Engineering an Education for the Future

Electrical and computer engineering has undergone rapid change and will continue to do so for the foreseeable future. These changes will have a profound effect on ECE education. In this article, we describe some changes in the practice of engineering and speculate on the educational implications.

Edward A. Lee

David G. Messerschmitt

University of California at Berkeley

Faced with rapid and unremitting change in the disciplines of electrical and computer engineering (ECE), some educators argue that we should deliberately not respond aggressively. Rather, we should focus on fundamentals that will serve the students well for an entire career. Although we strongly agree, this approach largely begs the question of what those fundamentals are. Of course it is desirable to impart all feasible fundamentals, but that seems impossible given the expanding breadth of knowledge required in the ECE field. Thus the question we address is: What are the fundamental skills and knowledge that are important for a future career in ECE? What should be the educational priorities?

In this context, *electrical engineering* encompasses integrated circuit and device design, microelectromechanics, electromagnetics, and so on. *Computer engineering* includes the design of computer systems hardware (such as processors, network switches, and peripherals) and related software (such as compilers, operating systems, and networking software). Somewhere in the midst of ECE are topics like signal processing, communications, control, application-specific hardware design, and the broad terrain of heterogeneous systems (including hardware, software, and physical channels). In the future there will be (or should be) considerable overlap between these curricula, and also an overlap with computer science.

CHANGES IN TECHNOLOGY

Engineering within the general domain of electricity, magnetism, electronics, and computers has undergone major shifts in emphasis over the past century. One of these was the shift from power transmission and rotating machinery to electronics. A second was the shift from vacuum tubes to semiconductors, from discrete circuits to integrated circuits. A third was the shift from analog to digital electronics. A fourth was programmable digital hardware.

Of course, none of these transitions was hard, absolute, or quick. Indeed, none of them is yet complete, and the fourth is in its early stages. Nevertheless, they all had watershed implications for the nature of an education. Our primary concern here is with the shift in emphasis from analog to digital, and from fixed to programmable. Is this as important a shift as the earlier ones, from the perspective of its impact on education? We believe so. This shift is similar to the shift from power to electronics and from vacuum tubes to semiconductors.

The shift from power to electronics hardly shifted fundamental knowledge and skills at all, but did profoundly change the nature of the design process and the character of the systems built on those fundamentals. The shift from vacuum tubes to semiconductors had an almost opposite effect: The essential fundamentals shifted profoundly (from free-space electron motion to holes and electrons in a semiconductor) and the design skills changed substantially, but the nature of the systems being realized didn't change in a discontinuous way. The shift from analog to digital, and from fixed to programmable, changes the most important fundamental knowledge, the skills used, and the nature of the systems being designed.

This latest shift raises three questions: What remains necessary, fundamental knowledge for all engineers designing digital, programmable systems? What design skills must engineers be taught? What are the characteristics of the systems being designed, and how must they be addressed in the curriculum?

Advances in performance

Advances in electronic technology have been simply extraordinary. As embodied in Moore's law, an underlying exponential increase in capability at a fixed price has profound implications over time. It implies

a trend away from performance limitations to limitations caused by our incomplete understanding of how to realize an application. For example, many signal processing investigations were not long ago criticized as being impractical to implement, but now Moore's law has consumed most of our algorithm and theoretical knowledge in these areas. Now much of the needed progress is constrained by shortcomings in our understanding of the task or application, rather than implementation feasibility or cost.

A clear implication is that performance and efficiency are somewhat less critical design issues than they used to be. There are, of course, notable exceptions, such as wireless communications, where traffic

An Integrated Curriculum

Raising the level of abstraction in the design of electronic systems has a major effect on the relationship of traditional electrical engineering with computer science. The classical overlap area, often called "computer engineering," captures only certain aspects of this relationship. In particular, it focuses on hardware, correctly viewing such disciplines as architecture, digital design, and computer-aided VLSI design as overlap areas. However, there is an equally important overlap area that deals with systems at a higher level of abstraction, and centers more on their application than on their hardware.

We view EECS as consisting of three core areas and three overlap areas, as illustrated in Figure A. The three core areas are:

1. Electronics (E, red)
2. Electronic information systems (EIS, yellow)
3. Computer science (CS, blue)

Some subtopics (which are meant to be suggestive, not exhaustive) are also shown in Figure A. The overlap areas are

1. Computer hardware (E and CS, purple)
2. Electronic systems (E and EIS, orange, which does not show because it rests behind the red)
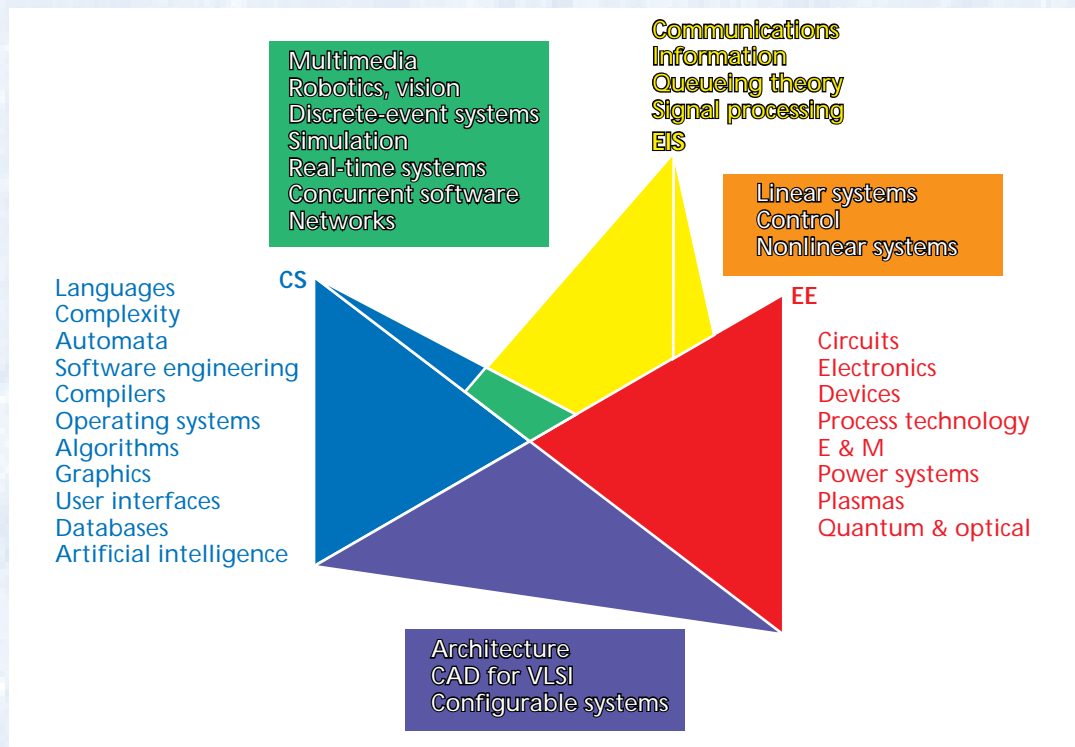3. Computer information systems (EIS and CS, green)

Figure A. The interaction between electrical engineering and computer science is not only the interaction between electronics and computer hardware, but also between what electrical engineers call "systems" (here labeled EIS for "electronic information systems") and computer science theory and software. In this diagram, we list some topics in the core and overlap areas. The orange area is entirely behind the red area.

capacity depends on efficient use of the scarce radio spectrum. But in a widening circle of applications, the goal of minimizing manufacturing cost or making the most efficient use of a resource is eclipsed by the desire to minimize design cost or time to market.

In certain cases, a design may not even be attempted because its complexity cannot be managed.

The overlap between EIS and CS forms a substantial part of the current dynamism in our field. Yet it is poorly served by most ECE, CS, and EECS curricula.

The choice of term "electronic information systems" is debatable, but more traditional terms fail to serve our purposes. We note that there is a strong overlap between the area we have described and what electrical engineers call "systems" or "systems theory." Use of these terms, however, conflicts with the use of the term "systems" in computer science, where it refers to software systems and operating systems. Since our discussion focuses on overlap areas, we have to choose an alternative term. Why not the simpler "information systems," since the intellectual underpinnings of EIS really do not depend on the implementation in electronics? We avoid this term to distinguish the subject from what a business school would call "information systems" or what a library school would call "information systems."

Any taxonomy of intellectual activities can be endlessly debated. Nevertheless, to design a curriculum, we have to take a stand on the specific content of each of these disciplines. We believe that the unifying principle of EIS is its formal approach to modeling systems. Such a formal approach has always been a strength of both traditional electrical engineering systems theory and computer science theory. That it becomes logical to reclassify these traditional approaches is a consequence of the rising level of abstraction in the design of electronic systems.

The fundamental ideas in EIS might include such traditional EE concepts as discrete-time and continuous-time signals, frequency domain, and feedback control, as well as overlap concepts such as discrete events, streams, state, automata, concurrency, and randomness. This differs from a purely EE view of systems in that continuous-time signals are merely one of many types of signals. At the opposite extreme in abstraction we have streams, which are discrete and possibly infinite sequences of values with no underlying notion of time. In between are events, in which values occur in time but irregularly. Most systems-oriented undergraduate EE curricula cover continuous-time signals, treat discrete-time signals as an advanced topic, and ignore the rest.

Most software today would not be designed without high-level languages, regardless of efficiency concerns. Thus, methodologies and tools that manage complexity, even if they sacrifice efficiency, become not just desirable but essential. In summary, the critical issue in the development of many products becomes not, "how do we design circuitry with sufficient speed?" but rather, "is it feasible to manage the complexity of the design?" and "how do we reduce the time to market?"

Importance of abstraction

While fundamental limits in electrical engineering were formerly always physical, today they are as likely to be complexity (and its cousins, decidability and computability). Carefully conceived architecture is necessary to manage complexity, sometimes at the expense of efficiency. An important tool for complexity management is abstraction. One of the most dramatic successes of abstraction is digital design, where circuits are modeled as Boolean functions. Another example is synchronous design, where time becomes discrete.

Of course, the synchronous and digital abstractions are not always successful. If a design is pushing the performance limits of the circuit technology, we may have to revert to an analog design that abandons these abstractions (as in RF and microwave frequencies today). It is also clear that the abstractions used in digital design will have to be changed for deep submicron technologies, where nonidealities like crosstalk and wire delays will have to be taken into account. However, this will have to be done by modifying the set of abstractions, rather than abandoning abstraction per se.

Many designs today simultaneously use multiple abstractions. For example, a microcontroller may be conceived as a synchronous digital circuit and then abstracted as an instruction set. The instruction set may be further abstracted by a compiler, in which case designs are carried out in a higher level language such as C, C++, or Java. Operating systems, software components (Java Beans or ActiveX), and middleware (such as CORBA or DCOM), offer still higher levels of abstraction for system design.

In some situations, the performance penalty incurred by basing designs on higher levels of abstraction can be substantial. This penalty, however, becomes perfectly acceptable when we realize that the design could not be conceptualized well at lower levels of abstraction, or that the cost or time to carry out the design would be excessive. A design carried out at too low a level of abstraction is likely to be poor, late, or both.

In other situations, performance and efficiency may actually improve by using higher levels of abstraction. A user interface for a microwave oven, for example,

has very modest speed requirements. If it is designed directly in digital hardware, the resulting design will very likely be capable of handling user interactions at superhuman speeds, quite unnecessarily. A programmable microcontroller, in contrast, achieves effective reuse of hardware modules to reduce hardware complexity and allow for better features, all the while reducing design effort.

Increasingly, designs combine elements better matched to programmability with elements better matched to custom hardware. The essential difference today is that software is sequential and hardware is concurrent, but this distinction is blurring. Technologies such as *behavioral compilers* are introducing a more sequential style into hardware design. Meanwhile, concurrent programming languages are introducing more concurrency into software design. The essential question becomes one of choosing the right abstraction, not one of choosing between hardware and software; in fact, it is arguable that the latter choice is nonsensical since every software design, when viewed at another level of abstraction, is a hardware design.

There are two consequences for curriculum development. First, we must prepare students to select abstractions, not just technologies. Second, just as designs can be built on top of higher level abstractions, so can courses. A course in synchronous digital design, for example, can build up from that abstraction, relegating the study of the underlying circuits to another part of the curriculum.

Advances in design technology

The design of today's multimillion-transistor chips would be very difficult to accomplish without computer-aided design tools. These tools manage evolving design information, aid the coordination of multiple designers, automate certain optimization tasks, and allow the designer to work with higher levels of abstraction. It is arguable that the latter role, while being the most important, is the slowest to evolve. In the context of digital design, the Mead-Conway approach and hardware design languages (now notably VHDL and Verilog) are two key milestones where new, higher level abstractions were made available to designers.

The prevalence of such tools has implications to the engineering curriculum. These tools, with their strengths and weaknesses, can change the relative priorities of subjects, all of which might be considered fundamental by many. Tasks that are routinely automated, such as Boolean logic minimization, should be relegated to more specialized courses where the principal concern is the design of the design tools, not the design of digital circuits.

New subjects are also made fundamental by the tools. Verilog and VHDL, for instance, are concurrent programming languages with a discrete-event model of computation. An informed consumer of design tools, therefore, needs to have some fundamental understanding of concurrency and discrete-event modeling. These subjects are entirely absent from most electrical engineering undergraduate curricula.

Given the inherent tension between depth and breadth, managing this change in education is difficult. We do not want to neglect fundamentals, including those fundamentals embedded deep within tools, where they are necessary to understand the tools and to use them more effectively. On the other hand, we do not want to waste precious classroom time on topics that are better delegated to the tools. It is difficult to draw generalizations in this area, but we believe that education has followed an appropriate middle ground. If there is a problem, it is in having inadequate instructional computing facilities and staff support for a curriculum that fully embraces design technology.

CHANGES IN THE ENVIRONMENT

We believe that the external environment for engineering employment for ECE graduates is changing in some fundamental ways. For some time, we have been discussing the need for more communications skills, cultural awareness, and group project experiences to better align education with the needs of the workplace. Taking this as a given, there are now other aspects of the outside environment that should be taken into account. Again, we don't intend to argue in favor of teaching overtly vocational skills, but rather to ask whether there is a more fundamental mismatch between the fundamentals of our educational experience and the workplace.

Organizational changes

Large, vertically integrated corporations are increasingly rare. Industries are reorganizing around a horizontal integration model, in which the application and the infrastructure supporting that application are unbundled. The new organization is the reality in computing, and is becoming the reality in communications. Chief among the reasons for this is the lower incremental investment in deploying new applications, where they leverage an existing infrastructure shared by many applications. The model also rewards the customer with greater competition in the industry, leading to lower prices, higher quality, and more features. Another, more technical reason, is integrated services and multimedia, where applications need to integrate many capabilities.

Even in the absence of horizontal integration, the industry is becoming more fragmented. No market displays this more than the home environment, where

the telephone, cable, consumer electronics, and computer industries are vying to find a niche at the expense of other players. With horizontal integration, each organization becomes at the same time more focused (concentrating on one horizontal layer) and less focused (not concentrating on any particular application, but rather trying to service them all).

No matter how this industry fragmentation occurs, it implies that many products can no longer be designed in isolation, and it is less frequent that one product provides value to the consumer in isolation. Rather, designers must be acutely aware of complementary, as well as competing, products. Often, there is the need for two, and often many, companies to collaborate opportunistically in designing, manufacturing, and marketing a product.

This industry fragmentation means that not only managers, but also designers, must be cognizant of many factors that impact their design choices. These factors include coordination with complementary products, standards, and available design tools. They also include basic considerations impacting the success of their designs, such as economics, human factors, standardization, regulation, and the basic strategic goals of the organization, such as minimizing time to market at the expense of manufacturing cost.

Changes impacting engineers

Many engineers will be working in smaller organizations, working on more narrowly focused markets. In that environment, each individual engineer must assume a broader role, encompassing areas like marketing, standardization, contractual relationships, and so on. Engineers who insist on remaining "strictly technical" are not as valued in such organizations.

There will always be an important role for engineers who choose to specialize in highly technical job skills. Current examples include analog circuit design, extremely high-speed digital design, device design, operating system and compiler design, and many others. Successful engineers in this category will more than likely have a graduate degree, often a PhD. We will call them the "technical experts."

Those engineers whose contribution remains largely technical, but who do not fall into the technical expert category are likely to find themselves designing digital systems consisting of some mixture of hardware and software. We call them the "designers," to contrast them with the technical experts. These designers are more likely to have a BS or possibly an MS.

For these designers, the traditional, formal, mathematical electrical engineering background has less value than it once did. These engineers do not need to design circuits, but rather systems. The hardware portion of their design task is actually more similar to programming than to circuit design. They are most concerned with underlying abstractions like events and concurrency. When they do encounter applications in which the performance cannot be achieved through standard design tools (increasingly rare), they will fall back on the technical experts.

As design tools increasingly take care of routine design tasks, there is less value added by routine design decisions, and more value added by overall system architecture, complexity management, and, most of all, relating the needs of the application to the design. Increasingly, routine designs of well-specified modules can be outsourced to design groups in other countries, where the engineers are willing to work at a fraction of the salaries US students wish to command. If our students wish to command high salaries, they will need to focus on the aspects of the design that will be tightly held in this country, including relating to application needs, system architecture, and management of the design process. Alternatively, they can become the technology pioneers and experts who are expanding the frontiers, should they choose to pursue a graduate education.

FUNDAMENTAL KNOWLEDGE AND SKILLS

We now turn to the question of what fundamental skills and knowledge come to the fore in this new world of programmable digital system design.

System modeling

A traditional strength in electrical engineering has been its formal approach to system modeling. Well-developed mathematical frameworks, such as Laplace and Fourier transforms, differential equations, random processes, and linear algebra, have provided the foundation for modeling of circuits, signal processing, feedback control, and the communication physical layer. All of these areas, however, have become the domain of the technical experts and are increasingly out of the purview of the designers.

To address the needs of the designers while preserving the traditional strength in formal modeling, we have to raise the level of abstraction. The notion of "signal" should no longer be only "voltage over time" but should be generalized to include discrete arrival of messages. The notion of "state" should no longer be the variables in a differential equation, but should be generalized to include the state of a finite automaton. The notion of "system" should no longer be a matrix defining a linear update, but should be generalized to include a Turing-complete computational engine.

An undergraduate program should put less emphasis on linear time-invariant, continuous-time models, and instead present a broader range of formal models for systems. Some institutions have already given

In smaller organizations, each engineer must assume a broader role, encompassing areas like marketing, standardization, and law.

equal or greater weight to discrete-time modeling compared to continuous-time modeling. The emphasis, though, is still on linear time-invariant systems and regular time steps, probably because the theory is so well developed. Nonlinear systems theory, unfortunately, is not well developed. Complete, understandable models are rare, and apply only to specialized scenarios. One aspect of nonlinear systems theory—chaotic systems—is valuable in a curriculum because it illustrates dramatically the fundamental limitations of deterministic formal analysis. It says, in effect, "this approach is hopeless."

Random processes, by contrast, provide a wealth of analytic techniques that have been applied successfully at very high levels of abstraction, such as artificial intelligence systems and communication networks. In many engineering curricula, however, random processes are introduced at the senior or graduate levels, too late to be of much use to the designers.

State space modeling, which originated in the control systems community, has shown its major strength with LTI systems. Independently, however, the complementary approach of finite automata theory, which originated in computer science, simplifies the state space to be discrete and finite, and exploits this simplification to develop a rich and powerful theory. That theory is potentially useful to both the designer and the technical expert, while the former control systems theory has become primarily useful only to the technical expert. Indeed, the control systems community has addressed some of the limitations in their traditional modeling techniques, and has married the two state space approaches to develop a theory of hybrid systems.

A challenge in designing a curriculum around this broad range of formal system modeling techniques is that the field is less mature than the tried-and-true LTI system theory. For example, an emerging area is the theory of discrete-event systems. Hybrid systems are part (but only part) of this. This theory could be particularly challenging to integrate into the curriculum because it is fundamentally based on an entirely different mathematical foundation than traditional system theory. Instead of differential equations, it uses discrete mathematics, mathematical logic, topology, and the theory of partial orders. These are not necessarily more difficult or advanced than differential equations, but they aren't included (or even hinted at) in courses entitled "engineering mathematics."

Architecture and complexity management

Arguably the most important phase of the design process, after the definition of goals, is the definition of the architecture. Architecture refers to the partitioning of functions into modules and the choice of abstraction defining the interaction of the modules. A successful architecture makes the internal workings of the modules as independent of one another as possible.

A successful architecture will exploit or promote reusability by either applying previously designed components or defining components that are sufficiently configurable to be reused in the future. Failing to reuse previous designs can cause a project to drown in complexity. Detractors will often argue that making components reusable compromises their efficiency or performance. But failing to do so has much more dire consequences: It makes only trivial designs feasible.

The design of architectures, with relevant concepts and theories and many design experiences, should be a part of every designer's education.

Hardware and software

Syntactically, hardware and software have completely merged. While it used to be that logic diagrams and schematics were used for hardware design, now it is more often Verilog and VHDL programs. These programs syntactically look much more like software than hardware. What makes them hardware is the concurrent semantics behind that syntax.

Software engineering and hardware engineering suddenly have a great deal in common. Both are concerned with source code control, interactive debugging, data structures, compilation techniques, and programming language semantics. Digital system design is largely a programming task. At the practical level, digital hardware design in the modern sense of employing computer-aided design tools and hardware description languages requires a set of programming skills remarkably similar to software design, even in the use of discrete-event models and concurrency.

From an educational perspective, the fundamental skills required of future digital system designers are not much differentiated from software designers. Of course, there remains a need for specialists in "pure software" (those concentrating on operating system and higher layers up through the application, the traditional domain of computer science) and "high-performance hardware" (those able to tread effectively below the digital abstraction, tuning performance for special-purpose designs by sophisticated circuit designs). But there is a broad middle ground where the fundamentals are, we believe, essentially the same for hardware and software, and where the engineers will have to be equally adept at both.

While one might argue that Moore's law is generally driving design from hardware toward software, software design is also increasingly benefiting from hardware-like design methodologies. There is today an increasing emphasis on reusability in software, and one approach that is gaining momentum is software components. There is also an increasing interest in

concurrency and communication within a software system.

The sophisticated concurrency model embodied in synchronous/reactive programming languages like Esterel and Lustre overtly uses the synchronous hardware metaphor, and compilers for these languages have borrowed methods from hardware synthesis. Concurrency in software, however, can assume more forms than it can in hardware. It need not be bound by notions of time or distance, for example, and can introduce the notion of transportable computation, such as agents or Java applets, that have no counterpart in hardware.

IMPACT ON EDUCATION

Thus far, our argument might be interpreted as requiring much greater breadth. Are we saying that students should know not only basic technologies, but also complexity management, design automation, economics, and human factors? Clearly something has to give.

One answer is to modularize engineering the way we modularize our engineering systems. If we can segment the curriculum into distinct and perhaps overlapping subcurricula, we can contain the knowledge explosion to some extent. The key issue is how that partitioning is to occur.

Another response is to extend the education. In our view, four years is simply inadequate to train an engineer in this environment, and even five years is marginal. Certainly this is the case for those aspiring to be the technical experts.

Our solution

If we designed a curriculum from scratch, we would

- target the undergraduate experience as a preparation for graduate school, and incidentally also to serve students who want to enter other fields (such as law or medicine) or concentrate on management, sales, and marketing;
- develop the masters program as a time when a deeper technical knowledge is obtained, preparing students for design careers. Their broader undergraduate education will serve them well in the new work environment we have described; and
- reserve doctoral programs so that, in addition to producing teachers and researchers, we prepare technical experts who will work in industry in frontier and extremely high-performance technologies. These experts will also carry forward the infrastructure and core technologies that continue to form the important foundation of the industry.

Teaching learning skills

Our students will face incessant change, and thus one of the most important skills we can impart is the ability to learn. In our view, too much of the current curriculum is aimed at covering everything, attempting to ensure that students will have collected the necessary background and skills for any conceivable professional activity they might encounter. This is, quite simply, hopeless.

We would argue in favor of an approach where a set of fundamentals are illustrated by current application and technology examples. We should depend on the students to pick up more of the latter themselves, including assignments in which they do exploration on the Internet to find current relevant examples. This will give them useful vocational knowledge, buttress the principles, and most importantly leave them with the skills and habit to pick up vocational knowledge on their own.

The pressure to focus education on vocational skills is huge. Industrial recruiters complain about students not having particular programming skills, or not having experience using some particular software or hardware tool. We should vigorously and unambiguously resist such pressure. In a field as dynamic as ours, no set of vocational skills has any significant longevity. It is far more important that our graduates be bright, curious, intellectual, and unafraid of learning new things.

Unfortunately, many engineering educators who would agree with this principle in fact preach a double standard. All too often we hear students being told to "improve their programming skills," or worse, "learn C++." Most designers should expect to need to learn several new programming languages during their career. Frequently, they will not even recognize these as programming languages, because they are highly specialized (as in a macro language for a spreadsheet) or have an unusual syntax (as in visual dataflow languages for signal processing).

Students who have been misled into believing that taking a course in C++ will give them all the programming skills they will ever need will be poorly equipped to adapt to changing technology. The more fundamental concepts of data structures, object-oriented design, and functional programming will better prepare them.

Using the Internet

Emerging technologies, particularly the Internet, will have a more profound impact on our education than we realize. One possible effect is that students will learn the boundlessness and often contradictory complexity in any particular technical discipline. Traditional engineering courses often try to circumscribe a body of material, creating a complete and coherent subject with consistent notation, and within which every problem

> **The pressure to focus education on vocational skills is huge. We should vigorously and unambiguously resist such pressure—no set of vocational skills has much longevity.**

has a right solution. Although pedagogically useful, when related to the real world this approach is seriously misleading. One use of the Internet is to illustrate for students the inconsistent notation and sometimes conflicting or even contradictory approaches described there.

The Internet will also help make connections between interrelated ideas without devoting precious lecture time to establishing connections with areas that some students will be able to relate to and others will not.

A third effect is that dynamic behaviors of engineering systems can be much more readily presented on a computer than on a blackboard or in a textbook. No longer do we have to rely on the imagination of students to interpret the dynamics in a mathematical formula.

What is fundamental

We now arrive at an essential and important question. We have identified a number of skills and fundamental knowledge that students need to navigate at higher levels of abstraction. We believe there is no room for expansion of the current engineering curriculum, and in fact argue that more room should be left for nontechnical subjects such as economics and humanities.

Thus we must give up something as well. Are there topics in the current engineering curriculum that are not so essential that they can be given up? If we poll the faculty at any major engineering school about whether the topic they teach is essential, the answer will be "of course." Moreover, they will probably be right, for a subset of the students.

The conclusion is simple: An undergraduate education in electrical and computer engineering will be different for different students. Some students will focus on technologies below the digital abstraction, and we believe most of these students should aspire to become technical experts and should consider it essential to go to graduate school. Others will focus on higher level abstractions, aspiring to be either designers or technical experts. This is in fact similar to the partitioning that has already occurred between electrical and computer engineering and computer science, where the latter focuses on software and software-defined systems as a separable issue. But a new requirement for the designers is to gain an understanding of more applications areas, which will doubtless become more important throughout their career.

On the other hand, the core EE areas of semiconductors, circuit design, and manufacturing will focus on increasingly difficult and sophisticated problems in both research and practice. The educational requirements of highly technical areas will include a virtually obligatory graduate degree, and, increasingly, a PhD.

Digital systems also have many difficult research agendas in areas such as algorithms, design systems, and concurrency management. There will be a continuing and expanding need for graduate education and research. At the same time, this area will become a mainstream topic for undergraduate education, with many more students to teach than in the core areas. This will be a difficult process to manage, as digital systems will require undergraduate teaching resources out of proportion with the research needs, and the core areas will be the opposite.

Relation to computer science

Digital systems will increasingly be programmatically similar to some areas of computer science. And computer science as a discipline has been moving in the direction of digital systems. Some of the most exciting developments in networking, like IP and tag switching, are actually a merger of the traditional ECE and CS viewpoints, exploiting the strengths of each. Likewise, mobile computing done correctly becomes intertwined with signal processing and network protocols. Multimedia applications in a networked computing environment encounter many issues of source coding and signal processing that are a traditional ECE focus.

Even areas of computer science like databases, AI, and theory, which appear to remain largely distinct from ECE topics, actually have interesting connections. Modern AI, for instance, uses stochastic modeling not unlike that commonly used in ECE. Computer science theory connects to traditional ECE topics such as coding theory and communications through its approach to computational complexity.

We believe that a modern curriculum in electrical and computer engineering cannot be logically separated from a computer science curriculum. The levels of abstraction used to design and model electronic systems increasingly coincide with those used in computer science, and any artificial separation will inevitably lead to significant redundancy in the two curricula.

We believe that the center of gravity of most undergraduate curricula today is too far on the side of attempting to train the small cadre of technical experts, a hopeless task within a four- or five-year program. Because of the hopelessness of the task, we cram too much content into the program, thinking it makes it better (and somehow less hopeless). This shuts out other fundamental knowledge that we believe will be extremely valuable to them in their design careers. Students have been seriously shortchanged by not understanding the big picture.

We advocate an alternative vision in which the undergraduate program focuses on a limited and carefully chosen set of core ideas, supplemented by real-world examples and importantly by student self-exploration and learning. Such an undergraduate program also emphasizes breadth, an exposure to a range of technical issues, as well as mathematics, science, humanities, and social sciences.

After the undergraduate experience, the students divide themselves into several groups. One group chooses to leave with an undergraduate degree, perhaps returning to school later to obtain a master's degree in engineering or business, or perhaps emphasizing a career in design management, marketing, or sales. A second group stays for a master's degree, leaving with the skills to be long-term design professionals. The third group stays for a doctorate, and becomes the cadre of technical experts who are prepared to tackle the difficult technical challenges and carry forward the core technologies.

In *Abstracting Craft*, McCullough reminds us that crafting things is at the core of humanity: "It is our talent to bring a mass of raw material into conformity with a vision. We fashion tools and coax materials." He builds a case that this essence of humanity is also the essence of technology today: "Our use of computers ought not be so much for automating tasks as for abstracting craft."

Computers, and more broadly, digital electronics, offer a new creative medium where "things" are crafted on top of abstractions, and every aspect of craft, its humanism and its cultural context, comes into play.

More than any prior human craft, this time we craft the medium as well as the artifact and the tools. The engineers that will do this best will be more creative than technical, and far more comfortable with abstraction. ❖

Edward A. Lee is professor of electrical engineering and computer science at the University of California at Berkeley. Together with Messerschmitt, he was one of the founders of the Ptolemy project, which currently serves as the focal point for his research. His research activities include design methodology for embedded systems, real-time software, discrete-event systems, visual programming, and architecture and software techniques for signal processing. Lee received a BS from Yale University, an SM from MIT, and a PhD from UC Berkeley. He won the 1997 Frederick Emmons Terman Award for engineering education.

David G. Messerschmitt is Roger A. Strauch professor of electrical engineering and computer science at the University of California at Berkeley. His research interests include the future direction of the merged network computing and telecommunications infrastructure, including wireless access, configuration of quality-of-service attributes, and the interaction of these functions with signal processing. Messerschmitt received a BS from the University of Colorado and MS and PhD degrees from the University of Michigan. He was chair of the EECS Department at UC Berkeley from 1993 to 1996.

Further Reading

S.W. Director et al., "Reengineering the Curriculum: Design and Analysis of a New Undergraduate Electrical and Computer Engineering Degree at Carnegie Mellon University," *Proc. IEEE*, Sept. 1995, pp.1,246-1,269.

D.A. Fraser, "Electrical Engineering Education: Twenty-Five Years on the Influence of Developments in Semiconductor Technology," *Int'l J. Electrical Eng. Education*, July 1988, pp. 219-227.

M.G. Hartley, "Trends in Electrical Engineering Education—A 25-Year Retrospective," *Int'l J. Electrical Eng. Education*, July 1988, pp. 209-217.

J.G. Harris, "The National Science Foundation Workshop on Undergraduate Education in Electrical Engineering," *Proc. 1987 Frontiers in Education Conf.*, IEEE Press, Piscataway, N.J.

W. Kline, "World War II: A Watershed in Electrical Engineering Education," *IEEE Technology and Society*, Summer 1994, pp.17-23.

J.Z. Lavi, B. Melhart, I. Pyle, "Engineering of Computer-Based Systems—A Proposed Curriculum for a Degree Program at Master Level," *Proc. Int'l Conf. and Workshop on Engineering of Computer-Based Systems*, IEEE CS Press, Los Alamitos, Calif., 1997, pp. 54-63.

J.H. McClellan et al., "Using Multimedia to Teach the Theory of Digital Multimedia Signals," *IEEE Trans. Education*, Aug. 1996, pp. 336-341.

M. McCullough, *Abstracting Craft*, MIT Press, Cambridge, Mass., 1996.

S.K. Mitra, "Re-Engineering the Electrical Engineering Curriculum," *Proc. 1997 IEEE Int'l Conf. Acoustics, Speech, and Signal Processing*, IEEE CS Press, Los Alamitos, Calif., 1997.

D.C. Munson Jr., "Elements of a New Electrical Engineering Curriculum at Illinois: A Shift from Circuits to Signal Processing," *Proc. 1995 IEEE Symp. Circuits and Systems, Vol. 1,* 1995, pp. 1-4Sf.

J. L. Pokoski, "Technological Evolution, Social Revolution, or Old Fogeyism (Electrical Engineering Education)," *Proc. 1989 Frontiers in Education Conf.*, IEEE Press, Piscataway, N.J., 1989, pp. 248-253.

A.B. Tucker et al., "Strategic Directions in Computer Science Education," *ACM Computing Surveys*, Dec. 1996. pp.836-45.