Embedded System Education: A New Paradigm for Engineering Schools?

Alberto Luigi Sangiovanni-Vincentelli and Alessandro Pinto
University of California at Berkeley
alberto,apinto@eecs.berkeley.edu

***Abstract*— Embedded systems are emerging as an essential component of modern electronic products. Embedded system design problems are posing challenges that involve entirely new skills for engineers. These skills are related to the combination of traditionally disjoint engineering disciplines. There is a shared concern that today's educational systems are not providing the appropriate foundations for embedded systems. We believe a new education paradigm is needed.**

We will argue this point using the example of an emerging curriculum on embedded systems at the University of California at Berkeley. This curriculum is the result of a distillation process of more than ten years of intense research work. We will present the considerations that are driving the 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.

***Index Terms*— Embedded System Design, Education.**

I. 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 where vertical research areas cover application domains such as automotive, avionics, energy, industrial control, and 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 that the enabling technologies are explicitly linked to the vertical application areas and are geared towards 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 [41] 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 started. In the automotive domain, there has also been strong interdepartmental interaction between mechanical engineering and electrical engineering/computer science.

In US 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 start with advanced graduate course offerings to migrate towards 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 and 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 with no errors.

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 [30], [32]) that has been now offered over the past five 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 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.

II. The Graduate Program Part 1: EECS249 Design of Embedded Systems: Models, Validation and Synthesis

This course [15] 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 Engineering, Nuclear Engineering and Civil Engineering.

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 [45] [46] 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 possible otherwise. 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.

A. 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 that are summarized in table I.

Introduction and Methodology. 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 [45]) and examples of its applications. We present many examples of embedded systems: cars, airplanes, smart buildings, elevators, and sensor networks. In the introductory lecture, we highlight commonalities among all the examples that we present to set the stage for the philosophy of the course 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.

Function. The notion of behavior is analyzed and the role of non-determinism 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

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

Languages, Data Flow Networks, Petri Nets, Discrete Event Systems and Hybrid Systems. For each model we present the computation, communication and coordination semantics with a particular emphasis on the properties that are guaranteed and verifiable. We outline the use of unified frameworks to compare the different models of computation and we present the Tagged-Signal Model (TSM) [31] and agent algebras [40] as unifying theories to allow the composition of different models of computation to describe a complete system. We introduce here the Ptolemy [44] and Metropolis [3] environments for analysis and simulation of heterogeneous specifications.

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. 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 [43]. We also presented the Ptolemy actor-oriented semantics and showed how this is another style for abstract semantics.

Architecture and Mapping 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 [38] environments are presented.

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. 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. We show a practical example of an architecture platform like Xilinx VirtexII Pro as a heterogeneous platform that can be used for fast prototyping.

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.

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 re-writing of 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.

Then, we 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. 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 [49] and from other platforms such as the Intel MXP5800.

Synthesis and Verification. 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 keep in the same environment 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 [34] 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 [21].

We then move to the software synthesis problem and we 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 [5] is compiled using this idea and using FSM optimization techniques based on MIS [7] 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 [8] 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 [20].

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 [2], [23] and we point to the evolution of implementation platforms that can make the optimized "compilation" problem increasingly difficult.

Finally we present two industrial tools for automatic code generation: the real-time workshop (RTW) [36] by Mathworks and Targetlink [14] by dSPACE.

The last few lectures of the course are 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 demonstrate a complete design flow in action, but the parallel development of the tools needed did not provide us in time with the flow as we hoped to.

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 capture well 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.

B. 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 form 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 publication for the best reports.

Following the course organization, projects are divided in the ones that are related to functional description, architectural description, mapping, 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 short 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 this kind of projects are rarely taken even though a couple were chosen and had worthwhile results. Projects that were developed in the past years include Time-Based Communication Refinement for CFSMs, Heterochronous dataflow domain in Ptolemy II, Hardware extension to ECL and Extending CFSMs to allow multirate operations.

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.

A successful project was about a complete methodology for embedded software design and system integration of a rotorcraft UAV [26] whose results were published both as a conference and as invited paper in the Proceedings of the IEEE. This project was carried out by three students and two mentors coming from different knowledge domain. In the area of wireless sensor networks, a group of students applied the methodology for studying an ad-hoc sleeping disciplines for nodes [29]. This application domain is very relevant not only for its distributed nature but also for the severe energy consumption constraints on each node.

Two other projects that had success to the point of being published in primary conferences were related to chip architectures. One was about the synthesis of on-chip communication architectures [42] and automatic hardware/software partitioning for reconfigurable platforms [4].

Projects in less obvious application domains have also been offered. In particular, two years ago a project on the design of a real-time eye detection system was assigned to two students [50]..

III. 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 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 jelled and there is enough interest in the student population. Since embedded systems are so important in our research agenda, there are several advanced courses that have a

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, Co-design 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 Co-design, Mathworks RTW, dSpace Target Link

TABLE I
CLASS SYLLABUS

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 1, 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.

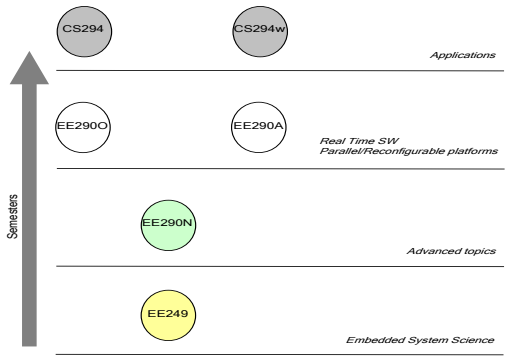

Fig. 1. A graduate program in embedded systems

A. 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.

1) Deeply Embedded Network Systems: This advanced class focuses on ubiquitous computing [11]. The course is based on the experience of researchers form 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, one week is dedicated to the presentation of several platforms and operating systems that are currently available.

The following four 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 objects in the environment in which they are embedded.

Students in the class are divided in 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 form the description of the application.

2) 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 in on new trends in mobile computing and integration of heterogeneous networks [12].

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 to solve all this problems are presented.

A network node is an hand-held device that presents severe limitations in power consumption. This problem, which is presented as an important constraints, 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 this kind of systems like WLAN, UMTS, GSM and GPRS are presented.

B. 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.

1) EE290N: Concurrent models of computation for embedded software: This advanced class focuses on concurrent models of computation for embedded software development [17]. It can be seen as the extension to and deep analysis of the first five 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 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 a particular emphasis on the problems that arise when using this technique for developing concurrent programs. Then, languages for particular applications, like NesC [22] and Lustre [10], are taken as representative of domain specific models for embedded software.

Then Process Networks (PN) are introduced and an abstract semantics based on 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.

After Process Networks, synchronous languages are introduced together with the notion of complete partial orders and the least fixed point semantics of synchronous programs. Dataflow models are extensively discussed: statically schedulable Data Flow, multidimensional and heterochronous data flow and boolean data flow.

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.

2) 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 [24], and explains why it is suitable for a class of embedded software [18]. The class is divided in three parts:
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 [25] is then described and its properties are formally explained: semantics of the E-machine, portability and predictability, deterministic behavior and logical execution time.
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. Then, early deadline first and 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).
RT Communication. The last part of the class deals with real time communication. Messages are modeled with deadline and worst case latency. Two protocols are presented in details: Controller Area Network (CAN) and Time Triggered Protocol (TTP) [28]. The problem of fault tolerance is introduced and the solution proposed by the TTP protocol is explained.

3) 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 [16]. This course can be considered as a follow-on to EECS249 with respect to Architecture and Mapping.

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 co-design finite state machines are introduced emphasizing the fact that each model is particularly suitable for a specific application domain. The Click [47] model of computation is explained for modeling the processing of streams of packets in routers.

The first example presented in the class is MPEG decoding together with an entire flow from specification (using a flavor

of Kahn process networks called YAPI [13]) to implementation. This example shows how the requirements of an application condition the selection of a model of computation and the underling implementation platform. Several platforms are then presented: the Nexperia [39] platform by Philips for multimedia application and the Ixp1200 [27] platform by Intel for network processing. For each platform, examples of what kind of application they target are given together with performances result.

The Giotto model of computation and the E-Machine are presented and an example of the software running on an autonomous helicopter is shown. Another example show the implementation of an IP router on a VirtexII Pro FPGA and the implication of using a multi-threaded implementation. The SCORE (Stream Computations Organized for Reconfigurable Execution) [9] project is also presented. In this project both function and architecture are described using Kahn process networks and 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. The last lectures give an overview of multiprocessors platforms like RAW and IWarp [6] and other programmable platforms like PSOC by Cypress and the Extensa Processor.

IV. 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.

A. 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 allow 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.

B. 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 much 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 [48] 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.

V. 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.

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 [30], [32]) is also at the root of a system science program whose structure is shown in Figure 2. 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.

A. EECS20N: Structure and interpretation of signals and systems

a) Motivation and History: EECS20N [19] 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 modeling techniques used were essentially continuous time ignoring for a large part the

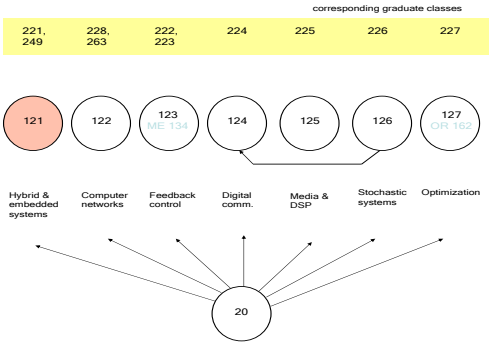

Fig. 2. Pre-requisite Structure for Undergraduate Program in Systems

discrete world. 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, Edward Lee and Pravin Varaija embarked in 1999 for the journey that eventually led to EECS20N. They wrote a book [33], built a course on the fundamentals needed to understand signals and systems, and adopted tools such as Matlab [37] to lower the barrier to abstract reasoning using extensively visualization of system behavior. The description of their approach can be found in two papers [30], [32] from which this section is taken.

b) 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, frequency domain analysis 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 a material that otherwise may be too arid and abstract at an early stage of engineering education.

The introduction to the course motivates forthcoming material by illustrating how signals can be modeled abstractly as functions on sets. The emphasis is on characterizing the domain and the range, 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, and examines the waveform in both the time and frequency domain. Systems are described 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.

The connection between imperative and declarative description is fundamental to set the framework for making the intellectual connection between the labs and the lecture material. The, finite state machines are introduced and analyzed. Nondeterminism and equivalence in state machines is explained. Equivalence is based on the notion of simulation, so 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). Composition of finite state machines and the feedback loop connection is introduced. The most useful concept to help subsequent material is that feedback loops with delays are always well formed.

The last part applies the theory to real examples that depend on the interests and expertise of the instructor, 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.

c) 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 [37] are chosen 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.

There are 11 lab exercises, each designed to be completed in one 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.

B. 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 in gestation will emphasize 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 [31], 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 [35] 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 non ideal for teaching embedded systems where heterogeneous models of computation play a fundamental role. We will add to the Matlab/simulink environment, Ptolemy II 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 develop completely 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 EECS249, the graduate class, we will borrow heavily from that experience while the material of EECS249 will evolve towards more advanced topics.

VI. 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 rigour with practical relevance. Given the present role of embedded systems in our research agenda and the traditional approach to education in the leading US 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 US system allows to change fairly easily the curriculum 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 do. 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 [1] Network of Excellence Education Team whose agenda is described in another paper of this special issue.

VII. Acknowledgements

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. Work (partially) done in the framework of the HYCON Network of Excellence, contract number FP6-IST-511368.

REFERENCES

[1] Artist, "http://www.artist-embedded.org/overview/."
[2] F. Balarin *et al.*, *Polis: A Design Environment for Control-Dominated Embedded Systems*. Kluwer, 1997.
[3] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An integrated electronic system design environment," *IEEE Computer*, Apr 2003.
[4] M. Baleani, F. Gennari, Y. Jiang, Y. Patel, R. K. Brayton, and A. Sangiovanni-Vincentelli, "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, USA*, May 2002.
[5] G. Berry and G. Gonthier, "The esterel synchronous programming language: Design, semantics, implementation," *Science of Computer Programming*, vol. 19, no. 2, pp. 87–152, 1992.
[6] S. Borkar, R. Cohn, G. Cox, S. Gleason, and T. Gross, "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, 1988, pp. 330–339.
[7] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "Mis: A multiple-level logic optimization system," *IEEE Trans. Comput.-Aided Design Integrated Circuits*, vol. 6, no. 6, pp. 1062–1081, Nov. 1987.
[8] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," vol. C-35, no. 8, pp. 677–691, Aug. 1986. [Online]. Available: mentok.cse.psu.edu/bryant86graphbased.html
[9] E. Caspi, M. Chu, R. Huang, J. Yeh, J. Wawrzynek, and A. DeHon, "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, 2000, pp. 605–614.
[10] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice, "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, 1987, pp. 178–188.
[11] CS294, "http://www.cs.berkeley.edu/~ culler/cs294-f03/."
[12] CS294w, "http://www.cs.berkeley.edu/~ adj/cs294-1.f00/."
[13] E. A. de Kock, G. Essink, W. J. M. Smits, P. van der Wolf, J.-Y. Brunel, W. M. Kruijtzer, P. Lieverse, and K. A. Vissers, "Yapi: Application modeling for signal processing systems," *Proceedings of the Design Automation Conference*, June 2000.
[14] dSpace, "http://www.dspaceinc.com/ww/en/inc/products/sw/targetli.htm."
[15] EE249, "http://www-cad.eecs.berkeley.edu/~ polis/class."
[16] EE290A, "
http://www-cad.eecs.berkeley.edu/respep/research/classes/ee290a/fall02/."
[17] EE290N, "http://embedded.eecs.berkeley.edu/concurrency/."
[18] EE290O, "http://www.cs.uni-salzburg.at/~ ck/teaching/eecs290o-spring-2002."
[19] EECS20N, "http://ptolemy.eecs.berkeley.edu/eecs20/index.html."
[20] F. B. et. al., "Synthesis of software programs for embedded control application," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, June 1999.
[21] C. Ferdinand and R. Wilhelm, "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, 1998, pp. 16–30.
[22] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," *SIGPLAN Not.*, vol. 38, no. 5, pp. 1–11, 2003.
[23] Giotto, "http://embedded.eecs.berkeley.edu/giotto/."
[24] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, pp. 84–99, 2003.
[25] T. A. Henzinger and C. M. Kirsch, "The embedded machine: Predictable, portable real-time code," in *Proceedings of the International Conference on Programming Language Design and Implementation*. ACM Press, 2002, pp. pp. 315–326.
[26] B. Horowitz, J. Liebman, C. Ma, J. Koo, T. A. Henzinger, A. Sangiovanni-Vincentelli, and S. Sastry, "Embedded software design and system integration for rotorcraft uav using platforms," in *Proceedings of the 15th IFAC World Congress on Automatic Control*. Elsevier,, 2002.
[27] IXP1200, "
http://www.intel.com/design/network/products/npfamily/ixp1200.htm."
[28] H. Kopetz and G. Grunsteidl, "Ttp-a protocol for fault-tolerant real-time systems," *Computer*, vol. 27, no. 1, pp. 14–23, 1994.
[29] F. Koushanfar, A. Davare, D. T. Nguyen, M. Potkonjak, and A. Sangiovanni-Vincentelli, "Distributed localized algorithms and protocols for power minimization in networked embedded systems," in *ACM/IEEE International Symposium On Low Power Electronics and Design*, 2003.
[30] E. A. Lee, "Designing a relevant lab for introductory signals and systems," *Proc. of the First Signal Processing Education Workshop*, 2000.
[31] E. A. Lee and A. L. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE Trans. Comput.-Aided Design Integrated Circuits*, vol. 17, no. 12, pp. 1217–1229, Dec. 1998.
[32] E. A. Lee and P. Varaiya, "Introducing signals and systems – the berkeley approach," *Proc. of the First Signal Processing Education Workshop*, 2000.
[33] ——, "Structure and interpretation of signals and systems," 2003.
[34] Y.-T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," in *Workshop on Languages, Compilers and Tools for Real-Time Systems*, 1995, pp. 88–98.
[35] J. Lygeros, C. Tomlin, and S. Sastry, "Controllers for reachability specifications for hybrid systems," in *Automatica, Special Issue on Hybrid Systems*.
[36] Mathworks, "http://www.mathworks.com/products/rtw/."
[37] Matlab, "http://www.mathworks.com/."
[38] Mescal, "http://embedded.eecs.berkeley.edu/mescal."
[39] Nexperia, "http://www.semiconductors.philips.com/products/nexperia/."
[40] R. Passerone, "Semantic foundations for heterogeneous systems," Ph.D. dissertation, University of California, Berkeley, 2004.
[41] PATH, "http://www.path.berkeley.edu/."
[42] A. Pinto, L. Carloni, and A. Sangiovanni-Vincentelli, "Constraint-driven communication synthesis," in *Proceedings of the Design Automation Conference 2002 (DAC'02)*, June 2002.
[43] A. Pnueli, "The temporal logic of programs," in *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, IEEE. Providence, Rhode Island: IEEE Computer Society Press, Oct. 31–Nov. 2 1977, pp. 46–57.
[44] PtolemyII, "http://ptolemy.eecs.berkeley.edu."
[45] A. Sangiovanni-Vincentelli, "Defining platform-based design," *EEDesign of EETimes*, February 2002.
[46] A. Sangiovanni-Vincentelli, L. Carloni, F. D. Bernardinis, and M. Sgroi, "Benefits and challenges for platform-based design," in *Proceedings of the 41st annual conference on Design automation*. ACM Press, 2004, pp. 409–414.
[47] N. Shah, W. Plishker, and K. Keutzer, *NP-Click: A Programming Model for the Intel IXP1200*, 1st ed. Elsevier, 2004, vol. 2, ch. 9, pp. 181–201.
[48] Teja, "http://www.teja.com/."
[49] Xilinx, "http://www.xilinx.com."
[50] L. Zimet, S. Kao, A. Amir, and A. Sangiovanni-Vincentelli, "An embedded system for an eye detection sensor," in *Computer Vision and Image Understanding: Special Issue on Eye Detection and Tracking*, 2004.