td>Mica2</td>
<td>On board humidity/temp. External sensors.</td>
</tr>
<tr>
<td>MTS400/420</td>
<td>Mica2</td>
<td>GPS weatherboard</td>
</tr>
<tr>
<td>Not released:</td>
<td>Mica2Dot</td>
<td>Weatherboards</td>
</tr>
</tbody>
</table>

See MTS/MDA Sensor and Data Acquisition Boards User's Manual
MTS101CA

- Light photo resistor - Clairex CL94L
- Thermistor - YSI 44006,
- Both sensor are highly non-linear.
- Good prototyping area.

To use this sensor board add (modify) the apps/app/makefile for:

```
SENSORBOARD = basicsb
```

MDA500CA

- Prototyping board for mica2dots
MTS300CA/MTS310CA

- Light (Photo)-Clairex CL94L
- Temperature-Panasonic ERT-J1VR103J
- Acceleration-ADI ADXL202
  - 2 axis
  - Resolution: ±2mg
- Magnetometer-Honeywell HMC1002
  - Resolution: 134μG
- Microphone
- Tone Detector
- Sounder
  - 4.5kHz

SENSORBOARD = micasb

MTS400/420 – GPS/Weather

- Gps (LeadTek 9546) - optional
  - SiRFstartII LP chipset (60ma)
  - External active antenna.
  - 12 channels
  - 15 Meter ( SA off); 7 Meter (WAAS corrected)
  - DC Booster to maintain required voltage
- Temperature & Humidity (Sensirion SHT11).
  - All digital (14 bits)
  - 3.5% RH accuracy, 0.5degC
  Temperature accuracy
MTS400/420 – GPS/Weather

- Barometric Pressure and Temperature (Intersema MS5534A)
  - All digital
  - 300 to 1100 mbar, 3% accuracy
  - -10 to +60 degC, 3% accuracy
- Ambient Light (TAOS TSL2250)
  - All digital
  - 400-1000nm response
- Acceleration-ADI ADXL202
  - 2 axis
  - Resolution: ±2mg
- 2 K EEPROM for user configuration info.

MDA300

- 8 External Analog Inputs
  - External Sensors
  - Hi and low level signals
  - Block Screw Terminal
- 8 channel digital I/O
- 2 relays
- On board 12-bit ADC
  - 0-2.5V, 0-3V, 0-5V Ranges
- Stable 2.5V Reference
- 3V and 5V power
- Designed by UCLA CENS w/ Crossbow and UCB

http://www.cens.ucla.edu/~mhr/daq/
PNI- Magnetometer/Compass
• Resolution: 400 μGauss
• Very low power
• Three axis

Ultrasonic Transceiver
• Used for ranging
• Up to 2.5m range
• 6cm accuracy
• Dedicated microprocessor
• 25kHz element
• Mica2 and Mica2Dot versions
Mica2Dot WB

- UCB environmentally packaged weatherboards for GDI
- Temperature & Humidity (Sensirion SHT11)
  - All digital (14 bits)
  - 3.5% RH accuracy, 0.5degC Temperature accuracy
- Barometric Pressure and Temperature (Intersema MS5534A)
  - All digital
  - 300 to 1100 mbar, 3% accuracy
  - -10 to +60 degC, 3% accuracy
- Ambient Light (TAOS TSL2250)
  - All digital
  - 400-1000nm response
  - Photosensitive Light Sensor.

Mote In Tires

- Real time control of vehicle dynamics.
- 3 bridge accelerometers (500g-1000g) mounted in tire.
- Sensor board has 3 channels of amplifiers, filters, programmable D/As for bridge balancing.
- Monitor and analyzed acceleration forces when tire is in contact with ground.
- Transmit results every revolution.
- 3 motes, 1 master, 2 slaves.
Micro Radar

- Darpa project: Detect intruders with micro-powered radar detectors and relay data through mote network.
- Drop detectors from UAV (ex: Predator)
- Ghz Doppler radar detector.
- Done with LLL and Advantaca

COTS-BOTS (UCB)

- 5” x 2.5” x 3” size
- <$250 total
- 2-axis accelerometer
Robomote (USC)

- Less than 0.000047m$^3$
- $150 each
- Platform to test algorithms for adaptive wireless networks with autonomous robots

MICAbot (Notre Dame)

- Designed for large-scale research in distributed robotics and ad-hoc wireless networking.
- $300 each
Ratiometric Adcs & Sensors

- Atmega128 is 10 bit (1024) ratio metric ADC
- If sensor is ratio metric then don’t have to measure battery voltage. (Sensor’s FS changes with battery voltage).
- Ratio metric sensors may not work over full range of battery voltage.
- ADC full scale is proportional to battery voltage.
- Must measure battery voltage to get accurate sensor readings:
  
  \[ \text{Battery Volts} = \text{RefVolt} \times \frac{\text{ADC FS}}{\text{data}} \]

- Mica2 and Mica2Dot have on-board voltage references to calibrate the ADC full scale.

\[ /\text{contrib/xbow/apps/XSensorMica2 } \]

Enclosures for Environmental Monitoring
Sensor Power Management

- **Simple Strategy for Low Power Sensors:**
  - Use Atmega output pins to source sensor power.
  - Will source ~5-10mA of current per pin.

- **Analog Switch Strategy**
  - Use hardware I2C (mica2) or software I2C (mica2dot) (in Sourceforge)
  - Switch connects sensor power to VCC.
  - ADG714 switch has 2.5 ohm on resistance

- **DC-DC Booster Strategy**
  - Create battery independent, constant supply voltage.
  - Create +5 V or more
  - Turn on booster from analog switch or Atmega
  - Boosters are ~80% to 90% efficient. Need good layout and decoupling. Not ratiometric
Mote Programming and Base Station Boards

Overview:

- MIB500 Parallel Port Programmer
- MIB510 Serial Port Programmer
- eMote
- USB

MIB500

- Programs mote through the PC’s parallel port
- Supports Mica, Mica2, Mica2Dot
- Voltage monitor to protect from low battery voltage. Low battery voltage can cause fuse errors.
- Serial port for base station operation
- Parallel port can cause flash corruption on some computers due to uisp parallel port drivers. THESE MAY BE IRRECOVERABLE
- Crossbow application note at www.xbow.com to help fix uisp problems.
- JTAG connector: AVRStudio and JTAG pod allows viewing and setting all fuses.
MIB510

• Q3 release

• Programming through the serial port. On board ISP uP is 3x faster than parallel port.

• Shares serial port with mote for base station operation.

• Voltage monitor to protect from low battery voltage

• Supports Mica (Atmega128 uP on Mica2, Mica2Dot

• JTAG port powered directly.

USB

• Q4 release

• USB interface for programming and base station operation.

• Power supplied thru USB.
Ethernet connection as serial forwarder.

- Programming through ethernet.
- Remote base station operation through ethernet.
- Remote powered ethernet sensor.
- Remote code debugging through ethernet. Ideal for mote network debug.

- Similar configuration (eprb) used extensively at UCB for mote development.