TEACHING OPTIMIZATION OF TIME AND ENERGY IN EMBEDDED SYSTEMS

Alex Dean
Dept. of ECE
NC State University
Overview

- Motivation
- Course Basics
- Time Optimization
- Power and Energy Optimization
- Conclusions
Motivation

- Teach methods for optimizing deeply embedded *systems*
  - Look at entire system, not just a fragment
- Compelling examples for students
  - Make it run faster
  - Make the battery last longer

- My motivation for this talk
  - Give you ideas on how to structure the programming projects so they work
Course Administrivia

- Piggy-backed senior (20%) and grad (80%) courses
  - Most grad students didn’t do undergrad at NCSU

- Assume students have some background
  - MCU and peripheral programming in C
    - GPIO, timer, ADC, DAC, UART,
    - Interrupts

- Need to teach hands-on material practically, given 102 students and two teaching assistants
Course Contents

- MCU, ISA and development board (QSK) introduction
- Interfacing projects: analog and digital, timers, serial communications
- Run-time Performance Analysis and Optimization + project
- Energy Optimization + project
- Schedulers (task ordering, preemption)
- Real-time Systems
- Application design with an RTOS
- Lightweight Software Engineering for Embedded Systems
- Embedded code porting project (TFT LCD, FAT on microSD card)
- Final project
Development Platform

- Renesas M16C architecture MCU
  - 16-bit CISC, 24 MHz
  - 64 KB on-chip flash, 10 KB SRAM
  - Rich set of peripherals

- QSK62P+ board
  - USB for power, debugger
  - LCD, thermistor, pot, switches, LEDs, RS232, many expansion headers

- Good tool support
  - Source-level debugging
  - Much easier to use than GNU toolchain

- Cheap enough - $70 at Digikey
  - Students purchase the board, keep it
  - Benefits of ownership
Optimization Lessons

- Be a slacker - Avoid all run-time work possible
  - Pre-compute when possible
  - Reuse computation results
  - Avoid unnecessary conversions

- Do whatever remains efficiently
  - Start with efficient algorithm and problem representation, then work down into implementation details
  - Avoid double-precision math
  - Use low-complexity algorithms

- Be scientific: let the data drive your activities
  - Identify functions which dominate time
  - Evaluate impact of optimization attempts
Time Optimization

Motivation

- Students often blindly optimize what they think is slow, rather than what really is slow.
- Students often skip ahead to low-level optimizations, forgetting about the big picture.
  - Algorithm -> implementation -> tweaks (convincing the compiler to be good)
  - Our other classes focus on the detailed instruction-level optimizations.

Methods

- Profile-driven optimization
- Program rich with algorithmic optimization opportunities
Profile-Driven Optimization

- Sample program counter periodically with timer interrupt, increment corresponding function’s count in statistics table
- ISR examines return address on stack
- Students use gawk script to create region address look-up table based on map file
- Does not require instrumentation of application code (more like oprofile than gprof)
Program for Speed Optimization

- Program requirements
  - Not a kernel (e.g. FFT)
  - Not a ginormous server or desktop application (SPEC)
- Find distance and bearing to nearest of 163 weather/sea state monitoring buoys
  - Spherical surface (tunnels and submarines are not allowed)
  - distance = \( \text{acos}(\sin(\text{lat}_1) \times \sin(\text{lat}_2) + \cos(\text{lat}_1) \times \cos(\text{lat}_2) \times \cos(\Delta \text{lon})) \times R \)
  - bearing = \( \text{atan2}(\sin(\Delta \text{long}) \times \cos(\text{lat}_2), \cos(\text{lat}_1) \times \sin(\text{lat}_2) - \sin(\text{lat}_1) \times \cos(\text{lat}_2) \times \cos(\Delta \text{lon})) \)
- Locations in list of lat/lon coordinates
- Test cases: two “undisclosed locations”
Optimization Opportunities

- Graded on a curve
  - Some of score depends on performance relative to class
- Intentionally provided a program rich with opportunities for optimization - not written badly, but a reasonable, structured program designed for correctness
  - Double-precision math (through math.h)
  - Progress updates on LCD
  - Unnecessary calculations of bearing
  - Repeated conversion of degrees to radians
  - Repeated calculation of sine and cosine of fixed values
  - Repeated calculation of PI/180
  - Alphabetically sorted data (not geographically sorted)
Project Post-Mortems

- Always hold these after projects
  - Great way to get many different ways of looking at a problem (crowdsourcing)

- Informal class discussion
  - How did you get such good performance?
  - What worked best?
  - What didn’t work well?
  - How would you change what you did, if you could repeat it?
  - How would you change the project assignment for next time?

- Lab reports include lessons learned section
One Student’s Optimizations

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Speed-Ups over Initial Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial program</td>
<td>1</td>
</tr>
<tr>
<td>Single-precision floating-point math</td>
<td>10</td>
</tr>
<tr>
<td>No LCD update or bearing calculation</td>
<td>100</td>
</tr>
<tr>
<td>Precompute constants, reuse trig results</td>
<td>1000</td>
</tr>
<tr>
<td>Use orthogonal coordinate system, precompute fixed coordinates</td>
<td>10000</td>
</tr>
<tr>
<td>Use fixed point math</td>
<td>10000</td>
</tr>
</tbody>
</table>

Copyright 2010, A.G.Dean
Overall Speed-Up Results

Number of Student Projects

Speed-Up

0  5  10  15  20  25
6   8   12   17   23   33   47   67   95  136  192  273  388  551  782  1,111  1,578  2,240  3,181  4,518  6,415  9,110
Power and Energy Optimization

- Challenge: how to measure power or energy consumption?
  - Should be inexpensive, fast, repeatable, sensitive

- Initial idea - build a cheap power-meter/integrator
  - Use op-amp circuits to multiply current by voltage, then integrate over time, then sample with MCU ADC.
  - Need to consider integrator drift, integrator reset, range, software support

- Solution - how long can a supercap power the circuit?
  - Cheap and easy, breadboardable
  - Elegant energy calculation: \[ E = \frac{C}{2} \left( V_1^2 - V_0^2 \right) \]
  - Provide a supercap to each student, but use one reference supercap for grading purposes
Supercapacitor Circuit

- Use 1F 5 V supercap
- Current limiting required
  - QSK is USB-powered -> need to keep current under ~125 mA
  - R1 provides initial current limiting
- D1 bypasses R1 when I > 8 mA
- Exponential charging = slow
  - R3 and R6 get switched in later (under software control) to reduce charge time
- Monitor supercap voltage with resistive divider
Program for Energy Optimization

- Program requirements
  - Idle time for CPU to sleep
  - Easy testing and measurement

- Device reports temperature every second:
  - Read thermistor with ADC in burst of 10 samples
  - Average the samples
  - Convert to formatted text string ("77.3°F")
  - Send string out UART at 57600 baud (e.g. to Bluetooth)

- Base program runs 3 seconds before supercap gives out

- And as the power fails...
  - LCD stops working below 4.3 V
  - Board forced into reset below 2.7 V by brown-out detector
Energy Optimization Techniques

- All the speed optimizations discussed previously
- Processor
  - Operating modes - covered in application note
  - Clock speeds - from 32 kHz to 24 MHz
- Peripheral devices
  - Can slow or disable clock, remove supply voltage for subsystems
  - Oscillator drive modes
  - GPIOs
- Board devices
  - LEDs
  - LCD
One Student’s Optimizations

Energy Savings Factor

- Base program
- Idle CPU in wait mode
- Disconnect power LED
- Set unused ports to inputs
- Use ADC and UART interrupts
- Disable peripheral clocks after UART idle
- Disable PLL if CPU idle
- CPU wait mode during ADC and transmission
- Use UART transmission
- Use 32 kHz clock (not 24 MHz) when idle

Energy Savings Factor:
- 1.00
- 10.00
- 100.00

Copyright 2010, A.G.Dean
One Student’s Optimizations (Linear Plot)

- Base program
- Idle CPU in wait mode
- Disconnect power LED
- Set unused ports to inputs
- Use ADC and UART interrupts
- Disable peripheral clocks after UART idle
- Disable PLL if CPU idle
- CPU wait mode during ADC and transmission
- Use UART transmission
- Use 32 kHz clock (not 24 MHz) when idle

Energy Savings Factor

Copyright 2010, A.G.Dean
Overall Energy Savings

![Bar chart showing energy reduction factor and number of student projects.](chart.png)

Copyright 2010, A.G. Dean
Student Feedback

- Likes
  - Projects!
  - Whiteboard rather than powerpoint
  - Open-ended nature of projects
  - In-class programming examples

- Dislikes
  - Too many A+’s!
  - Embedded coding style should be taught
  - Too much pre-written code
  - Cover less introductory material, assume students will be able to read manual
  - In-class programming examples
Conclusions and Changes

- Very effective - students were excited by competitive aspect of optimization
- Programs which offer a range of optimizations encourage outside the box thinking

Changes for next time
- Switch from supercap to supply rail when discharged, keeping board running
- Reduce introductory projects
- Add ARM Cortex-A8 for breadth and comparison
  - $50 processor vs. $5 processor

Course materials available from me at agdean@ncsu.edu or at Renesas University program at http://renesasrulz.com