Incorporating System-Level Concepts into Undergraduate Embedded Systems Curricula

Kenneth G. Ricks
The University of Alabama
Electrical and Computer Engineering
Tuscaloosa, Alabama 35487-0286
(205)-348-9777
kricks@eng.ua.edu

David J. Jackson
The University of Alabama
Electrical and Computer Engineering
Tuscaloosa, Alabama 35487-0286
(205)-348-2919
jjackson@eng.ua.edu

ABSTRACT

As more and more embedded systems concepts are integrated into academic curricula, the incorporation of system-level concepts must keep pace with lower-level topics. However, there are several challenges faced by educators trying to integrate system-level concepts into embedded systems curricula. Four such challenges include: the breadth of the embedded systems field limits opportunities for system-level content; lack of adequate laboratory platforms addressing system-level alternatives; limited student exposure to diverse software tools; and assessment associated with system-level activities. Each of the challenges is described in detail and suggestions for overcoming them are offered.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education, information systems education*; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems – *real-time and embedded systems*

General Terms

Design

Keywords

Embedded systems education, engineering curriculum

1. INTRODUCTION

Embedded systems education has gained much momentum as embedded systems concepts are being integrated into many undergraduate electrical and computer (ECE) engineering curricula [3, 8, 12, 15, 17, 19]. To create capable embedded systems engineers, educators must present both low-level material and high-level system concepts. Low-level and subsystem design is common and closely associated with most of the course-specific embedded systems content being delivered. However, concepts related to system-level design, integration, and especially testing are often lacking as students progress through the curricula. In many cases, the system-level concepts are left to be presented in a project-based course at the end of the curriculum, such as a capstone design course, or in graduate courses. This last minute effort to include system-level concepts in an undergraduate curriculum does not offer students the appropriate opportunities to become effective at solving embedded systems problems at the system level.

In this paper, "system-level" concepts are considered to be those that pertain to the overall functionality of the entire system, as opposed to low-level or subsystem-level concepts related to individual parts of the whole system. System-level concepts include several key ideas that embedded systems designers must understand to be successful and focus on choices that must be made when designing the embedded system. Some typical system-level concepts include: the number and type of processors required; interconnection networks interfacing processors, peripherals, and memory; memory organization and its effect on system performance; types of peripherals needed; software tools and development environments; various allocations of tasks to hardware or software; system-level integration; testing, and verification; and high-level abstraction.

These types of concepts center on design decisions and operational verification of the whole system and its parts. While low-level and subsystem-level details are critical aspects of any embedded systems curricula, it is a mistake to assume that students can convert their detailed knowledge of subsystem behavior to the skills necessary to evaluate alternatives at the system-level and to understand the nuances associated with system-level integration and verification. In fact, recent research indicates that more qualitative thinking should be incorporated into engineering curricula [5]. Thus, it is critical that these system-level skills be directly addressed in embedded systems undergraduate curricula. The significance of this is well documented for several engineering disciplines [4, 7, 20].

In this expanded version of [11], the incorporation of system-level concepts into embedded systems curricula is discussed. Section 2 describes several challenges educators face when trying to incorporate system-level activities into the curriculum. Some possible suggestions for overcoming the challenges are presented in section 3. Finally, conclusions are presented in section 4.

2. CHALLENGES

Despite the variety of educational approaches used to integrate embedded systems concepts into ECE curricula, there are several challenges that seem to be universal when it comes to addressing system-level concepts.

2.1 The Breadth Problem

Embedded systems is a broad field incorporating topics from electrical engineering, computer engineering, computer science, and many application-specific areas. The broad scope of this field has led to the development of educational aids such as the 2004 IEEE/ACM Computer Engineering Model Curriculum [1]. This model curriculum includes curricular guidelines for embedded systems education in the form of a broad set of topics and learning outcomes addressing both low-level and system-level concepts. To quantify the scope of the embedded systems component of the model curriculum, there are 11 knowledge areas containing a total of 63 topics and 34 learning outcomes.

One common problem faced by embedded systems educators is the difficulty associated with incorporating such a broad set of topics and learning outcomes into undergraduate curricula, often referred to as the breadth problem [6]. Since all concepts cannot be covered and decisions have to be made, there is a tendency for curricula to become too focused on the low-level subject matter at the expense of system-level concepts. The breadth problem can be mitigated to some degree by having a larger set of course offerings. However, for smaller programs having fewer courses in the embedded systems area, the breadth problem is made worse.

2.2 Laboratory Platforms

In many cases, the integration of system-level concepts into the curriculum is hindered by the lack of an appropriate laboratory platform. Many educational and development platforms offer only a fixed hardware and software configuration, and laboratory activities are reduced to using this configuration in different ways. This could explain the common approach used in many embedded systems laboratories where students concentrate on the operational characteristics of a broad set of peripherals and lab assignments differ only in the peripheral being used.

Such a platform discourages system-level design decisions. Typically there is only one processor and it has already been chosen. Similarly, the set of I/O peripherals, the I/O interfacing design, the memory system, and the software environment are fixed and static. Students are left with few, if any, system-level design decisions. There are no opportunities to evaluate different software allocations to see if multiple processors or hardware accelerators would improve system performance. Nor can students look at different processors to determine the best cost/performance for the application. In the best cases, students can evaluate the performance of the memory hierarchy by turning on or off the cache system. However, most of the time the hierarchy design is fixed and various characteristics such as mapping and replacement policies cannot be changed.

Simulation and emulators are sometimes used to fill some of the holes left by these types of platforms. Laboratory activities based upon simulation are good exercises for students and reinforce the benefits of simulation such as reduced costs and improved time-to-market. However, simulation plays only a part in the overall design cycle of an embedded system and students should be exposed to other parts of this cycle including prototyping.

Another limiting factor for laboratory platforms that address system-level concepts is cost. There are platforms available that map well to the system-level topics and learning outcomes of the model curriculum. One is the VMEbus [10, 13], and another is PC/104 [2]. However, these systems are designed for industrial applications in harsh environments and carry a high price compared to most educational platforms.

2.3 OS and Software Environments

One of the most significant factors influencing many system-level design decisions for embedded systems is the operating system (OS) and the software development environment [18]. Although there is no shortage of OSs or development environments and they are generally available at little to no cost for educational uses, their use and influence at the system-level is somewhat limited in the curriculum. One reason for this is that educators are limited in the number of such systems students can be expected to master. Without detailed knowledge of several software environments, students are limited in their ability to evaluate tradeoffs among them.

Efforts to expose students to a larger number of software tools and environments are susceptible to degrading to a “teach-the-tool” approach due to the limited time available and steep learning curves. Thus, educators are often pushed in the opposite direction where a single tool is used throughout a course sequence with students gaining additional skills using the tool as they progress through the sequence. This single tool approach undermines the ability of educators to integrate high-level software evaluation concepts into the curriculum.

2.4 Assessment Challenges

The difficulty with the assessment of system-level concepts is another obstacle to their integration into the average embedded systems curriculum. Assessment of system-level concepts can be difficult for several reasons. First, the scope of system-level problems, especially those that are design related, is significantly larger than that for component-level questions partially due to the exploding architecture space [14]. Given the larger solution spaces associated with such questions, it is more challenging to devise assessment rubrics capable of accurately capturing student performance.

Similarly, the scope of system-level questions is often difficult to integrate into written assignments making them hard to assess in some courses. This can be less of a problem in courses with labs or project-oriented courses, such as a capstone design course. However, students need to be exposed to many system-level concepts before they encounter the capstone design course. Also, teaming aspects of a capstone design project or course lab assignments can make it more difficult to collect assessment data related to individual student understanding of system-level concepts.

Another difficulty with assessment is that system-level knowledge is difficult for students to judge and therefore assessment data based upon student feedback can be misleading. It is easy for students to mistakenly equate detailed component-level knowledge to system-level understanding especially if they have little system-level experience. For example, consider a memory hierarchy problem. A student who knows all about the mapping and replacement policies defining the operation of the hierarchy may easily assume knowledge of how the memory system effects overall system performance without ever evaluating the memory system in that context. This observation is supported by anecdotal evidence collected from a capstone design course. In this specific course, it is common for students to report total surprise at the time required to perform system integration and to validate system performance even after each subsystem was independently tested and verified to subsystem requirements.

3. OVERCOMING THE CHALLENGES

The previous section outlines several challenges to the integration of system-level concepts into embedded systems curricula. This section discusses some possible ways to overcome these challenges.

3.1 The Breadth Problem

As discussed earlier, because of the breadth problem, there is a tendency for many embedded systems curricula to focus on the low-level concepts instead of system-level concepts. In some cases, this concentration occurs accidently as a side effect of how the curriculum is constructed and maintained. If each course in the curriculum is designed and maintained individually, it is easy for each course to focus only on the details for that course, often heavily weighted toward low-level concepts. If the curriculum is designed and maintained as an associated set of courses so that all the courses work together, then it is easier to see where system-level concepts can be integrated into the course sequence. For example, it may be difficult to design several complex components and integrate them into a larger system in a single one-semester course. Therefore, the instructor for such a course may decide to limit the course to the design of the components. However, if the course is closely aligned with other courses within the curriculum, the components designed in the first course can be used and integrated into a larger system in subsequent courses. This approach requires that educators view their curricula from both high, curricular-levels and low, course-levels. This approach also requires coordination among faculty teaching the different courses so that related projects can extend across course boundaries.

On the other hand, the concentration on low-level concepts may occur intentionally as educators are faced with making decisions regarding topics to include and omit in the curriculum. Concentration on the low-level concepts is often justified by educators that believe the system-level concepts are merely extensions of the lower-level material and by those that consider the low-level concepts to be more fundamental, therefore more important. However, the breadth problem and the abundance of low-level topics should not prevent the integration of system-level concepts into a curriculum. The first step is to realize that system-level and low-level concepts do not have to be mutually exclusive. If the proper approach is taken, a given set of topics can be presented in such a way as to emphasize both low-level and system-level concepts.

In the remainder of this section, several examples are presented describing how common topics and learning outcomes from the model curriculum can be presented to enhance a system-level understanding. This approach works well with existing topics and learning outcomes from the model curriculum making it easy to integrate the approach with existing curricular guidelines.

3.1.1 Input/Output Interfacing

One of the general knowledge areas in the embedded systems component of the model curriculum is *CE-ESY1 Embedded Microcontrollers*. This knowledge area includes a topic related to the understanding of input/output (I/O) peripherals and another topic related to polled and interrupt-driven I/O [1]. Both of these concepts rightfully belong in an embedded systems curriculum. However, there are two completely different skill sets that students can gain from a series of lectures and labs covering these concepts.

The first possible skill set is mastery of the operation and the required interfacing to various I/O peripherals. Often presentation of this material focuses on the operational specifics of the popular peripherals of the day whether it is an A/D converter or an LCD. Students learn the operational details with the unrealistic expectation that they will interface to the same peripherals throughout their careers.

The second skill set is the mastery of the interfacing from the processor to the peripherals, whatever they may be. The lecture and lab materials to achieve this skill set are different from that used to achieve skill set one. Specifically, students are presented with polled I/O techniques and interrupt-driven I/O techniques including the various ways interrupt vectors are obtained, the use of multiple levels of interrupts, and contention among interrupting peripherals. In this approach, the peripheral is a secondary concern since the students are focusing on the interfacing techniques. Popular and complex peripherals can be used, but their complexity will likely pose an obstacle to the student learning the interfacing concepts.

From a system-level perspective, interfacing and evaluating the benefits of polled vs. interrupt-driven I/O is more significant than detailed knowledge of several different peripherals. Peripherals will come and go and likely students will be interfacing to peripherals in the future that have not even been conceived yet. However, the basic skills of interfacing have not changed in years and are not likely to change dramatically in the near future. Students having these skills are more likely to be able to make system-level decisions and evaluate tradeoffs associated with different I/O designs to achieve stated performance goals of any embedded system.

3.1.2 Processor Organization

Distributed across multiple knowledge areas in the model curriculum, there are several topics and learning outcomes specifically related to basic processor organization, operation, and interaction within the context of the overall system. These topics are specified for basic systems as well as more complex multiprocessor systems [1]. These topics can be presented in many different ways leading to many different skill sets for the students.

For example, it is easy for the internal architecture and the performance evaluation of a processor to become the central theme for curriculum materials covering processor-related topics. In fact, anecdotal evidence collected over several years through conversations with colleagues at various universities suggests that the processor performance is the main reason to select a processor for an embedded application with little thought given to the overall system performance. While this approach will produce students having in-depth knowledge of the organization and performance of one processor, it will not address any of the system-level concerns including the role of the processor within the overall system and how other system components affect overall system performance.

Another example concerns multiprocessor systems. In many cases, the interconnection network topology is the centralized theme. Students evaluate the benefits of different topologies and analyze communication routes utilizing skills from networking theory, communications, and queuing theory. This addresses many of the multiprocessor related topics and learning outcomes of the model curriculum, but it limits the students' view of the overall system in which the multiple processors exist. A more system-level approach should include discussions of hardware and software allocation decisions that might lead to the need for multiple processors. Once multiple processors are deemed necessary, the interconnection network needs to be chosen. This is where the network topology information plugs into the puzzle. However, from a system-level perspective, students would benefit more from understanding the effects of topology decisions rather than knowing every detail related to a set of popular topologies. The choice of interconnection topology influences communication mechanisms for inter-processor communication as well as shared and distributed memory designs. A good embedded system designer should understand these relationships and be able to make educated decisions related to overall system design as opposed to being exposed to low-level details of one or more topologies.

3.1.3 Memory Hierarchy Design

Another example is related to memory system design. Memory hierarchies and caches are specifically listed as a topic in the model curriculum and understanding how memory system design affects program design and performance is listed as a learning outcome [1]. However, the presentation of this material can be memory system-centric targeting the topic mentioned in the model curriculum. Or, this material can be presented having a system-level focus helping to achieve system-level design skills.

For example, memory hierarchies and caches can be presented with an emphasis on the mechanical details of replacement policies, write-back policies, and mapping policies. Students can then be evaluated on their ability to demonstrate their understanding of the operation of the memory system. On the other hand, these same concepts can be presented with overall system-level performance in mind. In the latter case, students should be evaluated on their ability to choose memory hierarchy design characteristics to optimize system performance given a set of memory access patterns that are part of the standard workload. This second skill requires in-depth knowledge of the operation of the memory system. However, the focus is to use this knowledge to address system-level performance questions. In both cases, students gain a useful set of skills. However, students with the ability to make system-level design decisions and to perform system-level evaluation are more valuable as embedded systems designers.

3.1.4 Verification and Testing

One final example is related to verification and testing, an area of critical need in embedded systems education. Verification and testing is included in one of the 11 knowledge areas (one of the seven core areas) in the model curriculum, *CE-ESY5 Reliable System Design*. This particular knowledge area concentrates on teaching students how to find hardware and software faults, and the overall benefits of design verification [1]. However, it does not differentiate among component-level, subsystem-level, and system-level testing and verification. Thus, educators can choose the context within which to place these concepts. It is important to include testing and verification at the lower levels for each component and subsystem. But, it is just as critical for students to understand the potential problems encountered during system integration. Furthermore, students should be able to relate component-level testing and system-level testing for the same project. This provides students with a global understanding of how testing at one level can impact the other, and helps students to identify potential problems and the corrective actions needed to address errors encountered at both levels.

3.2 Appropriate Laboratory Platforms

Future work should focus on the development of more system-friendly laboratory platforms. The importance of laboratory assignments in an embedded systems curriculum is well documented. Without laboratory support for system-level concepts, there are fewer opportunities to integrate these concepts into the curricula. There are three possible approaches to solving this problem.

3.2.1 System-on-a-Chip Platforms

The eventual solution to the platform problem could lie in the development of system-on-a-chip (SoC) designs. Powerful programmable logic devices (PLDs) such as FPGAs are now available that are capable of supporting multiple softcore processors, various I/O peripherals, limited on-chip memory, and associated glue logic. Using such a system, multiple processor designs are feasible, different I/O system designs can be evaluated, and different memory designs are possible.

Currently, SoC platforms are attractive as a complementary technology, but improvements are required if they are to become the primary platform for teaching system-level concepts. First, there is a steep learning curve associated with the design tools. Because of this, students are limited to a single development environment contributing to the single tool problem described above. Also, the steep learning curve pushes the use of these systems further back in the curriculum giving students less time to use the platform for more advanced activities. Additionally, there are fewer softcore processors and compatible OSs available for PLD implementations than are available using traditional system implementations. Finally, the PLD devices are not complex enough to provide for a significant amount of on-chip memory to support a complex multiprocessor design. This limits possible memory system designs and requires that the SoC design interface to off-chip resources.

3.2.2 Model Traditional Platform

There is no one laboratory platform that can address every possible system-level concept associated with embedded systems and that will please every embedded systems educator. However, the following paragraphs identify some general characteristics defining a well-rounded, capable, model laboratory platform that could, if it existed, help to address system-level concepts in embedded systems curricula.

The platform should be modular to support easy transition from one design to another. System-level design decisions include number and types of peripherals, number and types of processors, amount and configuration of memory, and desired software environment. Thus, processor modules, I/O modules, and memory modules should be available and should be easily connected and disconnected from the other system components.

Although various component interconnection networks should be supported, this capability is not realistic in a low-cost educational platform. Since a basic system interconnection architecture must be provided, a basic bus-based architecture is suggested. This is a straightforward architecture that will support the easy integration of system modules. A bus-based system provides the opportunity to address many system-level concepts such as bus arbitration, the effects of bus saturation on system performance, and the effects of memory system operation on bus traffic and system performance. Again, since many different bus communication protocols are in use, it is suggested that a simple asynchronous protocol be used. The bus protocol should support multiple masters and have multiple levels of interrupts for flexibility in the design of the I/O system.

Processor modules based upon different types of processors having various speeds and data widths and supporting various operating systems should be offered. This allows for heterogeneous or homogeneous multiprocessor design, both common design methodologies used in practice. Processor modules must support interrupt driven I/O using multiple levels of interrupts and should support bus arbitration for multiprocessor systems. There is no requirement that I/O administration and bus arbitration logic physically exist on the processor modules. However, that would result in a smaller design requiring fewer modules and fewer connection slots to the system bus. Also, processor modules should include the capability to support on-board memory so distributed non-uniform memory access (NUMA) designs can be created.

Memory modules should also be available for connection to the system bus. This memory can serve as global shared memory providing support for uniform memory access (UMA) designs to compare to the NUMA designs supported by distributed memory. Cache memory should be provided and the memory hierarchy should be reconfigurable to facilitate the evaluation of various hierarchical structures.

I/O modules should also be available implementing various common peripheral devices. These modules should support multiple levels of interrupts and should have various response times. In addition, special purpose modules can be offered that host FPGAs and other PLDs so that advanced SOC designs can be supported.

The form factor for the bus backplane, the individual modules, and an enclosure should be relatively small and fit easily on the desktop or lab bench. There is no need for rack mounting capabilities or ruggedized performance given the educational environment. The enclosure should have an integrated power supply to create a self-contained design.

Costs must be kept low. The enclosure and backplane, two of the more expensive components of the system, would be provided by the institution creating permanent lab stations. However, the overall goal would be to make the processor, memory, and I/O modules inexpensive enough that students would purchase them as part of a toolbox to be used as they progress through the entire curriculum. During lab activities, individuals or student teams would design their systems by choosing appropriate modules to connect to the backplane within the enclosure at a given lab station. Since many embedded systems are not based upon state-of-the-art performance and this system is dedicated to educational purposes, none of the modules require the best performance available. This can help to keep costs low.

One of the biggest hurdles to overcome to make this platform feasible is that of software. Operating systems, software development environments, drivers, and board support packages are necessary. However, it is not feasible to require significant amounts of custom in-house development of these components either from an administration perspective or from a student perspective. This also conforms to the goal to incorporate system-level concepts. Low-level software and driver development requirements are contrary to the system-level approach the platform is supposed to create.

Based upon the description of the desired characteristics, it may appear that such a model platform is out of reach. This may indeed be the case if a custom design starting from scratch is required. However, the preferred approach is to find an existing platform having most of the desired characteristics and a low cost/performance ratio and to modify it to fit this use. One such platform is the VMEbus platform. As mentioned earlier, the VMEbus platform maps well to most of the system-level concepts in the model curriculum and has many of the characteristics of the desired lab platform. Its primary drawbacks include costs and form factor. Interestingly, if a subset of the standardized VMEbus specification is used and the mechanical requirements are relaxed, the form factor could be altered to make it more suitable for educational use and the resulting costs would correspondingly be reduced. Other existing bus-based platforms are also possibilities. For example, the PCI-based PC/104 platform offers many of the desired characteristics.

3.2.3 Virtual Platform

Some of the difficulties associated with providing a model traditional laboratory platform can be overcome by using a virtual platform. Virtual tools have become quite popular in embedded systems education [9, 16]. So far, these tools have been introduced at the component and subsystem levels and are a valuable means of abstracting some of the low-level details that often overwhelm students. This approach can be extended to provide a software tool capable of designing virtual embedded systems. In this case, the tool provides a set of components (processors, I/O peripherals, memory modules, bus arbiters, etc.) and interconnection networks (buses, point-to-point networks with various topologies) from which to choose. The students pick the desired components to build a complete system, placing instances of the components in an editing pane within the software environment and connecting them. In the way, students are basically drawing their virtual embedded system.

However, the software design tool does not just provide a means to draw box and line type of diagrams. It also supports user input of component parameters. Each component has associated parameters and characteristics that must be set by the user. The parameters include clock speed, communication protocol, interrupt level, data width, execution times for hosted software, etc. In this way, the user builds a virtual system and specifies its behavior through component and interface specifications.

To really capture the power of this approach, the tool must also provide analysis of the overall design created by the user. This analysis must include subsystem verification as well as analysis of overall system-level behavior. Typical system characteristics that might be analyzed include mismatched communication protocols, analysis of memory access times, interrupt latency, bus throughput, etc. The analysis performed by the software tool should be user-configurable. Therefore, the user would be responsible for specifying a test plan for the virtual system he/she created. This plan could contain both subsystem-level and system-level tests using different analysis capabilities built into the software tool to help address both the low and high-level testing and verification concepts in the model curriculum.

Due to the virtual nature of the tool, there is no limit to the system modules that can be added as potential system components. All the components identified as necessary in the model traditional lab platform can have virtual counterparts for use with this tool. In addition, the necessity to limit options in the traditional platform does not exist with the virtual platform. There is no need to limit designs to bus-based architectures or to asynchronous buses. All options are still available to the designer. In this way, there are no restrictions on the design space based upon available resources, and students can explore a larger portion of the architecture space for each assignment.

The use of a virtual tool would be easy for universities to adopt. The software tool would execute on a PC or workstation already present in most ECE laboratories thus requiring minimal infrastructure or investment.

A word of caution is needed here. As virtual tools become more common, the natural side effect will be more abstraction of low-level details. Students will migrate to higher levels of understanding but will miss the underlying details. Thus, it is important that virtual tools be incorporated into a curriculum slowly and not at the expense of the low-level engineering details.

The authors are currently implementing a software tool capable of designing virtual embedded systems. The tool is intended for use in a senior-level/graduate-level embedded systems course. Once completed, the software tool will be used and assessed alongside a more traditional laboratory platform based upon the VMEbus and used in this same course. In this way, direct comparisons can be made between the actual VMEbus platform and the software tool. The assessment process will focus on both low-level and system-level concepts identified in the IEEE/ACM Computer Engineering Model Curriculum for embedded systems [1].

3.3 Software Tools

The limited exposure to multiple software environments to facilitate comparisons and trade-off analysis is a difficult problem to overcome. As software tools continue to become more complex, the learning curve associated with their use will remain an obstacle to educational efforts and will continue to push educators to the single environment approach. One positive associated with complex software tools is that they often access hardware from high levels of abstraction. So, by default, these tools are supporting a higher, system-level approach. Still, teaching only one environment limits students' ability to analyze systems from this perspective.

The negative effects of using a single tool or environment can be minimized if software components are presented by focusing on general software features. By teaching students the different types of features that software tools have and what types of features map well to various applications, students will begin to be able to compare and contrast software components within a system context. The one specific software environment students use within the curriculum can then be used as an example and justification can be given for its selection.

Another alternative to introducing more tools is to focus on languages. Students are typically introduced to several different languages in the average embedded systems curriculum including assembly language, C, and C++. Efforts should be made to make sure students master programming skills in appropriate languages and to educate students when use of a particular language is appropriate. This means that specific embedded programming skills must be presented above and beyond the commonplace general programming concepts presented to all ECE students [12].

3.4 Assessment

As more system-level activities are incorporated into the curriculum, the challenge to accurately assess them minimizes. For example, an appropriate traditional lab platform or a virtual platform can make it more feasible for students to perform individual lab work. This makes assessment of individual effort more straightforward. Secondly, with an appropriate platform integrated into more courses, design problems having a smaller scale can be used. These problems are easier to assess accurately. Finally, giving students more experience with system-level concepts will inherently improve their judgment making student feedback a more reliable assessment mechanism.

4. CONCLUSIONS

The 2004 IEEE/ACM Computer Engineering Model Curriculum for embedded systems includes both low-level and system-level concepts. Despite the wide-spread integration of these embedded systems concepts into academic curricula, there is still work to be done to integrate system-level embedded systems concepts. Because of the breadth problem, introducing even more concepts into the curriculum is difficult. One possible solution is to apply a system-level slant to many of the component and subsystem concepts already being used.

However, there are other inherent challenges that academia faces regarding the integration of system-level concepts into embedded systems curricula. These other challenges include the lack of appropriate laboratory platforms, limited student exposure to software tools and development environments, and difficulties associated with the assessment of system-level concepts. SOC, a new traditional laboratory platform addressing a wide range of design options, or a design tool capable of creating virtual embedded systems having a wide range of architectural characteristics offer possible solutions to the first challenge. The learning curve associated with modern complex software tools continues to present a difficult obstacle to education, while assessment difficulties associated with system-level concepts are shown to be minimized as more system-level concepts appear in the curriculum.

5. REFERENCES

[1] Joint Task Force on Computer Engineering Curricula, IEEE Computer Society, Association for Computing Machinery. *Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering.* IEEE Computer Society, Los Alamitos, CA, 2006, pp. A.43 – A.45.

[2] PC/104 Embedded Consortium, [Online]. Available: http://www.pc104.org/.

[3] Bruce, J. W., Harden, J. C., Reese, R. B. Cooperative and Progressive Design Experience for Embedded Systems. *IEEE Transactions on Education*, *47*, 1 (February 2004), 83-92.

[4] Chen, T. From System Design to IC in 14 Weeks – Teamwork makes it Possible. *IEEE Transactions on Education, 36*, 1 (February 1993), 137-140.

[5] Goldberg, D. E., "Bury the Cold War Curriculum", PRISM, P. 68, April 2008.

[6] Haberman, B., Trakhtenbrot, M. An Undergraduate Program in Embedded Systems Engineering. In *Proceedings of the 18th Conference of Software Engineering Education and Training (CSEET'05)*, April 18-20, 2005, pp. 103-110.

[7] Hung, D., Vien, J., Chan, W., Fu, C. Developing a Teaching Environment for Rapid Design and Verification of Complex Digital/Computing Systems. In *Proceedings of the IEEE International Conference on Microelectronic Systems and Education (MSE'03)*, Anaheim, California, June 1-2, 2003, pp. 131-133.

[8] Paulik, M., Krishnan, M., Al-Holou, N. Work in Progress – Development of an Innovative Curriculum for Undergraduate Electrical and Computer Engineering Students. In *Proceedings of 34th ASEE/IEEE Frontiers in Education Conference*, Savannah, Georgia, October 20-23, 2004, pp. S2C-13 – S2C-14.

[9] Phalke, A., Lysecky, S., eBlocks. In *Proceedings of the Workshop on Embedded Systems Education* (held in conjunction with EMSoft 2008), ISSN: 1943-801X, Atlanta, Georgia, October 23, 2008, pp. 49-56.

[10] Ricks, K. G., Jackson, D. J. A Case for the VMEbus Architecture in Embedded Systems Education. *IEEE Transactions on Education, 49*, 3 (August, 2006), 332-345.

[11] Ricks, K. G., Jackson, D. J., Addressing System-Level Concepts in Embedded Systems Education. In *Proceedings of the Workshop on Embedded Systems Education* (held in conjunction with EMSoft 2007), Salzburg, Austria, October 4-5, 2007.

[12] Ricks, K. G., Jackson, D. J., Stapleton, W. A. Incorporating Embedded Programming Skills into an ECE Curriculum. *SIGBED Review*, ISSN: 1551-3688, 4, 1 (January 2007), pp. 17-26. [Online]. Available: http://www.cs.virginia.edu/sigbed/vol4_num1.html.

[13] Ricks, K. G., Jackson, D. J., Stapleton, W. A. An Evaluation of the VME Architecture for Use in Embedded Systems Education. In *Proceedings of the Workshop on Embedded Systems Education* (held in conjunction with EMSoft 2005), Jersey City, New Jersey, September 22, 2005, pp. 59-65.

[14] Schaumont, P., Hardware/Software Co-design is a starting point in Embedded Systems Architecture Education. In *Proceedings of the Workshop on Embedded Systems Education* (held in conjunction with EMSoft 2008), Atlanta, Georgia, October 23, 2008, pp. 18-24, ISSN: 1943-801X.

[15] Seviora, R. A Curriculum for Embedded System Engineering. *ACM Transactions on Embedded Computing Systems*, *4*, 3 (August 2005), 569-586.

[16] Sirowy, S., Sheldon, D., Givargis, T., Vahid, F., Virtual Microcontrollers. In *Proceedings of the Workshop on Embedded Systems Education* (held in conjunction with EMSoft 2008), ISSN: 1943-801X, Atlanta, Georgia, October 23, 2008, pp. 57-62.

[17] Sztipanovits, J., Biswas, G., Frampton, K., Gokhale, A., Howard, L., Karsai, G., Koo, T. J., Koutsoukos, X., Schmidt, D. C. Introducing Embedded Software and Systems Education and Advanced Learning Technology in an Engineering Curriculum. *ACM Transactions on Embedded Computing Systems, 4*, 3 (August 2003), 549-568.

[18] Turley, J. Survey says: Software Tools More Important Than Chips. *Embedded Systems Design*, April 11 2005, [Online]. Available: http://www.embedded.com/showArticle.jhtml?articleID=160700620.

[19] Vallino, J. R., Czernikowski, R. S. Thinking Inside the Box: A Multi-Disciplinary Real-Time and Embedded Systems Course Sequence. In *Proceedings of the 35th ASEE/IEEE Frontiers in Education Conference*, Indianapolis, Indiana, October 19-22, 2005, pp. T3G-12 – T3G-17.

[20] Wirthlin, M. Senior-Level Embedded System Design Project using FPGAs. In *Proceedings of the IEEE International Conference on Microelectronic Systems and Education (MSE'05)*, Anaheim, California, June 12-14, 2005, pp. 91-92.