# Introduction to Real-Time Systems

## ECE 397-1

Northwestern University

Department of Computer Science

Department of Electrical and Computer Engineering

| | | |
|---|---|---|
| Teachers: | Robert Dick | Peter Dinda |
| Office: | L477 Tech | 338, 1890 Maple Ave. |
| Email: | dickrp@ece.northwestern.edu | pdinda@cs.northwestern.edu |
| Phone: | 467–2298 | 467-7859 |
| Webpage: | http://ziyang.ece.northwestern.edu/EXTERNAL/realtime | |




# Homework index

1 Reading assignment . . . . . 29
2 Lab six . . . . . 32



# 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
  - $e_i' = p_{i+1} - p_i + \varepsilon = e_i + \varepsilon$
- Must compensate by decreasing another execution time
- This always results in decreased utilization
  - $e_k' = e_k - \varepsilon$
  - $U' - U = \frac{e_i'}{p_i} + \frac{e_k'}{p_k} - \frac{e_i}{p_i} - \frac{e_k}{p_k} = \frac{\varepsilon}{p_i} - \frac{\varepsilon}{p_k}$
  - 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
- $\mu$C/OS-II
- Linux
- Static list scheduler



## TinyOS

- Most behavior event-driven
- High rate $\longrightarrow$ 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



## $\mu$C/OS-II

- Similar to BD threads
- More flexible
- Bigger footprint



## Old linux scheduler

- Single run queue
- $\mathcal{O}(n)$ scheduling operation
- Allows dynamic goodness function

## $\mathcal{O}(1)$ 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

- Read Chapter 12 in J. W. S. Liu, *Real-Time Systems*. Prentice-Hall, Englewood Cliffs, NJ, 2000
- Read K. Ghosh, B. Mukherjee, and K. Schwan, "A survey of real-time operating systems," tech. rep., College of Computing, Georgia Institute of Technology, Feb. 1994



## 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 the 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.

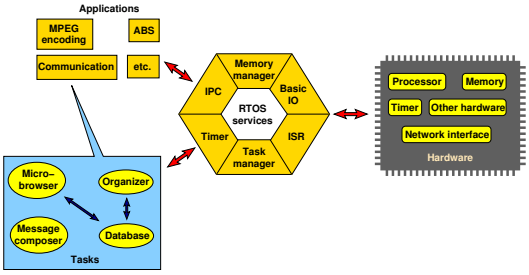


# Introduction

- 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



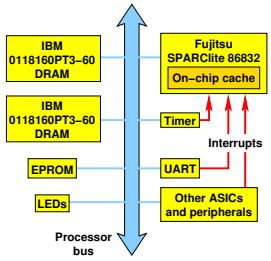



# Related work and contributions

- **Instruction level power analysis**
  V. Tiwari, S. Malik, A. Wolfe, and T.C. Lee, Int. Conf. VLSI Design, 1996
- **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



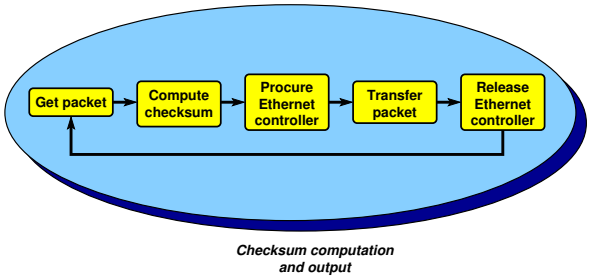
# Simulated embedded system




- Easy to add new devices
- Cycle-accurate model
- Fujitsu board support library used in model
- $\mu$C/OS-II RTOS used



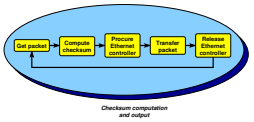
# Single task network interface



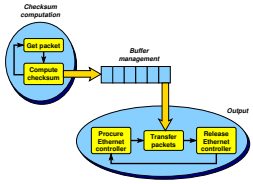

Procuring Ethernet controller has high energy cost



# TCP example

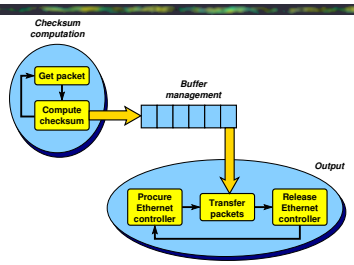



Straight-forward implementation




Multi-task implementation

# Multi-tasking network interface

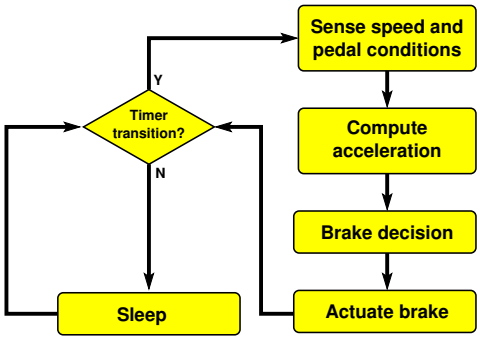



RTOS power analysis used for process re-organization to reduce energy

21% reduction in energy consumption. Similar power consumption.

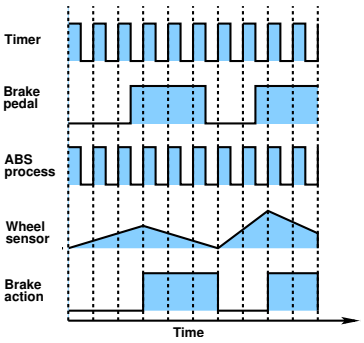


# ABS example



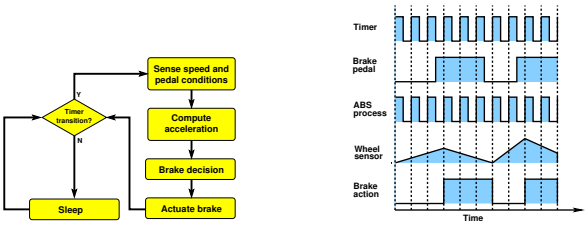



# ABS example timing



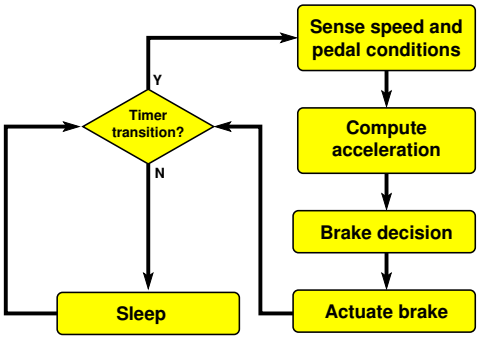



# Straight-forward ABS implementation



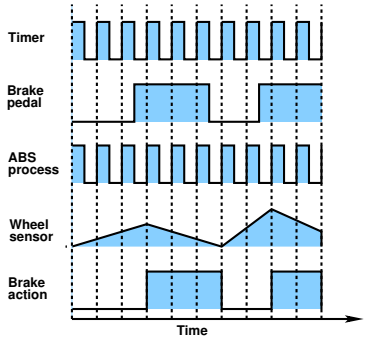



# Periodically triggered ABS



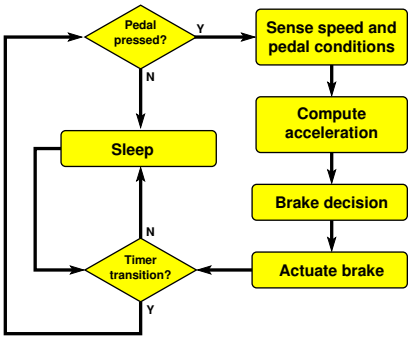



# Periodically triggered ABS timing



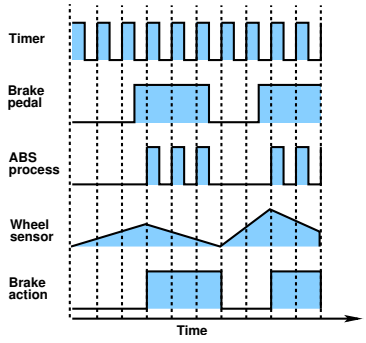



# Selectively triggered ABS



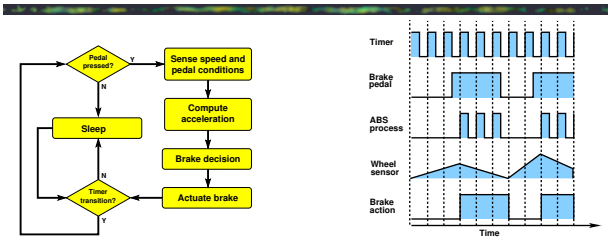



# Selectively triggered ABS timing



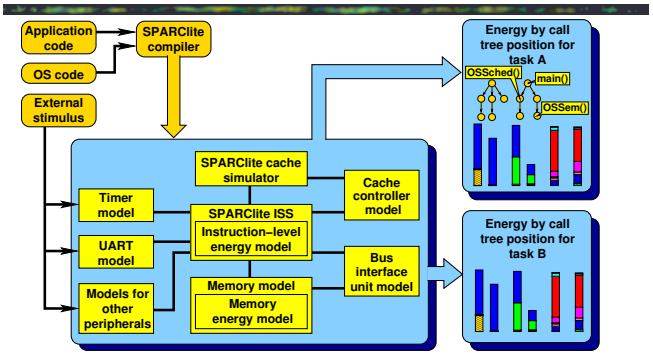

63% reduction in energy and power consumption

# Power-optimized ABS example



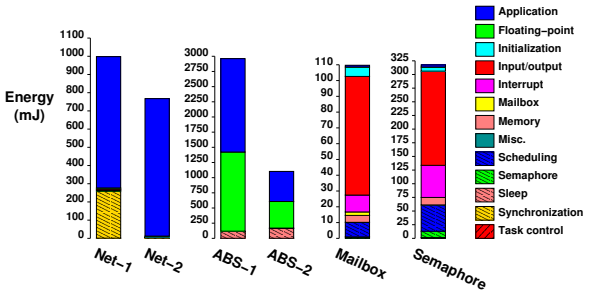



# Infrastructure



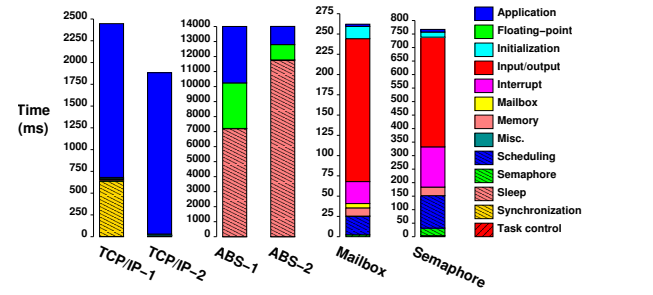



# Experimental results



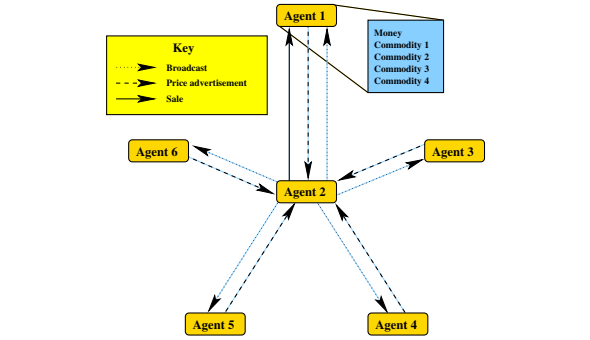



# Experimental results – time



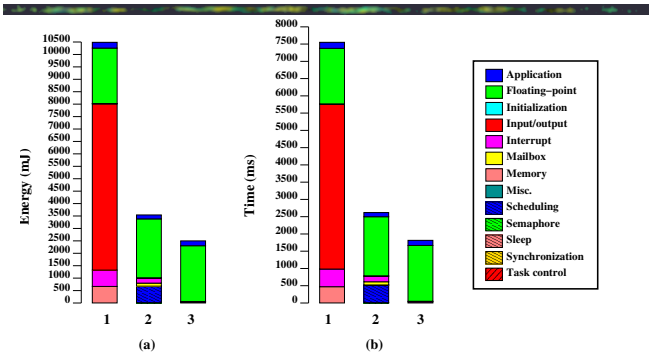



# Agent example



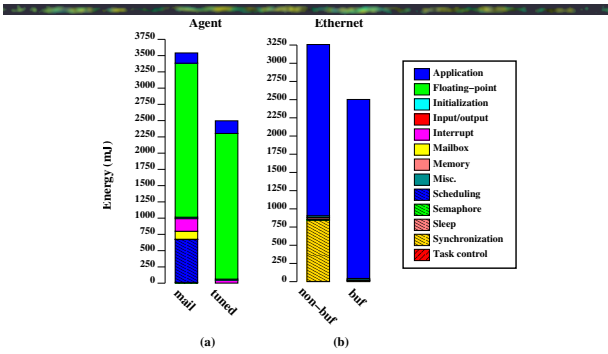



# Experimental results






# Experimental results






# 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

| | | Function | Energy/invocation (uJ) | Energy (%) | Time (mS) | Calls |
|---|---|---|---|---|---|---|
| realstart | init_tvecs | | 0.41 | 0.00 | 0.00 | 1 |
| 6.41 mJ total | init_timer | liteled | 1.31 | 0.00 | 0.00 | 1 |
| 2.02 % | 5.51 mJ total | | | | | |
| | 1.74 % | | | | | |
| | startup | do_main | 887.44 | 0.28 | 2.18 | 1 |
| | 0.90 mJ total | save_data | 1.56 | 0.00 | 0.00 | 1 |
| | 0.28 % | init_data | 1.31 | 0.00 | 0.00 | 1 |
| | | init_bss | 0.88 | 0.00 | 0.00 | 1 |
| | | cache_on | 2.72 | 0.00 | 0.01 | 1 |
| Task1 | win_unf_trap | | 1.90 | 1.20 | 9.73 | 1999 |
| 155.18 mJ total | _OSDisableInt | | 0.29 | 0.09 | 0.78 | 1000 |
| 48.88 % | _OSEnableInt | | 0.32 | 0.10 | 0.89 | 1000 |
| | sparcsim_terminate | | 0.75 | 0.00 | 0.00 | 1 |
| | OSSemPend | win_unf_trap | 2.48 | 0.78 | 6.33 | 999 |
| | 31.18 mJ total | _OSDisableInt | 0.29 | 0.18 | 1.59 | 1999 |
| | 9.82 % | _OSEnableInt | 0.29 | 0.18 | 1.59 | 1999 |
| | | OSEventTaskWait | 3.76 | 1.18 | 9.22 | 999 |
| | | OSSched | 19.07 | 6.00 | 47.97 | 999 |
| | OSSemPost | _OSDisableInt | 0.29 | 0.09 | 0.78 | 1000 |
| | 2.90 mJ total | _OSEnableInt | 0.29 | 0.09 | 0.78 | 1000 |
| | 0.91 % | | | | | |
| | OSTimeGet | _OSDisableInt | 0.27 | 0.08 | 0.70 | 1000 |
| | 1.43 mJ total | _OSEnableInt | 0.29 | 0.09 | 0.78 | 1000 |
| | 0.45 % | | | | | |
| | CPUInit | BSPInit | 1.09 | 0.00 | 0.00 | 1 |
| | 0.09 mJ total | exceptionHandler | 4.77 | 0.02 | 0.17 | 15 |
| | 0.03 % | | | | | |
| | printf | win_unf_trap | 2.05 | 0.65 | 5.06 | 1000 |
| | 112.90 mJ total | vfprintf | 108.89 | 34.30 | 258.53 | 1000 |
| | 35.56 % | | | | | |



## Energy per invocation for $\mu$C/OS-II services

| Service | Minimum energy ($\mu$J) | Maximum energy ($\mu$J) |
|---|---|---|
| `OSEventTaskRdy` | 18.02 | 20.03 |
| `OSEventTaskWait` | 7.98 | 9.05 |
| `OSEventWaitListInit` | 20.43 | 21.16 |
| `OSInit` | 1727.70 | 1823.26 |
| `OSMboxCreate` | 27.51 | 28.82 |
| `OSMboxPend` | 7.07 | 82.91 |
| `OSMboxPost` | 5.82 | 84.55 |
| `OSMemCreate` | 19.40 | 19.75 |
| `OSMemGet` | 6.64 | 8.22 |
| `OSMemInit` | 27.41 | 27.47 |
| `OSMemPut` | 6.38 | 7.91 |
| `OSQInit` | 20.10 | 20.93 |
| `OSSched` | 6.96 | 52.34 |
| `OSSemCreate` | 27.87 | 29.04 |
| `OSSemPend` | 6.54 | 73.64 |
| etc. | etc. | etc. |



## 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