EE249 Embedded System Design: Models, Validation and Synthesis
Alberto Sangiovanni Vincentelli
"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”

“Client”

Scalable, Reliable, Secure Services

Gigabit Ethernet
Massive Cluster
Embedded 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
- System Behavior
- Synthesis: HW and SW
- Quantity estimation
- Mapping
- Assign functionality to arch elements
- HW/SW partitioning
- Scheduling
- Refinement
- How to Implementation

Comm. and comp. resources

- Polis (1990-1996)
- Metropolis (2003-present)
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)

- Teaching Assistant:
  - **Kelvin Lwin**, klwin@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 (possibly 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….)
Links

- Class website
  - http://inst.eecs.berkeley.edu/~ee249
<table>
<thead>
<tr>
<th>Part 1: Introduction</th>
<th>Design complexity, Example of embedded systems, traditional design flow, Platform-Based Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 3: Architecture and performance abstraction</td>
<td>Definition of architecture, examples. Distributed architecture, coordination, communication. Real time operating systems, scheduling of computation and communication.</td>
</tr>
<tr>
<td>Part 5: Verification</td>
<td>Validation vs Simulation. Verification of hybrid system. Interface automata and assume guarantee reasoning.</td>
</tr>
<tr>
<td>Part 6: Applications</td>
<td>Automotive: CAN, Flexray, Autosar Architecture, GM car architecture, scheduling and timing analysis</td>
</tr>
<tr>
<td></td>
<td>Building automation: BanNet, LonWorks, ZigBee with applications to monitoring and security</td>
</tr>
</tbody>
</table>
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
- GM
- Ford
- Toyota
- DaimlerChrysler
- VW
- Honda
-

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

Tier 1 Suppliers
- Visteon
- Denso
- Johnson Controls
- Bosch
- Delphi
-

90%+ of revenue from automotive
2004 Revenue ~$200B
CAGR 5.4% (2004-2010)

IC Vendors
- Freescale
- Renesas
- ST
- Infineon
- NEC
- ~15% of revenue from automotive

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

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

- More functions & features
- Less hardware
- Faster

Potential inflection point. Now!

Vehicle Integration
System Connection
Subsystem Controls & Features
Forefront of Innovation

**Value from Electronics & Software**
- More functions & features
- Less hardware
- Faster

**1970s 1980s 2000s 2010s 2020s**

**1990s**
- BCM
- ABS
- TCC
- EGR
- Electric Fan
- Electric Ignition
- Fuel Cell
- Wheel Motor
- Hybrid PT

**2000s**
- GDI
- ACC
- Rear Vision
- Passive Entry
- Side Airbags
- Head Airbags
- OnStar
- OBD II
- HI Spd Data
- Rear aud/vid

**Challenge: Electronics, Controls and Software**
Shifting the Basis of Competition in Vehicles

<table>
<thead>
<tr>
<th>Year</th>
<th>Mechanical %</th>
<th>Software %</th>
<th>Other %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1970s</td>
<td>76%</td>
<td>2%</td>
<td>9%</td>
</tr>
<tr>
<td>1980s</td>
<td>76%</td>
<td>2%</td>
<td>9%</td>
</tr>
<tr>
<td>1990s</td>
<td>55%</td>
<td>13%</td>
<td>8%</td>
</tr>
<tr>
<td>2000s</td>
<td>55%</td>
<td>13%</td>
<td>8%</td>
</tr>
<tr>
<td>2010s</td>
<td>24%</td>
<td>13%</td>
<td>8%</td>
</tr>
<tr>
<td>2020s</td>
<td>24%</td>
<td>13%</td>
<td>8%</td>
</tr>
</tbody>
</table>

**Value from Electronics & Software**

- **1970s**
  - Mechanical $76%
  - Software $2%
  - Other $9%

- **1980s**
  - Mechanical $76%
  - Software $2%
  - Other $9%

- **1990s**
  - Mechanical $55%
  - Software $13%
  - Other $8%

- **2000s**
  - Mechanical $55%
  - Software $13%
  - Other $8%

- **2010s**
  - Mechanical $24%
  - Software $13%
  - Other $8%

- **2020s**
  - Mechanical $24%
  - Software $13%
  - Other $8%

**Categories**
- **Mechanical**
- **Software**
- **Other**

**Percentage Breakdown**
- **Mechanical**
- **Software**
- **Other**

**Cost Analysis**

- **1970s**
  - Mechanical $400
  - Software $50
  - Other $40
  - Total $500

- **1980s**
  - Mechanical $400
  - Software $50
  - Other $40
  - Total $500

- **1990s**
  - Mechanical $1182
  - Software $196
  - Other $150
  - Total $1528

- **2000s**
  - Mechanical $1182
  - Software $196
  - Other $150
  - Total $1528

- **2010s**
  - Mechanical $400
  - Software $50
  - Other $40
  - Total $500

- **2020s**
  - Mechanical $400
  - Software $50
  - Other $40
  - Total $500
GM SAC Vehicular Electronics, Controls and Software Study

• Software content in automobiles could increase by 100 X over the next 5-6 years. Challenges will include:
  – Software system architecture
  – Partitioning for modularity & system reliability
  – Reuse
  – Standardization of interfaces
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; 4 m/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**

- **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**

---

**Supplied by ST**

---

**2003 & Beyond**
Consumer segments

Common technology elements

- Gaming
- Broadcasting
- Locality
- e-commerce
- Internet
- Auto electronics

‘Systems within systems’

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

Today, more than 80 Microprocessors and millions of lines of code

Source: Bosch
Challenge: The Physical Internet

- Mainframe
- Minicomputer
- Workstation
- PC
- Laptop
- PDA
- Cellular phone

Log (people per computer)

Year

Number Crunching Data Storage

Productivity Interactive

Streaming information to and from physical world

Ubiquitous Sensor Networks
Exponentials Bound to Continue

• 5 Billion people to be connected by 2015 (Source: NSN)
• The emergence of Web2.0
  – The “always connected” community network
• 7 trillion wireless devices serving 7 billion people in 2017 (Source: WirelessWorldResearchForum (WWRF))
  – 1000 wireless devices per person?

[Courtesy: Niko Kiukkonen, Nokia]
The Emerging IT Scene

Infrastructural core

Sensory swarm

Mobile access
The Technology Gradient: Computation

Driven by Moore’s Law

Driven by “More Than Moore” and “Beyond Moore”
The Technology Gradient: Communication

Mostly wired

Almost uniquely wireless
Energy = upper bound on the amount of available computation

- Total Energy of Milky Way Galaxy: $10^{59}$ J
- Minimum switching energy for digital gate (1 electron@100 mV): $1.6 \times 10^{-20}$ J (limited by thermal noise).
- Upper bound on number of digital operations: $6 \times 10^{78}$
- Operations/year performed by 1 billion 100 MOPS computers: $3 \times 10^{24}$
- Energy consumed in 180 years assuming a doubling of computational requirements every year.
Challenge: Parallel Architectures

Scaling enabled integration of complex systems with hundreds of millions of devices on a single die

Intel KEROM dual core
ISSCC 07, 290M trans.

SUN Niagara-2
ISSCC 07, 500M trans.

IBM/Sony Cell
ISSCC 05, 235M trans.
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$_5$

Source: K. Pister, Berkeley
Wireless Sensor Networks


Berkeley Dust Mote\(^1\)

Berkeley Mote\(^1\)

\(^1\)From Pister et al., *Berkeley Smart Dust Project*
Creating a Whole New World of Applications

From Monitoring

To Automation
Energy Management and Conservation

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

Cal ISO Daily Peak Loads
January 1, 2000 - December 31, 2000

Peak Day August 16 - 43.5 GW
Commercial AC
Residential AC
Automotive Electronics: Occupant Safety

Occupant Safety Systems Portfolio

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

Bosch
Tire to Vehicle

Sensing Device

RF Link

Computing

Power Management

Energy Scavenging

Rx/Tx Antenna

SW &

Stability Control System

computer

EE249Fall08
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

Source: Xbow
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
Silicon-Processed Micro-needles

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

IP

Interfaces

Manufacturing

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
Automotive Supply Chain: Tier 1 Subsystem Providers

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

1. Transmission ECU
2. Actuation group
3. Engine ECU
4. DBW
5. Active shift display
6/7. Up/Down buttons
8. City mode button
9. Up/Down lever
10. Accelerator pedal position sensor
11. Brake switch

Tier 1 Subsystem Providers:

- Bosch
- Magneti Marelli
Automotive Supply Chain: Subsystem Providers

Application Platform layer
(≈ 10% of total SW)

SW Platform layer
(> 60% of total SW)

HW layer

Platform Integration
Software Design

"firmware" and "glue software"
"Application"
Automotive Supply Chain: Platform & IP Providers

- **Application Platform layer** (≈ 10% of total SW)
  - Application Libraries
  - Application Specific Software

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

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

- “Software” platform
- “Hardware” platform

**RTOS and communication layer**
- Hardware and IO drivers

**“Software” platform**
- “Hardware” platform
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
<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>Memory</td>
<td>256 Kb</td>
<td>128 Kb</td>
<td>184 Kb</td>
<td>8 Mb</td>
</tr>
<tr>
<td>Lines Of Code</td>
<td>50,000</td>
<td>30,000</td>
<td>45,000</td>
<td>300,000</td>
</tr>
<tr>
<td>Productivity</td>
<td>6 Lines/Day</td>
<td>10 Lines/Day</td>
<td>6 Lines/Day</td>
<td>10 Lines/Day*</td>
</tr>
<tr>
<td>Residual Defect Rate @ End Of Dev</td>
<td>3000 Ppm</td>
<td>2500 ppm</td>
<td>2000 ppm</td>
<td>1000 ppm</td>
</tr>
<tr>
<td>Changing Rate</td>
<td>3 Years</td>
<td>2 Years</td>
<td>1 Year</td>
<td>&lt; 1 Year</td>
</tr>
<tr>
<td>Dev. Effort</td>
<td>40 Man-yr</td>
<td>12 Man-yr</td>
<td>30 Man-yr</td>
<td>200 Man-yr</td>
</tr>
<tr>
<td>Validation Time</td>
<td>5 Months</td>
<td>1 Month</td>
<td>2 Months</td>
<td>2 Months</td>
</tr>
<tr>
<td>Time To Market</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
Software Bugs Cost $59.5 Billion a Year

INFOWORLD, JUNE 28, 2002 – BY PAUL KRILL

Software bugs cost $59.5 billion a year, study says

Software bugs cost the U.S. economy an estimated $59.5 billion per year, or 0.6 percent of the gross domestic product, according to a newly released study by the U.S. Department of Commerce National Institute of Standards and Technology (NIST). In a statement released on Friday, NIST said more than half the costs are borne by software users and the remainder by software developers and vendors. Additionally, the study found that although errors cannot be removed, more than a third of the costs, or an estimated $22.2 billion, could be eliminated by improved testing that enables earlier and more effective identification and removal of defects. Currently, more than half of errors are not found until “downstream” in the development process or during post-sale use of software, according to NIST.
Embedded Software Architecture Today
We Live in an Imperfect World!

What’s Bugging the High-Tech Car?

On a hot summer trip to Cape Cod, the Mills family minivan did a peculiar thing. After an hour on the road, it began to sputter. Mom and Dad were cool and comfortable up front, but heat was blasting into the rear of the van and it could not be turned off.

Fortunately for the Mills children, their father, W. Nathaniel Mills III, an expert on computer networking at IBM, is persistent. When three dealership visits, days of waiting and the cumbersome replacement of mechanical parts failed to fix the problem, he took the van out and drove it until the oven fired up again. Then he rushed to the mechanic to look for a software error.

Additionally, the study found that although errors cannot be removed, more than a few took two minutes for them to hook up their diagnostic tool and find the fault. "It's a car," said Mr. Mills, senior technical staff member at IBM's T.J. Watson Research Center in Hawthorne, N.Y. "I can almost see the software code, a sensor was bad!"

Indeed, the high-tech comfort cars are confusing customers. The 2001 Lexus was found to be freezing up when loyal van drivers got up, third, and up. More than a billion, they said.

NHTSA To Probe Reports Of Sudden Engine Stalls In Prius Hybrids

The National Highway Traffic Safety Administration said yesterday it is investigating reports that a software problem can cause the engine of Toyota’s Prius hybrid to stall without warning at highway speeds. No accidents have been reported thus far.

NHTSA has received 33 reports of stalling in Prius cars from model years 2004 and 2005, according to the agency’s initial report. More than 85 percent of the cars that stalled did so at speeds between 35 and 65 miles per hour.
How is Embedded Software Different from Ordinary Software?

• It has to work

• One or more (very) limited resources
  – Registers
  – RAM
  – Bandwidth
  – Time

Source: Alex Aiken
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

Source: Alex Aiken
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

*Source: Alex Aiken*
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

Source: Alex Aiken
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*

Source: Alex Aiken
Examples of the Standard Trick

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

*Summary: abstract and hide complexity of resources*

*Source: Alex Aiken*
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

Source: Alex Aiken
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

Source: Alex Aiken
Everything “computable” can be given by a terminating sequential program.

- Functions on bit patterns
- Time is irrelevant
- Non-terminating programs are defective

Alan Turing

Source Ed Lee
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.

Source Ed Lee
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.

Source: Ed Lee
Non-compositional formalisms lead to very awkward architectures.
What About Real Time?

“Make it faster!”
First Challenge on the Cyber Side: Real-Time and Power-aware Software

Correct execution of a program in C, C#, Java, Haskell, etc. has nothing to do with how long it takes to do anything. All our computation and networking abstractions are built on this premise.

Timing of programs is not repeatable, except at very coarse granularity.

Programmers have to step outside the programming abstractions to specify timing and power behavior.
Second Challenge on the Cyber Side: Concurrency

Threads dominate concurrent software.

- **Threads**: Sequential computation with shared memory.
- **Interrupts**: Threads started by the hardware.

Incomprehensible interactions between threads are the sources of many problems:

- Deadlock
- Priority inversion
- Scheduling anomalies
- Nondeterminism
- Buffer overruns
- System crashes
Common Features

- Systems are assembled out of heterogeneous components
- Systems are distributed
- Interactions difficult to define
The Intellectual Agenda

To create a modern computational systems science and systems design practice with

- Concurrency
- Composability
- Time
- Hierarchy
- Heterogeneity
- Resource constraints
- Verifiability
- Understandability
Chess: Center for Hybrid and Embedded Software Systems

Principal Investigators
- Thomas Henzinger (EPFL)
- Edward A. Lee (Berkeley)
- Alberto Sangiovanni-Vincentelli (Berkeley)
- Shankar Sastry (Berkeley)
- Janos Sztipanovits (Vanderbilt)
- Claire Tomlin (Berkeley)

Executive Director
- Christopher Brooks

Associated Faculty
- David Auslander (Berkeley, ME)
- Ahmad Bahai (Berkeley)
- Ruzena Bajcsy (Berkeley)
- Gautam Biswas (Vanderbilt)
- Ras Bodik (Berkeley, CS)
- Bella Bollobas (Memphis)
- Karl Hedrick (Berkeley, ME)
- Gabor Karsai (Vanderbilt)
- Kurt Keutzer (Berkeley)
- George Necula (Berkeley, CS)
- Koushik Sen (Berkeley, CS)
- Sanjit Seshia (Berkeley)
- Jonathan Sprinkle (Arizona)
- Masayoshi Tomizuka (Berkeley, ME)
- Pravin Varaiya (Berkeley)

This center, founded in 2002, blends systems theorists and application domain experts with software technologists and computer scientists.

Applications
- Building systems
- Automotive
- Synthetic biology
- Medical systems
- Instrumentation
- Factory automation
- Avionics

Some Research Projects
- Precision-timed (PRET) machines
- Distributed real-time computing
- Systems of systems
- Theoretical foundations of CPS
- Hybrid systems
- Design technologies
- Verification
- Intelligent control
- Modeling and simulation
Why can’t we make Software Reliable?

Uptime: 125 years

Source: T. Henzinger
Why can’t we make Software reliable?

Engineering
- Theories of estimation.
- Theories of robustness.

Computer Science
- Theories of correctness.

Source: T. Henzinger
Why can’t we make Software reliable?

Engineering
Theories of estimation.
Theories of robustness.

Goal: build reliable systems.

Computer Science
Theories of correctness.

Temptation: programs are mathematical objects; hence we want to prove them correct.

Source: T. Henzinger
The CHESS Premise:
The pendulum has swung too far

Source: T. Henzinger
Embedded Systems are a perfect playground to readjust the pendulum.

Source: T. Henzinger
Execution constraints

- CPU speed
- Power
- Failure rates

Reaction constraints

- Deadlines
- Throughput
- Jitter

Embedded Systems

Computation

- Algorithms
- Protocols
- Reuse

Source: T. Henzinger
Embedded System Design is generalized hardware design (e.g. System C).

**Execution constraints**
- CPU speed
- power
- failure rates

**Reaction constraints**
- deadlines
- throughput
- jitter

**Computation**
- algorithms
- protocols
- reuse

Source: T. Henzinger
Embedded System Design is generalized control design (e.g. Matlab Simulink).

Execution constraints
- CPU speed
- Power
- Failure rates

Reaction constraints
- Deadlines
- Throughput
- Jitter

Computation
- Algorithms
- Protocols
- Reuse
Execution constraints
- CPU speed
- power
- failure rates

Reaction constraints
- deadlines
- throughput
- jitter

Embedded Systems

Computation
- algorithms
- protocols
- reuse

Embedded System Design is generalized software design (e.g. RT Java).

Source: T. Henzinger
Execution constraints
- CPU speed
- power
- failure rates

Reaction constraints
- Deadlines
- throughput
- jitter

Computation
- Algorithms
- protocols
- reuse

Source: T. Henzinger
The CHESS Challenge

We need a new formal foundation for embedded systems, which systematically and even-handedly re-marries computation and physicality.

Source: T. Henzinger
Integration of the Two Cultures

Engineering
Component model: transfer function
Composition: parallel
Connection: data flow

Computer Science
Component model: subroutine
Composition: sequential
Connection: control flow

[Hybrid Systems; Ptolemy; Metropolis; Metamodels]
### Equational Models

**Strengths:**
- Concurrency
- Quantitative constraints
  (time, power, QoS)

**Tool support:**
- Best-effort design
- Optimization

### Abstract-Machine Models

**Strengths:**
- Dynamic change
- Complexity theory

**Analysis:**
- Worst-case analysis
- Constraint satisfaction

---

Engineers must understand both complexities and trade-offs.

Source: T. Henzinger
The Embedded Software SCIENCE Dilemma
Software Architecture Today

Poor common infrastructure. Weak specialization of functions. Poor resource management. Poor planning.
Software Architecture Tomorrow?