"I believe we are now entering the Renaissance phase of the Information Age, where creativity and ideas are the new currency, and invention is a primary virtue, where technology truly has the power to transform lives, not just businesses, where technology can help us solve fundamental problems."

Carly Fiorina, CEO, Hewlett Packard Corporation
eMerging Societal-Scale Systems

New System Architectures
New Enabled Applications

Diverse, Connected, Physical, Virtual, Fluid

Embedded Systems
MEMS
BioMonitoring

Information Appliances

“Server”

Scalable, Reliable, Secure Services

“Client”

Gigabit Ethernet
Massive Cluster
Clusters
Embedded Software Systems

• Computational
  – but not first-and-foremost a computer

• Integral with physical processes
  – sensors, actuators

• Reactive
  – at the speed of the environment

• Heterogeneous
  – hardware/software, mixed architectures

• Networked
  – shared, adaptive

Source: Edward A. Lee
Observations

• We are on the middle of a revolution in the way electronics products are designed

• System design is the key (also for IC design!)
  – Start with the highest possible level of abstraction (e.g. control algorithms)
  – Establish properties at the right level
  – Use formal models
  – Leverage multiple “scientific” disciplines
Course overview

Managing Complexity

- Orthogonalizing Concerns
  - Behavior Vs. Architecture
  - Computation Vs. Communication
Behavior Vs. Architecture

Models of Computation

Comm. and comp. resources

Quantity estimation

Assign functionality to arch elements

Synthesis: HW and SW

HW/SW partitioning, Scheduling

Mapping

Refinement

How To Implementation

- Polis (1990-1996)
Behavior Vs. Communication

- Clear separation between functionality and interaction model
- Maximize reuse in different environments, change only interaction model
Administration

• Office hours: *Alberto*: Tu-Th 12:30pm-2pm or (better) by appointment (2-4882)

• **Co-Instructor:** *Alessandro Pinto*, apinto@eecs.berkeley.edu

• Teaching Assistant:
  – *Qi Zhu*, zhuqi@eecs.berkeley.edu
Grading

• Grading will be assigned on:
  – Homework (~30%)
  – Project (~50%)
  – Reading assignments (~10%)
  – Labs (10%)

• Bi-weekly homework.
  – HW #n is due the same day HW #n+1 is handed out
Schedule

• Labs (Th. 4-6):
  – Presentation of tools followed by hands-on tutorial and assignments (to turn in after 2 weeks, we might have to skip some labs….)

• Discussion Session (Tu. 5-6)
  – Each student (in groups of 2 people) will have to make one or more oral presentations during the class

• Last two weeks of class dedicated only to projects (usually due the 1st or 2nd week of Dec.)

• Auditors are OK but please register as P-NP (resources are assigned according to students…)

Schedule is tight
Don’t fall behind!!!
Links

- Class website
  - http://inst.eecs.berkeley.edu/~ee249

- But also
  - http://www.eecs.berkeley.edu/~apinto/esd/
### Outline of the course

**Part 1: Introduction**
- Design complexity, Example of embedded systems, traditional design flow, Platform-Based Design

**Part 2: Functional modeling, analysis and simulation**

**Part 3: Architecture and performance abstraction**
- Definition of architecture, examples. Distributed architecture, coordination, communication. Real time operating systems, scheduling of computation and communication.

**Part 4: Mapping**
- Definition of mapping and synthesis. Software synthesis, quasi static scheduling. Behavioral synthesis. Communication Synthesis and communication-based design

**Part 5: Verification**
- Validation vs Simulation. Verification of hybrid system. Interface automata and assume guarantee reasoning.

**Part 6: Applications**
- Automotive: CAN, Flexray, Auotosar Architecture, GM car architecture, scheduling and timing analysis
- Building automation: BanNet, LonWorks, ZigBee with applications to monitoring and security
Outline for the Introduction

• Examples of Embedded Systems
• Their Impact on Society
• Design Challenges
• Embedded Software and Control
Electronics and the Car

• More than 30% of the cost of a car is now in Electronics
• 90% of all innovations will be based on electronic systems
Automotive Industry
Three Levels of Players

Automakers
- 2005 Revenue: $1.1T
- CAGR 2.8% (2004-2010)

Tier 1 Suppliers
- 2004 Revenue ~$200B
- CAGR 5.4% (2004-2010)

IC Vendors
- 2005 revenue $17.4B
- CAGR 10% (2004-2010)

Source: Public financials, Gartner 2005
Electronics, Controls & Software
Shifting the Basis of Competition in Vehicles

- More functions & features
- Less hardware
- Faster

Potential inflection point. Now!

Value from Electronics & Software

Mechanical $
Electronics $
Software $
Other $

Electric Ignition
Electric Fan
Rear Vision
Head Airbags
... 

ACC
OnStar
Rear View
Passive Entry
... 

Fuel Cell
Wheel Motor
Hybrid PT
... 

Source: Matt Tsien, GM

1970s 1980s 1990s 2000s 2010s 2020s

17

System Connection
Subsystem Controls & Features
Forefront of Innovation

Vehicle Integration

EV249Fall07
FUNCTION OF CONTROLS

Typical minivan application

Configure
Sense
Actuate
Regulate
Display
Trend
Diagnose
Predict
Archive
CARRIER CONTROLS BUSINESS

Market segments

2001 ($ millions)

- Refrigeration: $87
- Residential HVAC: $212
- Commercial HVAC: $175

Total: $474
FUNCTION OF CONTROLS
Typical commercial HVAC application

Configure
Sense
Actuate
Regulate
Display
Trend
Diagnose
Predict
Archive
OTIS Elevators

1. EN: GeN2-Cx
2. ANSI: Gen2/GEM
3. JIS: GeN2-JIS
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type 1</th>
<th>Type 2</th>
<th>Type 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stops/Rise</td>
<td>&lt; 20 stops</td>
<td>&lt; 64 stops</td>
<td>&lt; 128 stops</td>
</tr>
<tr>
<td></td>
<td>Opportunity: &lt; 6 stops (20m)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group Size</td>
<td>Simplex</td>
<td>1 – 8 cars</td>
<td>1 – 8 cars</td>
</tr>
<tr>
<td>Speed</td>
<td>&lt; 4m/s</td>
<td>&lt; 4 m/s</td>
<td>&lt; 15 m/s</td>
</tr>
<tr>
<td></td>
<td>&lt;= .75 m/s (ANSI)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op Features</td>
<td>Basic</td>
<td>Advanced</td>
<td>Hi-End Dispatch</td>
</tr>
<tr>
<td>Motion Features</td>
<td>Basic Perf.</td>
<td>Limited Perf.</td>
<td>Advanced Perf.</td>
</tr>
<tr>
<td></td>
<td>Basic FM</td>
<td>Advanced FM</td>
<td>Advanced FM</td>
</tr>
<tr>
<td>Code</td>
<td>EN, ANSI, JIS</td>
<td>EN, ANSI, JIS</td>
<td>EN, ANSI, JIS</td>
</tr>
<tr>
<td>Remote Service</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Price Sensitivity</td>
<td>High</td>
<td>High, Med</td>
<td>Med</td>
</tr>
<tr>
<td>Market</td>
<td>Utility</td>
<td>Utility, Design</td>
<td>Design</td>
</tr>
</tbody>
</table>
**System Above Chip - SAC**

- **ST Reference Designs** (Qualified Software, Certification, Cost Effective Turnkey Manufacturing Tooling & Specifications)
- Application (Navig., Electr. Guide, Browsing, ...)
- Middleware A.L. (MediaHighway, OpenTV)
- ST Drivers (audio, video, OSD, demux, tuner, smartcard, teletext...)
- RTOS (STLite, VxWorks, PSOS)
- Hardware Adaptation Layer
- OMEGA Silicon Platforms

---

**Supplied by ST**

- **System-Above-Chip** (Boards, Chips, & Software)
  - NO value in customer owning/writing drivers. (TMM,E*, HNS)
  - Customer added value is Application, Conditional Access, Brand Name

- ST supplies the complete base system BELOW MIDDLEWARE to save time to market

---

**CMG-Design**

STMicroelectronics Confidential and Proprietary
Consumer segments

Common technology elements

‘Systems within systems’

- Multimedia processors
- Embedded µP
- Wireless connectivity
  Baseband processing, RF transceivers
- Power Amps
- Flat panel displays
- Digital signal processor technologies
- VOIP

Gaming

Broadcasting

Locality

e-commerce

Internet

Auto electronics
Common Situation in Industry

- Different hardware devices and architectures
- Increased complexity
- Non-standard tools and design processes
- Redundant development efforts
- Increased R&D and sustaining costs
- Lack of standardization results in greater quality risks
- Customer confusion
Outline for the Introduction

• Examples of Embedded Systems
• The Future of Embedded Systems and Their Impact on Society
• Design Challenges
• Embedded Software and Control
Smart Dust

Passive CCR comm.
MEMS/polysilicon

Laser diode
III-V process

Active beam steering laser comm.
MEMS/optical quality polysilicon

Sensor
MEMS/bulk, surface, ...

Analog I/O, DSP, Control
COTS CMOS

Power capacitor
Multi-layer ceramic

Solar cell
CMOS or III-V

Thick film battery
Sol/gel V$_2$O$_5$

1-2 mm

Source: K. Pister, Berkeley
Energy Scavenging: Vibration

Source: P. Wright, Berkeley

Berkeley Dust Mote¹

Berkeley Mote¹

¹From Pister et al., Berkeley Smart Dust Project
Creating a Whole New World of Applications

From Monitoring

To Automation
Demand response:
Make energy prices dependent upon time-of-use

- Advanced thermostats operate on required level of comfort, energy cost, weather forecast and distributed measurements to offload peak times
- Appliances are energy and cost aware
Automotive Electronics: Occupant Safety

Occupant Safety Systems Portfolio

- ECU incl. Rollover
- Pedestrian Sensing
- Peripheral Front Sensors
- Peripheral Side Sensors
- Pressure Sensor
- iVision™
- iBOLT™

BOSCH
Tire to Vehicle

SW &

Rx/Tx Antenna

Sensing Device  RF Link  Computing

Power Management

Energy Scavenging

Stability Control System

computer

EE249Fall07
Industrial Plants

Monitoring:
Vibrations, Temperature, Humidity, Position, Logistics

Current solution:
Wired Infrastructure

Future solution:
WIRELESS

Wireless advantages:
Reduce cabling
Enhance flexibility
Easy to deploy
Higher safety
Decreased maintenance costs
Temperature Tracking

- No or little real-time data on assets, environment, or activity
  - Inventory/supply management
    - Pharmaceutical
    - Foods
  - Automated meter reading

Source: Xbow
Weyerhaeuser 20 Million Seed Management

**Task:**
- Manage 20 million fast growing seeds annually

**Issue:**
- Seed dormancy depends on a complex combination of water, light, temperature, gasses, mechanical restrictions, seed coats, and hormone structures
Tree Growth Rate Variability

• Old Method
  • Trust nature
  • Monitor local atmospheric conditions

• Sensor Network Way:
  • Monitor soil temperature and moisture at various locations
  • Adjust irrigation schedule accordingly
Preventative Maintenance Program on Oil Tankers

• The task:
  – Engine monitoring is critical for both keeping the ship operational and complying with insurance policy.

• Old Methods
  – Manually record vibration profile with data loggers.
  – Post process data for engine health and diagnostics.

Source: Xbow
Personal Themes

- Data driven, remote feedback control
- Government or industry mandates
- Personal computing themes
  - Ubiquitous computing
  - Safety
  - Convenience
  - Health and performance
  - Entertainment

Source: Xbow
Applications

Distributed Bio-monitoring

- Wristband bio-monitors for chronic illness and the elderly
- Monitored remotely 24x7x365
- Emergency response and potential remote drug delivery
- Cardiac Arrest
  - Raise out-of-hospital survival rate from 6% to 20% => save 60K lives/year
Silicon-Processed Micro-needles

- Neural probe with fluid channel for bio-medical appl.
- Two micro-needles penetrating porterhouse (New-York)

Applications

Disaster Mitigation (natural and otherwise)

- Monitor buildings, bridges, lifeline systems to assess damage after disaster
- Provide efficient, personalized responses
- Must function at maximum performance under very difficult circumstances
What is Disaster Response?

- Sensors installed near critical structural points
- Sensor measure motion, distinguish normal deterioration and serious damage
- Sensors report location, kinematics of damage during and after an extreme event
  - Guide emergency personnel
  - Assess structural safety without deconstructing building
Seismic Monitoring of Buildings: Before CITRIS

$8,000 each
Seismic Monitoring of Buildings: With CITRIS Wireless Motes

$70 each
Stability of Masada North Face:

The Foundations of King Herod’s Palace
Discussion

• What are the most challenging aspects of these applications (and how does a company make money) ?
  – Interaction mechanisms: sensors, actuators, wireless networks
  – Reliability and survivability
  – Infrastructure
  – Services
  – Legislation
  – ……
Picoradio Sensor Networks (BWRC)

- Control Environmental parameters (temperature, humidity...)
- Minimize Power consumption
- Cheap (<0.5$) and small (< 1 cm³)
- Large numbers of nodes — between 0.05 and 1 nodes/m²
- Limited operation range of network — maximum 50-100 m
- Low data rates per node — 1-10 bits/sec average
- Low mobility (at least 90% of the nodes stationary)

Key challenges

- Satisfy tight performance and cost constraints (especially power consumption)
- Identify Layers of Abstraction (Protocol Stack)
- Develop distributed algorithms (e.g. locationing, routing) for ubiquitous computing applications
- Design Embedded System Platform to implement Protocol Stack efficiently
Secure Network Embedded SystEms (SENSE)

- Networked embedded systems and distributed control creates a new generation of future applications: new infrastructures
- We need to think about how to prevent the introduction of vulnerabilities via this exciting technology
- Security, Networking, Embedded Systems
Outline for the Introduction

- Examples of Embedded Systems
- Their Impact on Society
- Design Challenges
- Embedded Software and Control
Opportunity: Electronic Systems Design Chain

Design Science

System Design → Implementation → Manufacturing → Interfaces → IP

Fabrics
Disaggregation:
Complex Design Chain Management

Supply Chain
- Movement of tangible goods from sources to end market
- Supply Chain Management is $3.8B market projected to be $20B in 2005

Design Chain
- Movement of technology (IP and knowledge) from sources to end market
- Design Chain Management is an untapped market
Supply Chain:
Design Roles -> Methodology -> Tools
Automotive Supply Chain: Car Manufacturers

Product Specification & Architecture Definition
(e.g., determination of Protocols and Communication standards)
System Partitioning and Subsystem Specification
Critical Software Development
System Integration
Today, more than 80 microprocessors and millions of lines of code.
### Automotive Supply Chain: Tier 1 Subsystem Providers

- **Subsystem Partitioning**
- **Subsystem Integration**
- **Software Design: Control Algorithms, Data Processing**
- **Physical Implementation and Production**

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Transmission ECU</td>
</tr>
<tr>
<td>2</td>
<td>Actuation group</td>
</tr>
<tr>
<td>3</td>
<td>Engine ECU</td>
</tr>
<tr>
<td>4</td>
<td>DBW</td>
</tr>
<tr>
<td>5</td>
<td>Active shift display</td>
</tr>
<tr>
<td>6/7</td>
<td>Up/Down buttons</td>
</tr>
<tr>
<td>8</td>
<td>City mode button</td>
</tr>
<tr>
<td>9</td>
<td>Up/Down lever</td>
</tr>
<tr>
<td>10</td>
<td>Accelerator pedal position sensor</td>
</tr>
<tr>
<td>11</td>
<td>Brake switch</td>
</tr>
</tbody>
</table>

![Diagram of automotive subsystems](image-url)
Automotive Supply Chain: Subsystem Providers

Application Platform layer
(≈ 10% of total SW)

OSEK

Transport

KWP 2000

CCP

Application Specific Software

I/O drivers & handlers
(> 20 configurable modules)

μControllers Library

OSEK COM

Hardware layer

μController Library

Platform Integration
Software Design

“firmware” and “glue software”

“Application”
Automotive Supply Chain: Platform & IP Providers

- **Application Platform layer**
  - (≅ 10% of total SW)
  - Application Libraries
  - Odometer Speedometer
  - Application Specific Software
  - Application Programming Interface

- **SW Platform layer**
  - (> 60% of total SW)
  - OSEK RTOS
  - I/O drivers & handlers
  - (> 20 configurable modules)
  - μControllers Library

- **HW layer**
  - Nec78k
  - HC08
  - HC12
  - H8S26
  - MB90

“Software” platform:
- RTOS and communication layer
  - Hardware and IO drivers

“Hardware” platform:
- Hardware and IO drivers
Outline for the Introduction

• Examples of Embedded Systems
• Their Impact on Society
• Design Challenges
• Embedded Software and Control
How Safe is Our Real-Time Software?
Computing for Embedded Systems
Mars, December 3, 1999
Crashed due to un-initialized variable
$4 billion development effort
40-50% system integration & validation cost
## Complexity, Quality, & Time To Market today

<table>
<thead>
<tr>
<th></th>
<th>PWT UNIT</th>
<th>BODY GATEWAY</th>
<th>INSTRUMENT CLUSTER</th>
<th>TELEMATIC UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory</strong></td>
<td>256 Kb</td>
<td>128 Kb</td>
<td>184 Kb</td>
<td>8 Mb</td>
</tr>
<tr>
<td><strong>Lines Of Code</strong></td>
<td>50.000</td>
<td>30.000</td>
<td>45.000</td>
<td>300.000</td>
</tr>
<tr>
<td><strong>Productivity</strong></td>
<td>6 Lines/Day</td>
<td>10 Lines/Day</td>
<td>6 Lines/Day</td>
<td>10 Lines/Day*</td>
</tr>
<tr>
<td><strong>Residual Defect Rate @ End Of Dev</strong></td>
<td>3000 Ppm</td>
<td>2500 ppm</td>
<td>2000 ppm</td>
<td>1000 ppm</td>
</tr>
<tr>
<td><strong>Changing Rate</strong></td>
<td>3 Years</td>
<td>2 Years</td>
<td>1 Year</td>
<td>&lt; 1 Year</td>
</tr>
<tr>
<td><strong>Dev. Effort</strong></td>
<td>40 Man-yr</td>
<td>12 Man-yr</td>
<td>30 Man-yr</td>
<td>200 Man-yr</td>
</tr>
<tr>
<td><strong>Validation Time</strong></td>
<td>5 Months</td>
<td>1 Month</td>
<td>2 Months</td>
<td>2 Months</td>
</tr>
<tr>
<td><strong>Time To Market</strong></td>
<td>24 Months</td>
<td>18 Months</td>
<td>12 Months</td>
<td>&lt; 12 Months</td>
</tr>
</tbody>
</table>

* C++ CODE

FABIO ROMEO, Magneti-Marelli
DAC, Las Vegas, June 20th, 2001
How is Embedded Software Different from Ordinary Software?

• It has to work

• One or more (very) limited resources
  – Registers
  – RAM
  – Bandwidth
  – Time
Devil’s Advocate

• So what’s different?

• All software works with limited resources

• We have compiler technology to deal with it
  – Various forms of program analysis
Example: Registers

• All machines have only a few registers

• Compiler uses the registers as best as it can
  – Spills the remaining values to main memory
  – Manages transfers to and from registers

• The programmer feels she has 1 registers
The Standard Trick

- This idea generalizes

- For scarce resource X
  - Manage X as best as we can
  - If we need more, fall back to secondary strategy
  - Give the programmer a nice abstraction
The Standard Trick

• This idea generalizes

• For scarce resource X
  – Manage X as best we can
  – *Any correct heuristic is OK, no matter how complex*
  – If we need more, fall back to secondary strategy
  – *Focus on average case behavior*
  – *Give the programmer a nice abstraction*
Examples of the Standard Trick

- Compilers
  - Register allocation
  - Dynamic memory management
- OS
  - Virtual memory
  - Caches

**Summary:** abstract and hide complexity of resources
What’s Wrong with This?

- Embedded systems have limited resources
- Meaning hard limits
  - Cannot use more time
  - Cannot use more registers
- The compiler must either
  - Produce code within these limits
  - Report failure
- The standard trick is anathema to embedded systems
  - Can’t hide resources
Revisiting the Assumptions

- **Any correct heuristic is OK, no matter how complex**
  - Embedded programmer must understand reasons for failure
  - Feedback must be relatively straightforward

- **Focus on average case behavior**
  - Embedded compiler must reason about the worst case
  - Cannot improve average case at expense of worst case

*Give the programmer a nice abstraction*
- Still need abstractions, but likely different ones
Everything “computable” can be given by a terminating sequential program.

- Functions on bit patterns
- Time is irrelevant
- Non-terminating programs are defective
Current fashion – Pay Attention to “Non-functional properties”

- Time
- Security
- Fault tolerance
- Power consumption
- Memory management

But the formulation of the question is very telling:

How is it that *when* a braking system applies the brakes is any less a *function* of the braking system than *how much* braking it applies?
Infinite sequences of state transformations are called “processes” or “threads”.

In prevailing software practice, processes are sequences of external interactions (total orders).

And messaging protocols are combined in ad hoc ways.
Interacting Processes – Concurrency as Afterthought

Software realizing these interactions is written at a very low level (e.g., semaphores). Very hard to get it right.
Interacting Processes – Not Compositional

An aggregation of processes is not a process (a total order of external interactions). What is it?

Many software failures are due to this ill-defined composition.
Non-compositional formalisms lead to very awkward architectures.
What About Real Time?

“Make it faster!”
Design “Practice”
Design Science: Build upon Solid Foundations
Software Architecture Tomorrow?
The Goal (CHESS Project)

- To create a modern computational systems science and systems design practice with
  - Concurrency
  - Composability
  - Time
  - Hierarchy
  - Heterogeneity
  - Resource constraints
  - Verifiability
  - Understandability
A Traditional Systems Science – Feedback Control Systems

- Models of continuous-time dynamics
- Stability analysis
- But not accurate for software controllers
Discretized Model –
A Step Towards Software

- Numerical integration techniques provided ways to get from the continuous idealizations to computable algorithms.
- Discrete-time signal processing techniques offer the same sophisticated stability analysis as continuous-time methods.
- But it’s still not accurate for software controllers

In general, $z$ is an $N$-tuple, $z = (z_1, \ldots, z_N)$, where $z_t: \text{Reals}_+ \rightarrow \text{Reals}$. The derivative of an $N$-tuple is simply the $N$-tuple of derivatives, $\dot{z} = (\dot{z}_1, \ldots, \dot{z}_N)$. We know from calculus that

$$\dot{z}(t) = \frac{dz}{dt} = \lim_{\delta \to 0} \frac{z(t+\delta) - z(t)}{\delta},$$

and so, if $\delta > 0$ is a small number, we can approximate this derivative by

$$\dot{z}(t) \approx \frac{z(t+\delta) - z(t)}{\delta}.$$

Using this for the derivative in the left-hand side of (5.50) we get

$$z(t+\delta) - z(t) = \delta g(z(t), v(t)). \quad (5.51)$$
Hybrid Systems – Reconciliation of Continuous & Discrete

- But it’s still not accurate for software controllers

This model gives two separate ordinary differential equations, one for each point mass attached to a spring. The ZeroCrossingDetector actor detects the collision of the point masses and emits the "touched" event.

V1 and V2 are velocities, and P1 and P2 are positions of the two masses.
Timing in Software is More Complex Than What the Theory Deals With

An example (Jie Liu) models two controllers sharing a CPU under an RTOS. Under preemptive multitasking, only one can be made stable (depending on the relative priorities). Under non-preemptive multitasking, both can be made stable.

Where is the theory for this?

This model shows two (independent) control loops whose controllers share the same CPU. The control loops are chosen such that it is unstable if the control signals are constantly delayed. By choosing different priority assignments and TM scheduling policies, different stability of the two loops may appear. For example, a non-preemptive scheduling can stabilize both control loops, but none of the preemptive ones can.
Foundational Theory Research …

• The science of computation has systematically abstracted away the physical world. The science of physical systems has systematically ignored computational limitations. Embedded software systems, however, engage the physical world in a computational manner.

• It is time to construct a Hybrid Systems Science that is simultaneously computational and physical. Time, concurrency, robustness, continuums, and resource management must be remarried to computation.