System Architecture Directions for Networked Sensors

By Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister
SMS messaging was first used in December 1992, when Neil Papworth, a 22-year-old test engineer for Sema Group used a personal computer to send the text message "Merry Christmas"
Context

- Sensor network motes are
  - Highly application specific
  - Broad span in the design space

- Problem
  - Sensor network specific operating system missing

- Approach
  - Modular, micro-threaded OS
Sensor Network Characteristics

- Small size, low power
- Concurrency intensive
- Limited hardware support
- Diversity in design and usage
- Robust operation
### Yet Another Mote

#### AT 90L8535

- **8-bit data bus**
- **SRAM**
- **PC**
- **Pgm. mem. (flash)**
- **Inst. Register**
- **Inst. Decoder**
- **Ctrl lines**
- **Regs**
- **ALU**
- **SR**
- **SP**
- **EEPROM**
- **Reference Voltage**
- **4 MHz clock**
- **32.768 MHz clock**

- **SPI**
- **UART**
- **IO pins**
- **Pwr data**
- **Ctrl**
- **IO pins**
- **RX**
- **TX**
- **RFM TR100 916 MHz transceiver**

- **Coprocessor AT90L2313**
- **EEPROM**
- **L2C**

---

#### Component Power Consumption Table

<table>
<thead>
<tr>
<th>Component</th>
<th>Active (mA)</th>
<th>Idle (mA)</th>
<th>Inactive (µA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU core (AT90S8535)</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MCU pins</td>
<td>1.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LED</td>
<td>4.6 each</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Photocell</td>
<td>.3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Radio (RFM TR1000)</td>
<td>12 tx</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>Radio (RFM TR1000)</td>
<td>4.5 rx</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>Temp (AD7416)</td>
<td>1</td>
<td>0.6</td>
<td>1.5</td>
</tr>
<tr>
<td>Co-proc (AT90LS2343)</td>
<td>2.4</td>
<td>.5</td>
<td>1</td>
</tr>
<tr>
<td>EEPROM (24LC256)</td>
<td>3</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>
## Vs. TelosB

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM</td>
<td>8 KiB</td>
<td>48 KiB</td>
</tr>
<tr>
<td>RAM</td>
<td>512 B</td>
<td>10 KiB</td>
</tr>
<tr>
<td>Clock</td>
<td>4 MHz</td>
<td>8 MHz</td>
</tr>
<tr>
<td>Power (active/idle)</td>
<td>5mA/2mA</td>
<td>2mA/1mA</td>
</tr>
<tr>
<td>Peripherals</td>
<td>I(^2)C, SPI, UART, ADC, Timers</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Radio</th>
<th>TelosB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>19.2 kbps</td>
</tr>
<tr>
<td>Type</td>
<td>Bit</td>
</tr>
<tr>
<td>Power (rx/tx)</td>
<td>4 mA / 12 mA</td>
</tr>
</tbody>
</table>
Challenge

- Meet sensor network characteristics AND run on mote-class device

- TinyOS
  - Event driven
  - Two-level scheduling: Tasks and events
  - Scalable hardware/software boundary
  - No blocking functions or polling
  - No dynamic memory
Running Example Application

- Environmental Monitoring
  - Sample sensors
  - Store measurements
  - Transmit measurements
  - Forward other’s measurements
Modularized OS

- Divide application and OS in components

Components
  - Self-contained
  - Well-defined interfaces (uses and provides)
  - Three layer abstraction:
    • Application
    • Hardware Independent Layer
    • Hardware Abstraction Layer
Components

- Interfaces
  - Provides (up)
  - Uses (down)

- Mutators
  - Call command (down)
  - Signal event (up)

- Tasks
  - Low priority
  - Functions, command calling, event signaling

- Asynchronous events
  - High priority
  - Hardware interrupt handler
void main( void )
{
    ...
    while(1) {
        while (taskQueue.notEmpty()) {
            task = taskQueue.next();
            task.run();
        }
        sleep();
    }
    ...
}

void taskAdc() { ... };
void taskRadio() { ... };
void taskTimer() { ... };
...

void interruptTimer()
{
    time = readTimer(); // 16–bit value
    taskQueue.insert(taskTimer);
}

void interruptAdc()
{
    value = readAdc(); // 16–bit value
    taskQueue.insert(taskAdc);
}

void interruptRadio()
{
    // 128 bytes packet – post task
    taskQueue.insert(taskRadio);
}
“Surreal, but nice”

- **Concurrency**
  - Tasks run atomically with regard to other tasks
  - Race conditions can be detected at compile time

- **Split-phase operation**
  - Commands are non-blocking
  - E.g. call command ‘start’, signal event ‘startDone’

- **Modular**
  - Code reuse
  - Flexible hardware/software boundary
## Code Size

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Code Size (bytes)</th>
<th>Data Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multihop router</td>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>AM_dispatch</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>AM_temperature</td>
<td>78</td>
<td>32</td>
</tr>
<tr>
<td>AM_light</td>
<td>146</td>
<td>8</td>
</tr>
<tr>
<td>AM</td>
<td>356</td>
<td>40</td>
</tr>
<tr>
<td>Packet</td>
<td>334</td>
<td>40</td>
</tr>
<tr>
<td>RADIO_byte</td>
<td>810</td>
<td>8</td>
</tr>
<tr>
<td>RFM</td>
<td>310</td>
<td>1</td>
</tr>
<tr>
<td>Photo</td>
<td>84</td>
<td>1</td>
</tr>
<tr>
<td>Temperature</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>UART</td>
<td>196</td>
<td>1</td>
</tr>
<tr>
<td>UART_packet</td>
<td>314</td>
<td>40</td>
</tr>
<tr>
<td>I2C_bus</td>
<td>198</td>
<td>8</td>
</tr>
<tr>
<td>Procesor_init</td>
<td>172</td>
<td>30</td>
</tr>
<tr>
<td>TinyOS scheduler</td>
<td>178</td>
<td>16</td>
</tr>
<tr>
<td>C runtime</td>
<td>82</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>3450</strong></td>
<td><strong>226</strong></td>
</tr>
</tbody>
</table>
178 bytes
- SMS messaging was first used in December 1992, when Neil Papworth, a 22-year-old test engineer for Sema Group used a personal computer to send the text message "Merry Christmas"

SMS message
- 160 bytes

Contiki OS Kernel
- 810 bytes
### Execution Time

<table>
<thead>
<tr>
<th>Operations</th>
<th>Cost (cycles)</th>
<th>Time (μs)</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>Post 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>Post a task to scheduler</td>
<td>46</td>
<td>11.5</td>
<td>6</td>
</tr>
<tr>
<td>Context switch overhead</td>
<td>51</td>
<td>12.75</td>
<td>6</td>
</tr>
<tr>
<td>Interrupt (hardware cost)</td>
<td>9</td>
<td>2.25</td>
<td>1</td>
</tr>
<tr>
<td>Interrupt (software cost)</td>
<td>71</td>
<td>17.75</td>
<td>9</td>
</tr>
</tbody>
</table>

- **Radio bit**: 40 μs
- **(QNX Context Switch)**: 72 μs
  - BlackBerry PlayBook
Signal Propagation

Diagram showing the process of signal propagation with labeled stages and timing notations.
## Profiling

<table>
<thead>
<tr>
<th>Components</th>
<th>Packet reception breakdown</th>
<th>Percent CPU Utilization</th>
<th>Energy (nJ/bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>0.05%</td>
<td>0.02%</td>
<td>0.33</td>
</tr>
<tr>
<td>Packet</td>
<td>1.12%</td>
<td>0.51%</td>
<td>7.58</td>
</tr>
<tr>
<td>Radio handler</td>
<td>26.87%</td>
<td>12.16%</td>
<td>182.38</td>
</tr>
<tr>
<td>Radio decode task</td>
<td>5.48%</td>
<td>2.48%</td>
<td>37.2</td>
</tr>
<tr>
<td>RFM</td>
<td>66.48%</td>
<td>30.08%</td>
<td>451.17</td>
</tr>
<tr>
<td>Radio Reception</td>
<td>-</td>
<td>-</td>
<td>1350</td>
</tr>
<tr>
<td>Idle</td>
<td>-</td>
<td>54.75%</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>100.00%</td>
<td>100.00%</td>
<td>2028.66</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Components</th>
<th>Packet transmission breakdown</th>
<th>Percent CPU Utilization</th>
<th>Energy (nJ/bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>0.03%</td>
<td>0.01%</td>
<td>0.18</td>
</tr>
<tr>
<td>Packet</td>
<td>3.33%</td>
<td>1.59%</td>
<td>23.89</td>
</tr>
<tr>
<td>Radio handler</td>
<td>35.32%</td>
<td>16.90%</td>
<td>253.55</td>
</tr>
<tr>
<td>Radio encode task</td>
<td>4.53%</td>
<td>2.17%</td>
<td>32.52</td>
</tr>
<tr>
<td>RFM</td>
<td>56.80%</td>
<td>27.18%</td>
<td>407.17</td>
</tr>
<tr>
<td>Radio Transmission</td>
<td>-</td>
<td>-</td>
<td>1800</td>
</tr>
<tr>
<td>Idle</td>
<td>-</td>
<td>52.14%</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>100.00%</td>
<td>100.00%</td>
<td>4317.89</td>
</tr>
</tbody>
</table>
Summary

- TinyOS
  - Event driven OS
  - 2-level scheduling (Tasks, Asynchronous Events)
  - Modular design
  - Component graph
  - Tradeoffs code complexity with high concurrency
Homework Assignment 1

- Due date:
  - Friday 30\textsuperscript{th} September

- Hand-in policy:
  - 10% deduction each day late

- Implement
  - Read/parse packets from the Bridge
  - Write/construct packets to the Bridge
  - Control panel GUI and message handler
Homework Assignment 1

We provide
- Skeleton code (only fill out the missing parts)
- Binaries for the Android, Bridge, and Probe
- Code: local database, remote database, graphs

Deliverables:
- Report
  - Approach, design choices, etc.
  - Implementation, evaluation (bug reporting), etc.
  - Future Work
- The files containing your code
Homework Assignment 1

- Grading:
  - 40% Report
  - 20% “It works!”
  - 20% Android UART communication
  - 20% Android GUI and message handler