CSE 237B Fall 2009

Software for Embedded Systems

Rajesh Gupta
University of California, San Diego
Welcome to CSE 237B!

- **Instructor:**
  - Rajesh Gupta, rgupta@ucsd.edu, 858 822-4391, EBU3B 2120
  - Office Hours: Wed 2-4, by appointment
- **Admin:**
  - Virginia McIlwain, vmcilwain@ucsd.edu, 534-3047
- **Class meets:**
  - TTh 5:00PM-6:20PM EBU3B-2154
- **Teaching Assistant:**
  - Thomas Weng, thomaslw@gmail.com
- **Class Website:**
  - [http://mesl.ucsd.edu/gupta/cse237b.html](http://mesl.ucsd.edu/gupta/cse237b.html)
About This Course

Part of a four-course group

- CSE 237A: Introduction to Embedded Systems (S’10)
- CSE 237B: Software for Embedded Systems (F’09)
- CSE 237C: Validation and Prototyping for Embedded Systems
- CSE 237D: Design Automation for Embedded Systems (W’10)

Since these courses are designed to be stand-alone, there is a small amount of overlap (1.5-2 lectures) related to introductory and modeling material.

• Related courses in this group
  - ECE 264: Wireless Embedded and Networked Systems, Curt Schurgers
  - CSE 218: Software modeling for distributed, reactive systems, Ingolf Krueger

• Depth sequence: Embedded Systems and Software
Course Requirements

• No official graduate course as prerequisite.
• But, many assumptions.
• R1: knowledge
  – Digital hardware, basic electrical stuff, computer architecture (ISA, organization), programming & systems programming, algorithms and data structures
• R2: skills
  – Using simulation and analysis tools
  – Advanced ability to program & use simulation/analysis tools
  – Basic calculus
  – Ability to look up references and track down pubs (Xplore etc)
  – Strong ability to communicate your ideas (talks, reports)
• R3: initiative
  – Definitely not a spoon-fed undergrad or basic grad course
  – Open-ended problems with no single answer requiring thinking and research
• R4: interest
  – Have strong interest in research in this or related fields
Course Grading
(distribution subject to change)

• Homeworks (4 to 5): 15%
  – Analysis, simulation, programming, library/web research, paper reviews
• Machine Problems (3 to 4): 15%
  – Programming on the Android platform

• Caveat: HW and MP may not be graded individually!

• One take home exam: 10%
  – Weekend between Week 5 and Week 6 for 2-3 days
• Project: 25%
  – 10% effort/results; 7% report; 8% presentation
• Final examination: 30%
  – December 11, 2009 7:00PM
• Class participation, attendance, engagement: 5%
Class Project

- Projects are centered around a new idea, even if minor
  - Must have a ‘utilitarian goal’, than just making a point
  - Related to embedded software ‘infrastructure’
    - Pure application software are normally not encouraged
- Mostly implementation projects
  - one or more of simulation, analysis, implementation
  - literature survey not accepted
- Error on the side of picking up a risky project even if the results turn out to be disappointing or prove to be contrary to expected
  - On the other hand, lack of poor results is no excuse for not putting the best effort. You must be able to explain your results. No results is decidedly worse.
- Group projects possible: 2 is OK, 3 is a stretch.
- 15 minute presentation; demo
- Project report with strong style guidelines
  - ACM style
    http://www.acm.org/sigs/pubs/proceed/template.htm
Class Projects

• We develop project proposals as part of the class work
  – some suggested project topics on class web page
    • I encourage you to think of your own topic
  – may relate to your own research
    • you may not “reuse” work already done or being done for some other purpose
  – come and discuss possible project ideas with me!

• What should be your goal?
  – something useful: tie it to specific tool, language, hardware
  – similar style/quality as a conference paper and talk
  – key is to keep the project simple, and focused
  – aim for high quality!
Project Timelines

• Timeline
  – Submit project ideas and groups by Thursday of Week 2
    • October 1, 2009
    • Rough sketch: goals, work, expected outcome, resources
  – Discuss project ideas in class on October 6, 2009
  – Detailed proposal, timeline by Thursday, October 8, 2009
  – Project mid-course report and presentation by Week 6
    • October 29, 2009
    • Five-minute presentation in class.
• Final project presentations in the examination week or just preceding it
  • Wednesday, December 9, 2009
Example Projects

• Accurate time synchronization (FTSP implementation)
• Remote notification for distributed event control
• Distributed music generation on iPhone
• Real-time robotic control using iPhone
• Hardware/device interface modeling for embedded sensors
• Architectural modeling of embedded processors in a given language
• DSP library functions for a given Reconfigurable Processor
• OS/RTOS Services for Energy Minimization (eCOS)
• Embedded code library for baseband / media functions
Reader & Textbooks

• No textbook
• But many books, I draw my lecture material from including
  – Kopetz, Real-Time Systems, Kluwer
  – Burns & Wellings, Real-Time Systems and Programming Languages, Addison-Wesley
  – Holzmann, The SPIN Model Checker, Addison-Wesley
  – Lewis, Fundamentals of Embedded Software, PH
  – Ledin, Embedded Control Systems in C/C++, CMPBooks
• A set of papers will be required reading
  – average of one paper per class
  – will relate to the core topic of that class
  – Papers that you are expected to read it BEFORE the class will be available >2 days in advance.
FYI: Some Other Books

Embedded Systems on the Web

- Berkeley Design technology, Inc.: http://www.bdti.com
- EE Times Magazine: http://www.eet.com/
- Linux Devices: http://www.linuxdevices.com
- Embedded Linux Journal: http://embedded.linuxjournal.com
- Embedded.com: http://www.embedded.com/
  - Embedded Systems Programming magazine
- Circuit Cellar: http://www.circuitcellar.com/
- Electronic Design Magazine: http://www.planetee.com/ed/
- Integrated System Design Magazine: http://www.isdmag.com/
- Sensors Magazine: http://www.sensorsmag.com
- Collections of embedded systems resources
  - http://www.ece.utexas.edu/~bevans/courses/ee382c/resources/
  - http://www.ece.utexas.edu/~bevans/courses/realtime/resources.html
- Newsgroups
  - comp.arch.embedded, comp.cad.cadence, comp.cad.synthesis, comp.dsp, comp.realtime, comp.software-eng, comp.speech, and sci.electronics.cad
Embedded Systems Courses on the Web

• Alberto Sangiovanni-Vincentelli @ Berkeley
  – EE 249: Design of Embedded Systems: Models, Validation, and Synthesis
    • http://www-cad.eecs.berkeley.edu/~polis/class/index.html
• Brian Evans @ U.T. Austin
  – EE382C-9 Embedded Software Systems
    • http://www.ece.utexas.edu/~bevans/courses/ee382c/index.html
• Edward Lee @ Berkeley
    • http://ptolemy.eecs.berkeley.edu/~eal/ee290n/index.html
• Mani Srivastava @ UCLA
  – EE202A: Embedded and Real Time Systems
    • http://nesl.ee.ucla.edu/courses/ee202a/2008f/
Some Conferences and Journals

• Conferences & Workshops
  – ACM/IEE DAC
  – IEEE ICCAD
  – IEEE RTSS
  – ACM ISLPED
  – IEEE CODES+ISSS
  – CASES
  – Many others…

• Journals & Magazines
  – ACM Transactions on Design Automation of Electronic Systems
  – ACM Transactions on Embedded Computing Systems
  – IEEE Transactions on Computer-Aided Design
  – IEEE Transactions on VLSI Design
  – IEEE Design and Test of Computers
  – IEEE Transactions on Computers
  – Journal of Computer and Software Engineering
  – Journal on Embedded Systems
## Tentative Outline of Topics Covered

### Four parts:
- **Programming in the large**
  - Modularity, Abstraction, Reuse, Fault tolerance, Correctness, Validation
- **Programming support**
  - for time, space
- **Programming in the small**
  - Estimation and optimization
- **Beyond programming**
  - Control systems
  - Media processing

### Topics

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Modeling and Specification for Embedded Systems</td>
</tr>
<tr>
<td>2.</td>
<td>Programming Embedded RT Systems</td>
</tr>
<tr>
<td>3.</td>
<td>Fault Tolerance and Recovery</td>
</tr>
<tr>
<td>4.</td>
<td>Exceptions and Exception Handling</td>
</tr>
<tr>
<td>5.</td>
<td>Low Level Programming</td>
</tr>
<tr>
<td>6.</td>
<td>Programming Language Support for Time</td>
</tr>
<tr>
<td>7.</td>
<td>Programming Language Support for Space</td>
</tr>
<tr>
<td>8.</td>
<td>Scheduling for Real-time</td>
</tr>
<tr>
<td>9.</td>
<td>Task Management, RTOS Issues</td>
</tr>
<tr>
<td>10.</td>
<td>Optimization of Embedded Software</td>
</tr>
<tr>
<td>11.</td>
<td>Programming Control Systems in Software</td>
</tr>
</tbody>
</table>
Optional Modules

• Topics that I have included or removed from the course
  – Time Synchronization
  – Location Modeling
  – Virtualization
  – (Formal) Verification of Software
  – Co-simulation and virtual prototyping

• In addition to picking from these, I may pick from
  – Embedded Security
  – Meta-data and Meta-models
  – Hybrid Control
  – Cyber-Physical Systems Software
  – Signal Processing Software on Embedded Platforms
  – Time-Triggered Systems
Logistics: one last thing...

• Fall is a terrible time to teach:
  – Travel: conferences, project reviews.
• I know I am going to miss the following days due to travel

• My strategy
  – Record classes
    • (not an option this quarter, though may recycle from last time😊)
  – Preemptive Make-up classes
    • Wednesdays 5-6:20PM?
  – Presentations, Examinations outside of the class times.
What are Embedded Computing Systems?

What are embedded systems?
Embedded system characteristics
An Embedded Computing System

- Generally, part of a larger system that may not be a "computer"
- works in a **reactive, time- and energy-constrained** environment.
- employs a combination of hardware & software (a “computational engine”) to perform a specific function;
  - Software is used for providing features and flexibility
  - Hardware = {Processors, ASICs, Memory,...} is used for performance (& sometimes security)

<table>
<thead>
<tr>
<th>DSP Code</th>
<th>Application Specific Gates</th>
<th>Analog I/O</th>
<th>Processor Cores</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: Automotive Embedded Systems

Dominated by software and networking functions
Control, processing, networking, ...

- Embedded, reactive controllers, for
  - Central locking system
  - Crash management
  - Power windows & seats
  - Airbag controller
  - etc.

- Embedded processing for
  - Multimedia system
  - Navigation system

- Infotainment alone uses up to 20 processors

90% of all innovations are SW-driven
Automotive embedded software: Highly complex, networked, and distributed

• **Processing**
  – 70-80 electronic control units (ECUs) supporting hundreds of features
  – ECUs delivered by multiple suppliers, with their own software chains

• **Networking**
  – Separate, integrated networks for power train, chassis, security, MMI, multimedia, body/comfort functions
  – Increasing interaction beyond car’s boundaries with devices, networks

• **Software development challenges:**
  – hardware independence, information interdependence among subsystems, system composition, validation.
But that is just the beginning...

Embedded Systems come in many (more) shapes.
Intel Montecito

- Two 2-threaded 64b EPIC cores
  - Two threads hide memory latency on a single thread (TMT)
- Total of 26.5 MB caches
- 1.72 Billion transistors, 596 mm², 90nm
Generated clock frequency is a function of the measured voltage -- 3X gain in power consumption

Power Distribution at Fixed V/F

Power Range with Fosston
Computers + Radios = Mobile Platforms
Sensor Nodes as Wireless Networked Embedded Systems

Sensor:
- Application specific.

Microcontroller:
- OS, networking.
- Signal processing.
- Power management.

Transceiver
- ISM Band (433/900 MHz, 2.4 GHz)

Energy Storage:
- Battery
- Fuel cell

Autonomous Power Supply:
- Solar power.
- Temperature differences.
- Vibration.
- ...

Source: Estrin, MOBICOM’03
New Computing Nodes

• High-end processing migrating to mobile computing, sensor computing
• Very likely sources of innovations in future for mainstream computing.
Characteristics of Embedded Systems

1. Application specific
2. Digital signal processing in ECS
3. Reactive
4. Real-time
5. Distributed
1. Application Specific

- **Application Specific**
  - Perform a single or tightly knit set of functions;
  - (not usually "general purpose")

- **System designed for a given application**
  - Application is known a priori before the system design begins
  - System flexibility is important for upgrades, product differentiation and design reuse, usually achieved through limited system re-programmability

- **In practice, however, the application development takes place concurrently with ES design**
  - Delayed partitioning of hardware and software is needed (though rarely achieved)

- **Constrained**
  - Power, cost and reliability are often important attributes that influence design;

- **Application specific processor design can be a significant component of some embedded systems.**
  - Customization yields lower area, power, cost, ...
  - Higher HW/software development overhead
    - design, compilers, debuggers, ...
2. Digital Signal Processing in ES

• Continued digitization of signals increasing the role of DSP in ES
  – signals are represented digitally as sequence of “samples”
  – ADC’s are moving closer to signals
    • even RF, IF processing in digital domain!
    • Three tiers of Software Defined Radio
      – Tier 1: Functional, Tier 2: Transport, Tier 3: Physical

• Typical DSP Processing
  – Filtering, averaging, DFT, device control etc etc.
  – speech: codecs, processing, user interface
  – modem: modulation, demodulation, noise/echo cancellation
  – channel: encoding, decoding, equalization, etc etc.
3. Reactivity in Embedded Systems

• Closed systems
  – execution indeterminacy confined to one source
  – causal relations are easily established.

• Open systems
  – indeterminacy from multiple sources, not controllable or observable by the programmer
  – not possible to infer causal relations.

– constraints are an important part of system functionality in building embedded computing systems.

*Embedded is no longer disconnected, remote or low performance: In fact, increasing applications are real-time.*
4. Real-Time Applications of ES

• A real-time system consists of tasks under deadline constraints
  – notion of time typically is global and “physical”
• Hard real-time versus soft real-time systems
  – in hard real-time systems, the tasks must complete by specified deadlines: flight control, collision alert.
  – in soft real-time systems, task execution may go beyond specified deadlines without catastrophic failures, may also be timed-out: display updates, connection establishment.
• Hard real-time systems are more often embedded
  – dedicated applications.
5. Distributed Systems

• Consist of components that may necessarily be physically distributed.
• Consist of communicating processes on multiple processors and/or dedicated hardware connected by communication links.
• Motivation:
  – economical
    • 4 8-bit micro-controller may be cheaper than a 32-bit processor
  – multiple processors to handle multiple time-critical tasks
  – physically distributed
    • devices under control may be physically distributed.
ES Design: Task Ingredients

- **Modelling**
  - the system to be designed, and experimenting with algorithms involved;

- **Refining (or “partitioning”)**
  - the function to be implemented into smaller, interacting pieces;

- **HW-SW partitioning: Allocating**
  - elements in the refined model to either (1) HW units, or (2) SW running on custom hardware or a general microprocessor.

- **Scheduling**
  - the times at which the functions are executed. This is important when several modules in the partition share a single hardware unit.

- **Mapping (Implementing)**
  - a functional description into (1) software that runs on a processor or (2) a collection of custom, semi-custom, or commodity HW.
Scope and Ingredients of ES Design Tasks

- The specific issues that need to be addressed in ES design depend to some extent on
  - the scope of the application at hand, and
  - the richness of the system delivered
A changing ecosystem of embedded systems

Data/External Uncertainty

Real-World Sensor Inputs

Timing-Dependent Data

"Text"

System/Internal Uncertainty

Robotics

Distributed Robotics

Sensor Networks

OS Kernels/Device Drivers

TCP

MS Word

Squid

cat

Single- or Few-Threaded

Multi-Threaded

Distributed, Timing Dependent System

Jeremy Elson, Microsoft
Embedded Software
Software Development Process Is Diverse!

- Different development styles and tools are used for different types of projects.
- We consider the following examples:
  - High-volume, low-cost, low-complexity
    - portable cassette player
  - Medium-volume, moderate-cost, moderate-complexity
    - handheld remote terminal
  - Low-volume, high-cost, high-complexity
    - air-traffic control system
  - very high performance
    - spread-spectrum wireless modem
Portable Cassette Player

• Low complexity
  – Functionality is easy to describe and test
• Quality of design (cost, power, size) related to value
• Generally use
  – 4-bit or 8-bit microcontrollers with on-chip RAM, ROM, IO
  – hand-coded assembly
  – single task, plus may be some interrupt service routines
  – 1K-8K bytes of code, software optimized for size
    • RAM size is critical
• Development tools
  – limited to assembler
• Code has a long life time.
Portable Terminal

- More functionality, more value
- Functionality complex enough to require documentation
- Generally 16-bit or 32-bit microcontrollers
  - on and/or off-chip memory and peripherals
- Software developed as a mixture of C and assembly
  - must be customizable
- More than one task, some time critical (in control loops)
- 64KB to 1MB of code
- Debuggers and simulators may be used.
- Code may be upgraded in new products or in field.
Traffic Control System

- Security and reliability are primary measures of quality
- Functionally is often “formally” (legally) specified in regulatory documents
- Significant development and maintenance costs
- Most are derivatives of desktop systems (32, 64 bit)
- ADA code development: standards.
- Full operating system, distributed environment
- Multiple (and redundant) tasks
- Many mega bytes of code
- Custom tools, continuous updates and fixes.
Wireless Modem

- Performance is important
- Functionality is “simple” but demanding
- Hardware enables the design, software must keep up with hardware capabilities
- Custom ASIP processor
- Custom development environment
  - abstract specification methods
- Integration with hardware design tools is often desired
  - co-simulation and co-synthesis tools
- Hard to do with available design and CAD technology
Specification Methods

• Formal specification of software
  – Visual languages (e.g., ViDEA from ISI)
  – Language-based development leading to “executable specifications”
    • C, C++, ADA, Forth etc.
• Requirements specifications
  – non-functional needs (power, area, performance)
  – usability requirements (interface needs)
• Specify by example
• Availability of tools determines design and specification methods
  – cross-development platforms, compilation, ICE support
  – currently the trend is towards ES development as general purpose software (functionality and performance are disjoint).
Embedded system software is a *design* problem, not a *programming* problem.

*Development system propaganda from a vendor, circa 1975:*

If you think a microprocessor is “just a small computer”...you will:
-- Turn design over to programmers,
-- Waste money on “software”,
-- Waste time talking to your “software”,
-- At least **double** your hardware costs,
-- Not have an effective way to service your systems in the field
-- Scare off the engineers who should be using the microprocessor in the first place.

---

courtesy A. Wolfe
A CAD Methodology for SW

• Automated software synthesis from specs.
• Synthesis tools generate implementation
  – Global optimization of the program.
  – One-time compilation costs.
• Optimization used to achieve design goals.
• Analysis and verification tools for feedback.
Software Optimizations

• Code compression
• Optimization for power
• Instruction-set generation
• Static memory allocation
What is With Compilers?

- "Compilers produce slow and bloated code"
  - The "hello world" phenomenon:
    - i960 C compiler produces 37KB of code for hello.c
    - it also uses library support for IO, vararg, floating point..
  - FFT is only 6.4KB
- C libraries are powerful and extensive
  - but not designed for the embedded system designer.
- "Compiler produced code is difficult to predict/control"
  - relatively minor code modifications have significant impacts
Summary

• The real problem is
  – **compilers do generate bad code**
    • easily times worse than assembly!
  – poor or no support for embedded libraries
• But not a fundamental issue
  – **compiler algorithms beginning to be developed for embedded systems**
• Software for embedded systems needs to be optimized
  – **just like hardware**
  – and needs to be specified just like hardware...
Prelude: Modeling Embedded Computation

• Typically, a program represents a discrete computation
  – Sorting, pattern search, graph algorithms, etc.
• Program is a function from inputs to outputs
• Theoretical basis
  – Finite-state computation: automata theory, regular languages, computability limits (Turing Machines)
  – Correctness through assertions, pre/post conditions, program analysis (type systems, model checking etc)
  – Efficiency analysis through complexity theory, architectural modeling and analysis, simulation tools.
Typical Embedded Computation

repeat {
    read the sensors;
    compute something;
    apply computation output to actuators;
} until forever;

- Reactive computation – non-terminating interactions
- Complex analysis with multiple non-terminating programs
- Even more complex, sincere interactions are time bound, exception may part of normal processing

- Analysis extends from computation, architecture to other domains
  - E.g., modeling dynamics of the car, power grid, ...
Emergence of Formal Methods in ES

• Embedded as a Hybrid System
  – Combination of DES, Dynamical systems

• Timed Languages and Timed Automata
  – An language is a set of words, word is a sequence of symbols over an alphabet
  – A language L is regular if these is a finite automaton that accepts or generates L
  – Extend alphabet to be a tuple with time
    • Timed words
    • Timed automata with continuous variables

Ref: CHARON, Alur/U Penn.