Undergraduate Embedded System Education at Carnegie Mellon

PHILIP KOOPMAN, HOWIE CHOSET, RAJEEV GANDHI, BRUCE KROGH, DIANA MARCULESCU, PRIYA NARASIMHAN, JoANN M. PAUL, RAGUNATHAN RAJKUMAR, DANIEL SIEWIOREK, ASIM SMAILAGIC, PETER STEENKISTE, DONALD E. THOMAS, and CHENXI WANG
Carnegie Mellon University

Embedded systems encompass a wide range of applications, technologies, and disciplines, necessitating a broad approach to education. We describe embedded system coursework during the first 4 years of university education (the U.S. undergraduate level). Embedded application curriculum areas include: small and single-microcontroller applications, control systems, distributed embedded control, system-on-chip, networking, embedded PCs, critical systems, robotics, computer peripherals, wireless data systems, signal processing, and command and control. Additional cross-cutting skills that are important to embedded system designers include: security, dependability, energy-aware computing, software/systems engineering, real-time computing, and human–computer interaction. We describe lessons learned from teaching courses in many of these areas, as well as general skills taught and approaches used, including a heavy emphasis on course projects to teach system skills.

Categories and Subject Descriptors: K.3.2 [**Computer and Information Science Education**]: C.3 [**Special-Purpose and Application-Based System**]: D.4.7 [**Organization and Design**]: J.7 [**Computers in other Systems**]

General Terms: Design, Experimentation, Human Factors, Performance, Reliability, Security

Additional Key Words and Phrases: Embedded systems education, curriculum

1. INTRODUCTION

Trying to teach embedded computing as a unified topic is a difficult task. Embedded applications are very diverse and span a tremendous range of complexity [Estrin et al. 2001]. Indeed, embedded computing is more readily defined by what it is not (it is not generic application software executing on the main CPU of a "desktop computer") than what it is. Embedded CPUs comprise the vast majority of processors [Turley 2002], although many of those processors are used in very high-volume applications.

Authors' address: Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh PA, 15213.

From an educational needs perspective, it is relevant to scope educational capacity according to the number of engineers involved. While precise numbers are difficult to come by, one useful data point is the relative size of relevant periodical subscriber bases. *Embedded Systems Programming* magazine, a qualified subscription monthly publication, currently has 45,000 subscribers, mostly in the United States [CMP 2005]. By way of comparison, *IEEE Computer* magazine, which is sent to every member of the IEEE Computer Society worldwide, has 94,841 subscribers [IEEE 2004]. From our experience, most embedded system developers have minimal or no formal training in methodical embedded system engineering approaches—most expertise comes as a result of on-the-job experience and self-training. Given the substantial number of engineers who practice in this area, we believe that the field could greatly benefit from an enhanced, widespread foundation of education in computer science and engineering, and are working to that end.

Carnegie Mellon University has offered embedded system courses of various sorts for three decades. Grason and Siewiorek [1975] describe an early embedded controls project course that included student projects on motion control, flight simulation, train control, and ultrasonic obstacle detection. Over time, various courses and areas of specialization have evolved to become those described in the remainder of this paper.

There are two important cultural factors inherent to work at Carnegie Mellon that influence our approach. First, we have a strong tradition of system building, both in research and coursework. This is reflected in the significant project content of many of our courses. Second, our engineering school has adopted a very flexible curriculum [Director et al. 1995] that minimizes prerequisite courses and encourages continuous innovation by faculty in their course coverage. This has lead to a situation in which the embedded "curriculum" is an organic result of evolutionary development based on feedback from industry, faculty perception of technology trends, faculty coordination, and student interests. It is definitely not the result of a centralized planning process.

The purpose of this paper is to describe how we at Carnegie Mellon University approach the task of organizing and teaching the diverse areas of embedded computing, as well as to discuss areas we consider important in this field, even if we do not happen to teach specific courses on them. The emphasis of the discussion is on undergraduate education (the first 4 years at a U.S. university, leading to a Baccalaureate degree) and characterizing various areas of knowledge at that level rather than by research topics more typical of graduate-level (post-Baccalaureate) course offerings. To reduce confusion over terminology, we will refer to portions of the curriculum by year, with the first 4 years being Baccalaureate study, years 5 and 6 generally referring to Masters and new Doctoral students, and years 7 and on referring exclusively to Doctoral students.

A relatively new requirement for ABET [2004] accreditation in a U.S. engineering education is the inclusion of what most universities term a "capstone" design course. A capstone course involves the completion of a significant engineering project, usually in the student's fourth year. Such projects are typically completed by teams within the structure of a normal university course. Embedded systems content is prevalent in our capstone design courses. Capstone

courses are distinct from the European Diploma thesis approach, which, in general, is not used within the United States educational system.

2. APPROACH TO EMBEDDED EDUCATION

We believe that it is the rare embedded system developer who will need, or even be able to absorb, expertise in all areas of embedded systems. Therefore, we divide expertise into several areas to guide our approach to teaching. From an application perspective, we think of embedded computing as falling into the following distinct categories: small and single microcontroller applications, control systems, distributed embedded control, system on chip, networking, embedded PCs, critical systems, robotics, computer peripherals, wireless data systems, signal processing, and command and control. More detailed explanations of these categories are provided in the following sections. Most embedded system developers will need expertise in several of these areas.

In addition to application categories, there are core skill areas that apply to most application areas rather than being specific to a focused end-use application or industry. These skill areas include: security, dependability, energy-aware computing, software/systems engineering, real-time computing, and human–computer interaction.

There are, of course, many institutions teaching various aspects of embedded systems. For example, the ARTIST [2003] project has issued guidelines for curricula on embedded software and systems. (ARTIST is also the subject of a paper elsewhere in this special issue.) That curriculum divides topics into the areas of control and signal processing, computing theory, real-time computing, distributed systems, optimization and evaluation, and system architecture and engineering, with a noted need for practice on real systems and simulators. We have organized our courses somewhat differently for a variety of reasons that are probably not crucial. But, more importantly, we have chosen to place significant emphasis on teaching a broad range of embedded systems in the first 4 years.

Other universities are contributing to understanding what it takes to create a broad and deep educational experience in embedded systems beyond point courses in the area. For example, Neilsen et al. [2002] at Kansas State University are teaching a four-course interdisciplinary sequence for real-time embedded systems that includes a remedial course (covering computer science and computer engineering topics), a real-time implementation course, a theory course, and a capstone design course. Motus [1998] describes a trio of courses at Tallinn Technical University on real-time software engineering. Pri-Tal et al. [2001] have reported an ambitious plan to combine research, education, and industry outreach to establish an "ecosystem" for embedded systems in Phoenix Arizona, including Arizona State University. Tempelmeier [1998] advocates adjusting the regular computer science curriculum to include real-time educational components as practiced at the Fachhochschule Rosenheim. Vahid [2003] describes a three-course embedded systems sequence at the University of California, Riverside. There are obviously many courses and sets of courses in embedded systems worldwide beyond those described in published papers.

Beyond university offerings, the series of Embedded Systems Conferences [2004] has a significant lecture track designed to provide continuing education as well as introductory material on embedded systems to practicing engineers.

Wolf and Madsen [2000] summarize some of the issues in embedded systems education. But for all the efforts to teach embedded systems thus far, the community is, in general, still finding its way. It is our hope that this paper contributes to understanding the depth and breadth of possible education approaches in this developing area.

3. EMBEDDED APPLICATION AREAS

The following sections discuss each embedded application area in turn. Our curriculum organization is that the first course on small and single microcontroller applications is used as the gateway course to the rest of the embedded curriculum. In general there are no embedded-specific constraints on what students can take (additional prerequisites tend to be area-specific skill courses to relevant domains, such as signal processing mathematics). We do not necessarily teach courses for the first 4 years of study in every area listed at the moment, but we believe that doing so is a reasonable goal to set for the long term. Unless otherwise noted, the term "course" refers to a semester-long university course that, in general, meets 3 or 4 hours per week (plus laboratory sessions) for 15 weeks.

3.1 Small and Single Microcontroller Applications

Distinguishing Features. Many low-end embedded applications use a 4- or 8-bit microcontroller running a relatively small program to control some device. This is the traditional origin of embedded computing, where education has often taken the form of an "introduction to microcontrollers" course and, which we feel, still serves well as an introduction to other embedded topics. The area is characterized by severe cost limitations and scarcity of almost every resource (memory, computing power, etc.). While embedded devices increasingly support networking capabilities (e.g., sensors in building-monitoring networks), most of the processors produced are still used as stand-alone systems. A comprehensive understanding of the operation of nonnetworked, single microcontroller-based embedded systems is crucial for students in year 2 or 3 to appreciate the emerging issues when these devices are connected together in networked, multimicrocontroller embedded systems.

Key Skills and Principles. Because embedded devices contain both hardware and software components, understanding hardware–software interactions is essential. Key skills in this area span both electrical engineering and computer science and address the interrelationships among processor architectures, performance optimization, operating systems, virtual memory, concurrency, task scheduling, and synchronization. It is important for students to have a breadth of exposure so that they appreciate the utility of other fields (e.g., digital signal processing, feedback control) in embedded systems. At the same time, students should appreciate what distinguishes embedded systems

from related areas, such as operating systems, and understand the tight coupling between embedded applications and the hardware platforms that host them.

Teaching Approach. As a prerequisite, students should have participated in some significant (C and assembly language) programming project so that they can concentrate on building system-level skills rather than on learning to write programs. An introduction to logic design course that is required of all students in the department provides sufficient hardware background.

Our course coverage includes writing assembly-language and C programs specifically for embedded system applications. The lecture topics include timers, interrupts, caches, virtual memory, direct memory access, double-buffering, profiling, performance optimization, A/D and D/A conversion, practical signal processing, mutual exclusion, watchdog timers, scheduling, system buses/backplanes, serial/parallel communications, flash memory, and device drivers.

Because this course covers a diverse array of topics, we have favored using a series of individual hands-on projects that build upon each other, with each project oriented toward a distinct topic. Students learn more effectively when motivated by exciting course projects. To this end, the course involves designing/implementing/testing eye-catching, interesting projects with underlying complex, detailed concepts. For instance, one project involves building a simple version of a video-arcade game. In the process of building something "fun," students learn about and implement, a great deal of embedded-system functionality, including watchdog/periodic timers, nested interrupts, software interrupt-handlers, flash memory, and terminal device I/O. We have successfully taught, and continue to teach, this course as a live, distance education offering to an audience consisting of students on our Pittsburgh campus and our campus in Athens, Greece.

In our current curriculum, this area is combined with real-time computing (discussed later), leading us to use 32-bit processors and relatively full-featured real-time operating systems for projects. This is a tradeoff that might be made differently in other curricula, where more introductory emphasis on hands-on 8-bit microcontroller projects in a course that covers only this area has merit. Our current course does not use a standard text because of the combination of topics and student audiences we address. Several textbooks and other books suitable for classroom use exist and should be considered by instructors teaching this area. Available books that span a range of approaches include those by Wolf [2001], Simon [1999], Lewis [2001], Catsoulis [2003], and Berger [2002].

Challenges. Because many embedded devices are used in safety-critical settings, incorporating and emphasizing a steady level of software engineering is important in training embedded system engineers. We have found that typical engineering students dislike "heavy" engineering processes and, when short on time, cut corners to meet course deadlines. A significant challenge is getting third-year students to appreciate the perspective that undocumented, unmaintainable code that appears to mostly "work" is not good enough for an embedded

systems course. A crucial end result to strive for is making good software engineering practices an intrinsic part of each student's mindset and approach, rather than merely an afterthought.

3.2 Control Systems

Distinguishing Features. Embedded control systems are used to close feedback loops. Consequently, in contrast to many embedded system applications where design requirements map directly into specifications for embedded software, the design requirements for control systems typically prescribe desired behaviors for the integrated closed-loop system. For example, requirements for an automotive cruise control system characterize the desired behavior of the automobile under various road conditions and driver commands rather than enumerating software functions. What embedded processors need to do to achieve the desired behavior is not immediately obvious from the requirements and requires models of the automobile dynamics as well as feedback control theory.

Key Skills and Principles. Industry is moving quickly toward model-based design of embedded control systems, where models are used throughout the entire design process, from requirements capture to the automatic generation of the production code for the target embedded processor. Students should learn how to create, analyze, and validate models of physical dynamic systems. They should also understand numerical simulation, how parameters need to be chosen for integration routines to obtain correct simulation results, and how control-oriented models are created using linearization and order reduction. They should be introduced to hardware-in-the-loop (HIL) testing and understand what features of the real implementation can be investigated using HIL techniques. They need to understand the elements of automatic code generation, including the influence of parameters for timing and target-specific features, as well as why final parameter tuning is usually needed in the implementation of embedded control systems, and how it can be accomplished to achieve the desired performance.

We are unaware of any textbooks that introduce students to the full range of issues described above. Traditional control systems textbooks have, at best, only a cursory discussion of implementation issues. Embedded systems textbooks, on the other hand, seldom include any discussion of the complexities of control system design.

Teaching Approach. The historical approach of simply having students take control courses along with embedded systems courses is unsatisfactory for several reasons. By focusing exclusively on the mathematics of feedback control system design, traditional control courses fail to introduce students to the realities of embedded control system implementation. Control-oriented models neglect important details such as sampling jitter, finite precision (which is typically much more restrictive on an embedded processor than in the computers used for design), data conversion, timing constraints, limits imposed by the device interfaces, and physical saturation in sensors and actuators. These issues are addressed and understood adequately only when the feedback control

design is carried through to a full implementation on an embedded processor. On the other hand, an embedded systems course that does not consider feedback control applications fails to introduce students to the types of constraints and limitations that arise when the "environment" for the embedded software is a complex physical dynamic system.

We have designed a course that introduces students to several of these topics through lectures, homework assignments, and, most importantly, through a sequence of laboratory exercises based on a commonly used 32-bit automotive embedded controller. Students coming into this course have taken a first course in signals and systems and, perhaps, a course in feedback control theory, as well as an introductory computer systems course covering concepts such as interrupts, operating systems, and device drivers. The emphasis in this course is model-based design using tools such as MATLAB Simulink/Stateflow and Real-Time Workshop (RTW). Experiments involving motor control systems increase in complexity, culminating in a full haptic interface system that emulates elements of control systems for the so-called X-by-Wire systems (e.g., steer-by-wire, brake-by-wire, fly-by-wire). To make sure they understand the real-time implementation, students write some code without using autocode generation and also modify some of the automatically generated code on selected projects.

Challenges. The biggest challenge in an embedded control systems course is to strike a balance between teaching control systems principles and embedded systems principles. The approach we have taken is to always maintain a focus on embedded implementation issues when introducing control material and to maintain a focus on feedback control systems when introducing embedded systems material. We have found that for all of the students there is an enormous benefit gained from the model-based approach. Having the students design systems completely in simulation gives them an understanding of the complete system from a "theoretical" perspective. Having them then implement the same system as a working embedded control system then creates for them a strong a connection between theory and practice that they would not gain from simply jumping immediately to implementation, in addition to giving them an appreciation of "real-world" issues. The modeling and simulation step makes the students realize the value of understanding a system design analytically and gives them insights into why the real system behaves as it does. This connection between theory and practice is an invaluable experience facilitated by the technology of model-based autocode generation.

3.3 Distributed Embedded Control

Distinguishing Features. Distributed embedded control systems have multiple CPUs connected by a low-bandwidth real-time communication network. Automotive control computing is a good example of this area (Leen and Heffernan [2002]), often involving several dozen CPUs connected via real-time networks at data rates often below 1 Mbit/sec. CPUs within such a system often include many small (4- or 8-bit), inexpensive microcontrollers coupled to sensors and actuators as well as a smaller number of large (often 32-bit) CPUs to perform compute-intensive functions.

Key Skills and Principles. The key design principles for distributed embedded control center on the need for coordinated system-wide real-time performance with extreme resource constraints. Many of the underlying theoretical principles of distributed computing apply, but with dramatically altered application. For example, typical desktop and enterprise computing systems are transaction-oriented (also known as being "event-triggered"), use dynamic software allocation within hardware components, use more or less homogeneous computing nodes, optimize throughput, and use Ethernet-based network technology. In contrast, distributed embedded systems are often based on periodic state variable updates (also known as being "time triggered;" Kopetz and Bauer [2003]), use static software allocation, execute on diverse heterogeneous processing nodes, concentrate on meeting a set of system-wide coordinated end-to-end deadlines, and often use non-Ethernet-based network technology. Understanding distributed embedded systems requires revisiting many traditional topics in a different, embedded, frame of thought.

Teaching Approach. Teaching distributed embedded systems requires imparting a detailed understanding of communication protocol tradeoffs and behaviors to support prediction of system-level timing properties. For example, a commonly used embedded real-time protocol is Controller Area Network (CAN) [Bosch 1991], which has a maximum message payload size of only 8 bytes, but is just right for many embedded systems because it supports prioritized message transmission and efficient communication of short messages.

The best-known text in this area is by Kopetz [1997], which covers distributed embedded systems with significant emphasis on the approach used by his Time-Triggered Architecture. Key topics include distributed time, atomic broadcast, group membership, and the need for some aspects of synchronous system operation to provide essential services. Additional topics that are important include end-to-end real-time operating system support, security, graceful degradation in the face of component failures, and management of redundancy in support of dependability.

This material is currently taught as a component of a capstone design course taken after the single-processor embedded system course. Students experience creating specifications, designs, and implementations for relatively simple fine-grain distributed components, and then watching simulations of complex emergent behavior from the interaction of those components. A portion of the assessment for the course is based building an elevator system simulation composed of more than 100 nodes, including demonstrating graceful degradation during fault injection trials. The project includes phases representative of industry with deliverables including requirements, architecture, design, test planning, implementation, failure mode analysis, system integration, verification/validation, and acceptance testing.

Challenges. The challenges in teaching this area include making distributed system theory accessible to students and providing a reasonably large testbed without incurring the space, cost, and maintenance overhead of a huge distributed hardware physical lab. This problem is addressed via the use of a

simulated system rather than a physical one. This strategy is effective, in part, because students have already had experience with embedded hardware in a prerequisite single-processor embedded system course.

3.4 Systems-on-Chip

Distinguishing Features. The Multi-Processor System on a Chip (MPSoCs)/ Systems-on-Chip (SoC) area is an outgrowth of the areas of Application-Specific Integrated Circuit (ASIC) design and embedded systems design. ASIC design was focused on creating a single chip hardware design dedicated to prespecified functionality. As technology has progressed, single-chip systems with 5–10 heterogeneous processors connected via buses to shared memories have become available [e.g., Wolf 2003]; future systems will likely have tens or even hundreds of processing elements of a dozen or more types. Thus, ASICs and many embedded system chips are evolving to become highly concurrent, programmable MPSoCs. Indeed, one vision for these systems is that of custom-designed supercomputers on single chips for portable and handheld devices [Austin et al. 2004].

Since these systems are on the cutting edge of IC manufacturing, design tools and strategies for these systems lag behind other more established areas. Current textbooks are organized around a collection of research topics and are primarily targeted at advanced (year 5 and later) students [e.g., Jerraya and Wolf 2004]. Thus, it is challenging to introduce MPSoC design into the first 4 years of a curriculum.

Key Skills and Principles. Hardware design knowledge is required as a prerequisite for MPSoC design. The knowledge needed is at the level of composing Intellectual Property (IP) blocks. At this level, IP may represent a fully custom-hardware portion of the system, such as a JPEG encoder, or an individual processing element. Unfortunately, since there is no widely accepted bus or network on chip standard [Benini and De Micheli 2002] these building blocks seldom "snap" together. Thus, some detailed hardware design is required.

As in most areas of embedded systems, software carries out much of the real-time functionality for MPSoCs. Since the software in these systems can be updated, a performance-oriented design of the system's hardware and software must be done with an eye to future applications—designing in hardware capacity for anticipated functionality.

The hardware/software codesign research area initially took the view that software and hardware could be conceived as monolithic, side-by-side design entities. However, this did not fully address the design issues, such as scheduling, that began to appear for these emerging systems. Currently, there is no well-established set of principles or even a common design language for MPSoC design.

Teaching Approach. In the absence of a well-established fundamental set of principles, instruction in the design of MPSoCs at Carnegie Mellon is primarily at the level of a fourth-year capstone design project course that concentrates on a large, team-created project.

Students who design an MPSoC learn to consider multidisciplinary tradeoffs among the areas of computer hardware design, computer system architecture, real-time embedded software design, and system modeling. Issues, such as the software's level of concurrency or portability in the context of overall concurrent system performance, must be considered. Additional considerations include the possibility of simplifying software at the expense of additional hardware, or trading off more complex scheduling decisions for fewer hardware resources. The evaluation of system performance, itself, is a new fundamental principle for many students, as are optimizing for maximum speed of execution, and minimum design effort.

The design of an MPSoC prototype in a capstone design course provides the basis for application of fundamentals as well as "just-in-time" learning of new technology, applications, and design tools. Since we cannot build an actual MPSoC due to the time and expense involved in chip fabrication, we apply a similar process using CAD tools to create designs built from processors, embedded controller chips, FPGAs, and SRAM. Although the whole class works from the same high-level design specification, alternate implementations are possible, as each team is expected to customize their design based on a set of self-selected quantifiable design goals. Project topics have included: the game of GO, chess, MP3 players, voice over IP, and face recognition.

The method of delivery for the capstone MPSoC course reflects the unique set of skills being taught. There are few formal lectures, no exams, no homework, and not even any formal lab sessions. After the project is introduced in the first week of class, the entire remainder of the class lecture time is designed to support the development of the term-long project, with each team having 24-hour access to a hardware lab bench for project work. Student assessment includes: in-class presentations, design reviews, interim reviews, weekly email journals, four demonstrations throughout the course, and a final project demo at the end of the course.

Challenges. The challenge of teaching this course lies in evolving the project specifications and implementation technology as new capabilities arise. Because there is no consensus yet as to the "best" way to design MPSOC systems, students must be taught general approaches that will serve them well as the area evolves and matures.

3.5 Networking

Distinguishing Features. Most commercial network devices are embedded systems. While desktop computers can function as a network device (they can route packets, filter packets, and do network address translation), they tend to be too slow or too expensive to deploy as everyday network devices. Network devices must have fairly predictable response time so they can keep up with the rate of packets flowing through the device, whereas traditional operating systems tend to be too complex to consistently keep up with high data rates. Moreover, the amount of processing per packet is typically small and, for many packets, processing is limited to the header, so the data locality of memory access is low, rendering caches ineffective. Because of these unique requirements,

high-end network devices tend to be multiprocessor architectures, using a combination of special-purpose ASICs, specialized programmable processors, and general-purpose processors.

Key Skills and Principles. The skills we teach to embedded system students in this area fall into three categories. First, students have to learn about the unique architectural considerations and tradeoffs that exist in network devices, both at the system and at the processor level. A traditional computer architecture course targets processors with large caches and a lot of material focuses on arithmetic and branch type instructions. In contrast, network processors have significant on-chip support for moving data and for rich interrupt support. Moreover, traditional multiprocessors are homogeneous, while most network processors are heterogeneous, combining a variety of specialized processors on a single chip. Second, students learn the different types of functions that are performed on network devices. Functions are typically classified as data processing and control functions. Data processing is time-critical since it has to keep up with the packet flow (e.g., route lookup based on the destination address in a packet). In contrast, control functions run in the background (e.g., building the packet forwarding tables that are used for route lookup). The third category of skill is mapping functions onto appropriate processors in the architecture. For example, packet forwarding is usually done on a microengine of special-purpose hardware, while routing tables can be calculated on general-purpose processors. Other implementation-oriented skills include programming the different types of processors and coordinating their operation.

Teaching Approach. We teach a fourth-year capstone design course on "Network Design and Evaluation" that uses a platform built around a network processor and a 4-by-4 router with Ethernet interfaces to introduce students to network device internals [Steenkiste 2003]. The platform currently in use has multiple microengines programmed in both microcode and C. Processor coordination is performed via shared memory, message passing, and interrupts.

Challenges. The biggest challenge in this area is dealing with the complexity of realistic network processor platforms. We address this by aggressively presenting platform-specific tutorials to students. In most teams, each team member specializes in specific aspects of the platform.

3.6 Embedded PCs

Distinguishing Features. Embedded PCs include applications such as wearable computers in which a personal computer hardware and software platform is adapted for an embedded environment. These applications are generally characterized by either the use of nontraditional peripherals [e.g., a head-mounted display (HMD) or small touchscreen rather than a full-size screen], or by dedication to a single task (e.g., use as a cash register). This category can easily blur with traditional computing if a desktop computer is used both for embedded applications, such as home climate control, as well as traditional desktop applications.

Originally, both the research and educational emphasis in this area was on building custom-printed circuit boards and custom housings (for example, the early VuMan wearable computers evolved in sophistication over a series of semester-long project courses, as reported by Siewiorek et al. [1994], Smailagic et al. [1995], and Amon et al. [1995]). Over time, wearable computer hardware, PDAs, and other highly mobile computing hardware have started to become commercially available products. Therefore, educational emphasis has shifted to integrating available or incrementally modified technology to address specific application needs in a quick-turnaround fashion (early examples of this include Smailagic et al. [1998] and Siewiorek et al. [1998]).

With the advent of rapid design methodologies and rapid fabrication technologies, it is possible to construct fully customized systems in a matter of months. We have developed a User-Centered Interdisciplinary Concurrent System Design Methodology (UICSM) that takes teams of electrical engineers, mechanical engineers, computer scientists, industrial designers, and human–computer interaction (HCI) students who work with an end-user to generate a complete prototype system during a 4-month long capstone design course run in conjunction with industry partners.

Key Skills and Principles. The design methodology that forms the basis of our teaching is web-based and defines intermediary design products that document the evolution of the design, including not only software and digital hardware, but also mechanical issues, thermal management, and other relevant aspects of system design. The methodology has been used in designing over two-dozen mobile and wearable systems with applications as diverse as aircraft manufacturing at Boeing, bridge inspection with PennDOT, offshore crane operation for Chevron, river environmental data collection on the Pittsburgh Voyager vessel, and new car/driver interactions for General Motors. The methodology includes monitoring and evaluation of the design process. While the complexity of the prototype artifacts has increased by over two orders of magnitude over the years, the total design effort in terms of person-hours per project has increased by only a factor of two.

While the artifact of the design process changes from year-to-year, the fundamental learning objectives do not. Upon completion, students are able to generate systems specifications from a perceived need, partition functionality between hardware and software, produce interface specifications for a system composed of numerous subsystems, use CAD tools, fabricate, integrate, and debug a hardware/software/mechanical system, and evaluate the system in the context of an end-user application. Within the course, students exercise their primary discipline skills, such as building small, embedded printed circuit boards with processors, memory, sensors, and wireless communications, housings for electronics, wireless communication services, software services, and novel user interaction modalities and interfaces.

Teaching Approach. The course evolves around our UICSM, with the industrial partner introducing the problem and interacting with students throughout the semester. Whenever possible, the students visit an actual work site for

firsthand observations, including, in one case, an offshore oil rig. People who will actually use the system, critique the design at each phase, with critique sessions including travel to campus and to remote application sites.

The course itself is divided into three phases: (1) conceptualization, (2) detailed design, and (3) implementation. During each phase, a set of work products are produced and placed on the course web site. At the end of each phase, student teams write a design document and orally present a design review that are critiqued by the instructors and industry representatives.

During the conceptualization phase, the class is introduced to the problem through presentations by faculty and the industry partner. Brainstorming and other methods are used to develop a visionary scenario from which functionality, cost, performance, and techniques for prototype construction are identified. Instructors introduce appropriate technology and students conduct research with instructors serving as consultants. Students are organized in discipline-specific teams with four to five participants. Students specify the system architecture and subsystems for interaction, hardware, software, and mechanical. Performance, interface specifications, and evaluation criteria are also defined.

The second phase leads to a detailed design document. Student teams for this and later phases are multidisciplinary, organized around functional capabilities identified in the first phase. Instructors provide "risk management," ensuring that students do not make decisions that will lead to undue difficulties. Students use task-appropriate CAD tools. Component mock-ups are used to conduct HCI studies.

The final phase consists of four main activities: implementation of subsystems using rapid-prototyping techniques; integration of subsystems; system evaluation through user experiments; and quantitative evaluation of the course methodology. The final presentation, final report, and prototype system form a comprehensive deliverable for the industry partner.

The instructors meet with students at the end of each phase to provide feedback through oral "annual reviews" including their current grade. Grading is based on: (1) visible, concrete contributions to the final project; (2) performance as a leader, presenter, or editor (students rotate through these positions); and (3) incremental activities in the work log.

The industry partner provides the application domain, domain expertise, and background material. In addition to providing companies with access to potential new hires, student projects are a way to explore high-risk ideas. For students, this is a truly unique course that exposes them to multiple disciplines. They take a project from concept to functional product prototype in just one semester. The class's start-up atmosphere—complete with a celebration and t-shirts for the "first customer ship"—excites students. Students and their potential employers recognize the course as valuable preparation for work in today's corporate environment.

Challenges. Managing multidisciplinary teams of students on a large coordinated project is a challenge both because students from different disciplines (e.g., fine arts designers and computer hardware engineers) are not used to working with each other and because of aggressive project goals. Making the

course succeed requires a substantial amount of faculty time. A future challenging topic for this course will be creating more "context-aware" computing (one in which an embedded computer is aware of its user's state and surroundings and modifies its behavior based on this information). New projects, exploiting context information to significantly reduce demands on human attention, will bring new challenges to this class.

3.7 Critical Systems

Distinguishing Features. Critical systems include traditional safety and mission-critical systems such as nuclear power, medical devices, aviation, and some process control applications. This area is characterized by an application need for assured levels of safety and, often, very high dependability as well. It is distinct from the cross-cutting skill of dependability discussed later in that emphasizes specific safety analysis techniques, which sometimes result in systems being shut down (a dependability violation) to maintain safety. One could also envision combining critical systems and dependability, but that is not the way we approach the material.

Key Skills and Principles. Key concepts taught include the principles of software safety, the general approach of embedded software safety standards, and common analysis techniques such as Failure Mode Effect Analysis (FMEA) and Fault Tree Analysis (FTA).

An additional topic that is essential to discuss in a critical systems course is ethics. Most safety critical situations involve inherent tradeoffs between safety, functionality, time to market, and cost. In more mature safety critical application areas, engineers can face ethical issues in ensuring that safety standards are being followed with an appropriate amount of rigor in the face of schedule and cost pressures. But there are other application areas in which such standards are immature, not widely followed, or nonexistent, and engineers working in these areas can face significant ethical dilemmas. A key concept that students are exposed to is that many safety decisions implicitly or explicitly put a monetary value on each human life lost or saved by a system design decision.

Teaching Approach. We incorporate an introduction to critical systems into our capstone design course on distributed embedded systems, since many such systems have safety critical aspects (for example, steer-by-wire is a safety critical automotive application in addition to being a distributed embedded system application). The capstone design project includes an FMEA exercise along with fault injection so that students can see how accurately their FMEA-predicted system responses to injected faults. An alternate approach for a course dedicated to safety would be to use the text by Storey [1996].

Challenges. As more systems acquire functions that are directly or indirectly safety critical, a significant challenge is exposing a broader base of nonspecialists to the basic principles of critical system design. Even a basic exposure to the concept of failure mode analysis could go a long way to avoiding the creation of brittle systems that become dangerous when something goes wrong.

3.8 Robotics

Distinguishing Features. Robotics clearly covers many facets of embedded computing, while offering exciting possibilities for motivating students. While a full discussion of robotics is beyond the scope of this paper, robotics is used as a vehicle to teach general engineering skills and general aspects of embedded systems.

Key Skills and Principles. We teach a third-year course entitled "General Robotics." The interrelated goals of this course are to inject basic mathematics into fundamental engineering education and tie basic theory together with pragmatic implementation for students who are not necessarily going to specialize in embedded systems. The approach uses robotic construction experiences to reinforce fundamental topics by having students build a LEGO robot every week. This significantly motivates students and gives them hands-on examples of engineering principles in the context of robotics projects.

Teaching Approach. Unlike conventional "cookbook" chemistry labs, labs are both hands-on and heads-on, requiring students to synthesize approaches and solve problem statements rather than follow prescriptive directions. Student self-evaluation of success is made easier by the fact that students can observe whether or not a robot is actually working, whereas theoretical homework can often be assessed only via a human grader. In addition, homework serves as small creative design experiences that expose students to working in teams.

There is a separate fourth-year capstone design course in robotics. A recent capstone project example is a mock search-and-rescue experience where student teams designed a robot to traverse rough terrain. Students drove the robots, but only using images from a camera on board the robot, with the students in a separate physical space from the operating robots, as would be the case in a real search-and-rescue scenario. The projects provided students with many lessons of robot integration which are difficult to teach in the classroom, but can be learned through direct experience. Results of that project have influenced the design of search-and-rescue robots.

3.9 Computer Peripherals

Distinguishing Features. A significant fraction of embedded computers are placed in printers, disk drives, and other similar computer peripheral applications. The computer peripheral industry has a market that differs dramatically from many other embedded systems, because it is driven by the market cycle and economics of the desktop computing industry. For example, disk drives themselves become obsolete in only a few years due to density improvements in newer drives rather than by wear-out of installed embedded hardware and software. Because peripherals are connected to computers that are, in turn, usually connected to the Internet, deploying software patches for peripherals is considerably easier than for most other embedded applications (for example, it is standard practice to update the firmware of a newly installed peripheral as part of installation or troubleshooting—something relatively rare in most other embedded applications).

Key Skills and Principles. We have neither identified a unique set of embedded system engineering principles that applies to the computer peripheral area, nor are we aware of a specific text or teaching methodology that is distinctively tailored to this area for students in years 1–4.

Teaching Approach. We rely upon students learning other areas (primarily the small microcontroller area, as well as non-embedded computer engineering areas) to gain the skills needed to work in this application area.

3.10 Signal Processing

Distinguishing Features. Signal processing as a discipline involves filtering, coding, detecting, analyzing, and otherwise using computers to process signals that include audio, video, and other data streams. Courses to teach the mathematics and theory of this area are well established. Beyond those prerequisite courses, however, there is an opportunity to teach how to implement theory subject to the constraints and limitations of real computing platforms. Representative application areas include radar, sonar, and image compression. Very often, such systems employ specialized embedded processing hardware, including Digital Signal Processing (DSP) chips.

Key Skills and Principles. We teach this area via a capstone design course that centers on using DSP hardware to implement a student-selected project of their choice. Topics have included: speech and music processing, digital communications, multimedia processing, data compression, data storage, wireless communications, CD drives, image processing, and signal processing. One month of introductory laboratories familiarize the students with DSP hardware and support software. Lectures address Z-transforms, IIR and FIR filter design using MATLAB and DSP hardware, LPC and adaptive filters, channel coding, time and frequency multiplexing, short-time Fourier and wavelet transforms, and spread spectrum techniques.

Teaching Approach. As one might expect, the first of two prerequisite courses is an introductory signals and systems course. Because this is a broad capstone design course, the second prerequisite is flexible, including a course in wireless communications, an introductory computer science data structures course, optical processing, multimedia encoding, image processing, or advanced digital signal processing. Projects are generally expected to build upon the combined strengths of the members of each project team in these or related areas.

Challenges. A key challenge in this area is similar to that experienced in the controls area—bridging the gap between the abstract mathematics of signal processing and the gritty details of implementation on resource-constrained hardware. This is currently addressed via students having to come to terms with real hardware limitations as part of a capstone design experience. In the future, bridging the two areas with lecture-based course material might also be useful.

3.11 Command and Control

Distinguishing Features. Command and control systems are exemplified by relatively complex military combat and aerospace systems, and so-called "systems of systems" built by defense contractors. This area is characterized by a need to integrate very large systems from different subsystem suppliers, often with soft real-time constraints and an emphasis on data coordination and aggregation.

Key Skills and Principles. It seems difficult to separate the specific embedded system skills necessary for this area from the more generalized skills required for complex software systems and general-purpose computing. Specifically, because they tend to be built from "Commercial Off The Shelf" (COTS) components, these systems increasingly use desktop and enterprise technologies.

Teaching Approach. We have not been able to pin down the essential topics for this area, nor do we offer a course in it. Nonetheless, this area represents a distinct approach to embedded systems, is one that employs a significant number of graduating students, and might be well served by being directly addressed at years 3–4 in some institutions.

One way this area can be addressed, in part, is to tailor an embedded real-time course to command and control applications, along the lines of textbooks, such as the one by Cooling [2003]. Of course, many of the difficult issues of command and control systems go beyond real-time computing, and result from the immense complexity encountered in system integration. Dealing with system complexity traditionally falls under the realm of software engineering, which we treat in years 5–6 at Carnegie Mellon with a curriculum created by the Software Engineering Institute [Ford and Gibbs 1989]. While some elements of that curriculum cover embedded computing, that program is not fundamentally focused on embedded systems.

Challenges. The principle challenge to teaching command and control system skills to embedded system engineers is that of motivating those students to invest serious attention in what amounts to software engineering, whereas most embedded engineering students are trained as and think of themselves as hardware engineers. Creating students with strength in both hardware and software engineering (as opposed to mere programming or only software systems) remains an open challenge.

3.12 Wireless Data Systems

Wireless data systems are a relatively new area of study that involve collections of nodes used to sense, aggregate, and distribute data. A popular alternate name for this area is "sensor networks." This is a relatively new area that we have taught to students in year 6 and later. Topics of interest in this area include: wireless network operation, image processing, security, privacy, distributed databases, portable data representation, and distributed systems. However, we have not yet attempted to teach this topic to year 1–4 students.

Hemingway et al. [2004] have addressed this topic in a capstone design course at the University of Washington.

4. CROSS-CUTTING EMBEDDED SKILL AREAS

Beyond specific application areas, there are cross-cutting technologies, skills, and teaching areas that are not applications in themselves, but rather tools and techniques that are frequently used by embedded system designers. In general, the difference between these skills and the previous embedded computing areas is that most people would not consider a course with one of the below titles to be an "embedded" course. Nonetheless, we have found in our interactions with industry that all of the skills are critical ones for practicing embedded system engineers. Thus, all these skills must be taught either within the embedded computing courses themselves or as strongly recommended/required auxiliary courses to produce well-rounded students.

4.1 Security

Distinguishing Features. Security for embedded applications presents different requirements and constraints from traditional applications [Koopman 2004]. First, many embedded applications execute in resource-constrained environments (both in terms of CPU power and memory capacity), making liberal use of heavy-weight security mechanisms, such as public-key cryptography, infeasible in many cases. Second, embedded applications typically have stringent reliability requirements. For instance, a controlled battery bank that provides power to critical applications may be required to stay up for 99.99999% of the time, even in the face of malicious attacks, while typical desktop applications can tolerate occasional interruptions. Last, embedded applications are often cost sensitive—a 10-cent increase in the manufacturing cost due to security is sometimes prohibitively expensive.

More than 20 years of security research suggests that it is impracticable to add security as an afterthought to an existing system design. Thus, it has to be built in by the original embedded system designers. However, embedded security is a subject area that is rarely taught (if at all) in academic institutions. Instilling some level of understanding of security concepts in embedded system engineers is a critical challenge for the near future as well as long-term educational planning.

Key Skills and Principles. The principal security issues behind embedded applications can be loosely categorized as follows:

Authentication and Access Control. This deals with determining the identity and access rights of the entity involved in certain operations. Many embedded applications have subtle authentication and access control issues. For example, Internet-capable devices often have no clear definitions of a user (consider embedded processors in a dishwasher), and yet can be accessed remotely [Bergstrom et al. 2001]. The consequence of being controlled by the wrong user can range from relatively benign (hot water wasted by an extra washing cycle) to disastrous (consider power loss for a life-support machine).

Data Privacy and Integrity. Many embedded devices store or communicate sensitive data. For instance, automobile location information can reveal where a person is at what time. Even seemingly benign devices can sometimes reveal information with privacy implications [Koopman 2004].

Software Security. Malicious code presents an ever-increasing threat as more and more embedded systems become interconnected. For example, a remote software update capability for automobiles is being pursued by car manufacturers. However, the security implications for remote programming are so significant that the industry is proceeding with extreme caution.

Security Policies. Security policies govern system operation. Cookie-cutter policies for desktop computing may not apply, because requirements for embedded applications can be considerably different.

The above issues are not fundamentally different from those found in traditional computer/network security. However, many established security solutions do not carry over to the embedded domain. In the near future, embedded security will become a stand-alone area for research and, possibly, teaching.

Teaching Approach. At this time, we teach security as a separate subject and students must map concepts onto embedded systems on their own. This can be challenging because general security courses, by and large, ignore memory and CPU constraints as well as other embedded-specific concerns.

Challenges. The main challenge for teaching embedded security is that the subject area has not been studied extensively and therefore lacks good teaching material. There is no textbook written specifically on the subject and no published body of knowledge summary that recommends course topics in this area. We believe that a course on embedded security should be based on a cross-fertilization of security and embedded computing, incorporating fundamental aspects of both. However, this is a new curriculum area and there will no doubt be many lessons to learn along the way.

4.2 Dependability

Distinguishing Features. Dependability includes fault-tolerant computing techniques for both hardware and software, along with related techniques to increase the amount of reliance that can justifiably be placed on a computer system to meet its specification. While critical applications emphasize dependability and safety, many noncritical embedded applications also need some level of assured dependability. Because dependability can apply to any application area and any system, we treat it separately from critical systems (discussed earlier), which traditionally have niche application domains.

We note that the term "dependability" is often considered to encompass security as well as other areas. From a teaching point of view, the dependability community (with a fault tolerant computing heritage) and the security community have not yet merged, so we treat these as separate educational areas. This will change as faculty from both areas are cross-trained and new course materials are developed.

Key Skills and Principles. Several aspects of dependability are important in building real-world, mission-critical systems. Of particular importance are reliability engineering, which aims to make systems less failure-prone, and fault-tolerance, which aims to compensate for failures when they do happen.

The fundamental building blocks needed for understanding dependable distributed systems include atomic multicast, reliable group communication, replication, architecture-based reliability, fault detection, network protocols, fault injection, fault models, and transaction processing. Each of these techniques is useful in specific contexts, and some of these techniques can be combined to obtain adequate guarantees of dependability. It is essential for students to learn these concepts (and to have actually implemented them in practical systems) before they can be entrusted with high dependability applications. As a part of this area, students should learn how to critically examine the dependability trade-offs (e.g., performance versus reliability, real-time versus reliability, availability versus safety) involved in making engineering and design choices.

Teaching Approach. An introduction to this area is incorporated into the distributed embedded system fourth-year capstone design course, along with an introduction to safety-critical computing concepts. This is because an increasing number of distributed embedded applications require heavy-weight approaches to dependability. (For example, automotive X-by-Wire systems are likely to incorporate group membership as a fundamental system service.)

However, there is far more specialized material than can be taught in Baccalaureate courses during the first 4 years of study. Therefore, we teach an addition fifth/sixth-year course on dependable distributed systems. Over time, we are transitioning the core concepts such as group membership from that later course into the fourth-year course. The topics in this course include: (1) individual and combined aspects of real time, performance and reliability, (2) basics of distributed systems, including concepts such as asynchrony, (3) tools and techniques for analyzing dependability, and (4) critique of current distributed technologies from the respective viewpoints of real time, fault tolerance, and scalability. While the emphasis is on fundamental concepts to enable students to apply their skills throughout their careers, projects use new technologies, e.g., recent release versions of middleware.

A substantial portion of course content involves a cooperative team software system implementation project. The project requires the design, implementation, empirical evaluation, and end-to-end analysis of a real-time, fault-tolerant high-performance distributed application. Lectures, along with regular project meetings with the instructor, allow students to design and implement realistic middleware applications (often equivalent to their commercial counterparts), to develop infrastructures to make these applications dependable, and to evaluate the effectiveness of their techniques. This allows students to get first-hand insight into the real-world aspects of reliable systems and to learn to appreciate both the practical and theoretical aspects of dependability. While the context for this course is enterprise computing, the material taught is applicable to distributed embedded systems and, for that reason, is taken by many embedded systems students.

While this course is targeted to fifth-year students, increasing numbers of fourth-year students have been taking it to target careers in specific industries. For instance, students seeking positions in mission-critical industries (e.g., defense contractors, financial industries, telecommunications companies) often take this course in their fourth year, prior to graduation. Software engineers from local industries also take this course, upon recommendation from their employers, who want their employees to have formal dependability knowledge and exposure to practical, hands-on experience. The combination of diverse sets of students (graduating fourth-year students with jobs awaiting them, fifth-year students, and employed software engineers with domain expertise) makes for a combination of complementary strengths, especially in team project settings to solve dependability problems.

Challenges. A future challenge will be addressing the unique needs of middleware that executes on embedded systems (for example, middleware in the infotainment system of an automobile). Some concepts will carry over intact from enterprise middleware, but some will need to be revisited entirely to be useful in an embedded environment.

4.3 Energy-Aware Computing

Distinguishing Features. Traditionally, mobile, portable, and battery-powered embedded systems have provided incentive for low-power design and energy-aware computing. Energy management has become a first-class design constraint and a mandatory ingredient of embedded system curricula. Energy-aware computing provides power reduction techniques that can be used and orchestrated in order to achieve the best performance within a given power budget, or the best power efficiency under prescribed performance constraints.

Key Skills and Principles. An energy-aware computing course must address not only low-power design aspects, but also needs to introduce students to various techniques that can be used in concert to achieve the best set of power-performance operating points. A representative example is the use of voltage scaling for power reduction. Since power consumption varies as the cube of supply voltage (assuming that speed is also scaled accordingly), scaling down the voltage is the technique that achieves the most dramatic results for power savings. However, such techniques are only practical when combined with a deep understanding of not only the low-level circuitry involved to support them, but also the application profile and platform used for designing the system.

Thus, a course on energy-aware computing must expose students to both: (1) available "knobs" or parameters subject to change that are provided by circuit technology; and (2) customized, advanced techniques that take advantage of these "knobs" and provide an overall energy-aware solution.

Teaching Approach. Although a mainstream textbook for energy-aware computing has yet to be published, there is a wide variety of material that one could use in support of such a course. We have used a few monographs that touch upon low-power design aspects such as the one by Rabaey and Pedram

[1996], as well as power-aware computing methodologies, such as the ones by Pedram and Rabaey [2002], and Melhem and Graybill [2002], in conjunction with literature on low-power and power-aware design. We start with transistor-level power modeling and progress through all levels of abstraction (gate, microarchitectural, and system level), culminating with dynamic power management through operating system control. Thus, the prerequisite knowledge of students must cover (albeit, not necessarily at the same level of expertise) various areas which, until recently, have been in separate tracks in traditional computer engineering curricula. The most important tool for assessing how students have assimilated this knowledge is a semester-long integrated circuit design project, which, depending on the level of abstraction, could be a pure design project fully simulated (if possible, including postlayout simulation), a simulation framework, or implementing a power management mechanism in a real prototype (in which case a demo becomes part of the project milestones).

Challenges. As uncertainties in design increase due to a larger impact of process variability on overall design flow, joint-power and fault-tolerance management will become relevant for future technologies. We have incorporated such ideas in our course in support of these upcoming challenges.

4.4 Software and System Engineering

Distinguishing Features. Embedded systems are just that—systems composed of many components having many different apects, not the least of which is software. For this reason, teaching software engineering principles at some level is vital to teaching embedded systems. Moreover, teaching system engineering principles is similarly important (and, to a large degree, parallels or overlaps with teaching software engineering principles).

Key Skills and Principles. Important topics that are included in various embedded system courses include design and life cycle phases such as requirements, design, implementation, verification/validation, deployment, and maintenance. In addition, interdisciplinary tradeoffs between hardware, software, and mechanical components, as well as human operator actions, are central to many areas of embedded system design. A key economic issue that we discuss is the progressive shift of many systems from having recurring hardware cost as their main economic concern to having nonrecurring software development as a significant system cost, even when software costs are amortized over a large production run.

Teaching Approach. Currently, elements of software and systems engineering are included in different ways in various courses, mostly via capstone design project courses, as discussed in other sections of this paper. A traditional third-year/fourth-year software engineering course is available and taken by many students. However, it is not specifically designed to address embedded system engineering concerns.

Challenges. Expecting every embedded system engineering student to double-major in software engineering as well is unrealistic. Yet, it is common

for practicing engineers to need both skill sets, typically acquiring one or the other set of skills on the job. An integrated way to teach both skills to more students without resulting in academic overloads would better prepare them for industry.

4.5 Real-Time Systems

Distinguishing Features. A real-time system is one whose timing behavior is part of its correctness, with many embedded systems having this property. For example, in an automotive engine control system, fuel needs to be injected at the right points in time to improve both engine power and fuel efficiency while avoiding engine damage. Examples of real-time systems can be found in robotics, transportation and motion control systems, automated manufacturing, process control, nuclear reactor control, aerospace applications, and defense systems, among others. In general, any embedded control system is a real-time system to some degree.

Key Skills and Principles. Real-time systems are often called upon to provide predictable and guaranteed worst-case real-time response to critical events, acceptable average-case response times to noncritical events, and satisfaction of critical needs if transient overloads occur. Relatively large real-time systems employ real-time operating systems that support capabilities, including multitasking, priority-based preemptive, or time-driven scheduling, real-time synchronization primitives that minimize priority inversion [Sha et al. 1990], real-time communications that support priority queuing and priority inheritance [Object Management Group 2002; RTJ 2000], high-resolution timers and clocks, and predictable memory management including wiring down of memory pages, periodic threads, and exceptions for missed deadlines.

Teaching Approach. Third-year students are exposed to a significant amount of real-time content as part of the single-processor embedded systems course already described. They get further exposure to end-to-end real-time scheduling in both the distributed embedded control capstone design course and the MPSoC capstone design course, as well as fifth-and-sixth year courses.

Students in the third-year introductory course learn concepts including real-time scheduling theory (Klein et al. 1993), principles of resource management, and composability of real-time properties. Students also learn various real-time system approaches, including the use of cyclic executives, multitasking systems, resource management schemes, client–server paradigms, and pipelined approaches. Additional topics include real-time synchronization schemes, such as mutual exclusion, producer-consumer patterns, and techniques to avoid both buffer overflow/underflow and priority inversion.

A series of five course projects give students hands-on exposure to real-time system skills. Students use tools to analyze worst-case timing behavior and various debugging strategies, including processor simulators. An important lab project involves implementing a real-time low-footprint embedded operating system that implements context-switching, task scheduling, real-time

operation, and semaphore management. Students also build interactive timed games and complete audio and/or video-based multitasking applications to test and validate the functionality of the underlying RTOS that they themselves built. Our treatment of real-time systems in combination with single processor system topics in a third-year course of necessity limits the amount of material that can be covered. A possible alternate approach for other institutions is to teach a course that is more exclusively focused on real-time systems, using one of the many available texts such as those by Liu [2000] or Burns and Wellings [2001].

Challenges. An important challenge is making real-time system theory, which can be quite math-intensive, accessible to third year nonspecialist students. This has been done to this point by devoting multiple class meetings to going over the intricate mathematical details, but other approaches might be more appropriate for other curricula.

4.6 Human-Computer Interaction

Distinguishing Features. It is common for embedded computers to have an interface for directly interacting with humans (for example, a digital watch, or a sewing machine with touch-screen LCD). Many embedded systems also interact with humans indirectly through system behavior or less computer-specific interaction devices (for example, a toaster, or an antilock braking system). Any time an embedded system interacts with people, using good approaches to human computer interaction is an important consideration.

Key Skills and Principles. Carnegie Mellon is fortunate to have a department-level emphasis on HCI in the form of the Human Computer Interaction Institute (HCII). Several courses in the area of HCI are offered, although none is specifically designed for embedded system engineers. An essential tenant in HCI is involvement of end users in the design process. We teach students that "The user is not I."

Teaching Approach. Techniques in user-centered design include contextual enquiry, coalescing concepts via affinity diagrams, and creating scenarios from which capabilities and, subsequently, functional requirements, are identified. There is an undergraduate second major in HCI that enrolls one-half dozen electrical and computer engineering students. Courses in the major are drawn from social science and psychology (Humanities College), design (College of Fine Arts), and computer science.

Two fourth-year capstone design courses already discussed have specific HCI content. The Rapid Prototyping of Computer Systems capstone design course specifically includes students from the School of Design, and is cotaught by the Department Head of the HCII. Projects in that course emphasize the human aspects of design and system construction, usually of embedded computing systems. The distributed-embedded systems capstone design course includes a full lecture on HCI and other courses, include varying levels of HCI content as well.

5. LESSONS LEARNED AND GENERAL OBSERVATIONS

Perhaps the salient feature apparent from our description of embedded system education is that it is not a single, monolithic approach. Teaching approaches, assessment approaches, and goals vary dramatically. We consider this a strength, because it gives our students exposure to a wide variety of experiences. Life for an engineer in an embedded consumer electronics startup company can be expected to differ dramatically from life at a large, established company producing safety-critical medical devices. Similarly, our course experiences differ based on application area and recognized faculty strengths in various technical and educational areas. They also differ based on position in the curriculum, from introductory courses to fourth-year capstone design courses in which up to one-half of enrolled students are actually postbaccalaureate students, to fifth-year courses in which up to one-half of the enrolled students are actually baccalaureate students. They also differ based on the demographics of students, which have included mainstream students enrolled in ordinary courses, experienced industry professionals, and overseas distance education students.

Some of the high level lessons learned across our various teaching experiences are listed below:

- Students who have experience in nonembedded versions of a skill area do not necessarily have an advantage. A previous course (in operating systems, or distributed systems, for example), might give them a grasp of some topics, but also gives them preconceived ideas that have to be unlearned for the embedded system environment (for example, techniques that consume too much memory, network bandwidth, or CPU power).
- Students retain knowledge better when working through actual implementations in realistic environments that force them to confront the very real limitations and quirks of embedded systems.
- Enforcing a methodical design process noticeably increases the success rate of capstone design projects (i.e., projects that actually "work" at the end of the semester). However, students who perceive they are "wasting" time on process instead of creating more system features can lose motivation, so finding the right amount of process is a difficult balancing act.
- Significant projects can be completed with dramatically different team sizes, with team sizes of 2 to 4 students common for capstone design projects. Thirty-student projects can also be successful, but require significant attention from staff or faculty to perform management coordination and enforce an appropriate system engineering process.
- Because embedded system terminology can be nonintuitive and differ from desktop computing, students are well served by having access to a lexicon, such as the one by Ganssle and Barr [2003]. This is especially important for students who are newly transitioning to the language being used for instruction (English, at our university).
- Noncomputer engineering students (for example, mechanical engineers from industry enrolled part-time in a fourth-year course) might not have any

background in discrete mathematics. This can be an unpleasant surprise for both students and faculty, because embedded-systems faculty usually take discrete mathematics preparation for granted.

- Because the embedded system industry gets little press in most professional publications, guest speakers from embedded system industries provide invaluable motivation and insight when they visit classes.
- When possible to arrange, using industry partners as customers for embedded-system projects injects a level of realism not possible in any other way. Most students do not have an intuitive sense for the requirements and tradeoffs inherent in embedded system design and such issues carry more credibility when presented by an industry practitioner.

Beyond exposing students to engineers from industry in any reasonable way possible, it is important to expose students to realistic problems, and realistic problem-solving situations. Students need to learn about themselves, including their own strengths and weaknesses, as they respond to different situations. Someday they will not be able to rely upon faculty or teaching assistants to help them out, nor will an 84% correct piece of software (which can be an acceptable grade in many situations) be good enough shipping a safety-critical application. For these reasons, educators need to, as closely as possible within the confines of an academic setting, approximate real situations. Approaches to this vary across our embedded curriculum. Some project courses give students a problem and let them make their own mistakes, trusting that with subtle guidance from the course instructors they will emerge with appropriate skills. Some project courses set forth a very rigorous design methodology and make students follow it so that they can try it on for size. Of course, many approaches have value and we strongly encourage students experience multiple such situations to truly learn what they need to acquire lifelong learning skills. In all cases it is important to realize that the specific tools or technologies being taught are transitory. The deeper lessons that must be ingrained in students involve handling open-ended, complex, multidisciplinary, and overconstrained situations, and learning how to adapt techniques as problems unfold. Our feedback surveys of students indicate that, by and large (depending on choices made by the students themselves on their course of study), our graduates have indeed attained a set of skills that has prepared them well for work in various embedded system application domains.

6. CONCLUSIONS

Embedded systems cover an enormous span of applications, technology, and system scale. In three decades of teaching increasing amounts of embedded computing material, we have learned two important lessons. The first is that significant hands-on course content is essential to teaching the systems aspects that form an inherent part of embedded systems. This is reflected, in part, by the fact that much of our embedded system course content is taught as capstone design courses. Even the noncapstone courses usually have significant project content. The second major lesson is that embedded systems can be taught without having a single, monolithic curriculum or, for that matter, strong centralized

planning of course content. Informal interaction among faculty and tailoring of each course to its relevant embedded system area has resulted in a robust set of courses that covers most, but not all (primarily due to limited faculty size), of what we consider to be the important areas that can be covered in an undergraduate embedded system curriculum.

Discovering how to best teach embedded systems is, of course, a continuing journey. Technology, applications, and people change over time. So must embedded system education.

ACKNOWLEDGMENTS

P. Koopman was supported in part by the General Motors Collaborative Research Laboratory at Carnegie Mellon University, Honeywell, DaimlerChrysler, and the Pennsylvania Infrastructure Technology Alliance. B. Krogh was supported in part by grants from General Electric, Lockheed Martin, Ford, the US Defense Advance Projects Research Agency (DARPA) contract nos. F33615-00-C-1701 and F33615-02-C-4029, US Army Research Office (ARO) contract no. DAAD19-01-1-0485, and the US National Science Foundation (NSF) grant no. EIA-0088064. D. Marculescu was supported in part by US National Science Foundation (NSF) Career Award no. CCR-008479 and Semiconductor Research Corporation (SRC) grant no. 2004-HJ-1189. P. Narasimhan was supported in part by the National Science Foundation (NSF) Career Award CCR-0238381 and the General Motors Collaborative Research Laboratory at Carnegie Mellon University. J. Paul was supported in part by grants from ST Microelectronics, General Motors, Altera, Xilinx, the Pittsburgh Digital Greenhouse, and The National Science Foundation (NSF) grant no. 0103706 and grant no. EIA-9812939. R. Rajkumar was supported in part by the General Motors Collaborative Research Laboratory at Carnegie Mellon University. D. Siewiorek and A. Smailagic were supported by the National Science Foundation (NSF) grants no. 0205266 and 0203448, the Defense Advanced Research Project Agency (DARPA) contract no. NBCHC030029, and the Pennsylvania Infrastructure Technology Alliance. P. Steenkiste's development of the course on "Network Design and Evaluation" was supported by Intel. C. Wang was supported in part by grants from the US National Science Foundation (NSF) contract nos. CCR-0208853 and ANI-0326472. The insights gained about industry needs for embedded system education have also been informed by collaboration with numerous partners that had a direct impact on what and how we teach, including: General Motors, Ford, Bombardier Transportation, Boeing, Lockheed Martin, General Electric, Chevron, Pennsylvania Department of Transportation, Hyundai Motors, NASA, Bosch, ABB, Emerson Electric, United Technologies, Intel, DaimlerChrysler, AT&T, Honeywell, Lutron, the US Navy, and Inmedius.

REFERENCES

ABET ACCREDITATION COMMITTEE. 2004. *Criteria for Accrediting Engineering Programs*. November 1.

AMON, C. H., FINGER, S., SIEWIOREK, D. P., AND SMAILAGIC, A. 1995. Integration of design education, research and practice at Carnegie Mellon University: A multi-disciplinary course in wearable computer design. *Frontiers in Education Conference*, 1–4 Nov. Vol. 2, pp. 4a1.14–4a1.22.

ARTIST PROJECT. 2003. *Guidelines for a Graduate Curriculum on Embedded Software and Systems*, Project IST-2001-34820 report, review version, May 6. Accessed at http://www.artist-embedded.org/Education/Education.pdf on August 28, 2004.

AUSTIN, T., BLAAUW, D., MAHLKE, S., MUDGE, T., CHAKRABARTI, C., AND WOLF, W. 2004. Mobile supercomputers. *IEEE Computer*, May 2004, 81–83.

BENINI, L. AND DE MICHELI, G. 2002. Networks on chips: a new SoC paradigm. *IEEE Computer*, January 2002, 70–78.

BERGER, A. 2002. *Embedded Systems Design: An Introduction to Processes, Tools and Techniques*. CMP Books, Manhassett, NY.

BERGSTROM, P., DRISCOLL, K., AND KIMBALL, J. 2001. Making home automation communications secure. *Computer*, October 2001, 50–56.

BOSCH, ROBERT GMBH. 1991. *CAN Specification*, Version 2.

BURNS, A. AND WELLINGS, A. 2001. *Real Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time C/POSIX (3rd Edition),* Addison Wesley, Reading, MA.

CATSOULIS, J. 2003. *Designing Embedded Hardware*, O'Reilly, Sebastopol, CA.

CMP 2005. *CMP Media Publication Information*, accessed at http://www.cmp.com/pubinfo/?pubID=50 on February 14, 2005.

COOLING, J. 2003. *Software Engineering For Real-Time Systems*. Addison Wesley, Reading, MA.

DIRECTOR, S. W., KHOSLA, P. K., ROHRER, R. A., AND RUTENBAR, R. A. 1995. Reengineering the curriculum: Design and analysis of a new undergraduate Electrical and Computer Engineering degree at Carnegie Mellon University. *Proceedings of the IEEE 83*, 9(Sep.), 1246–1269.

EMBEDDED SYSTEMS CONFERENCES. 2004. *Embedded Systems Conferences home page*, http://www.esconline.com/ accessed August 28, 2004.

ESTRIN, D., BORRIELLO, G., COLWELL, R., FIDDLER, J., HOROWITZ, M., KAISER, W., LEVESON, N., LISKOV, B., LUCAS, P., MAHER, D., MANKIEWICH, P. L., TAYLOR, R., AND WALDO, J. 2001. *Embedded Everywhere: A Research Agenda for Networked Systems of Embedded Computers*. National Academy Press, Washington, DC.

FORD, G. AND GIBBS, N. 1989. A Master of software engineering curriculum: recommendations from the Software Engineering Institute. *Computer 22*, 9 (Sep.), 59–71.

GANSSLE, J. AND BARR, M. 2003. *Embedded Systems Dictionary*, CMP Books.

GRASON, J. AND SIEWIOREK, D. 1975. Teaching with a hierarchically structured digital systems laboratory. *IEEE Computer*, December 1975, 73–81.

IEEE 2004. Advertise in Computer, accessed at http://www.computer.org/computer/ad.htm on August 18, 2004.

HEMINGWAY, B., BRUNETTE, W., ANDERL, T., AND BORRIELLO, G. 2004. The Flock: Mote Sensors Sing in Undergraduate Curriculum. *IEEE Computer*, August 2004, 72–78.

JERRAYA, A. AND WOLF, W., EDS. 2005. *Multiprocessor Systems-on-Chips*. Morgan Kaufmann, San Francisco, CA.

KLEIN, M., RALYA T., POLLAK B., OBENZA R., AND HARBOUR M. 1993. *A Practitioner's Handbook for Real-Time Analysis*. Kluwer Academic Publ., Boston, MA.

KOOPMAN, P. 2004. Embedded system security. *IEEE Computer*, July 2004, 95–97.

KOPETZ, H. 1997. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Acadimic Publ., Boston, MA.

KOPETZ, H. AND BAUER, G. 2003. The time-triggered architecture. *Proceedings of the IEEE 91*, 1 (Jan.), 112–126.

LEEN, G. AND HEFFERNAN, D. 2002. Expanding automotive electronic systems. *IEEE Computer*, January 2002, 88–93.

LEWIS, D. 2001. *Fundamentals of Embedded Software: Where C and Assembly Meet*. Prentice Hall, New York.

LIU, J. 2000. *Real-Time Systems*, Prentice Hall, New York.

MELHEM, R., AND GRAYBILL, R., EDS. 2002. *Power Aware Computing*, Kluwer Academic Publ., Boston, MA.

MOTUS, L. 1998. Teaching software-intensive embedded systems at Tallinn Technical University, *Proceedings of Real-Time Systems Education III*. Poznan, Poland, 30–35.

NEILSEN, M., LENHER, D., MIZUNOL, M., SINGH, G, ZHANG, N., AND GROSS, A. 2002. An interdisciplinary curriculum on real-time embedded systems. In *Proceedings of the 2002*

American Society for Engineering Education Annual Conference & Exposition, session 1526.

PEDRAM, M., AND RABAEY, J., EDS. 2002. *Power Aware Design Methodologies*, Kluwer Academic Publ., Boston, MA.

PRI-TAL, S., ROBERTSON, J., AND HUEY, B. 2001. An Arizona ecosystem for embedded systems. In *20th IEEE International Performance, Computing, and Communications Conference (IPCCC 2001)*. Phoenix AZ, April 4–6, 131–134.

RABAEY, J., AND PEDRAM, M., EDS. 1996. *Low Power Design Methodologies*, Kluwer Academic Publ., Boston, MA.

OBJECT MANAGEMENT GROUP. 2002. *Real-Time CORBA, CORBA 2.0 Specification*, accessed at http://www.omg.org.

RTJ 2000. *JSR-000001*, *The Real-Time Specification for Java*, accessed at http://www.rtj.org

SIEWIOREK, D. P., SMAILAGIC, A., AND LEE, J. C. 1994. An interdisciplinary concurrent design methodology as applied to the Navigator wearable computer system. *Journal of Computer and Software Engineering*. *2*, 3, 259–292.

SIEWIOREK, D. P., SMAILAGIC, A. ET AL. 1998. Adtranz: a mobile computing system for maintenance and collaboration. *Proceedings of The Second IEEE International Symposium on Wearable Computers*, IEEE Computer Society Press. 25–32.

SMAILAGIC, A., SIEWIOREK, D. P. ET AL. 1995. Benchmarking an interdisciplinary concurrent design methodology for electronic/mechanical design. *Proc. ACM/IEEE Design Automation Conference*. 514–519.

SMAILAGIC, A., SIEWIOREK, D.P., STIVORIC, J., AND MARTIN, R. 1998. Very rapid prototyping of wearable computers: a case study of custom versus off-the-shelf design methodologies, *Journal on Design Automation for Embedded Systems 3*, 217–230.

SHA, L., RAJKUMAR, R., AND LEHOCZKY, J. 1990. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*. *39*, 9, 1175–1185.

SIMON, D. 1999. *An Embedded Software Primer*, Addison-Wesley, Reading, MA.

STEENKISTE, P. 2003. A network project course based on network processors. *ACM Technical Symposium on Computer Science Education (SIGCSE 2003)*. Reno, Feb. 9–23, 262–266.

STOREY, N. 1996. *Safety-Critical Computer Systems*, Addison-Wesley, Reading, MA.

TEMPELMEIER, T. 1998. "Embedding practical real-time education in a computer science curriculum. In *Proceedings of the 3rd IEEE Real-Time Systems Education Workshop*. Poznan Poland, 21 November, 149–153.

TURLEY, J. 2002. Embedded processors (Parts 1–3), January 2002. Accessed at http://www.extremetech.com on August 18, 2004.

VAHID, F. 2003. Embedded system design: UCR's undergraduate three-course sequence. *Proceedings of the 2003 IEEE International Conference on Microelectronic Systems Education (MSE'03)*. 72–73.

WOLF, W. AND MADSEN, J. 2000. Embedded systems education for the future. *Proceedings of the IEEE 88*, 1, (Jan.), 23–30.

WOLF, W. 2001. *Computers as Components: Principles of Embedded Computing System Design*. Morgan Kaufmann. San Francisco, CA.

WOLF, W. 2003. How many system architectures? *IEEE Computer*, March 2003, 93–95.

Received August 2004; revised February 2005; accepted May 2005