Integrating Concurrency Control and Energy Management in Device Drivers

Chenyang Lu
Why worry about energy?

No Moore’s Law in batteries: 2-3%/year growth!
## Low-Power Sensor Platforms

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>AT90LS8535</td>
<td>ATmega163</td>
<td>ATmega128</td>
<td></td>
<td>TI MSP430</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program memory (KB)</td>
<td>8</td>
<td>16</td>
<td>128</td>
<td></td>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAM (KB)</td>
<td>0.5</td>
<td>1</td>
<td>4</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Active Power (mW)</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>60</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sleep Power (μW)</td>
<td>45</td>
<td>45</td>
<td>75</td>
<td>75</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wakeup Time (μs)</td>
<td>1000</td>
<td>36</td>
<td>180</td>
<td>180</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonvolatile storage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chip</td>
<td>24LC256</td>
<td></td>
<td>AT45DB041B</td>
<td></td>
<td>ST M24M01S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection type</td>
<td>I²C</td>
<td></td>
<td>SPI</td>
<td></td>
<td>I²C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size (KB)</td>
<td>32</td>
<td></td>
<td>512</td>
<td></td>
<td>128</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio</td>
<td>TR1000</td>
<td>TR1000</td>
<td>CC1000</td>
<td></td>
<td>CC2420</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data rate (kbps)</td>
<td>10</td>
<td>40</td>
<td>38.4</td>
<td></td>
<td>250</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modulation type</td>
<td>OOK</td>
<td>ASK</td>
<td>FSK</td>
<td></td>
<td>O-QPSK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive Power (mW)</td>
<td>9</td>
<td>12</td>
<td>29</td>
<td></td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmit Power at 0dBm (mW)</td>
<td>36</td>
<td>36</td>
<td>42</td>
<td></td>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Consumption</td>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
<td></td>
<td>1.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Active Power (mW)</td>
<td>24</td>
<td>27</td>
<td>44</td>
<td></td>
<td>89</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming and Sensor Interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expansion</td>
<td>none</td>
<td>51-pin</td>
<td>51-pin</td>
<td>none</td>
<td>51-pin</td>
<td>19-pin</td>
<td>51-pin</td>
<td>10-pin</td>
</tr>
<tr>
<td>Communication</td>
<td>IEEE 1284 (programming) and RS232 (requires additional hardware)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrated Sensors</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Power State Transitions

- System view when switching from sleep to active

Leaving/entering sleep states costs time & energy

Overview

- **Concurrency Control:**
  - Concurrency of I/O operations alone, not of threads in general
  - Synchronous vs. Asynchronous I/O

- **Energy Management:**
  - Power state of devices needed to perform I/O operations
  - Determined by pending I/O requests using Asynchronous I/O

![Diagram]

- Application
  - read()
  - read()
  - write()
  - read()

- **OS Flash Driver**
  - read()
  - write()
  - setPowerState()

- **Physical Flash**
  - read()
Overview

- **Concurrency Control:**
  - Concurrency of I/O operations alone, not of threads in general
  - Synchronous vs. Asynchronous I/O

- **Energy Management:**
  - Power state of devices needed to perform I/O operations
  - Determined by pending I/O requests using Asynchronous I/O

*The more workload information an application can give the OS, the more energy it can save.*
Motivation

- Difficult to manage energy in traditional OS
  - Hard to tell OS about future application workloads
  - API extensions for hints?
Existing OS Approaches

- **Dynamic CPU Voltage Scaling**
  - Vertigo - Application workload classes
  - Grace OS - Explicit real-time deadlines

- **Disk Spin Down**
  - Coop-IO - Application specified timeouts
Existing OS Approaches

- **Dynamic CPU Voltage Scaling**
  - Vertigo: Application workload classes
  - Grace OS: Explicit real-time deadlines

- **Disk Spin Down**
  - Coop-IO: Application specified timeouts

_Saving energy is a complex process_
Existing OS Approaches

- Dynamic CPU Voltage Scaling
  - Vertigo - Application workload classes
  - Grace OS - Explicit real-time deadlines

- Disk Spin Down
  - Coop-IO - Application specified timeouts

Saving energy is a complex process
A little application knowledge can help us a lot
Sensor Networks

- Domain in need of unique solution to this problem
  - Harsh energy requirements
  - Very small source of power (2 AA batteries)
  - Must run unattended from months to years
Sensor Networks

- Domain in need of unique solution to this problem
  - Harsh energy requirements
  - Very small source of power (2 AA batteries)
  - Must run unattended from months to years

- First generation IoT OSes (TinyOS, Contiki...)
  - Push all energy management to the application
  - Optimal energy savings at cost of application complexity
ICEM: Integrated Concurrency and Energy Management

- Device driver architecture that automatically manages energy
- Introduces **Power Locks**, split-phase locks with integrated energy and configuration management
- Defines three classes of drivers: dedicated, shared, virtualized
- Provides a component library for building drivers
- Implemented in TinyOS 2.0 -- all drivers follow it
Advantages of ICEM

- Energy efficient – 98.4% as hand-tuned implementation
- Reduces code complexity – 400 vs. 68 lines of code
- Enables natural decomposition of applications
The TelosB Platform

- Six major I/O devices
- Possible Concurrency
  - I²C, SPI, ADC
- Energy Management
  - Turn peripherals on only when needed
  - Turn off otherwise
Representative Logging Application

Producer

Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

Consumer

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

Sensors

Radio

Flash
Representative Logging Application

### Producer

Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

### Consumer

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

---

**Flash**

**Radio**

**Sensors**
Representative Logging Application

**Producer**

Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

**Consumer**

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

**Flash**

**Sensors**

**Radio**
Every 12 hours:
For all new entries:
Send current sample
Read next sample

Every 5 minutes:
Write prior samples
Sample photo active
Sample total solar
Sample temperature
Sample humidity

Producer

Consumer

Flash

Sensors

Radio
Representative Logging Application

Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

Producer

Consumer

Sensors

Radio

Flash
Every 12 hours:
For all new entries:
Send current sample
Read next sample

Every 5 minutes:
Write prior samples
Sample photo active
Sample total solar
Sample temperature
Sample humidity

Producer
Consumer

Sensors
Radio

Flash
Every 12 hours:
For all new entries:
Send current sample
Read next sample

Every 5 minutes:
Write prior samples
Sample photo active
Sample total solar
Sample temperature
Sample humidity
Representative Logging Application

Producer
Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

Consumer
Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

Flash

Sensors

Radio
Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

Producer

Consumer

Flash

Sensors

Radio
Hand-Tuned Application

Every 5 minutes:
  Turn on SPI bus
  Turn on flash chip
  Turn on voltage reference
  Turn on I\textsuperscript{2}C bus
  Log prior readings
  Start humidity sample
  Wait 5ms for log
  Turn off flash chip
  Turn off SPI bus
  Wait 12ms for vref
  Turn on ADC
  Start total solar sample
  Wait 2ms for total solar
  Start photo active sample
  Wait 2ms for photo active
  Turn off ADC
  Turn off voltage reference
  Wait 34ms for humidity
  Start temperature sample
  Wait 220ms for temperature
  Turn off I\textsuperscript{2}C bus
Hand-Tuned Application

Every 5 minutes:
- Turn on SPI bus
- Turn on flash chip
- Turn on voltage reference
- Turn on I²C bus
- Log prior readings
- Start humidity sample
- Wait 5ms for log
- Turn off flash chip
- Turn off SPI bus
- Wait 12ms for vref
- Turn on ADC
- Start total solar sample
- Wait 2ms for total solar
- Start photo active sample
- Wait 2ms for photo active
- Turn off ADC
- Turn off voltage reference
- Wait 34ms for humidity
- Start temperature sample
- Wait 220ms for temperature
- Turn off I²C bus
Every 5 minutes:
- Turn on SPI bus
- Turn on flash chip
- Turn on voltage reference
- Turn on I²C bus
- Log prior readings
- Start humidity sample
- Wait 5ms for log
- Turn off flash chip
- Turn off SPI bus
- Wait 12ms for vref
- Turn on ADC
- Start total solar sample
- Wait 2ms for total solar
- Start photo active sample
- Wait 2ms for photo active
- Turn off ADC
- Turn off voltage reference
- Wait 34ms for humidity
- Start temperature sample
- Wait 220ms for temperature
- Turn off I²C bus
Hand-Tuned Application

Every 5 minutes:
- Turn on SPI bus
- Turn on flash chip
- Turn on voltage reference
- Turn on I²C bus
- Log prior readings
- Start humidity sample
- Wait 5ms for log
- Turn off flash chip
- Turn off SPI bus
- Wait 12ms for vref
- Turn on ADC
- Start total solar sample
- Wait 2ms for total solar
- Start photo active sample
- Wait 2ms for photo active
- Turn off ADC
- Turn off voltage reference
- Wait 34ms for humidity
- Start temperature sample
- Wait 220ms for temperature
- Turn off I²C bus

ICEM Application

Every 5 minutes:
- Log prior readings
- Sample humidity
- Sample total solar
- Sample photo active
- Sample temperature
Split-Phase I/O Operations

- Implemented within a single thread of control
- Application notified of I/O completion through direct upcall
- Driver given workload information before returning control
- Example: `read() → readDone()`

```c
void readDone(uint16_t val) {
    next_val = val;
    read();
}
```
ICEM Architecture

- Defines three classes of drivers
  - Virtualized – provide only functional interface
  - Dedicated – provide functional and power interface
  - Shared – provide functional and lock interface
Dedicated Device Drivers

- Provide **Functional** and **Power Control** interfaces
- Assume a single user
- **No** concurrency control
- *Implicit* energy management
- Low-level hardware and bottom-level abstractions have a dedicated driver
Virtualized Device Drivers

- Provide only a **Functional** interface
- Assume multiple users
- **Implicit** concurrency control through buffering requests
- **Implicit** energy management based on pending requests
- Implemented for higher-level services that can tolerate long latencies
Shared Device Drivers

- Provide **Functional** and **Lock** interfaces
- Assume multiple users
- *Explicit* concurrency control through Lock request
- *Implicit* energy management based on pending requests
- Used by users with stringent timing requirements
ICEM Architecture

- Defines three classes of drivers
  - Virtualized – provide only functional interface
  - Dedicated – provide functional and power interface
  - Shared – provide functional and lock interface

- Power Locks, split-phase locks with integrated energy and configuration management
Power Locks

Lock

HW-Specific Configuration

Power Control

Functional

Dedicated Driver
Power Locks

- Lock
- HW-Specific Configuration
- Power Control
- Functional

Dedicated Driver
Power Locks

- HW-Specific Configuration
- Power Control
- Dedicated Driver
- Functional

Lock
Power Locks

HW-Specific Configuration

Power Control

Functional

Dedicated Driver
Power Locks

- Lock

- Power Locks
  - HW-Specific Configuration
  - Power Control

- Functional

- Dedicated Driver
Power Locks

- Dedicated Driver
- Functional
- HW-Specific Configuration
- Power Control

Lock
Power Locks

Lock

Power Locks

HW-Specific Configuration

Power Control

Functional

Dedicated Driver
Power Locks

- Lock
- Power Locks
- HW-Specific Configuration
- Power Control
- Functional
- Dedicated Driver
Power Locks
Power Locks

HW-Specific Configuration

Power Control

1
2
3

Lock

Dedicated Driver

Functional
Power Locks

Dedicated Driver

- Functional
- Power Control
- HW-Specific Configuration
- Lock
Power Locks

- HW-Specific Configuration
- Power Control
- Functional

Dedicated Driver
ICEM Architecture

• Defines three classes of drivers
  - Virtualized – provide only functional interface
  - Dedicated – provide functional and power interface
  - Shared – provide functional and lock interface

• Power Locks, split-phase locks with integrated energy and configuration management

• Component library
  - Arbiters – manage I/O concurrency
  - Configurators – setup device specific configurations
  - Power Managers – provide automatic power management
Component Library

Power Locks

- HW-Specific Configuration
- Power Control
- **Lock** interface for concurrency control (FCFS, Round-Robin)
- **ArbiterConfigure** interface automatic hardware configuration
- **DefaultOwner** interface for automatic power management
- **Lock** interface for concurrency control (FCFS, Round-Robin)
- **ArbiterConfigure** interface for automatic hardware configuration
- **DefaultOwner** interface for automatic power management
- Implement **ArbiterConfigure** interface
- Call hardware specific configuration from dedicated driver
- Implement **DefaultOwner** interface
- Power down device when device falls idle
- Power up device when new lock request comes in
- Currently provide *Immediate* and *Deferred* policies
Msp430 USART (Serial Controller)

- Three modes of operation – SPI, I^2C, UART
Shared Driver Example

- **Msp430 USART (Serial Controller)**
  - Three modes of operation – SPI, I\(^2\)C, UART
 MSP430 USART (Serial Controller)

- Three modes of operation – SPI, I\(^2\)C, UART

DIAGRAM:
- SPI User
- Lock
- Arbiter
- Immediate Power Manager
- Power Control
- Msp430 USART
- SPI Configurator
Shared Driver Example

 MSP430 USART (Serial Controller)

- Three modes of operation – SPI, I²C, UART
Flash Storage
Virtualized Driver Example

Flash Storage

Diagram:
- Log User
- Log Virtualizer
- Flash Driver
- SPI User
- Arbiter
- Immediate Power Manager

Connections:
- Functional
- Power Control
- Lock
Virtualized Driver Example

Flash Storage

- Log User
- Log Virtualizer
- Flash Driver
- SPI User
- Arbiter
- Immediate Power Manager
- Power Control
- Lock
- Functional

CPSL Cyber-Physical Systems Laboratory
Virtualized Driver Example

Flash Storage

Diagram with nodes labeled as follows:
- Log User
- Log Virtualizer
- Flash Driver
- SPI User
- Arbiter
- Immediate Power Manager

Connections and labels:
- Functional
- Lock
- Power Control
Virtualized Driver Example

- Flash Storage

Diagram:
- Log User
- Log Virtualizer
- Flash Driver
- SPI User
- Arbiter
- Immediate Power Manager
- Functional
- Lock
- Power Control
- Lock
Virtualized Driver Example

Flash Storage

- Log User
- Log Virtualizer
- Flash Driver
- SPI User
- Arbiter
- Immediate Power Manager

Connections:
- Functional
- Lock
- Power Control
Virtualized Driver Example

Flash Storage

- Block User
  - Block Virtualizer
- Log User
  - Log Virtualizer
- Flash Driver
- SPI User
- Arbiter
- Immediate Power Manager
Virtualized Driver Example

- Flash Storage

Diagram:

- Block User
- Log User
- Block Virtualizer
- Log Virtualizer
- Flash Driver
- Arbiter
- SPI User
- Immediate Power Manager
- Power Control
- Lock
Applications

- **Hand Tuned** – Most energy efficient
- **ICEM** – All concurrent operations
- **Serial +** – Optimal serial ordering
- **Serial -** – Worst case serial ordering

**Producer**

Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

**Consumer**

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

**Sensors**

**Flash**

**Radio**
Average energy consumption for application operations
Application energy consumption with 5-minute sampling interval and one send batch every 12 hours.
Application energy with 5 minute sampling interval and one send batch every 12 hours
Application energy consumption with 5 minute sampling interval and one send batch every 12 hours.
Overhead of ICEM to Hand-Tuned Implementation

= ADC Timeout + Power Lock Overheads

With 288 samples per day

≈ 2.9 mAs/day
≈ 1049 mAs/year

Insignificant compared to total

5.60% of total sampling energy

0.03% of total application energy
Expected Node Lifetimes

![Graph showing the relationship between sampling interval and lifetime. The x-axis represents sampling interval in minutes, ranging from 0.01 to 10. The y-axis represents lifetime in years, ranging from 0 to 3. Points labeled 1 sec, 10 sec, 50 sec, and 5 min indicate specific points on the graph.](image)
Expected Node Lifetimes

![Graph showing expected node lifetimes over different sampling intervals. The x-axis represents the sampling interval (in minutes) ranging from 0.01 to 10. The y-axis represents the lifetime as a percentage of hand-tuned performance, ranging from 70% to 100%. Arrows indicate the sampling intervals: 1 sec, 10 sec, 50 sec, and 5 min.]
Expected Node Lifetimes

The graph shows the expected node lifetimes as a function of the sampling interval. The y-axis represents the lifetime in percent (% of Hand-Tuned), and the x-axis represents the sampling interval in minutes. The graph includes three sets of data points:

- **Icem**: Represented by red diamonds.
- **Serial+**: Represented by blue triangles.
- **Serial-**: Represented by green triangles.

Key observations:
- At a sampling interval of 1 minute, the lifetime is approximately 70%.
- At a sampling interval of 5 minutes, the lifetime is approximately 100%.
- The lifetime increases with an increase in the sampling interval.

The graph indicates that for a shorter sampling interval, the lifetime is lower, and for a longer sampling interval, the lifetime is higher, approaching 100% for longer intervals.
Evaluation Conclusions

Conclusions about the OS
- Small RAM/ROM overhead
- Small computational overhead
- Efficiently manages energy when given enough information

Conclusions for the developer
- Build drivers with short power down timeouts
- Submit I/O requests in parallel
Integrated Concurrency and Energy Management

- Device driver architecture for low power devices
- At least 98.4% as energy efficient as hand-tuned implementation of representative application
- Simplifies application and driver development
- Questions the assumption that applications must be responsible for all energy management and cannot have a standardized OS with a simple API