# Homework index

<table>
<thead>
<tr>
<th></th>
<th>Assignment</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reading assignment</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>Lab six</td>
<td>32</td>
</tr>
</tbody>
</table>
Goals for lecture

• Lab four

• Example scheduling algorithm design problem
  – Will initially focus on static scheduling

• Real-time operating systems

• Comparison of on-line and off-line scheduling code
Lab four

- Talk with Promi SD101
- Sample sound at 3 kHz
- Multihop
Example problem: Static scheduling

• What is an FPGA?

• Why should real-time systems designers care about them?

• Multiprocessor static scheduling

• No preemption

• No overhead for subsequent execution of tasks of same type

• High cost to change task type

• Scheduling algorithm?
Problem: Uniprocessor independent task scheduling

• Problem
  – Independent tasks
  – Each has a period = hard deadline
  – Zero-cost preemption

• How to solve?
Rate monotonic scheduling

Main idea

• 1973, Liu and Layland derived optimal scheduling algorithm(s) for this problem

• Schedule the job with the smallest period (period = deadline) first

• Analyzed worst-case behavior on any task set of size \( n \)

• Found utilization bound: \( U(n) = n \cdot (2^{1/n} - 1) \)

• 0.828 at \( n = 2 \)

• As \( n \to \infty \), \( U(n) \to \log 2 = 0.693 \)

• Result: For any problem instance, if a valid schedule is possible, the processor need never spend more than 71% of its time idle
Optimality and utilization for limited case

- Simply periodic: All task periods are integer multiples of all lesser task periods
- In this case, RMS/DMS optimal with utilization 1
- However, this case rare in practice
- Remains feasible, with decreased utilization bound, for in-phase tasks with arbitrary periods
Rate monotonic scheduling

• Constrained problem definition
• Over-allocation often results
• However, in practice utilization of 85%–90% common
  – Lose guarantee
• If phases known, can prove by generating instance
Critical instants

Main idea:

A job’s critical instant a time at which all possible concurrent higher-priority jobs are also simultaneously released

Useful because it implies latest finish time
Proof sketch for RMS utilization bound

• Consider case in which no period exceeds twice the shortest period

• Find a pathological case
  – Utilization of 1 for some duration
  – Any decrease in period/deadline of longest-period task will cause deadline violations
  – Any increase in execution time will cause deadline violations
RMS worst-case utilization

- In-phase

- \( \forall k \text{ s.t. } 1 \leq k \leq n-1 : e_k = p_{k+1} - p_k \)

- \( e_n = p_n - 2 \cdot \sum_{k=1}^{n-1} e_k \)
Proof sketch for RMS utilization bound

- See if there is a way to increase utilization while meeting all deadlines
- Increase execution time of high-priority task
  \[ e'_i = p_{i+1} - p_i + \epsilon = e_i + \epsilon \]
- Must compensate by decreasing another execution time
- This always results in decreased utilization
  \[ e'_k = e_k - \epsilon \]
  \[ U' - U = \frac{e'_i}{p_i} + \frac{e'_k}{p_k} - \frac{e_i}{p_i} - \frac{e_k}{p_k} = \frac{\epsilon}{p_i} - \frac{\epsilon}{p_k} \]
  \[ \text{Note that } p_i < p_k \rightarrow U' > U \]
Proof sketch for RMS utilization bound

• Same true if execution time of high-priority task reduced

• \( e''_i = p_{i+1} - p_i - \varepsilon \)

• In this case, must increase other \( e \) or leave idle for \( 2 \cdot \varepsilon \)

• \( e''_k = e_k + 2\varepsilon \)

• \( U'' - U = \frac{2\varepsilon}{p_k} - \frac{\varepsilon}{p_i} \)

• Again, \( p_k < 2 \rightarrow U'' > U \)

• Sum over execution time/period ratios
Proof sketch for RMS utilization bound

- Get utilization as a function of adjacent task ratios
- Substitute execution times into $\sum_{k=1}^{n} \frac{e_k}{p_k}$
- Find minimum
- Extend to cases in which $p_n > 2 \cdot p_k$
Notes on RMS

- Other abbreviations exist (RMA)
- DMS better than or equal RMA when deadline $\neq$ period
- Why not use slack-based?
- What happens if resources are under-allocated and a deadline is missed?
Essential features of RTOSs

• Provides real-time scheduling algorithms or primatives

• Bounded execution time for OS services
  – Usually implies preemptive kernel
  – E.g., linux can spend milliseconds handling interrupts, especially disk access
Threads

- Threads vs. processes: Shared vs. unshared resources
- OS impact: Windows vs. Linux
- Hardware impact: MMU
Threads vs. processes

- Threads: Low context switch overhead
- Threads: Sometimes the only real option, depending on hardware
- Processes: Safer, when hardware provides support
- Processes: Can have better performance when IPC limited
Software implementation of schedulers

- TinyOS
- Light-weight threading executive
- μC/OS-II
- Linux
- Static list scheduler
TinyOS

- Most behavior event-driven
- High rate $\rightarrow$ Livelock
- Research schedulers exist
BD threads

- Brian Dean: Microcontroller hacker
- Simple priority-based thread scheduling executive
- Tiny footprint (fine for AVR)
- Low overhead
- No MMU requirements
µC/OS-II

• Similar to BD threads
• More flexible
• Bigger footprint
Old linux scheduler

- Single run queue
- $O(n)$ scheduling operation
- Allows dynamic goodness function
\( O(1) \) scheduler in Linux 2.6

- Written by Ingo Molnar
- Splits run queue into two queues prioritized by goodness
- Requires static goodness function
  - No reliance on running process
- Compatible with preemptible kernel
Real-time linux

• Run linux as process under real-time executive

• Complicated programming model

• RTAI (Real-Time Application Interface) attempts to simplify
  – Colleagues still have problems at $>18$ kHz control period
Real-time operating systems

- Embedded vs. real-time
- Dynamic memory allocation
- Schedulers: General-purpose vs. real-time
- Timers and clocks: Relationship with HW
Summary

• Static scheduling
• Example of utilization bound proof
• Introduction to real-time operating systems
Reading assignment


Goals for lecture

- Lab four?
- Lab six
- Simulation of real-time operating systems
- Impact of modern architectural features
Lab four

- Please email or hand in the write-up for lab assignment four
- Problems? See me.
  - Will need everything from lab four working for lab six
Lab six

• Develop priority-based cooperative scheduler for TinyOS that keeps track of the percentage of idle time.

• Develop a tree routing algorithm for the sensor network.

• Send noise, light, and temperature data to a PPC, via the network root.

• Have motes respond to send audio samples and buzz commands.

• Play back or display this data on PPCs to verify that the system functions.
Outline

- Introduction
- Role of real-time OS in embedded system
- Related work and contributions
- Examples of energy optimization
- Simulation infrastructure
- Results
- Conclusions
Introduction

- Real-Time Operating Systems are often used in embedded systems.
- They simplify use of hardware, ease management of multiple tasks, and adhere to real-time constraints.
- Power is important in many embedded systems with RTOSs.
- RTOSs can consume significant amount of power.
- They are re-used in many embedded systems.
- They impact power consumed by application software.
- RTOS power effects influence system-level design.
Introduction

• Real Time Operating Systems important part of embedded systems
  – Abstraction of HW
  – Resource management
  – Meet real-time constraints

• Used in several low-power embedded systems

• Need for RTOS power analysis
  – Significant power consumption
  – Impacts application software power
  – Re-used across several applications
Role of RTOS in embedded system

Applications
- MPEG encoding
- ABS
- Communication
- etc.

RTOS services
- IPC
- Memory manager
- Basic IO
- Timer
- Task manager
- ISR

Tasks
- Micro-browser
- Organizer
- Message composer
- Database

Hardware
- Processor
- Memory
- Timer
- Other hardware
- Network interface

Applications
- Communication
- etc.
Related work and contributions

• **Instruction level power analysis**

• **System-level power simulation**
  Y. Li and J. Henkel, Design Automation Conf., 1998

• **MicroC/OS-II**: J.J. Labrosse, R & D Books, Lawrence, KS, 1998

• **Our work**
  – First step towards detailed power analysis of RTOS
  – Applications: low-power RTOS, energy-efficient software architecture, incorporate RTOS effects in system design
Simulated embedded system

- Easy to add new devices
- Cycle-accurate model
- Fujitsu board support library used in model
- μC/OS-II RTOS used
Single task network interface

Get packet → Compute checksum → Procure Ethernet controller → Transfer packet → Release Ethernet controller

Checksum computation and output

Procuring Ethernet controller has high energy cost
TCP example

Checksum computation and output

Get packet → Compute checksum → Procure Ethernet controller → Transfer packet → Release Ethernet controller

Bufecksum computation

Buffer management

Output

Procure Ethernet controller → Transfer packets → Release Ethernet controller

Get packet → Compute checksum

Multi-task implementation

Straight-forward implementation
RTOS power analysis used for process re-organization to reduce energy
21% reduction in energy consumption. Similar power consumption.
ABS example

Timer transition?

Y

Sense speed and pedal conditions

N

Compute acceleration

Y

Brake decision

N

Actuate brake

Sleep
ABS example timing

Timer

Brake pedal

ABS process

Wheel sensor

Brake action

Time
Straight-forward ABS implementation

- Timer transition?
- Sense speed and pedal conditions
- Compute acceleration
- Brake decision
- Actuate brake

Graph showing:
- Timer
- Brake pedal
- ABS process
- Wheel sensor
- Brake action

Time
Periodically triggered ABS

Timer transition?

Y  Sense speed and pedal conditions

N  Compute acceleration

Brake decision

Actuate brake

Sleep
Periodically triggered ABS timing
Selectively triggered ABS

Pedal pressed?

Sense speed and pedal conditions

Compute acceleration

Brake decision

Actuate brake

Sleep

Timer transition?

Y

N

Y

N
Selectively triggered ABS timing

63% reduction in energy and power consumption
Power-optimized ABS example

- Pedal pressed?
  - Y: Sense speed and pedal conditions
  - N: Sleep

- Sleep
  - N: Timer transition?
    - Y: Actuate brake
    - N: Sleep

- Compute acceleration
- Brake decision

- ABS process

- Brake pedal
- Wheel sensor
- Brake action

Time
Infrastructure

Application code

OS code

External stimulus

SPARClite compiler

SPARClite cache simulator

SPARClite ISS

Instruction-level energy model

Memory model

Memory energy model

Cache controller model

Bus interface unit model

Timer model

UART model

Models for other peripherals

Energy by call tree position for task A

OSSched()

main()

OSSem()

Energy by call tree position for task B
Experimental results
Experimental results – time
Agent example

Key

- - - - - - Broadcast
- - - - Price advertisement
- - - Sale

Agent 1

Agent 2

Agent 3

Agent 4

Agent 5

Agent 6

Money
Commodity 1
Commodity 2
Commodity 3
Commodity 4
Experimental results

Energy (mJ)

Time (ms)

10500
10000
9500
9000
8500
8000
7500
7000
6500
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0

0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
5500
6000
6500
7000
7500
8000
8500
9000
9500
10000

(a)

(b)
Experimental results

(a) Sleep Synchronization Task control
(b) Semaphore

Agent

Energy (mJ)

Ethernet

- Application
- Floating-point
- Initialization
- Input/output
- Interrupt
- Mailbox
- Memory
- Misc.
- Scheduling
- Semaphore
- Synchronization
- Sleep
- Task control
Optimization effects

TCP example:

• 20.5% energy reduction
• 0.2% power reduction
• RTOS directly accounted for 1% of system energy

ABS example:

• 63% energy reduction
• 63% power reduction
• RTOS directly accounted for 50% of system energy

Mailbox example: RTOS directly accounted for 99% of system energy

Semaphore example: RTOS directly accounted for 98.7% of system energy
## Partial semaphore hierarchical results

<table>
<thead>
<tr>
<th>Function</th>
<th>Energy/invocation (µJ)</th>
<th>Energy (%)</th>
<th>Time (mS)</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>realstart</td>
<td>0.41</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>init_vecs</td>
<td>1.31</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>init_timer</td>
<td>0.41</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>lifefed</td>
<td>887.44</td>
<td>0.28</td>
<td>2.18</td>
<td>1</td>
</tr>
<tr>
<td>do_main</td>
<td>1.56</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>save_data</td>
<td>1.31</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>init_data</td>
<td>0.88</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>init_bss</td>
<td>2.72</td>
<td>0.00</td>
<td>0.01</td>
<td>1</td>
</tr>
<tr>
<td>Task1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>win_unf_trap</td>
<td>1.90</td>
<td>1.20</td>
<td>9.73</td>
<td>1999</td>
</tr>
<tr>
<td>_OSDisableInt</td>
<td>0.29</td>
<td>0.09</td>
<td>0.78</td>
<td>1000</td>
</tr>
<tr>
<td>_OSEnableInt</td>
<td>0.32</td>
<td>0.10</td>
<td>0.89</td>
<td>1000</td>
</tr>
<tr>
<td>sparc_sim_terminate</td>
<td>0.75</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>OSSemPend</td>
<td>2.48</td>
<td>0.78</td>
<td>6.33</td>
<td>999</td>
</tr>
<tr>
<td>OSSemPost</td>
<td>0.29</td>
<td>0.18</td>
<td>1.59</td>
<td>1999</td>
</tr>
<tr>
<td>OSSemPost</td>
<td>0.29</td>
<td>0.18</td>
<td>1.59</td>
<td>1999</td>
</tr>
<tr>
<td>OSSched</td>
<td>3.76</td>
<td>1.18</td>
<td>9.22</td>
<td>999</td>
</tr>
<tr>
<td>OSSched</td>
<td>19.07</td>
<td>6.00</td>
<td>47.97</td>
<td>999</td>
</tr>
<tr>
<td>OSSemPost</td>
<td>0.29</td>
<td>0.09</td>
<td>0.78</td>
<td>1000</td>
</tr>
<tr>
<td>OSSemPost</td>
<td>0.29</td>
<td>0.09</td>
<td>0.78</td>
<td>1000</td>
</tr>
<tr>
<td>OSSemPost</td>
<td>0.27</td>
<td>0.08</td>
<td>0.70</td>
<td>1000</td>
</tr>
<tr>
<td>OSSemPost</td>
<td>0.29</td>
<td>0.09</td>
<td>0.78</td>
<td>1000</td>
</tr>
<tr>
<td>OSTimeGet</td>
<td>108.89</td>
<td>34.30</td>
<td>258.53</td>
<td>1000</td>
</tr>
<tr>
<td>printf</td>
<td>2.05</td>
<td>0.65</td>
<td>5.06</td>
<td>1000</td>
</tr>
<tr>
<td>printf</td>
<td>1.09</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>exceptionHandler</td>
<td>4.77</td>
<td>0.02</td>
<td>0.17</td>
<td>15</td>
</tr>
<tr>
<td>CPUInit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.09 mJ total</td>
<td>112.90</td>
<td>35.56</td>
<td>112.90</td>
<td>1000</td>
</tr>
</tbody>
</table>

57
## Energy per invocation for \( \mu \text{C/OS-II} \) services

<table>
<thead>
<tr>
<th>Service</th>
<th>Minimum energy (( \mu \text{J} ))</th>
<th>Maximum energy (( \mu \text{J} ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSEventTaskRdy</td>
<td>18.02</td>
<td>20.03</td>
</tr>
<tr>
<td>OSEventTaskWait</td>
<td>7.98</td>
<td>9.05</td>
</tr>
<tr>
<td>OSEventWaitListInit</td>
<td>20.43</td>
<td>21.16</td>
</tr>
<tr>
<td>OSInit</td>
<td>1727.70</td>
<td>1823.26</td>
</tr>
<tr>
<td>OSMboxCreate</td>
<td>27.51</td>
<td>28.82</td>
</tr>
<tr>
<td>OSMboxPend</td>
<td>7.07</td>
<td>82.91</td>
</tr>
<tr>
<td>OSMboxPost</td>
<td>5.82</td>
<td>84.55</td>
</tr>
<tr>
<td>OSMemCreate</td>
<td>19.40</td>
<td>19.75</td>
</tr>
<tr>
<td>OSMemGet</td>
<td>6.64</td>
<td>8.22</td>
</tr>
<tr>
<td>OSMemInit</td>
<td>27.41</td>
<td>27.47</td>
</tr>
<tr>
<td>OSMemPut</td>
<td>6.38</td>
<td>7.91</td>
</tr>
<tr>
<td>OSQInit</td>
<td>20.10</td>
<td>20.93</td>
</tr>
<tr>
<td>OSSched</td>
<td>6.96</td>
<td>52.34</td>
</tr>
<tr>
<td>OSSemCreate</td>
<td>27.87</td>
<td>29.04</td>
</tr>
<tr>
<td>OSSemPend</td>
<td>6.54</td>
<td>73.64</td>
</tr>
<tr>
<td>etc.</td>
<td>etc.</td>
<td>etc.</td>
</tr>
</tbody>
</table>
Conclusions

• RTOS can significantly impact power

• RTOS power analysis can improve application software design

• Applications
  – Low-power RTOS design
  – Energy-efficient software architecture
  – Consider RTOS effects during system design
Impact of modern architectural features

- Memory hierarchy
- Bus protocols ISA vs. PCI
- Pipelining
- Superscalar execution
- SIMD
- VLIW
Summary

• Labs

• Simulation of real-time operating systems

• Impact of modern architectural features