Are “Embedded Systems" Just Systems Made with Small Computers?

Invited Talk
Artist International Collaboration Days
Education Day
In conjunction with EMSOFT
Philadelphia, PA, Oct 11, 2003

Edward A. Lee
Professor
UC Berkeley

Chess

Chess: Center for Hybrid and Embedded Software Systems

Abstract

Occasionally I hear the argument that embedded computing is just computing with extreme resource limitations, where key resources are memory and CPU cycles. By this argument, embedded computing does not constitute a new discipline, since optimizing resource usage has always been a part of computer science.

Traditional EE systems theory, including signal processing and control, is often a good match for the application domains of embedded systems. Moreover, this theory has steadily been evolving towards software-based realizations, for example with the emergence of hybrid systems theory in the control systems community. Therefore it is arguable that embedded systems is just an evolutionary outgrowth of these disciplines, and again does not constitute a new discipline.

In this talk, I will argue that the traditional CS theory of computation and traditional EE systems theory both fail to effectively model embedded systems.

Traditional EE systems theory offers powerful analytical techniques for proving "correctness" of systems, for example by demonstrating that designs are stable. However, when these designs are realized in software, often the "correctness" proofs are no longer formally valid, and engineers have to resort to bench testing to validate behavior. Implementations of discrete time systems under an RTOS, or worse, under a non-real-time operating system, no longer have the formal structure assumed by the analytical tools. For example, how should an engineer choose priorities for tasks, or whether processes should be preempted by higher priority processes? How should an engineer assess the effect of asynchronous events or mode changes? The theory breaks down, and the engineer is stuck with guesswork.

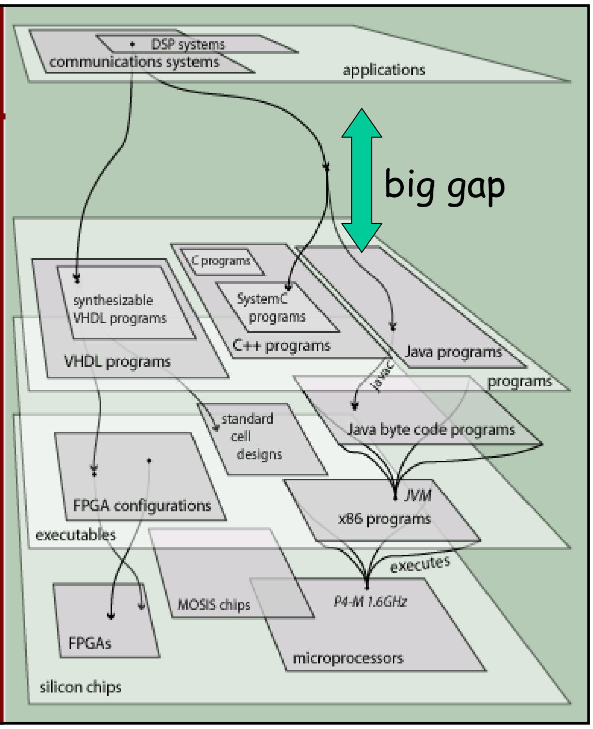
Unfortunately, the standard engineering curriculum cements this flaw. Courses in signals and systems, controls, and communications systems rely heavily on frequency-domain techniques, transforms, and linear systems theory. The beauty and richness of the subject, particularly compared to the relatively immature fields of hybrid systems analysis and embedded software design, seduces instructors to weave an ever more elaborate fiction. Real systems aren't like that, but this theory is so pretty, that we do it anyway.

In this talk, I will show how we can begin to adapt the traditional EE systems curriculum to embrace the real world of software. First, we have to show that even non-linear systems have formal structure. Next, the theory of hybrid systems shows how to leverage the theory of linear systems when the violations of the linear hypothesis are through mode changes. Finally, concurrent models of computation are available that enable the specification of software that is assured of meeting the assumptions of the formal framework. These cannot be based directly on the rather weak abstractions of threads, processes, priorities, preemption, and synchronization. They are based, instead, on synchronous and time-driven languages.

Platforms

A *platform* is a set of designs.

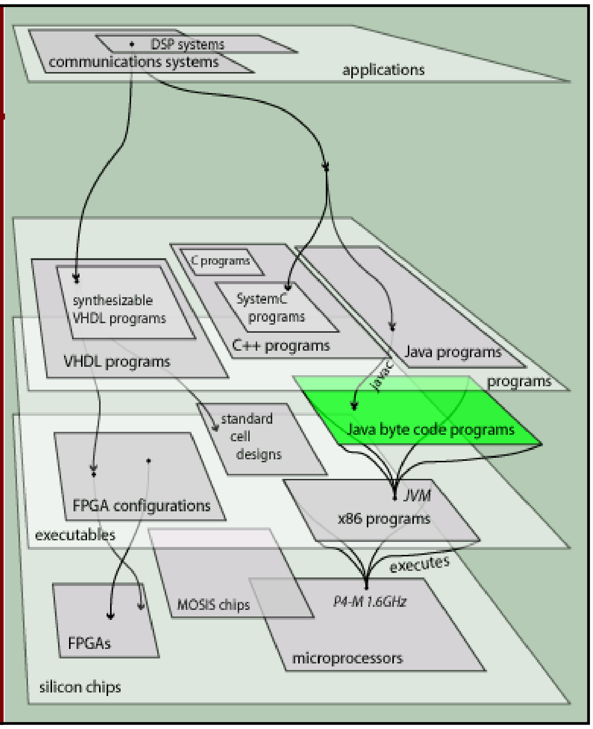
Relations between platforms represent design processes.



Progress

Many useful technical developments amounted to creation of new platforms.

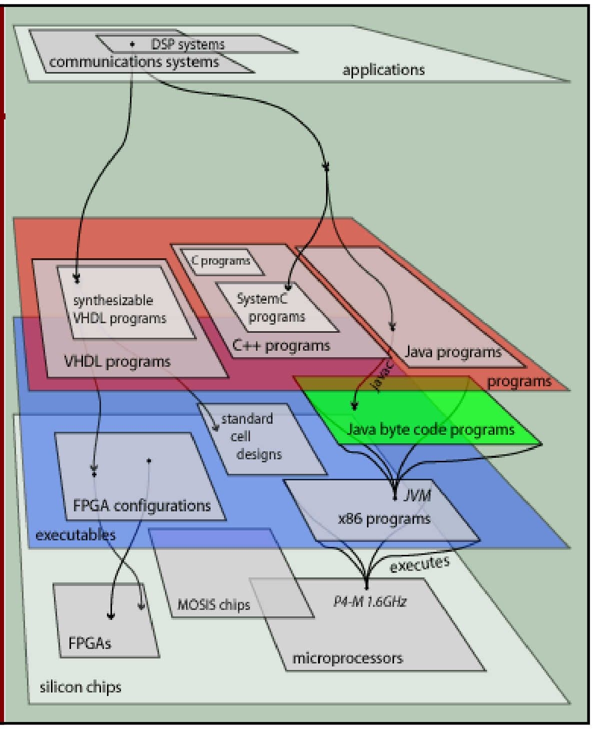
- microarchitectures
- operating systems
- virtual machines



Recent Action

Giving the red platforms useful modeling properties (e.g. verification, UML, MDA)

Getting from red platforms to blue platforms (e.g. correctness, efficiency)



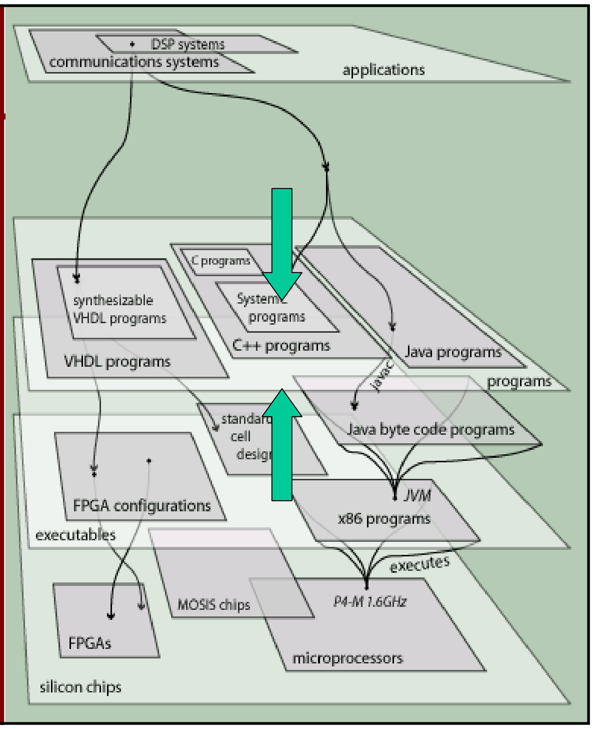
Desirable Properties

From above:

- modeling
- expressiveness

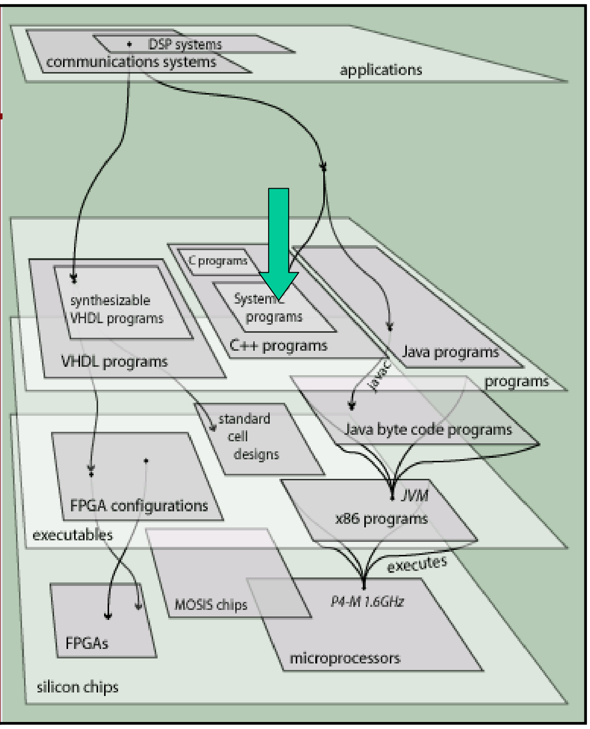
From below:

- correctness
- efficiency



Model-Based Design

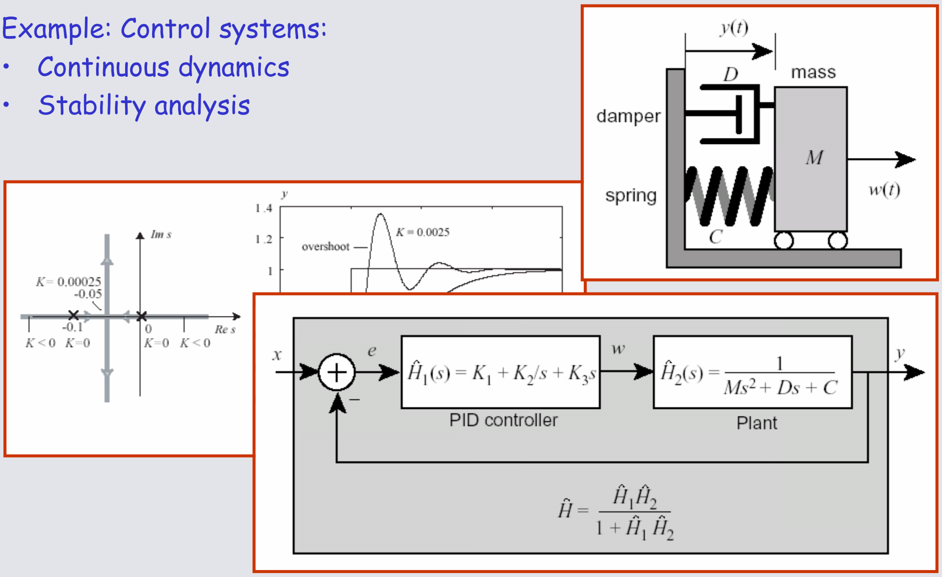
Model-based design is specification of designs in platforms with "useful modeling properties."



"Useful Modeling Properties" for Embedded Systems

Example: Control systems:

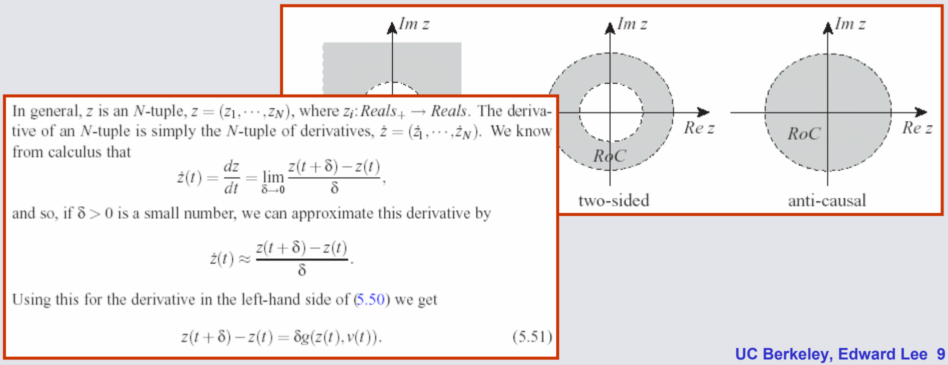
- Continuous dynamics
- Stability analysis


y(t)
D
mass
damper
M
spring
w(t)
C
Im s
K = 0.00025
-0.05
-0.1
0
Re s
K < 0
K = 0
K = 0
K < 0
y
1.4
1.2
1
overshoot
K = 0.0025
x
e
w
y
−
$\hat{H}_1(s) = K_1 + K_2/s + K_3 s$
PID controller
$\hat{H}_2(s) = \dfrac{1}{Ms^2 + Ds + C}$
Plant
$\hat{H} = \dfrac{\hat{H}_1 \hat{H}_2}{1 + \hat{H}_1 \hat{H}_2}$

Discretized Model
A Step Towards Software

- Numerical integration techniques provided sophisticated 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 not accurate for software controllers (fails on correctness)


Im z
Im z
Im z
Re z
Re z
RoC
RoC
two-sided
anti-causal

In general, z is an N-tuple, $z = (z_1, \cdots, z_N)$, where $z_i \colon Reals_+ \to Reals$. The derivative of an N-tuple is simply the N-tuple of derivatives, $\dot{z} = (\dot{z}_1, \cdots, \dot{z}_N)$. 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

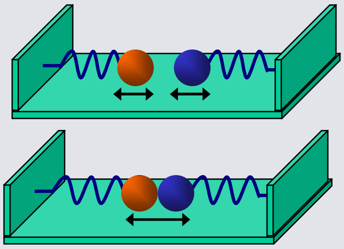
$$z(t+\delta) - z(t) = \delta g(z(t), v(t)). \qquad (5.51)$$

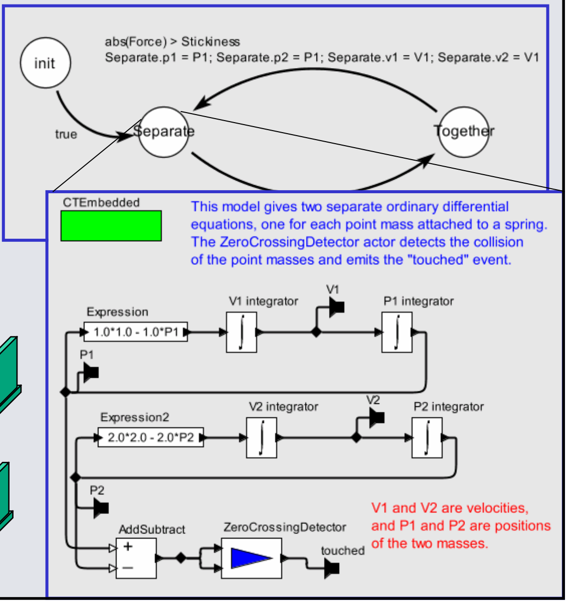
Hybrid Systems –
Union of Continuous & Discrete

Suffers from problems with

- modeling
- expressiveness
- correctness

E.g. Consider building a hybrid system model for software running under a multitasking real-time OS.





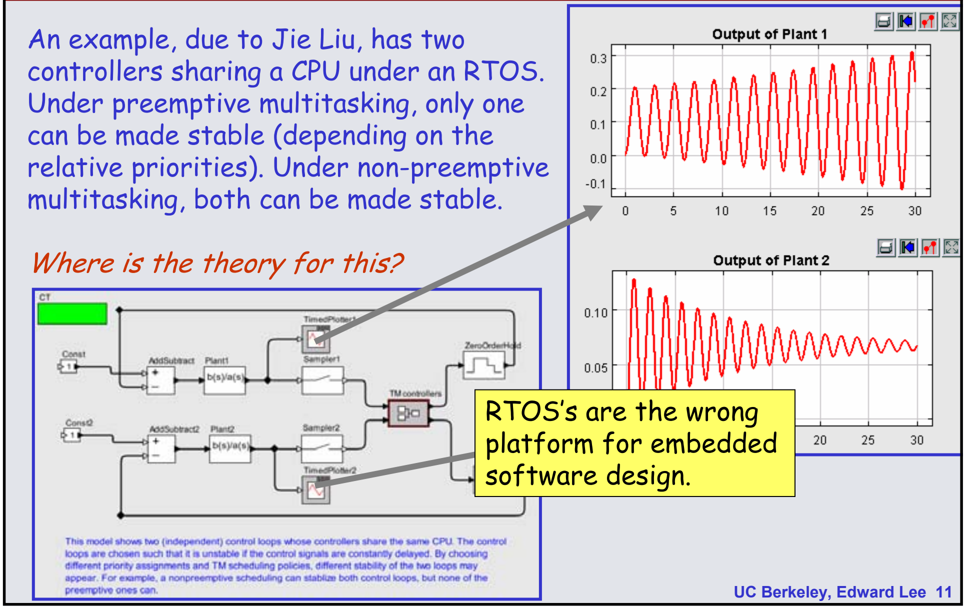
This model gives two separate ordinary differential equations, one for each point mass attached to a spring. The ZeroCrossingDetector actor detects the collision of the point masses and emits the "touched" event.

V1 and V2 are velocities, and P1 and P2 are positions of the two masses.

The Timing of Software is the Wrong Thing to Model

An example, due to Jie Liu, has 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?

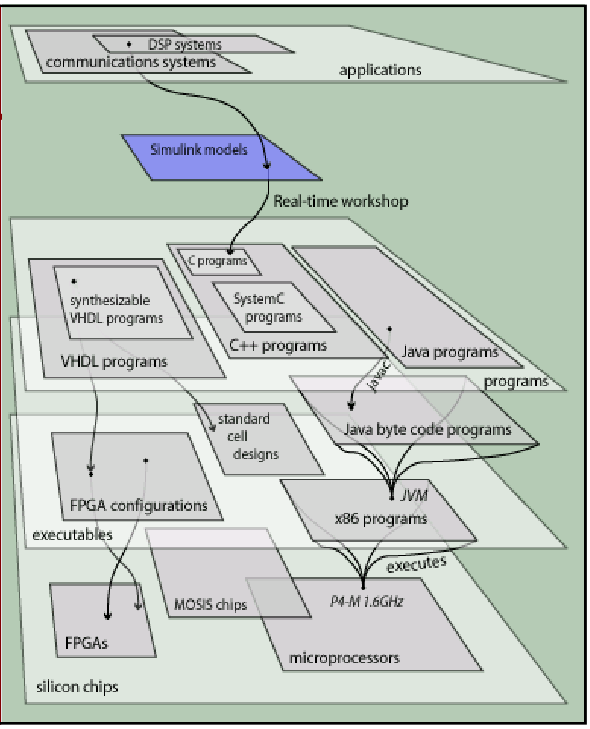


This model shows two (independent) control loops whose controllers share the same CPU. The control loops are chosen such that it is unstable if the control signals are constantly delayed. By choosing different priority assignments and TM scheduling policies, different stability of the two loops may appear.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

RTOS's are the wrong platform for embedded software design.

Better Platforms

Platforms with modeling properties that reflect requirements of the application, not accidental properties of the implementation.



How to View This Design

From above: Signal flow graph with linear, time-invariant components.

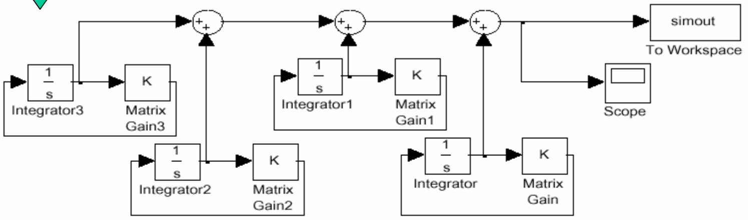
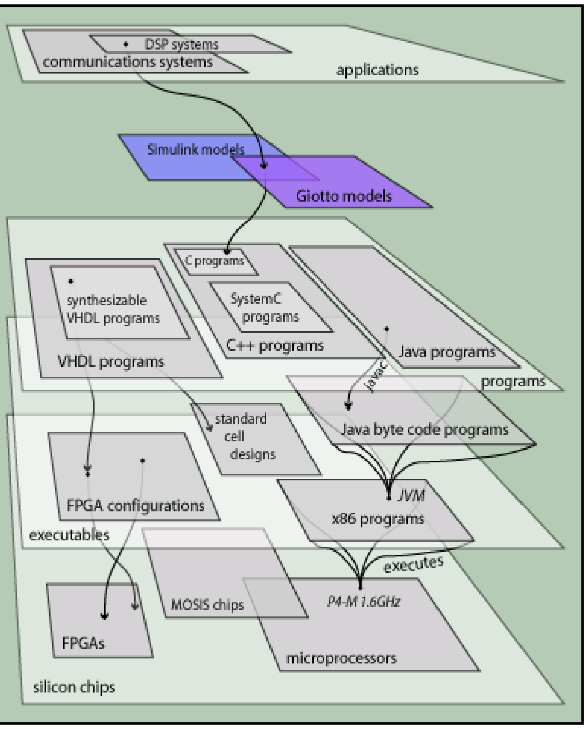


Figure C.12: A block diagram generating a plucked string sound with a fundmental and three harmonics.

From below: Synchronous concurrent composition of components

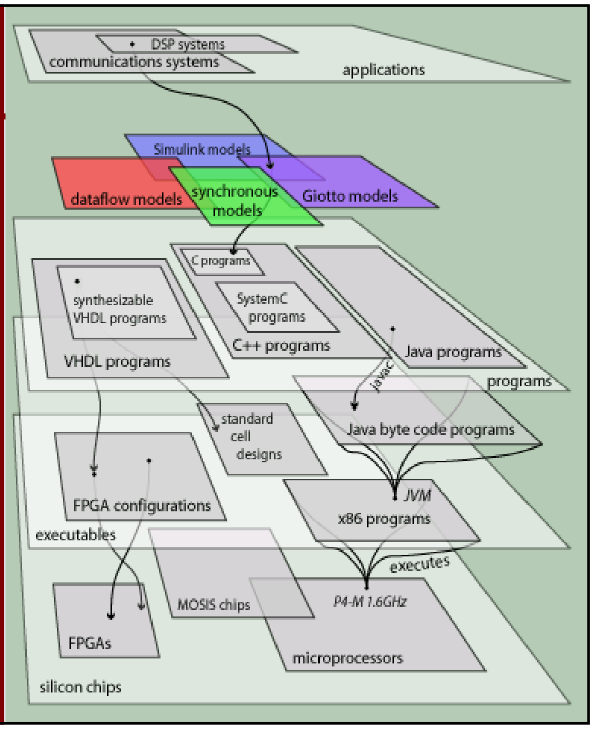
Embedded Platforms

The modeling properties of these platforms are about the application problem, not about the implementation technology.



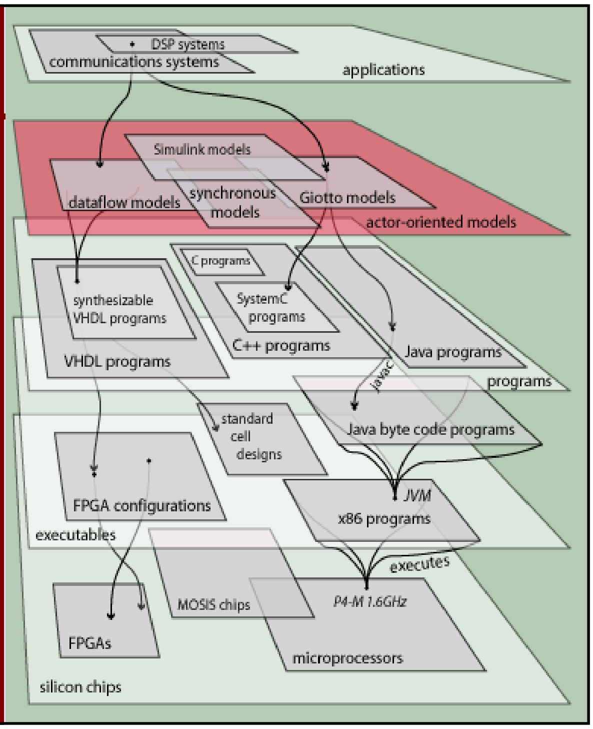
Embedded Platforms

The modeling properties of these platforms are about the application problem, not about the implementation technology.



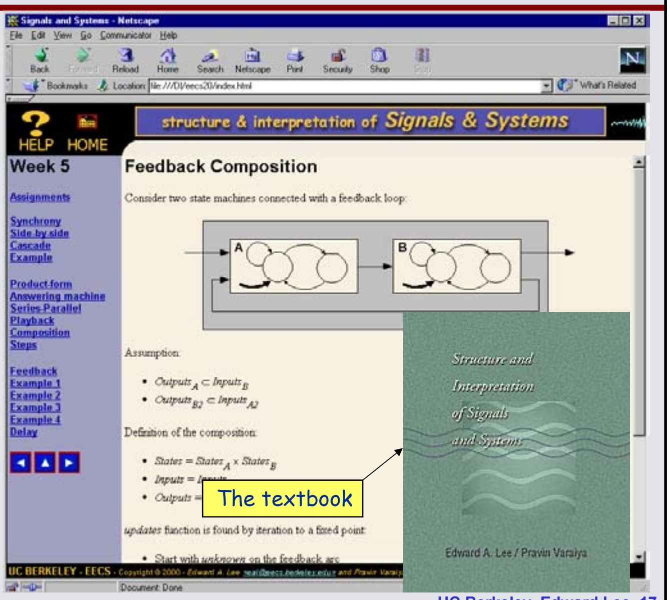
Embedded Platforms

Actor oriented models compose concurrent components according to a model of computation.



Education Changes
The Starting Point at Berkeley

Berkeley has a required sophomore course that addresses mathematical modeling of signals and systems from a computational perspective.


Signals and Systems - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: file:///D|/eecs20/index.html
What's Related
structure & interpretation of Signals & Systems
HELP HOME
Week 5
Assignments
Synchrony
Side by side
Cascade
Example
Product-form
Answering machine
Series-Parallel
Playback
Composition
Steps
Feedback
Example 1
Example 2
Example 3
Example 4
Delay
A
B
UC BERKELEY - EECS - Copyright © 2000 - Edward A. Lee

Feedback Composition

Consider two state machines connected with a feedback loop:

Assumption:

- $Outputs_A \subset Inputs_B$
- $Outputs_{B2} \subset Inputs_{A2}$

Definition of the composition:

- $States = States_A \times States_B$
- $Inputs = Inputs$
- $Outputs =$

updates function is found by iteration to a fixed point:

- Start with *unknown* on the feedback arc

Document: Done

Structure and Interpretation of Signals and Systems

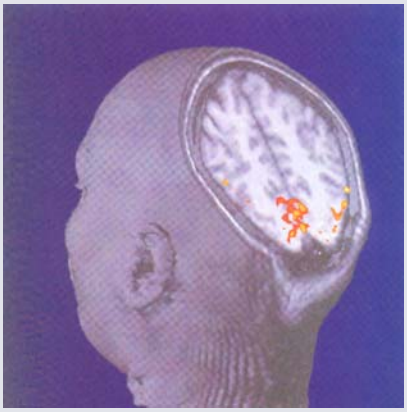
Edward A. Lee / Pravin Varaiya

The textbook

The web page at the right illustrates a broad view of feedback, where the behavior is a fixed point solution to a set of equations. This view covers both traditional continuous feedback and discrete-event systems.

Themes of the Berkeley Course

- The connection between *imperative* and *declarative* descriptions of signals and systems.
- The use of *sets and functions* as a universal language for declarative descriptions of signals and systems.
- Concurrent state machines and frequency domain analysis as complementary tools for designing and analyzing signals and systems.
- Early and often discussion of applications, with MATLAB and Simulink design for laboratory experimentation.

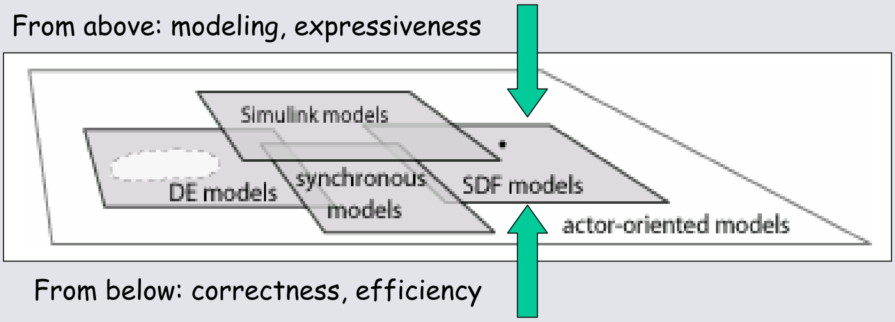


Brain response when seeing a discrete Fourier series.

Conclusion

- The distinction between modeling and design is narrowing
- We can teach system theory with a connection to computation

From above: modeling, expressiveness



From below: correctness, efficiency