An Overview of Embedded System Design Education at Berkeley

ALBERTO L. SANGIOVANNI-VINCENTELLI and ALESSANDRO PINTO
University of California, Berkeley

Embedded systems have been a traditional area of strength in the research agenda of the University of California at Berkeley. In parallel to this effort, a pattern of graduate and undergraduate classes has emerged that is the result of a distillation process of the research results. In this paper, we present the considerations that are driving our curriculum development and we review our undergraduate and graduate program. In particular, we describe in detail a graduate class (EECS249: Design of Embedded Systems: Modeling, Validation and Synthesis) that has been taught for six years. A common feature of our education agenda is the search for fundamentals of embedded system *science* rather than embedded system design techniques, an approach that today is rather unique.

Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Curriculum*

General Terms: Design, Standardization, Theory, Verification

Additional Key Words and Phrases: Graduate and undergraduate education, embedded systems, embedded software, functional design, architectural design, sourcework

1. INTRODUCTION

Embedded systems have been a strong research area for the University of California at Berkeley. We will briefly review this intense research activity as a preamble to present the Berkeley effort in embedded system education that is intimately related to the research program.

The research activities on embedded systems at Berkeley can be cast in a matrix organization (see Figure 1) where vertical research areas cover application domains, such as automotive, avionics, energy, and industrial control; horizontal areas cover enabling technologies such as Integrated Circuits, Sensors, Wireless Networks, Operating Systems, Embedded Software, Automatic Control, Design Methodologies, and Tools. The important aspect of our approach is

Authors' addresses: Department of EECS, University of California, Berkeley, Berkeley, California 94720; email: {alberto,apinto}@eecs.berkeley.edu.

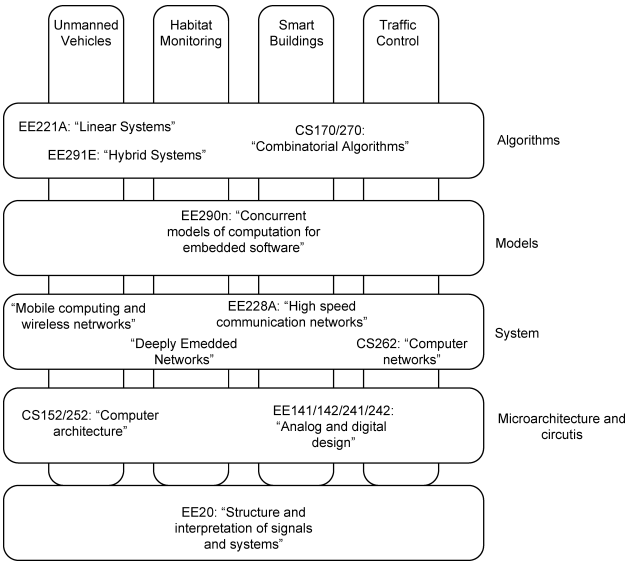

Fig. 1. Research activities on embedded systems at berkeley.

that the enabling technologies, are explicitly linked to the vertical application areas and are geared toward the embedded system domain.

At Berkeley, we have traditionally based our research programs on a strong interaction with industry and collaboration among faculty in different disciplines; embedded system research is no exception.

Among embedded system application domains, automotive has been an area of interest for many years. The PATH project [PATH] of CALTRANS (California Transportation Department) has been a test bed to develop new concepts in control of distributed systems, modeling, tools, and methodologies with a strong experimental part and an intense interaction with industry. The automotive emphasis of our design methodology work dates back to 1988 when a joint program on formal approaches to embedded controller design with Magneti Marelli for their Formula one robotized gear shift for Ferrari began. In the automotive domain, there has also been strong interdepartmental interaction between mechanical engineering and electrical engineering/computer science.

In U.S. universities, bottom-up aggregation of interests and approaches to education is more common than top-down planning. Hence, education initiatives in novel areas almost always begin with advanced graduate course offerings to migrate toward coordinated graduate programs and eventually into undergraduate courses. Thus, it is no wonder that course offering in Berkeley on embedded systems has been strong for years in the advanced course series (the EE and CS 290 series) that are related to faculty research activities. One such course has migrated to a regular offering in the graduate program—(*EECS249: Embedded System Design: Modeling, Analysis and Synthesis*)—the main topic of this paper.

The guiding principle in our teaching and research agenda related to embedded systems is to bring closer together system theory and computer science. The

two fields have drifted apart for years while we believe that the core of embedded systems intended as an engineering discipline lies in the marriage of the two approaches. While computer science traditionally deals with abstractions where the physical world has been carefully and artfully hidden to facilitate the development of application software, system theory deals with the physical foundations of engineering where quantities such as time, power, and size play a fundamental role in the models upon which this theory is based. The issue then is how to harmonize the physical view of systems with the abstractions that have been so useful in developing the CS intellectual agenda. We argue that a novel system theory is needed that, at the same, time is computational and physical. The basis of this theory cannot be but a set of novel abstractions that partially expose the physical reality to the higher levels and methods to manipulate the abstractions and link them in a coherent whole. The research community is indeed developing some of the necessary results to build this novel system theory. We believe it is time to inject these findings in the teaching infrastructure so that students can be exposed to this new way of thinking that should advance the state of embedded system design to a point where reliable and secure distributed systems can be designed quickly, inexpensively and error free.

The paper presents the guiding principles we have followed in our education effort and the set of courses offered that have direct relevance to the field of embedded system design. We list *only* the courses whose embedded system content is explicitly addressed. Otherwise, we may end up with a comprehensive list of all courses offered in engineering (except maybe a few) as today electronic system design is almost a synonym with embedded system design. In particular, we present first (Section 2) the graduate program: we zoom in on EECS249 and then we briefly review a set of advanced topical courses on embedded systems. In Section 3, we present an overview of our undergraduate program centered on a sophomore core course[1] (EECS20N [Lee 2000; Lee and Varaiya 2000]) that has been now offered over the past 5 years. This course for EE and CS students addresses mathematical modeling of signals and systems from a computational perspective. This reflects an effort of Berkeley faculty to set new foundations for the education in electrical engineering that is based on fundamentals rather than application domains. In this section, we also offer a view on our programs for the near future to address specifically embedded systems at the junior and senior level. In Section 4, we offer concluding remarks that could be of use for setting up a graduate or advanced undergraduate class in embedded system design in other institutions.

2. THE GRADUATE PROGRAM PART 1: EECS249 DESIGN OF EMBEDDED SYSTEMS: MODELS, VALIDATION, AND SYNTHESIS

This course [EE249] is part of the "regular" graduate program in EECS. It is taken by first-year graduate students as well as by senior graduate students in EECS and other departments such as mechanical, nuclear, and civil engineering.

[1]A core course is a required course for the educational programs offered by the department.

The idea of the course is to emphasize the commonality among the variety of application fields and to use a design methodology as the unified framework where the topics of the course are embedded. In this way, the variety of the advanced courses offered in our curriculum benefits from the foundations laid out by EECS249. The course is rather unique as it aims at bringing together the behavioral aspects of embedded system design with implementation within a rigorous as possible mathematical framework. Behavior capture, verification, and transformation are taught using the concepts pioneered by Edward Lee associated to models of computation. The implementation design is seen as a companion to the behavioral design as opposed to a more traditional academic view where implementation follows in a top-down fashion behavioral design. We adopt the view presented in Sangiovanni-Vincentelli 2002 and Sangiovanni-Vincentelli et al. 2004 to provide the intellectual background. In this methodology, the design proceeds by refinement steps where the mapping of behavior into "architecture" is the step to move to the next level of abstraction. Using this view, embedded software design is the process of mapping a particular behavior on a computing platform. By the same token, the choice of a particular distributed architecture due to geographical distribution or to performance optimization is handled in a unified way. The choice of components including reconfigurable and programmable parts is the result of architectural space exploration where cost functions and constraints guide the search.

From this brief overview, it should be clear that our motivation is to bring out the fundamental issues and the formalization that enables verification and synthesis at a level that would not be otherwise possible. This particular aspect should be seen as the quest for a new system science that serves as a framework to teach design that transcends the traditional discipline boundaries.

Given the large scope of the course, it has a heavy load; four contact hours and two lab hours per week. The contact hours are broken into traditional lectures and discussion of papers presented by the students. The verification of the learning process is left to weekly homework that are a mix of exercises and of theory, and to a final project that is fairly advanced, so much so that often the project reports see the light in the community as conference or journal papers.

The contents and the organization of the class has been the result of a number of advanced courses in hybrid systems and system level design that date back to 1991, when the first such class was taught.

2.1 Organization of the Class

The basic tenet of the methodology that forms the skeleton of the class is orthogonalization of concerns, and, in particular, separation of function and architecture, computation, and communication. This methodology, called platform-based design, is presented as a paradigm that incorporates these principles and spans the entire design process, from system-level specification to detailed implementation. We place particular emphasis on the analysis and optimization of the highest levels of abstraction of the design where all the important algorithmic and architectural decisions are taken.

The course has four main parts.

1. After a presentation of the motivation for the class extracted from a variety of examples in different industrial domains, we introduce the methodology followed (Platform-Based Design [Sangiovanni-Vincentelli 2002]) and examples of its applications.
2. The notion of behavior is analyzed and the role of nondeterminism in specification is explained. We present the basic models of computation that are needed to represent the behavior of most designs: Finite-State Machines, Synchronous Languages, Data Flow Networks, Petri Nets, Discrete Event Systems and Hybrid Systems. We outline the use of a unified framework to compare the different models of computation and we present the Tagged-Signal Model (TSM) [Lee and Sangiovanni-Vincentelli 1998] as a unifying theory to allow the composition of different models of computation to describe a complete system. We introduce here the Ptolemy [PtolemyII] and Metropolis [Balarin et al. 2003] environments for analysis and simulation of heterogeneous specifications.
3. We then introduce the notion of *architecture* as a set of components (interconnections for communication and building blocks that are capable of carrying out computation) that are providing *services* to the behavior that we wish to implement. *Optimal* architecture choice is presented as the selection of a particular set of computational blocks in a *library* of available components and of their interconnection. The evaluation of the quality of a selected architecture is based on a *mapping* process of behavior to the computation blocks, including programmable components. The mapping process should be accompanied by an estimate of the performance of the design once mapped onto the architecture. Communication representation is illustrated. The representation of architectures in the Metropolis and Mescal [Mescal] environments are presented.
4. Embedded software design is considered as part of the mapping process. In this context, we review the approaches followed to estimate physical quantities such as time and power for embedded software. We present scheduling as a way of mapping concurrent processes in a resource limited architecture and we address the characteristics of real-time operating systems. We review code generation techniques used in widely used industrial tools such as Real Time Workshop and Target Link. We present software synthesis approaches that exploit the mathematical properties of the design. We also address the problem of synthesizing communication structures given the system level requirements. The different coordination policies used at different levels of abstraction are introduced with particular emphasis on the desynchronization paradigm where a synchronous specification is mapped onto an asynchronous implementation that is guaranteed functionally equivalent to the specification.

Since we believe that design methods without applications are easily forgotten and not well assimilated, application examples are drawn from the multimedia and the automotive industrial sectors.

The class is complemented with discussion sessions and labs. Discussion is important to bring up questions on specific topics for which deep understanding of the general concepts is required. Labs are used to demonstrate how the theory can be applied to build tools and design flows and how the flows can be used to do real designs.

Students are required to work on a project from the beginning to the end of the class. A group of mentors follow the students who consequently are exposed to advanced research topics.

A detailed description of the course follows.

2.2 Week 1: Introduction and Methodology

The first two lectures serve as an introduction to the class and to the methodology upon which the class is structured. We present many examples of embedded systems: cars, airplanes, smart buildings, elevators, and sensor networks. All are real industrial applications so that the students are aware of the fact that embedded systems are not only an advanced research area but are a reality in the industrial world that is struggling with increasing complexity, reduced time to market, and increased competition.

In the introductory lecture, we highlight commonalities among all the examples that we present to set the stage for the philosophy of the course, which is aimed at defining the common methods that can be used across different application domains; the course is intended to solve "the embedded system design problem" rather than particular instances of it.

An entire lecture is devoted to an overview of the platform-based design principle. The method is justified by illustrating how it can be used to solve, or at least, to formalize, design problems that are common to the entire class of embedded systems.

2.3 Week 2–5: Function

Lectures. A *function* is defined as a denotational description of what a system is supposed to do without any implementation detail. Examples of functions from different application domains are shown to the students.

This part of the course presents several models of computation. For each model we present the computation, communication, and coordination semantics with a particular emphasis on the properties that are guaranteed and verifiable.

The first model that we introduce is finite state machines (FSM) that, in general, students are already somewhat familiar with. We also present an extension of FSMs to a globally asynchronous-locally synchronous (GALS) model, the so called Codesign FSM (or CFSM) that was the basis for Polis [Balarin et al. 1997] and VCC [Martin and Salefski 1998] because of its appeal to applications that have distributed nature.

We then present Kahn process networks (KPN) and data flow networks. A homework assignment is given to the students who have to explore FSM and KPN as candidates for describing control applications and multimedia or digital signal processing applications.

Table I. Class Syllabus

Course Section	Lectures	Discussions	Labs
Introduction	Definition of embedded systems, examples of applications, challenges, future applications	"The tides of EDA"	Introduction to the Metropolis meta-model language
Function	Finite state machines, Codesign finite state machines, Kahn process networks and data flow, Tagged signal model, Agent algebras, Petri nets, Synchronous languages and desynchronization.	StateCharts, Data flow with firings Desynchronization	Introduction to PtolemyII Building a model of computation in Metropolis, Esterel.
Architecture	Performances, Architecture modeling, Modeling concurrency: Scheduling, Interconnection architectures, Reconfigurable platforms, Programmable platforms, Fault tolerant architectures.	Formal modeling of processors, Rate monotonic hyperbolic bound, Interface synthesis	Modeling architectures in Metropolis Xilinx, PSoC
Mapping	Mapping specification, Design exploration, Software estimation, and synthesis, Static analysis, Quasi static scheduling.	Synthesis of software from CFSMs specification	Virtual component codesign Mathworks RTW, dSpace Target Link

After these two examples, we motivate the introduction of two frameworks for describing heterogeneous systems, comparing models of computation and understanding refinement and abstraction. Complex examples are shown where a system spans multiple application domains. We use the PicoRadio [PicoRadio] project as an example of embedded system that is distributed, has sensors and (possibly) actuators, has digital and analog components, a protocol stack, and a digital signal processing subsystem.

The two denotational frameworks we use in the class are: the tagged signal model for comparing models of computation and agent algebras [Passerone 2004] for explaining abstraction and refinement and giving a meaning to the communication between agents described in different models.

Petri nets [Murata 1989] are introduced using the frameworks and their use for functional modeling is analyzed. Then, we dedicate a lecture to introduce hybrid systems [Lygeros et al. 1999], i.e., a combination of continuous and discrete dynamics. We present the application of hybrid systems to model an automotive engine and to derive the appropriate control law.

TSM has been very well received by students that for the first time are exposed to the abstract concepts of signals as sets of events, pairs of values,

and tags. Building on the student intuition of sequence of events ordered by time stamps, we explain the more general concepts of partially ordered tags and express the semantics of a model of computation in terms of this simple set theoretic framework.

We then ventured in the presentation of the model of computation used in Metropolis: the Metropolis Meta-Model (MMM). The MMM can be considered an abstract semantics since it can accommodate a variety of models of computation that are obtained by refinement of this model. In some sense, an abstract semantic is "incomplete" and this incompleteness is the mechanism that allows the refinement into other models. We presented the additional constructs that are used in Metropolis to capture the specification of a design in a declarative style (a unique feature of Metropolis): the Language of Constraints (LOC) and a more conventional language for logical constraint specification, LTL [Pnueli 1977]. We also presented the Ptolemy actor-oriented semantics and showed how this is another style for abstract semantics.

Discussions. Classic papers are presented by students during the discussion sessions. The first one is a comprehensive introduction to the StateCharts syntax and semantics [Harel 1987] while the second is a paper on the properties of data flow networks [Lee and Messerschmitt 1987] where firing rules can be specified for each actor.

Discussions regard the meaning of a model instance and the properties that a model guarantees. For example, StateCharts is a graphical syntax with very few restrictions and students have to reason about the meaning of a feedback connection. In the case of data flow networks, properties like determinacy of a model are discussed. Students reason on the implication of this properties on the design steps. A discussion on heterogeneous models of computation follows.

Labs. Two environments for heterogeneous modeling are presented: Ptolemy and Metropolis. After introducing the environments, students are guided in modeling applications using models of computation already available as libraries in these frameworks.

A lab is dedicated to the development of a model of computation library in Metropolis. Students learn how the denotational definition of models can be implemented with programs. They also learn to separate the computation, communication, and coordination aspects. Using the MMM language, students are required to build a library for modeling data flow networks. They have to model a FIFO channel and an empty process, which can be extended by adding the description of the process behavior.

2.4 Week 6–8: Architecture

Lectures. We present formal models for describing architectures. We first give some examples of architectures and how they are used by system designers to implement functions. In this introductory lecture the concept of mapping a function to an architecture is presented as assignment of components of functions to processing elements. This exposes the students to the problem of matching two different concurrency models: the model of computation used for functional description and the amount of concurrency available in the architecture

that is often limited due to resource limitation. We introduce scheduling in the first lecture as a way of "impedance" matching between funtion concurrency and architecture concurrency.

We emphasize the communication aspect as one of the most important in architecture development. We introduce *communication-based design* as a design paradigm where computation is abstracted away and communication becomes the main objective of the design process. The methodology is motivated by two facts. First, embedded applications are usually distributed on networks of processing elements and it is important to formalize the definition of protocols and scheduling of communication resources. Second, SoC platforms are assembled from a library of available IPs meaning that the concern of a designer is not the optimization of each core, but rather their interconnection. We present a formal definition of the communication problem using the tagged signal model framework that explains the communication phenomena as a restriction of the behaviors of the connected processes.

Discussions. While the majority of discussion sessions in the first part of the course are theoretical digressions on models of computation, discussions on architectures are more practical and deal with specialized solution to distributed systems design, scheduling policies, on-chip networking, programmable platforms, and languages for architectural description.

The first set of papers show how processors can be modeled and how their performances can be abstracted. The Language for Instruction Set Architectures (LISA) [Zivojnovic et al. 1996] is first presented to give the students some basic intuitions of the main features that a microprocessor model should have. LISATek is also shown as the industrial set of tools that can simulate a description of a processor and generate the compilation tool chain in an automatic way. A modeling approach using Petri Nets is then presented, which gives an intuition of how models of computation can be used for modeling architectural components.

The second set of papers focuses on communication architectures. We discuss standard communication architectures, like busses and crossbars, and show the problems that are encountered when using these approaches in current systems on chip. The students also present a set of papers on networks on chip that has been proposed as a solution to such problems. We then deal with specialized protocols for on chip communication, like the latency insensitive protocol [Carloni 2004].

Labs. During lab sessions, students are guided toward developing models for different parts of an architecture: computation and communication components.

We first show the Liberty Simulation Environment [Liberty; Vachharajani et al. 2004] and how it can be used to model behavior, performances, and costs of a microprocessor.

We then teach students to develop models of architecture components using the Metropolis environment. During this lab, students are divided in groups and each group develops a simple model of an assigned IP. Those models are then collaboratively integrated into a model of a complete architecture.

Finally, we show a practical example of an architecture platform. We recently introduced the Xilinx VirtexII Pro as an example of heterogeneous platform that can be used for fast prototyping. The purpose of the lab is to introduce a platform that has hardware, software, and communication components, which can be composed in many ways. It is also the platform that will be used to show how an implementation can be automatically derived as a successive refinement of the original specification using suitable synthesis tools.

2.5 Week 9–11: Mapping

Lectures. Mapping functions to architectures is possibly the most relevant aspect of the platform-based design methodology taught in this class. The power of the MMM is evident here where the use of the same modeling constructs for function and architecture allows a particularly efficient way of performing mapping and analyzing the quality of the result. In particular, we first show how to put a function in relation to the supporting architecture. By using the MMM notion of events, we show how the function netlist can be placed in correspondence with the architecture netlist by introducing the so-called mapping netlist. In this way, events in the functional netlist trigger events in the architecture netlist via the mapping netlist. We show how the mechanism can be exploited to change the mapping of function to architecture elements in a straightforward manner that does not require rewriting the architectural and functional description. We present the scheduling problem as an essential part of the allocation of functionality to shared resources. In this respect, we review the fundamental results of the scheduling literature.

We then show how mapped functions can be simulated and how the performance of the mapping can be extracted. At this point, we introduce the notion of quantity managers as tools that compute quantities, such as power and time, used by architectural components when executing the part of the functionality mapped onto them. This mechanism allows evaluation of the quality of the mapping and provides important information about bottlenecks in the design. The feedback provided to the designer can be used to modify the mapping by changing assignments of functionality to architectural elements to modify architectures by adding or changing elements and to modify functionality by repartitioning or eliminating components.

Finally, we present mechanisms to feed quantity managers information about the basic execution "costs" (e.g., power and time) and we show examples drawn from Xilinx programmable platforms [Xilinx] and from other platforms, such as the Intel MXP5800.

Discussions. A crucial aspect of the mapping process is the scheduling of functional process on shared resources. The first paper is the classical paper on scheduling by Liu and Layland [Liu and Layland 1973]. Students discuss the complexity of the scheduling problem and the implication of the assumptions that are made in this paper. On the same topic, students present more recent results on real-time scheduling and, in particular, an hyperbolic bound on the processor utilization.

The following set of papers deals with recent approaches of mapping an application on a platform of processing elements interconnected by a regular network. This discussion focuses on the definition of an objective function to optimize during the mapping process. The cost of the communication architecture is the power it consumes and an optimal mapping is computed as the solution of an optimization problem subject to communication constraints.

Discussions are used to give practical examples of how the matching of performance abstraction and constraints propagation can be cast into optimization problems.

Labs. The two tools presented in the lab sessions are the Virtual Component Co-design (VCC) from Cadence and Metropolis. In the approach followed by VCC, mapping is the assignment of pieces of functions to architectural components. In the Metropolis framework, mapping is implemented as intersection of the function and architecture execution in a common domain, which serves as common refinement.

2.6 Weeks 12–14: Verification and Synthesis

Lectures. We review the notions of verification and synthesis and present how verification is not a synonym of simulation but contains static analysis tools as well as formal verification. In particular, we discuss the notion of successive refinement as the process used in the PBD methodology to go from specification to final implementation. We demonstrate the use of the MMM to maintain the same environment of both the more abstract and the more concrete representation to simplify the use of refinement verification techniques. We also show the simulation approach followed in the Metropolis environment to reduce or even eliminate the overhead that comes with the flexibility of maintaining both architecture and functionality present as separate entities of the design.

Then, we focus on the methods available in literature for software estimation, an important component of any verification methodology that mixes hardware and software implementations. The approach by Malik [Li and Malik 1995] is first introduced for static analysis of programs based on pipeline and cache modeling and integer linear programming followed by the abstract interpretation work of Wilhelm that yielded the well-known analysis program AbsInt [Ferdinand and Wilhelm 1998].

We then move to the software synthesis problem and present the model-based design approach where code is automatically generated from mathematical models. After reviewing shortly Real Time Workshop of MathWorks, we discuss a different way of generating code from models that follows the same paradigm used in hardware synthesis of optimizing the original description (software representation) before mapping it onto a given execution platform. In this case, we show that we have a "technology-independent" phase followed by technology mapping. We show how Esterel [Berry and Gonthier 1992] is compiled using this idea and using FSM optimization techniques based on MIS [Brayton et al. 1987] originally developed at Berkeley for logic synthesis to generate implementation code. We then present another method to optimize the original description based on the Ordered Binary Decision Diagram [Bryant

1986] representation of programs. We show how to use the variable ordering methods developed to manipulate OBDDs in logic synthesis to generate efficient programs from Co-Design Finite State Machines [Balarin et al. 1999].

We also present evaluation techniques to compute the time taken by synthesized programs to execute on a given platform that are used to guide the optimization search. These techniques are shown to be accurate when the software is automatically synthesized. We then move to the problem of optimizing code for data flow dominated applications with limited data-dependent control. We show also how operating system features such as hardware-software communication mechanisms and scheduling policies can be synthesized from requirements [Balarin et al. 1997; Giotto] and we point to the evolution of implementation platforms that can make the optimized "compilation" problem increasingly difficult.

Discussions. Most of the papers that we discuss in this part of the course are about synthesis. Students are exposed to recent approaches to software synthesis and the difference between synthesis and compilation. We start with a discussion on the Polis approach to software synthesis emphasizing why it is suitable for control applications and not for signal processing. In the discussion, we emphasize what are the implications of having a formal model (CFSM) of computation on the solution of the synthesis problem.

Then we move from the control domain to the synthesis of software for digital signal processing applications. The synthesis algorithm starts from a dataflow model and generates code that can be optimized for a specific processor. We highlight the difference between control and digital signal processing applications: the description that we start from has different assumptions (different models of computations) and also the target platforms are different.

Labs. We present two industrial tools for automatic code generation: the real-time workshop (RTW) [Mathworks] by Mathworks and Targetlink [dSpace] by dSPACE.

One lab presents the xGiotto [Giotto] approach to software synthesis and verification. xGiotto starts from the description of a program in a language with a formal semantics and is able to perform analysis like schedulability and race condition detection. Once a program has been verified, code can be automatically generated for a target platform.

2.7 Week 15: Applications and Summary

The last week of the course is dedicated to placing the material presented in perspective. An application such as automotive control is used to show the complete flow from modeling the functionality with hybrid systems to mapping onto execution platforms that are modified according to the results of the analysis. We had in mind to use Metropolis and x-Giotto as well as the Xilinx back-end to show a complete design flow in action. However, the tools were not mature at the time we taught the course and the flow could not be demonstrated. Next year we are confident we will be able to show the entire flow to students and have them play with it in more than one application. We find this summary

particularly useful as the course is loaded with material and it is understandable that the students may loose track of the overall vision that we try to convey. The feedback given during the last few years has been consistently good and encouraging.

Another application space that we explored is wireless sensor networks where our view of design leads naturally to the definition of layers of abstraction that identify clearly the need of a "middleware" that can well capture the performance of a particular physical implementation of the network to offer the application programmer an abstraction that enables reuse across different implementations with appropriate abstract analysis of the effects of the physical network on the application program.

After the end of the course, the projects that are used to evaluate the students, are presented in front of the entire class to open a wide range discussion among students, teaching assistant, mentors, and faculty on the results and the ideas on how to improve the course.

2.8 Projects

The course is graded on the basis of a set of homework and on a final project. After the first week of class, a list of project proposals is given to the students. We rely on a group of highly qualified mentors from industry and academia that help the students in reviewing previous work and conducting the research to complete their projects on time. We push students to start as early as possible and we motivate them by mentioning the possibility of submitting for the best reports for publication.

Table II gives an idea of the projects that have been carried out since 1998. Following the course organization, the table is divided in projects that are related to functional description, architectural description, and mapping. We also added two columns for projects that cross all of the three main sections of the course: case studies and design methodologies.

The choices of students are often concentrated on the definition of formal models for describing a function and on the development of algorithms for verification and synthesis. The case studies were mainly developed by students coming from departments other than EECS that are interested in studying how the methodology can be applied to solve specific design problems. Given the shortness of available time, it is not possible, in general, for students to develop and show the effectiveness of a complete design methodology for a specific application domain. This is the reason why these kind of projects are rarely taken even though a couple were chosen and had worthwhile results.

Few projects were given on architecture. This situation reflects the status of our research in the field that only recently has taken an important turn, determined by the extensive introduction of the use of the MMM and of the characterization work carried out in conjunction with the Xilinx project. We expect that more projects on this subject will be proposed in the future, particularly in the area of syntax and semantics of languages for architectural description and automatic verification and refinement tools for architectures.

Table II. Examples of Projects

Function	Architecture	Mapping and tools	Case Studies	Design Methodologies
—Time-Based Communication Refinement for CFSMs	—Reconfigurable architecture exploration [Baleani et al. 2002]	—Hybrid Systems Verification	—Efficient low-power network discovery algorithm for wireless embedded sensor networks [KooHenzinger et al. 2003]	—Synchronous platform based design of unmanned aerial vehicle [Horowitz et al. 2003]
—Heterochronous dataflow domain in Ptolemy II	—Modeling and refinement of a simple microprocessor using YAPI	—Protocol Converter Synthesis	—Pupil detection and tracking system [Zimet et al. 2004]	
—Hardware extension to ECL		—Task response time optimization using cost-based operation motion	—Multi-injection driver specification for engine management	
—Extending CFSMs to allow multirate operations		—Communication architecture synthesis [Pinto et al. 2002]		
		—Microcode compression for embedded processors		
		—Real-time distributed fault tolerant scheduling: timing analysis		
		—Metropolis SystemC simulator		
		—Hybrid systems simulation in PtolemyII		

As the citations in Table II show, a fair amount of projects led to publications in important journals and conference proceedings.

3. THE GRADUATE PROGRAM PART 2: ADVANCED GRADUATE COURSES

Advanced courses are an important feature of the Berkeley graduate program. These courses reflect very recent advances in the state of the art of a particular knowledge domain. They are topical courses; their content changes from year to

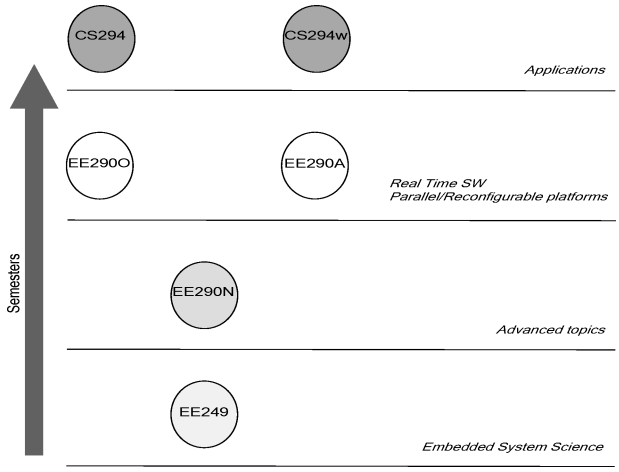

Fig. 2. A graduate program in embedded systems.

year and can be taken by students multiple times. In general, they are taken by PhD students who are interested in research topics in the area covered by the course. Regular graduate courses are often derived from the advanced series after the content has stabilized and there is enough interest from the student population. Since embedded systems are so important in our research agenda, there are several advanced courses that have a direct relationship with the topics of this paper. These courses are labeled EE290 and CS294 followed by a letter indicating the area these courses belong to.

In Figure 2, we show a the backbone of a graduate program in embedded systems that traverses the courses presented in this paper. A well thought out course program in embedded systems should include domain-specific courses that provide the reference framework for the students to be productive in the outside world.

3.1 Computer Science Courses

The Computer Science Division of our Department offers a graduate-level class on embedded network and mobile computing platforms. The course is "CS294-1." As is always the case for advanced graduate classes that belong either to the EE290 series or the CS294 series, its content changes every semester.

In the last five years, this course has always been centered around applications that are embedded in the environment with which they interact.

Deeply Embedded Network Systems. This advanced class focuses on ubiquitous computing [CS294]. It is based on the experience of researchers from different universities on wireless sensor networks.

The schedule of the class starts with an introduction to the emerging computing platforms composed of a large number of simple nodes communicating on wireless channels. Habitat-monitoring applications are taken as representative of embedded networks of nodes that can sense and communicate. The design of the embedded network is driven by the application that sets the requirements

to satisfy. After the introduction, 1 week is dedicated to the presentation of several platforms and operating systems that are currently available.

The following 4 weeks of the class give an overview of all the proposed protocols for wireless sensor networks: network discovery, topology information, aggregation, broadcast, and routing. The design of all these protocols takes into account the constraints imposed by the application. The class puts a lot of emphasis on power consumption, since nodes cannot be replaced.

The second part of the class gives directions for the implementation and deployment of these networks. Two weeks are devoted to the problems arising from the distributed nature of the applications. In particular, distributed storage of information and distributed control are presented as important research areas. Finally, privacy is considered as a potential problem, since embedded networks have the capability of monitoring every object in the environment in which they are embedded.

Students in the class are divided into groups and each of them works on a project. Possible topics range from programming models and simulations of large-scale networks to new protocols for routing and localization. In Fall 2003, a considerable number of projects investigated the problem of programming the network starting from the description of the application.

Mobile Computing and Wireless Networking. The level of abstraction of the networks considered in this class is much higher than the one considered in the previous section. The focus is on new trends in mobile computing and integration of heterogeneous networks [CS294w].

The class presents challenges in mobile computing where the end user is a person that uses a device to be constantly connected to the rest of the world. The requirements on the protocol implementation are derived by looking at the issues that mobility brings up: routing, network registration, and security. Some protocols that solve all these problems are presented.

A network node is an handheld device that presents severe limitations in power consumption. This problem, which is presented as an important constraint, presents some commonality with wireless sensor networks of the deeply embedded networks class, but it is not the only one. Connectivity and distributed information storage are also investigated and some solutions are presented.

Finally, some common platforms for these kind of systems, like WLAN, UMTS, GSM and GPRS, are presented.

3.2 Advanced Electrical Engineering Courses

The electrical engineering division of the EECS Department offers advanced courses that are focused on formal models for system level design and embedded software. Two classes are particularly relevant to our discussion on embedded systems.

EE290N: Concurrent Models of Computation for Embedded Software. This advanced class focuses on concurrent models of computation for embedded software development [EE290N]. It can be seen as the extension to and deep

analysis of the first 5 weeks of EECS249. It has been taught four different times with contents that are converging to a unified view so that there is a plan of making it a regular graduate course. Abstract semantics, concrete semantics, and implementation of some of the most commonly used models are presented. Besides homework assignments, students are required to work on a project.

The class is organized as follows:

Week 1–2. The first two lectures present the main differences between embedded software and software for standard applications. In particular, threads are formally defined and their limitations are underlined, with particular emphasis on the problems that arise when using this technique for developing concurrent programs. Language for particular applications, like NesC [Gay et al. 2003] and Lustre [Caspi et al. 1987], are then taken as representative of domain specific models for embedded software.

Week 3–5. In this part of the class, Process Networks (PN) are used to introduce a formalism that will be used in the rest of the class. First, the abstract semantics of PN is presented together with the notion of partial ordering and prefix ordering on sequences of events. In this abstract settings, properties like monotonicity, continuity, and determinacy of a process are described as properties of the input–output function on streams that characterize a process. The fixed-point semantics is also introduced when multiple processes are connected together and loops are present in the corresponding functional graph. A concrete semantics is then presented and, finally, the concrete implementation of PN semantics, following the Kahn-McQueen execution semantics, is shown by using PtolemyII as a platform for implementing and composing models of computation. The problem of bounded execution (an execution that used a bounded amount of memory) is introduced and simulation techniques are presented.

Week 6. This week is devoted to synchronous languages. After describing their semantics, the problem of connecting processes in a feedback configuration is considered. Complete partial orders are explained to define the least fixed-point semantics of synchronous programs. Simulation and implementation issues are presented using the PtolemyII synchronous reactive-model implementation.

Week 7. The fixed-point semantics is presented by defining a metric on the set of signals. The first lecture introduces the notion of tags and events as defined in the tagged-signal model. Different kinds of metrics and metric spaces are defined. The Cantor metric and the concept of ultrametric are introduced as ways of measuring the distance between two signals. Finally, a fixed-point theorem is presented that is used to define the semantics of the composition of processes.

Week 8–10. Three weeks are devoted to the different varieties of data flow models of computation. The first kind of data flow is called Statically Schedulable Data Flow (originally defined by Edward Lee in his thesis as Synchronous Data Flow and more recently referred to as Static Data Flow). The concept of

balance equation and scheduling of static data flow models is explained. A more concrete execution semantics is explained as a sequence of firings of data flow actors. Firing rules and fixed-point computation are presented. The second part of this section presents extensions to the basic data flow model like multidimensional and heterochronous data flow. The last part presents boolean data flow networks to express data dependent flow of tokens.

Week 11. The last part of the class introduces continuous time models and hybrid systems. The emphasis is on the semantics and the techniques that are used to solve systems of differential equations. In particular, problems like event detection and Zeno behaviors for hybrid systems are considered and the impact on the simulation engine are explained.

EE290O: Embedded Software Engineering. While the previous class explores the models that should be used in developing embedded software, this class focuses on a particular design flow. The class presents a model of computation, the Giotto model [Henzinger et al. 2003], and explains why it is suitable for a class of embedded software [EE290O]. The class is divided into three parts:

1. RTOS Concepts. A real-time operating system is characterized by the services that it provides: environment communication services, event triggering services, software communication services, and software scheduling. Tasks and threads are defined and a model for them is explained. A simple example of an RTOS is given. In their first homework, students have to implement an RTOS on the LEGO brick. The students now have a feeling of the RTOS abstraction level and the problems in modeling software at this level. The E-machine [Henzinger and Kirsch 2002] is then described and its properties are formally explained: semantics of the E-machine, portability and predictability, deterministic behavior, and logical execution time.
2. RTOS Scheduling. Some classic scheduling algorithms are presented. First, a task is modeled with execution time and deadline and the concept of preemption is explained. First, the early deadline and then rate monotonic scheduling are explained. The last part of this section is devoted to schedulability tests like rate monotonic analysis (RTA) and model-based schedulability analysis (where tasks and schedulers are modeled as timed automata).
3. RT Communication. The last part of the class deals with real-time communication. Messages are modeled with deadline and worstcase latency. Two protocols are presented in detail: Controller Area Network (CAN) and Time-Triggered Protocol (TTP) [Kopetz and Grunsteidl 1994]. The problem of fault tolerance is introduced and the solution proposed by the TTP protocol is explained.

EE290a: Concurrent System Architectures for Applications and Implementations. This experimental class was offered in the Spring 2002. The focus of the class is on models for concurrent applications, heterogeneous platforms, and the mapping of the former to the latter [EE290A]. This course can be considered as

a follow-on to EECS249 with respect to Architecture and Mapping. The course is organized as follows:

Week 1–3. The first part of the class introduces the content of the course by giving examples of applications and platforms. Several models of computation, like finite state machines, process networks, data flow, synchronous/reactive, communicating sequential processes, and codesign finite state machines are introduced, emphasizing the fact that each model is particularly suitable for a specific application domain. The Click [Shah et al. 2004] model of computation is explained for modeling the processing of streams of packets in routers.

Week 4. During this week a set of representative applications are presented. MPEG decoding is presented together with an entire flow from specification (using a flavor of Kahn process networks called YAPI [de Kock et al. 2000]) to implementation. This example shows how the requirements of an application condition, the selection of a model of computation, and the underling implementation platform.

Week 5–7. The motivations for using programmable platforms are explained. Several platforms are then presented: the Nexperia [Nexperia] platform by Philips for multimedia application and the Ixp1200 [IXP1200] platform by Intel for network processing. For each platform, examples of what kind of application they target are given together with performances result. In week 7, a broad overview of the available platforms for VoIp is given.

Week 8–10. This part of the class gives two examples of mapping the concurrency of the application onto the concurrency of the platform. The Giotto model of computation and the E-Machine are presented and an example of the software running on an autonomous helicopter is shown. A guest lecture from the Xilinx research group shows the implementation of an IP router on a VirtexII Pro FPGA and the implication of using a multithreaded implementation. The last presentation is given by the SCORE (Stream Computations Organized for Reconfigurable Execution) [Caspi et al. 2000] project team. In this project both function and architecture are described using Kahn process networks. The challenge is to find a schedule of processes for bounded execution. The architecture is composed of a set of processors that communicate over a shared network. Buffers are allocated on memory segments.

Week 11–13. The last lectures give an overview of multiprocessors platforms like RAW and IWarp [Borkar et al. 1988] and other programmable platforms like PSOC by Cypress and the Extensa Processor.

4. THE GRADUATE PROGRAM PART 3: CIVIL AND MECHANICAL ENGINEERING

These two departments have traditionally been interested in embedded applications. They have some graduate courses where embedded topics are featured.

4.1 Mechanical Engineering 230: "Real-Time Software for Mechanical System Control"

The Mechanical Engineering Department offers a class that teaches students how to control mechanical systems using embedded software. It is a lab-oriented class in the sense that students are taught how to implement a controller in Java on an embedded processor.

Even if methodology is not really the focus of this class, controlling a mechanical system implies understanding the continuous dynamics governing it and the constraints that are imposed on the reaction time of the software controller. The class gives an overview of the real-time control problems. Students are exposed to Java as a technology that allows the development of real-time systems and how complicated control algorithms can be implemented on an embedded processor. This part takes one third of the entire class. This technology is applied to the control of motors.

Students learn real-time programming through a set of labs. The first two labs are meant to teach students how to write a control algorithm for an embedded processor. The third lab show how the software world interfaces with a physical component like a motor. In the following lab sessions, students are guided in implementing a feedback control algorithm for a motor.

4.2 Civil and Mechanical Engineering 290I: "Civil Systems, Control and Information Management"

The possibility of sensing the environment and communicating over a dense network of tiny objects is of great interest for the civil engineering community. This course starts with an introductory lecture that motivates the use of embedded networks with several applications: automatic control of the BART (Bay Area Rapid Transportation) system, earthquakes monitoring, and mesh stable formation flight of unmanned air vehicles.

The emphasis of the class is on the formal specification of complex networked systems. The Teja [Teja] environment is used as example of formal language to specify systems of users and resources. Syntax and semantics of the language are explained in the class and students are trained in using the environment with labs and homework.

5. THE UNDERGRADUATE PROGRAM

Our approach to embedded systems has been to marry the physical and the computational world to teach students how to reason critically about modeling and abstractions. Traditionally undergraduate EE courses have focused on continuous time and detailed modeling of the physical phenomena using partial and ordinary differential equations, while undergraduate CS courses have focused on discrete representations and computational abstractions. We noted that students then were "boxed" in this dichotomy and had problems linking the two worlds. The intellectual agenda was to teach students how to reason about the meaning of mathematical models, their limitations, and power. EECS20N (Structure and Interpretation of Signals and Systems) was born with this idea in mind.

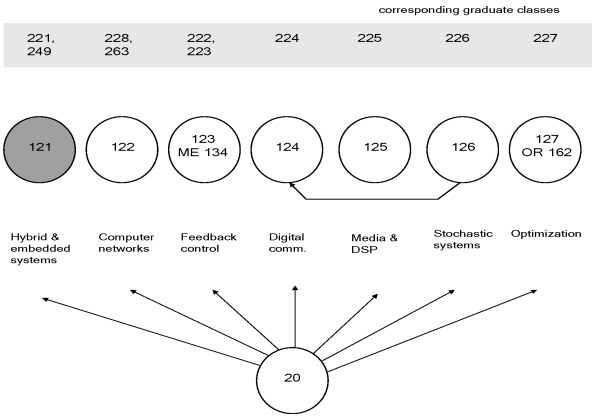

Fig. 3. Pre-requisite structure for undergraduate program in systems.

EECS 20N together with EECS 40 (Introduction to Microelectronic Circuits), CS 61A (The Structure and Interpretation of Computer Programs), CS 61B (Data Structures), and CS 61C (Machine Structures) are mandatory courses for any Berkeley undergraduate EECS student. In addition to having an important role in the general undergraduate education, EECS20N (see Lee 2000; Lee and Varaiya 2000) is also at the root of a system science program whose structure is shown in Figure 3. In this diagram, EE149, Hybrid and Embedded Systems, is an upper-division class that is under design and will provide the ideal follow-on to EECS20N in a curriculum that focuses on the foundations of embedded systems.

5.1 EECS20N: Structure and Interpretation of Signals and Systems

Motivation and History. EECS20N [EECS20N] is a course that electrical engineering and computer science students take in their second year at Berkeley. Traditionally, our undergraduates were exposed to a rigorous approach to systems in classes offered in the junior and senior year dealing with circuit theory, communication systems, and control. The contents of the courses were thought in terms of the application domain and exposed a certain degree of duplication, as ordinary differential equations and transforms such as Laplace and Fourier, were common tools. In addition, the models used were mostly continuous-time. Little, if any, attention was paid to discrete-time or event-driven models. From conversations among the system faculty, the idea of revamping substantially the foundations of systems and signals emerged to provide a strong basis for embedded system design.

Traditional courses such as circuit theory (and the resulting emphasis on linear time-invariant systems) were no longer deemed core courses and a unified approach to the basics of signals and systems took form as the seed for launching an initiative to bring our students to appreciate the mathematical underpinnings of embedded system analysis, including continuous and discrete abstractions and models. In particular, in 1999, Edward Lee and Pravin Varaija embarked on a journey that eventually led to EECS20N. They wrote a book [Lee and Varaiya 2003], developed a course on the fundamentals needed to

understand signals and systems, and adopted tools such as Matlab [Matlab] to lower the barrier to abstract reasoning using extensively visualization of system behavior. The description of their approach can be found in two papers [Lee 2000; Lee and Varaiya 2000] from which this section is taken.

The Program of the Course. The themes of the course are:

- The connection between imperative (computational) and declarative (mathematical) descriptions of signals and systems.
- The use of sets and functions as a universal language for declarative descriptions of signals and systems.
- State machines and frequency domain analysis as complementary tools for designing and analyzing signals and systems.
- Early and frequent discussion of applications.

State machines were the means to introduce EE students to reason about digital abstractions and frequency domain analysis was used to introduce CS students to reason about the continuous time world. The use of applications is essential to keep the students interested and motivated in absorbing material that otherwise may be too dry and abstract at an early stage of engineering education.

The course is designed to last 15 weeks. Below we summarize the contents on a weekly basis. The first 5 weeks and the last are the most relevant to embedded systems.

Week 1. The first week introduces forthcoming material by illustrating how signals can be modeled abstractly as functions on sets. The emphasis is on characterizing the domain and the range, and not on characterizing the function itself. The startup sequence of a voiceband data modem is used as an illustration, with a supporting applet that plays the very familiar sound of the startup handshake of V32.bis modem, examining the waveform in both the time and frequency domain.

Week 2. The second week introduces systems as functions that map functions (signals) into functions (signals). Again, the focus is not on how the function is defined, but rather on what is the domain and range. Block diagrams are defined as a visual syntax for composing functions.

Week 3. The first lecture in the third week is devoted to the problem of relating declarative and imperative descriptions of signals and systems. This sets the framework for making the intellectual connection between the labs and the lecture material. The rest of the week is devoted to introducing the notion of state and state machines. State machines are described by a function update that, given the current state and input, returns the new state and output. In anticipation of composing state machines, the concept of stuttering is introduced.

Week 4. The fourth week deals with nondeterminism and equivalence in state machines. Equivalence is based on the notion of simulation. Simulation

relations and bisimulation are defined for both deterministic and nondeterministic machines. These are used to explain that two state machines may be equivalent even if they have a different number of states, and that one state machine may be an abstraction of another in that it has all input/output behaviors of the other (and then some).

Week 5. The fifth week is devoted to composition of state machines. The deep concepts are synchrony, which gives a rigorous semantics to block diagrams and feedback. The most useful concept to help subsequent material is that feedback loops with delays are always well formed.

Week 6–14. In these weeks, the course deals with traditional linear system topics, such as time-domain response, frequency domain analysis and response, filtering, convolution, Fourier transforms, sampling and aliasing, and filter design.

Week 15. This week develops applications that showcase the techniques presented in the course. The precise topics depend on the interests and expertise of the instructors, but we have specifically covered vehicle automation, with emphasis on feedback control systems for automated highways. The use of discrete magnets in the road and sensors on the vehicles provides a superb illustration of the risks of aliasing. The few exercises in the text that require calculus provide any integration formulas that a student might otherwise look up. Although series figure prominently, we only lightly touch on convergence, raising, but not resolving, the issue.

The Lab. A major objective of the course is to introduce applications early, well before the students have built up enough theory to fully analyze the applications. This helps to motivate the students to learn the theory. In the Lab, we emphasize the use of software to perform operations that could not possibly be done by hand, operations on real signals, such as sounds and images.

While the mathematical treatment that dominates in the lecture and textbook is declarative, the labs focus on an imperative style, where signals and systems are constructed procedurally. Matlab and Simulink [Matlab] have chosen this as the basis for these exercises because they are widely used by practitioners in the field, and because they are capable of realizing interesting systems.

The labs are divided into two distinct sections: in-lab and independent. The purpose of the in-lab section is to introduce concepts needed for later parts of the lab. Each in-lab section is designed to be completed during a scheduled lab time with an instructor present to clear up any confusing or unclear concepts. The independent section begins where the in-lab section leaves off. It can be completed within the scheduled lab period, or may be completed on the students own time. They write a brief summary of their solution, following a supplied template, and turn it in at the beginning of the next scheduled lab period.

There are 11 lab exercises, each designed to be completed in 1 week. The exercises include: Arrays and Sound, Images, State Machines, Control Systems, Difference and Differential Equations, Spectrum, Comb Filters, Modulation and Demodulation, Sampling and Aliasing. Two of the weeks are quite interesting (State Machines and Control Systems) from the point of view of

embedded systems and models of computation. The third lab uses Matlab as a low-level programming language to construct state machines according to a systematic design pattern that will allow for easy composition. The theme of the lab is establishing the correspondence between pictorial representations of finite automata, mathematical functions giving the state update, and software realizations. The main project in this lab exercise is to construct a virtual pet. This problem is inspired by the Tamagotchi virtual pet made by Bandai in Japan. Tamagotchi pets were popular in the late 1990s and had behavior considerably more complex than that described in this exercise. The students design an open-loop controller that keeps the virtual pet alive. This illustrates that systematically constructed state machines can be easily composed.

In the Control System lab, students modify the pet so that its behavior is nondeterministic. They are asked to construct a state machine that can be composed in a feedback arrangement such that it keeps the cat alive. The semantics of feedback in this course are consistent with tradition in signals systems. Computer scientists call this style synchronous composition, and define the behavior of the feedback system as a (least or greatest) fixed point of a monotonic function on a partial order. In a course at this level, we cannot go into this theory in much depth, but we can use this example to explore the subtleties of synchronous feedback. Most students find this lab quite challenging, but also very gratifying when they figure it out. The concepts behind it are complex and better students realize that.

5.2 Discussion and Future Directions

EECS20N has reached a degree of maturity that will allow it to be taught by any faculty in the department given the large amount of teaching material available. During the first years, the course was not among the favorites of the students given its generality and mathematical rigor. However, the fine-tuning of labs and lectures and of their interrelation has had a positive effect on the overall understanding of the material and the students now express their satisfaction with this approach. Having laid out the foundation of the work, we are now considering extending the present offering in the upper division. The EE149 class emphasizes three main aspects:

- The idea of introducing signals and systems as described here with the operational and the denotational view can be traced to the research work that led to the development of the Lee-Sangiovanni-Vincentelli (LSV) tagged signal model [Lee and Sangiovanni-Vincentelli 1998], a denotational framework to compare models of computation. While the concept of time is not as abstract as in the LSV model, the course does present the denotational view of systems along similar lines. EE149 will focus on the semantics of embedded systems and introduce a consistent family of models of computation with the relative applications, strengths, and weaknesses.
- Using Matlab and Simulink as a virtual prototyping method and being exposed to both the digital and the analog abstraction, the students have an early exposure to hybrid systems [Lygeros et al.] as an important domain

for embedded systems where a physical plant described in the continuous-time domain is controlled by a digital controller. We will devote a substantial portion of the course to the discussion of the properties of hybrid models as paradigms for the future of large-scale system monitoring and control.

- A substantial problem is the way in which Simulink combines discrete and continuous-time models. Simulink essentially embeds the discrete in the continuous domain, i.e., the numerical integration techniques determine the time advancement for both continuous and discrete models. Thus *de facto* Simulink implements a single model of computation: the synchronous reactive model where logical time is determined by the integration algorithm. This may make Simulink nonideal for teaching embedded systems where heterogeneous models of computation play a fundamental role. We will add Ptolemy II to the Matlab/simulink environment, as a design capture and verification tool to obviate this problem.
- Implementation platforms are important as they determine the real-time properties of the system as well as its cost, weight, size, and power consumption. The course will present methods to capture implementation platforms and to map functionality to platforms.
- Applications will be emphasized as drivers and as test cases for the methods presented in the class. In particular, during the course, students will be asked to completely develop an embedded system in a particular application domain using the methods taught. This design work will be carried out in teams whose participants will have enough diversity as to cover all aspects of embedded system design.

Since many of these topics are covered in the graduate class of EECS249, we will borrow heavily from that experience, while the material of EECS249 will evolve toward more advanced topics.

6. CONCLUSIONS

We outlined the embedded system education program at the University of California at Berkeley. We stressed the importance of foundations in education as opposed to technicalities. Embedded systems are important enough to warrant a careful analysis of the mathematical bases upon which we can build a solid discipline that marries rigor with practical relevance. Given the present role of embedded systems in our research agenda and the traditional approach to education in the leading U.S. Universities, the first courses to be developed are advanced graduate courses. The natural evolution is to solidify the teaching material to a point where regular graduate classes can be taught and finally move the contents to the undergraduate curriculum while the graduate courses adjust continuously to the advances in the field brought about by research. The flexibility of the U.S. system allows the curriculum to change fairly easily and to strive for relevance to the ever-changing society and cultural landscape.

While we believe that our program has achieved a set of important goals, we do realize that much remains to be done. At the undergraduate level, we are planning to introduce an upper division class on hybrid and embedded

software systems. At the graduate level, we are considering the addition of regular courses on theoretical foundations focusing on function description and manipulation as well as one on reconfigurable and programmable architectural platforms to follow the basic graduate course (EECS249) on embedded systems. We also believe that embedded system courses should be considered foundational courses for the entire college of engineering and we are working with our Dean and Department Chairpersons to address this issue.

The views presented here are for a large part shared with the Artist [Artist] Network of Excellence Education Team, whose agenda is described in another paper of this special issue.

ACKNOWLEDGMENTS

We wish to acknowledge the long-time collaboration with Edward Lee, Tom Henzinger, Richard Newton, Jan Rabaey, Shankar Sastry, and Pravin Varaija in the research and teaching agenda on embedded systems at Berkeley. The help and support of the Metropolis group is gratefully acknowledged. Important impacts on our approach come from interactions with Albert Benveniste, Gerard Berry, Paul Caspi, Hugo DeMan, Luciano Lavagno, Joseph Sifakis, and the ARTIST European Network of Excellence team, Alberto Ferrari and his colleagues of PARADES, and last, but not least, our industrial associates too numerous to be mentioned. Funding for this work came from CHESS NSF ITR, and the GSRC.

REFERENCES

ARTIST. http://www.artist-embedded.org/overview/.

BALARIN, F. ET AL. 1997. *Polis: A Design Environment for Control-Dominated Embedded Systems*. Kluwer, Boston, MA.

BALARIN, F. ET AL. 1999. Synthesis of software programs for embedded control application. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

BALARIN, F., WATANABE, Y., HSIEH, H., LAVAGNO, L., PASSERONE, C., AND SANGIOVANNI-VINCENTELLI, A. 2003. Metropolis: An integrated electronic system design environment. *IEEE Computer*.

BALEANI, M., GENNARI, F., JIANG, Y., PATEL, Y., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 2002. HW/SW partitioning and code generation of embedded control applications on a reconfigurable architecture platform. In *Proceedings of the 10th International Symposium on Hardware/Software Codesign (CODES), Estes Park, Colorado*.

BERRY, G. AND GONTHIER, G. 1992. The esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming 19*, 2, 87–152.

BORKAR, S., COHN, R., COX, G., GLEASON, S., AND GROSS, T. 1988. Warp: An integrated solution of high-speed parallel computing. In *Supercomputing '88: Proceedings of the 1988 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society Press. 330–339.

BRAYTON, R. K., RUDELL, R., SANGIOVANNI-VINCENTELLI, A., AND WANG, A. R. 1987. Mis: A multiple-level logic optimization system. *IEEE Trans. Comput.-Aided Design Integrated Circuits 6*, 6 (Nov.), 1062–1081.

BRYANT, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput. C-35*, 8 (Aug.), 677–691.

CARLONI, L. P. 2004. Latency-Insensitive Design. Ph.D. thesis, University of California at Berkeley, Electronics Research Laboratory, College of Engineering, Berkeley, CA 94720. Memorandum No. UCB/ERL M04/29.

CASPI, P., PILAUD, D., HALBWACHS, N., AND PLAICE, J. A. 1987. Lustre: A declarative language for real-time programming. In *POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. ACM Press, New York. 178–188.

CASPI, E., CHU, M., HUANG, R., YEH, J., WAWRZYNEK, J., AND DEHON, A. 2000. Stream computations organized for reconfigurable execution (score). In *FPL '00: Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications*. Springer-Verlag. 605–614.

CS294. http://www.cs.berkeley.edu/~culler/cs294-f03/.

CS294W. http://www.cs.berkeley.edu/~adj/cs294-1.f00/.

DE KOCK, E. A., ESSINK, G., SMITS, W. J. M., VAN DER WOLF, P., BRUNEL, J.-Y., KRUIJTZER, W. M., LIEVERSE, P., AND VISSERS, K. A. 2000. Yapi: Application modeling for signal processing systems. *Proceedings of the Design Automation Conference*.

DSPACE. http://www.dspaceinc.com/ww/en/inc/products/sw/targetli.htm.

EE249. http://www-cad.eecs.berkeley.edu/~polis/class.

EE290A. http://www-cad.eecs.berkeley.edu/respep/research/classes/ee290a/fall02/.

EE290N. http://embedded.eecs.berkeley.edu/concurrency/.

EE290O. http://www.cs.uni-salzburg.at/~ck/teaching/eecs290o-spring-2002.

EECS20N. http://ptolemy.eecs.berkeley.edu/eecs20/index.html.

FERDINAND, C. AND WILHELM, R. 1998. On predicting data cache behavior for real-time systems. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*. Springer-Verlag, 16–30.

GAY, D., LEVIS, P., VON BEHREN, R., WELSH, M., BREWER, E., AND CULLER, D. 2003. The nesc language: A holistic approach to networked embedded systems. *SIGPLAN Not. 38*, 5, 1–11.

GIOTTO. http://embedded.eecs.berkeley.edu/giotto/.

HAREL, D. 1987. Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8*, 3 (June), 231–274.

HENZINGER, T. A., HOROWITZ, B., AND KIRSCH, C. M. 2003. Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE 91*, 84–99.

HENZINGER, T. A. AND KIRSCH, C. M. 2002. The embedded machine: Predictable, portable real-time code. In *Proceedings of the International Conference on Programming Language Design and Implementation*. ACM Press, New York, 315–326.

HOROWITZ, B. LIEBMAN, J., MA, C., KOO, J., SANGIOVANNI-VINCENTELLI, A., AND SASTRY, S. 2003. Platform-based embedded software design and system integration for autonomous vehicles. In *Proceedings of the IEEE*.

IXP1200. http://www.intel.com/design/network/products/npfamily/ixp1200.htm.

KOPETZ, H. AND GRUNSTEIDL, G. 1994. Ttp-a protocol for fault-tolerant real-time systems. *Computer 27*, 1, 14–23.

KOUSHANFAR, F., DAVARE, A., NGUYEN, D. T., POTKONJAK, M., AND SANGIOVANNI-VINCENTELLI, A. 2003. Distributed localized algorithms and protocols for power minimization in networked embedded systems. In *ACM/IEEE International Symposium On Low Power Electronics and Design*.

LEE, E. A. 2000. Designing a relevant lab for introductory signals and systems. *Proc. of the First Signal Processing Education Workshop*.

LEE, E. A. AND MESSERSCHMITT, D. G. 1987. Synchronous data flow. In *Proceedings of the IEEE*.

LEE, E. A. AND SANGIOVANNI-VINCENTELLI, A. L. 1998. A framework for comparing models of computation. *IEEE Trans. Comput.-Aided Design Integrated Circuits 17*, 12 (Dec.), 1217–1229.

LEE, E. A. AND VARAIYA, P. 2000. Introducing signals and systems—the Berkeley approach. *Proc. of the First Signal Processing Education Workshop*.

LEE, E. A. AND VARAIYA, P. 2003. Structure and interpretation of signals and systems. Addison-Wesley, Reading, MA.

LI, Y.-T. S. AND MALIK, S. 1995. Performance analysis of embedded software using implicit path enumeration. In *Workshop on Languages, Compilers and Tools for Real-Time Systems*. 88–98.

LIBERTY. http://liberty.cs.princeton.edu/software/lse/.

LIU, C. L. AND LAYLAND, J. W. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM 20*, 1, 46–61.

LYGEROS, J., TOMLIN, C., AND SASTRY, S. 1999. Controllers for reachability specifications for hybrid systems. In *Automatica, Special Issue on Hybrid Systems*.

MARTIN, G. AND SALEFSKI, B. 1998. Methodology and technology for design of communications and multimedia products via system-level ip integration. In *Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE Computer Society.

MATHWORKS. http://www.mathworks.com/products/rtw/.
MATLAB. http://www.mathworks.com/.
MESCAL. http://embedded.eecs.berkeley.edu/mescal.
MURATA, T. 1989. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*. 541–580. NewsletterInfo: 33. Published as *Proceedings of the IEEE*, *77*, 4.
NEXPERIA. http://www.semiconductors.philips.com/products/nexperia/.
PASSERONE, R. 2004. Semantic foundations for heterogeneous systems. Ph.D. thesis, University of California, Berkeley.
PATH. http://www.path.berkeley.edu/.
PICORADIO. http://bwrc.eecs.berkeley.edu/research/pico_radio.
PINTO, A., CARLONI, L., AND SANGIOVANNI-VINCENTELLI, A. 2002. Constraint-driven communication synthesis. In *Proceedings of the Design Automation Conference 2002 (DAC'02)*.
PNUELI, A. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*. IEEE, IEEE Computer Society Press, Providence, Rhode Island, 46–57.
PTOLEMYII. http://ptolemy.eecs.berkeley.edu.
SANGIOVANNI-VINCENTELLI, A. 2002. Defining platform-based design. *EEDesign of EETimes*.
SANGIOVANNI-VINCENTELLI, A., CARLONI, L., BERNARDINIS, F. D., AND SGROI, M. 2004. Benefits and challenges for platform-based design. In *Proceedings of the 41st Annual Conference on Design Automation*. ACM Press, New York, 409–414.
SHAH, N., PLISHKER, W., AND KEUTZER, K. 2004. *NP-Click: A Programming Model for the Intel IXP1200*, 1 ed. Vol. 2. Elsevier, Chapter 9, 181–201.
TEJA. http://www.teja.com/.
VACHHARAJANI, M., VACHHARAJANI, N., PENRY, D. A., BLOME, J., AND AUGUST, D. I. 2004. The liberty simulation environment, version 1.0. *Performance Evaluation Review: Special Issue on Tools for Architecture Research 31*, 4 (Mar.).
XILINX. http://www.xilinx.com.
ZIMET, L., KAO, S., AMIR, A., AND SANGIOVANNI-VINCENTELLI, A. 2004. An embedded system for an eye detection sensor. In *Computer Vision and Image Understanding: Special Issue on Eye detection and Tracking.*
ZIVOJNOVIC, V., PEES, S., AND MEYR, H. 1996. Lisa—machine description language and generic machine model for hw/sw co-design. *IEEE Workshop on VLSI Signal Processing*.

Received December 2004; revised March 2005; accepted May 2005