Bringing Embedded Software Closer to Computer Science Students

Jogesh K. Muppala
Dept. of Computer Science and Engineering
The Hong Kong University of Science and Technology
Clearwater Bay, Kowloon, Hong Kong
Tel: +852 2358 6978
muppala@cse.ust.hk

ABSTRACT
Computer Science (CS) students have often shied away from the field of embedded systems owing to their perception of this area as "hardware" oriented, not without reason. But recent trends in embedded systems, with the growing importance of the software component, has brought about new opportunities for computer science students to participate and contribute to embedded system development. In this paper we present our views and experience gained by teaching Computer Science students, on how we can bring embedded systems closer to them, to provide them the opportunity to fully participate in this growing field. We believe that by building on the CS students' strengths, while lessening the emphasis on the 'hardware' aspects, we can still allow them to master sufficient knowledge and skills to participate in this field.

Categories and Subject Descriptors
K.3.2 [Computer and Information Science Education]: *computer science education, embedded software education.*

General Terms
Computer Science Education, Curriculum.

Keywords
Embedded software, embedded systems education.

1. INTRODUCTION
Embedded systems design as a discipline has long been pursued in the industry in various application domains including avionics, aerospace, automobile, and industrial control etc. As pointed out by Lee [12], this area has largely been ignored by academics because it has not thrown up sufficient complex research challenges. However, all this has changed recently with increasing interest being paid by the academic community towards embedded systems education, as evidenced by the papers published in the recent special issue of ACM Transactions on Embedded Computing Systems dedicated to embedded systems education [1] and the successful organization of the Workshop on Embedded Systems Education (WESE) for the past two years [7][8].

Embedded systems as a field, is often perceived as a conglomeration of different areas, and is truly interdisciplinary [4][7]. Thus teaching "embedded systems" as a unified topic is viewed as a difficult task [6]. This field is often perceived by undergraduate Computer Science (CS) students as "hardware" oriented, not without reason. In our experience at our university, this perception hinders CS students from considering this dynamic field as an option for their future career, despite the fact that embedded systems offers exciting new opportunities for them. This is truer these days with the growing importance of the "software" component of embedded systems. In this situation it is imperative that CS students' misconceptions about this field be removed and they be introduced to the exciting new possibilities.

As we reported earlier [13], most university courses tend to emphasize the "systems" in embedded systems. The emphasis on the "software" side is not that prevalent. We take the latter approach by specifically designing a course emphasizing the embedded software aspects, especially the programming, software design and development and software engineering practice. The hardware aspects are deemphasized by considering standardized platforms, running real-time operating systems like Windows CE or Embedded Linux, which are easier to handle from the software development perspective for CS students. At the same time, many of the constraints encountered in embedded systems, like limited memory, limited power source and other resource limitations and their impact on software design can still be conveyed to the students.

The recent papers on embedded education in [1] have mostly emphasized the design of a comprehensive embedded systems curriculum. These have often originated in universities with established embedded systems research groups. Many universities may not have the expertise, resources or flexibility to introduce comprehensive curricula dedicated to embedded systems. Instead, a more feasible alternative would be to offer elective courses on embedded systems and software. It is from this perspective that we introduced two new courses into our curriculum at the Hong Kong University of Science and Technology (HKUST), one aimed at embedded systems and the other at embedded software, to give our undergraduate students the flexibility to take these as electives in order to gain knowledge and familiarity with embedded systems.

In this paper, we describe our experience in designing and offering an embedded software course specifically concentrating on the software requirements and design for embedded systems. We briefly revisit the course that we described earlier [13] including the list of topics, the hands-on laboratory exercises and some reflections on the experience with the second offering of the course. We also give a brief account of the students' perception and opinions on the course, after some modifications were introduced into the course based on feedback gathered in the earlier offering of the course [13].

The paper is organized as follows. Section 2 reviews some of the background for the course. Section 3 reviews some of the strengths and shortcomings of CS students. Section 4 provides the details of the course. Section 5 reflects on the students' opinions. Finally we give conclusions in Section 6.

2. BACKGROUND

Many universities have been actively engaged in designing new embedded systems curricula as evidenced by the papers published in the recent special issue of ACM Transactions on Embedded Computing Systems (TECS) dedicated to embedded systems education [1]. Similarly a number of papers describing faculties' experiences with designing and offering embedded systems related courses were published in the previous workshops on embedded systems education [7][8].

It is interesting to note that most of the papers in the special issue of TECS [1] were dedicated to describing embedded systems courses offered at the graduate level or about comprehensive undergraduate curricula. The emphasis has been on comprehensive curricula but with systems perspective. Two papers [2][19] did put emphasis on embedded software.

The observations by Sztipanovits et al. [19] about computer science students' strength is worth noting. They mention that CS students tend to be trained well on abstractions, especially about creating formal abstractions of computational processes and to relate layers of abstractions to each other. However they lack training on the relationship to "physical" processes and systems. These students tend to be trained in formal automata-based and algebraic models, but lack training in the continuous-time domain. Thus designing embedded courses for CS students should keep in mind their strengths and weaknesses.

Given the lack of flexibility in introducing embedded systems concepts into existing curricula without overtly disturbing the curricula, we need to find creative approaches to bring about the modifications. One possible approach that we suggested [13] was to introduce the embedded software concepts into various related CS courses like operating systems, compilers, architecture, programming languages etc. It is interesting to note that such an approach at least in the context of operating systems was also considered by Prof. Nutt [14]. In particular, he suggests that a traditional OS course could be reorganized to emphasize the growing importance of small computer systems. He views the computer systems space as consisting of three types based on the use of interrupts and the CPU mode bit: dedicated systems (DS), Truasted Process (TP) systems and Managed Process (MP) systems, each with increasing capability and complexity. In particular, the DS and TP systems covers what we normally view as the embedded systems space.

Prof. Lee [11] advocates a drastic rethinking on the way embedded software is developed, putting emphasis on the timing aspects of software. He emphasizes that the traditional approach to embedded software has paid more attention to optimizing the software for the resource limitations imposed by the platforms, rather than on timing issues. This overemphasis on efficiency with the consequent neglect of functionality and reliability, ill-serves the embedded software domain. Similarly he mentions that the thread model adopted for concurrent programming is not the best suited approach [10]. Threads introduce uncontrolled non-determinism which is detrimental to the functional and reliability requirements that most embedded software need to exhibit.

The model-driven approach [17][19] to embedded software development has also been advocated. Most of the existing courses that follow this approach are targeted at the graduate level because of the inherent complexity and the need to have sufficient background to appreciate this approach. As mentioned in [19] such concepts are often difficult to teach at the undergraduate level.

Grimheden and Törngren [4] present didactic analysis of embedded systems with detailed discussions to establish the legitimacy and identity of embedded systems. They establish that embedded systems have a functional legitimacy, implying emphasis on skills, and a thematic identity. They advocate an example driven approach with interactive communication.

Ricks et al. [16] discuss the specific needs of Electrical and Computer Engineering students to prepare them for embedded systems education. In particular they focus on the programming skills, or the lack thereof, of these students. They point out the shortcomings of the current approaches to teach programming, especially from the perspective of the skill requirements for developing software for embedded systems.

Sharad [18] discusses the use of graphical programming approaches to enable non-electrical engineering (and computer science) students to participate in the development of embedded systems. This approach enables the designers to concentrate more on the domain requirements, rather than get bogged down with the specific details of the embedded hardware and programming of the embedded platform.

3. STRENGTHS AND SHORTCOMINGS OF COMPUTER SCIENCE STUDENTS

Our own thinking and approach to teaching embedded software as reflected in this paper has been influenced by the views, opinions and suggestions presented in the literature mentioned above, and also by our own experience in teaching CS students over the past several years.

While it is very easy to bemoan the lack of suitable skills among our students to learn and appreciate the field of embedded systems, we believe that this leads us nowhere and may prove counterproductive. Instead if we focus on identifying the strengths and weaknesses of our students, we may be able to tailor our teaching approach and style to build upon the students' strengths while trying to address their shortcomings through appropriate corrective measures.

Towards this end, we start out by identifying the strengths and shortcomings of computer science students. These issues set the CS students apart from the Electrical Engineering/Computer Engineering (EE) students, who tend to concentrate more on the hardware aspects, while getting some software skills. We hope through this exercise to identify a suitable approach to teach embedded software to these students. Below, we list some of the strengths of CS students:

1. CS students often receive extensive training in high-level language programming, especially in two or more languages. These include both procedural and scripting languages, and object-oriented techniques. This should give them sufficient

skill to organize and implement reasonably complex software.

2. CS students receive sufficient training in formal models, finite automata and abstractions. Thus they are adept at expressing complex computational processes using abstract formalisms.
3. CS students are well trained in the mathematical techniques including discrete mathematics, required for analysis and design of systems.
4. They receive good training in algorithms and algorithmic way of expressing data representation and transformation. This includes analyzing the time and space complexity of the algorithms.
5. Software engineering is typically included in the CS curriculum, thus exposing the students to rigorous techniques for requirements analysis, design, implementation, testing and evaluation of complex software.
6. Most CS curricula will include courses on computer organization and architecture, and operating systems. These tend to be the only "systems" courses that CS students typically encounter during their undergraduate education.

We believe that the above (non-comprehensive) list of strengths gives CS students an advantage in working in the embedded software area. Their skills in formal techniques, algorithm design and software engineering should provide them distinct skills that can complement the skills of embedded systems designers with EE background.

We do however recognize some shortcomings of CS students which need to be addressed to give them a more well-rounded training to help them contribute to their full potential to embedded systems. These shortcomings include:

1. CS students get insufficient exposure to hardware and hardware interfacing. This topic is usually dealt with in a microcontroller/microprocessor course which normally EE students take as part of their curriculum.
2. The general programming skills imparted to CS students does not give them sufficient background to develop embedded software. The list of shortcomings of the training in programming skills imparted to students as pointed out by Ricks et al. [16] equally well applies to CS students. Thus the problem needs to be addressed not only for EE students, but also for CS students.
3. The systems related courses like Computer Architecture and Organization, and Operating Systems are not well understood or appreciated by CS students. We have been teaching these courses at our university for several years to CS students and speak from our own experience on this issue. The pre-conceived view among the students of these courses are "hardware" courses acts as a barrier for them which is difficult to break through and make them appreciate the need to understand the "hardware" aspects of computers.

4. OUR COURSE

In this section we give the details of our course, including the list of course topics, the hands-on laboratory exercises, and the student projects. We also reflect on our experience with the second offering of the course where modifications were introduced based on student feedback from the first offering of the course.

One important question that we need to consider is how a systems course should be structured. We believe that any systems course should emphasize the three "S" of systems: Science, Skills and State-of-the-art. We summarize this in Figure 1.

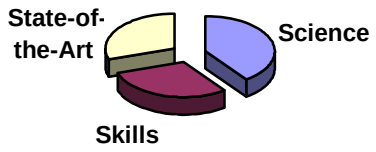

Figure 1: The Three "S" of Systems Courses.

By science, we mean the theory behind the systems. By skills we mean the practical skills required to apply the science learned to a real life environment. This includes mastering the tools and techniques used to develop real systems. By state-of-the-art we mean the current technology in the industry so that students that get trained can put their knowledge and skills to use. This is especially true for undergraduate courses which tend to have greater emphasis on the state-of-the-art in the field. We used this thinking as a guideline in designing our embedded software course, details of which are given next.

4.1 What Should Be Taught?

After having seen the review of various approaches and opinions presented in the earlier sections, the situation begs the question what should be taught to computer science students? More generally, what is feasible to be taught to CS students, given their background knowledge by the time the students reach their Junior/Senior year in their undergraduate education? Indeed, we were faced with this dilemma when we first set out to design and introduce embedded systems "elective" courses into our computer science/computer engineering curriculum. In this section, we reflect upon the guiding principles, thought processes and reasoning behind selecting various topics that we chose to include in the course on embedded software. While we do not claim to answer all the questions, we believe that this sharing of ideas might help elicit similar reflections on part of other faculty facing similar circumstances.

Through our own deliberations on this issue we came up with a set of topics that we felt will add sufficiently to the CS students' knowledge from the perspective of embedded systems. We felt that three broad categories of topics as listed below can be taught to CS students, leveraging on the students' background and the topics that are already part of the curricula. The emphasis was on embedded software development, with hardware concepts being only lightly touched where appropriate.

- Embedded Software Development: CS students are quite well-versed with software development. The main emphasis of this section therefore is how embedded software development differs from traditional software development. We found that the increasing use of integrated development environments like Microsoft Visual Studio™ isolates the students from the entire

process of compilation. Thus we felt it was important to emphasize this point, and especially make clear the concept of cross-platform development and issues related to cross-compilation. The students were introduced to the ideas of cross-platform development including host based embedded software development and embedded target environments, integrated development environments, interrupts and interrupt handling, and embedded software architectures.

- Real-Time Operating Systems (RTOS): CS students take operating systems as a core course during their study. The emphasis in such courses is usually placed on efficiency and fairness of resource sharing among processes, and the use of techniques like resource abstraction, virtual memory management and file systems. In the embedded world, the emphasis is more on reliability and timeliness. Thus in an embedded software course, real-time scheduling techniques with deadline constraints need to be emphasized. In our course, task scheduling topics including rate monotonic scheduling, the priority inversion problem and its solution were covered. Task synchronization issues including the use of semaphores and events were covered. Inter-task communication mechanisms including message queues, mailboxes and pipes were covered. Memory management issues including dynamic memory management, memory leak and dangling pointer problems were covered. Several example RTOS were also introduced in the course. As a simple and efficient real-time kernel, the μc/OS-II was first introduced. Two full-fledged real-time OS viz., Windows CE and Embedded Linux were introduced.

- Embedded Software Engineering: Trying to find a suitable set of material to cover under the embedded software engineering heading was the most difficult, because of the diffuse nature of the area. We leveraged on the existing techniques from software engineering and briefly covered several topics including embedded software development, software development lifecycle, software development models, including the waterfall model, the spiral model, rapid application development model, object-oriented approaches. Sufficient coverage was also given to software testing. Universal modelling language (UML) was introduced as an important formal method for software engineering, and its use in the different parts of the software development lifecycle was illustrated. We also concentrated on specific techniques for testing, verification and validation for embedded systems. It is interesting to note the observations made in Josesfsson [9] that industrial software engineering approach taught by universities are adequate from the technical perspective, but need more emphasis on business, teamwork and practical skills.

While these three are the broad categories of topics that we decided to cover in our course, we do acknowledge that other topics can be considered for inclusion. In particular we decided not to include topics on interfacing and device drivers in this course, leaving it to a companion course on embedded systems. Application programming interfaces like the approach adopted by Phidgets Inc. [15] makes it feasible to do programming with a conceptual view of the hardware, rather than being concerned with the details.

We do notice that the detailed curricula proposed in several papers in [1] at the undergraduate level typically include courses on real-time systems and software engineering. Our course is designed more as an umbrella course covering these topics within a single course, which within a semester offers a reasonable coverage of the aforementioned topics.

4.2 Course Topics and Structure

The list of topics covered in the course is given in Table 1.

Table 1: List of Course Topics

4. Introduction
   - Introduction to Embedded Systems
   - Examples of Embedded Systems
   - Embedded System Characteristics
5. Embedded Systems Architecture
   - Hardware Fundamentals: Processors, Memory, Bus, etc.
   - Software: OS, Application Software
6. Embedded Software Development
   - Hosts and Targets
7. Interrupts
   - Introduction to Interrupts
   - Interrupt Handlers and Interrupt Service Routines
8. Embedded Software Architectures
9. Real-Time Operating Systems (RTOS)
   - Review of Operating Systems Basics
     - Tasks, Processes and Threads
     - Task Scheduling: Rate Monotonic Scheduling, Priority Inversion
     - Task Synchronization and Coordination
     - Intertask Communication
     - Memory Management
   - Example RTOS: μC/OS-II, Windows CE, Embedded Linux
10. Embedded Software Engineering
    - Basics of Software Engineering
    - Software Engineering Models
    - Unified Modeling Language (UML)
    - Software Testing
11. Testing and Debugging Embedded Systems

4.3 Hands-on Laboratory Exercises

The hands-on laboratory component concentrated mainly on introducing the students to embedded software development using Windows CE. A general purpose teaching laboratory equipped with standard PCs was used for the laboratory exercises. The

majority of the labs were organized around the Microsoft Windows Embedded software including Platform Builder 4.2 and Windows CE. Students were introduced to the Platform Builder IDE, Visual Studio environment including the Embedded Visual C++ and ".NET" compact framework. Then the students did several laboratory exercises which were aimed at illustrating several RTOS concepts including threads, task scheduling, task synchronization, and memory management, including memory leaks. The laboratory exercises were mainly aimed at preparing the students for designing and implementing the course projects.

The students also had access to Ebox-II Windows CE 5.0 jump-start embedded software development kits from ICOP Technology Inc. [5]. This system is built around a Vortex X86 system on a chip technology with 128 MB system memory and 64 MB IDE bootable flash storage. Also additional Intel PXA-255 based embedded development kits from Emdoor Inc. [3] were also available.

4.4 Course Projects

The course project was an important part of the student assessment, in addition to quizzes and examinations. Students formed teams of up to 3 students per team and proposed their own project based on their interest. The students had about 2 months to develop their idea, propose the project, design and implement it. Students were encouraged to apply the software engineering principles learnt in the course during the design and implementation of the project, starting from requirements analysis to final implementation and testing.

The students were very enthusiastic and proposed and implemented interesting projects. While there were the typical projects implementing games on handheld devices, several unique projects were also designed. A list of some of the most interesting project topics with a brief description are presented below:

- Automatic "Dim Sum" Ordering System: the students designed a restaurant menu based ordering terminal for ordering Chinese "Dim Sum" which can be deployed at each table enabling customers to enter their orders online. The terminals are connected to a back-end order tracking system in the kitchen and pantry to schedule and supply the ordered dishes to the tables.
- On-line Retail Management System (RMS): This system enabled retail merchants to keep track of their inventories. The system was designed with a Pocket PC handheld connected to a barcode scanner, with the backend database support.
- Stock Manager: This system was implemented for a Pocket PC handheld with the provision for online stock information display and trading.
- Podcast CE: A Windows CE based device was designed to support podcast reception and playback.
- Video Surveillance System: This system enabled surveillance video being streamed to a Pocket PC.
- "Bomberlady" – an embedded linux game: This project implemented the classic "bomberman" video game on an embedded linux platform
- NewStation: This project designed a news kiosk based on Windows CE to display news headlines subscribed from various sources through RSS subscription
- Real Time Operating System – USTOS: This project aimed at designing a very compact real-time operating system with basic functionality from the scratch.

The final project reports of these projects is available online at http://www.cse.ust.hk/~muppala/comp355/project/reports.html.

As can be seen from the long list of topics, several interesting ideas were developed by the students. The students felt that this experience illustrated to them that a whole new arena of small device programming was easily accessible to them and provided them with alternate avenues for future career, compared to the traditional data processing field which is often the biggest employer of CS students in Hong Kong.

5. STUDENTS

As already mentioned the course was designed as a senior undergraduate course. In the second offering of the course, most of the students were in their third year (final year) of their undergraduate education at the university with a small number of students from the second year of study. It must be noted that the Hong Kong higher education system is based on a three year bachelor's degree program. Students entering the university are at the sophomore level of a typical US university. Thus students in the second and third year at the university here are equivalent to students in their junior and senior year of a 4-year bachelor's degree at a typical US university.

5.1 Background

Unlike the first offering of the course in Spring 2005 where most of the students taking the course (almost 99%) were doing their bachelor's degree in computer engineering, the second offering of the course in Spring 2006 attracted a balanced mix of students both from the Computer Science and the Computer Engineering stream. This was more heartening to note because the course was designed to attract students with both computer science and computer engineering background. Almost all the students were in their final year of study.

Two undergraduate courses were listed as prerequisites for our course, viz., Computer Architecture, and Principles of Systems Software (Operating Systems). Most students enrolled in the bachelor's programs in computer science or computer engineering at our university typically take these two courses by the time they have completed the first semester of the second year of their study. Since these two courses provide the necessary background required for introducing embedded systems, we had to devote only a short time at the beginning of our course to review the relevant materials before delving into the topics of our course.

5.2 Student Feedback

The course was very well received by the students in the second offering, surpassing the evaluations from the first offering of the course. At the end of the semester, a comprehensive survey of the students' opinions was carried out in order to assess how well the course met the students' expectations. The results of the survey indicated that the students were satisfied with the course. Some suggestions for improvement were given which are noted below:

1. Most students expressed interest in having more hands-on labs that currently available. The students expressed a desire to see at least some of the labs organized in a step-by-step

manner to illustrate the development of an example embedded system from conception to completion. Currently the labs are more focused on providing the students the skills in illustrating the RTOS concepts and illustrating the capabilities of Windows CE.

2. The topics that attracted the most interest among the students were embedded development, and RTOS. The least popular topic was software engineering, perhaps because of lack of demonstrative case studies. This was similar to the opinions expressed by the students in the first offering of the course.
3. In the earlier offering, the students expressed the opinion that the coverage of the RTOS should be limited to an in-depth coverage of only two or three major ones, rather than an overview of several RTOS. Thus in the second offering we restricted ourselves to only Windows CE and Embedded Linux. This was much better received by the students.
4. Some students expressed the opinion that some of the topics had overlap with other related courses that they had taken, and hence suggested that the overlap should be minimized. This was especially true for software engineering which is covered in detail in another course dedicated to the topic. We tried to address this problem to some extent in the second offering of the course, but could not completely avoid the overlap.

6. CONCLUSIONS

In this paper we discussed our effort at introducing an embedded software course, with the principal aim of bring embedded systems closer to computer science students. We described the structure of the course, the list of topics, the labs and the course projects implemented by the students. We also discussed some of the findings of the survey on students' opinions of the course. Through this effort we wished to illustrate that it is feasible to introduce computer science students to this exciting new field without getting trapped into the intricacies of the underlying hardware.

7. ACKNOWLEDGMENTS

The author wishes to thank the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology for providing him the opportunity to explore the issues presented in this paper.

8. REFERENCES

[1] Burns, A., and Sangiovanni-Vincentelli, A. Editorial for the Special Issue on Embedded Systems Education, *ACM Trans. Embedded Computing Systems, 4,* 3 (Aug. 2005), 469-471.

[2] Caspi, P. et al. Guidelines for a Graduate Curriculum on Embedded Software and Systems. *ACM Trans. Embedded Computing Systems, 4,* 3 (Aug. 2005), 587-611.

[3] Emdoor Inc. http://www.emdoor.com.

[4] Grimheden, M. and Törngren, M. What is Embedded Systems and How Should It Be Taught? – Results from a Didactic Analysis. *ACM Trans. Embedded Computing Systems, 4,* 3 (Aug. 2005), 633-651.

[5] ICOP Technology Inc., http://www.icop.com.tw.

[6] Koopman, P., et al. Undergraduate Embedded System Education at Carnegie Mellon. *ACM Trans. Embedded Computing Systems, 4,* 3 (Aug. 2005), 500-528.

[7] Jackson, D. J. and Caspi, P. Embedded Systems Education: Future Directions, Initiatives, and Cooperation. *ACM SIGBED Review Special Issue on the First Workshop on Embedded Systems Education (WESE 2005), 2,* 5 (Oct. 2005), 1-4.

[8] Jackson, D. J. *Proceedings of the Second Workshop on Embedded Systems Education (WESE 2006),* Seoul, South Korea, (Oct 2006).

[9] Josefsson, M., Ed. 2003. *Industriell Programvaruutveckling*. ITQ Nordic Institute.

[10] Lee, E. A. The Problem with Threads. *IEEE Computer, 39,* 5 (May 2006), 33-42.

[11] Lee, E. A. Absolutely Positively On Time: What Would It Take? *IEEE Computer, 38,* 7 (July 2005), 85-87.

[12] Lee, E. A. What's Ahead for Embedded Software? *IEEE Computer, 33,* 9 (Sep 2000), 18-26.

[13] Muppala, J. K. Experience with an Embedded Systems Software Course, *ACM SIGBED Review Special Issue on the First Workshop on Embedded Systems Education (WESE 2005), 2,* 5 (Oct. 2005), 29-33.

[14] Nutt, G. An OS Course for Small Computer Systems, submitted for publication, http://www.cs.colorado.edu/~nutt/SCC-OS-paper.pdf.

[15] Phidgets, Inc. http://www.phidgets.com/.

[16] Ricks, K. G. et al. Addressing Embedded Programming Needs within an ECE Curriculum, *Proceedings of the Second Workshop on Embedded Systems Education (WESE 2006),* Seoul, South Korea, (Oct 2006), 22-28.

[17] Sangiovanni-Vincentelli, A. L., and Pinto, A. An Overview of Embedded System Design Education at Berkeley. *ACM Trans. Embedded Computing Systems, 4,* 3 (Aug. 2005), 472-499.

[18] S. Sharad, Methodologies to Bring Embedded Systems to Non-EE Students, *Proceedings of the Second Workshop on Embedded Systems Education (WESE 2006),* Seoul, South Korea, (Oct 2006), 41-44.

[19] Sztipanovits, J. et al. Introducing Embedded Software and Systems Education and Advanced Learning Technology in an Engineering Curriculum. *ACM Trans. Embedded Computing Systems, 4,* 3 (Aug. 2005), 549-568.