Towards laying common grounds for embedded system design education

Peter Marwedel, *Senior Member, IEEE*

Abstract—**In this paper, we propose to introduce a common introductory course for embedded system education. The course puts the different areas of embedded system design into perspective and avoids an early over-specialization. Also, it motivates the students for attending more advanced theoretical courses. The content, the structure and the prerequisites of such a course are outlined. The course requires a basic understanding of computer hardware and software and can typically be taught in the second or third year.**

Index Terms—**Embedded systems, education, introduction, curriculum**

I. Introduction

According to many forecasts, the importance of embedded systems will be growing over the coming years. It is obvious, that traditional education focusing either mostly on hardware (as in many EE programs) or mostly on software (as in many CS programs) will not be sufficient. According to [1], there is a lack of vision and a lack of maturity of the domain and many courses do not present a sufficiently wide perspective. According to the same source, the result is that industry has difficulty finding adequately trained engineers, fully aware of design choices. Consequently, new educational programs have to be designed to provide graduates with the required knowledge and skills to design embedded systems. It would be feasible to design a special program for embedded system design. This was done, for example, at ALARI (see www.alari.ch). However, due to the limited resources at most universities and in order to avoid an inflation of programs, we suggest to incorporate the required education into existing EE and CS programs. This allows the efficient use of resources and avoids over-specialization of students. However, this leaves us with the problem of identifying the areas to be covered in embedded system education and also with the problem of properly integrating embedded system education into existing curricula. Mutual dependences have to be identified. This paper presents answers to the above problems.

It is structured as follows: related work will be presented in section II, and the proposed course will be described in section III. In section IV, we will discuss our experience with the course structure. Section V will provide a conclusion.

Manuscript received July 30, 2005.

Peter Marwedel is with the University of Dortmund, 44221 Dortmund, Germany; phone: +49 231 755 6111; fax: +49 231 755 6116; e-mail: peter.marwedel@udo.edu.

II. Related Work

To a major extent, course structures are influenced by available text books. Hence, currently available text books (together with information about courses on the web) give an impression on how embedded systems are currently taught.

Traditionally, text books on embedded system design have focused on the very specific problem of interfacing computers to physical environments and the programming of these computers with interrupts and memory maps. There are a significant number of such books [2]-[6].

However, this view of embedded systems is much too restricted. The scope of embedded system design has recently been described in a book by Sifakis et al. [7]. While this book lists the content that should be covered in research, it does not present approaches for how embedded system education can be taken into account by universities.

Other text books that are available cover other specific areas of embedded system design. For example, the book by Jantsch [8] focuses on models of computation. A similar remark applies to the text book by Vahid [9]. There, the focus is on implementing finite state machines. The book by Wolf [10] is used for many courses. However, a number of important topics are not covered in the book. In general, covered areas are certainly important, but courses based on those books fail to provide a broad overview over issues in embedded system design. A broader view is also requested in [1]. It is stated clearly that “training takes place continuously during professional life, and it is not easy to distinguish what should be learned during primary education and during continuous training. Yet, it seems that fundamental bases are really difficult to acquire during continuous training if they haven’t been initially learned, and we can think we must focus on them.” The proposed course is consistent with this view and can be seen a key element in the implementation of the requirements. However, our approach aims at the undergraduate level, whereas [1] tries to avoid making proposals for already tightly packed undergraduate curricula. Reference [1] also introduces the distinction between “a deductive style of education, where students go from theory to practice” and “a more inductive approach, which adopts the reverse order.” Our approach is more of the deductive style, but does include references to applications. A broad view of embedded system design is also implemented in the Berkeley approach to embedded system design [11]. The Berkeley approach seems to be focussing more on the use of tools. Due to the differences between the two approaches, some of the tools used in the Berkeley course are left for more advanced courses in our approach, the "Dortmund approach".

III. Outline of the Proposed Course

A. Content of the proposed course

The course proposed in this paper has been designed over a period of almost ten years. During this period, material has been added to and deleted from the course. The selection of the material for the current version of the course is based on an analysis of presentations at conferences, reviewed papers, discussions with industry and other personal talks. Overlaps with existing courses (e.g. on control theory or digital signal processing) have been removed.

The resulting scope of the course is the following:

1. Introduction (Definitions, scope, examples, common properties);
2. Specification techniques: Models of computation, communication methods, StateCharts, SDL, VHDL, Petri nets, UML diagrams;
3. Embedded system hardware: hardware in the loop, discretization, communication, processors, FPGAs, memory, D/A-converters;
4. Scheduling, operating systems for embedded systems and other standard software: standard real-time scheduling algorithms, properties of RTOSes, fundamentals of middleware;
5. Implementing embedded systems with hardware/software codesign: high-level transformations (loop transformations), array folding, task concurrency management, hardware/software partitioning, optimizations for power reduction, specialized compiler techniques for embedded systems, design flows;
6. Validation: simulation, types of models in formal verification, introduction to issues in testing.

This course is designed for about 60 hours (at 45 mins) of lectures and 30 hours of labs.

The order of the presentation is the order used above and is consistent with the dependencies between design information (see fig. 1).

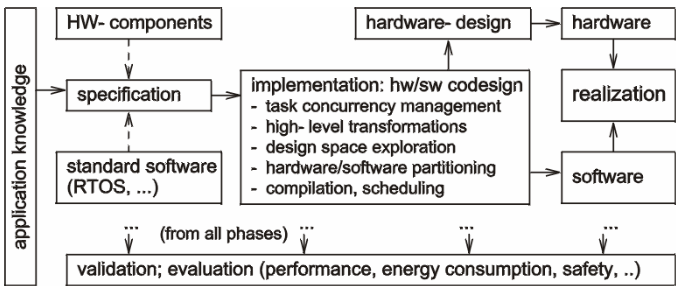
application knowledge
HW- components
specification
standard software (RTOS, ...)
hardware- design
implementation: hw/sw codesign
- task concurrency management
- high- level transformations
- design space exploration
- hardware/software partitioning
- compilation, scheduling
hardware
realization
software
(from all phases)
validation; evaluation (performance, energy consumption, safety, ..)

Fig. 1: Design information flow

Application knowledge, hardware design and evaluation are not covered in the course. Application knowledge can only be taught in more specialized courses and only one or two examples can be included in a curriculum. Teaching hardware design is deferred: since many CS students will actually not be involved in hardware design, it is taught in a more specialized "EDA" course (see below). It would be nice to include evaluation in the introductory course. However, we found it rather difficult to select material that is applicable to a wide range of situations. Reliability evaluation is a notable exception, since standard techniques are known for that area.

The course is complemented by a text book [12] and slides on a web site [13]. The web site also contains links to related information and to courses referring to the text book.

Obviously, it is not feasible to cover all potential topics that colleagues might want to teach in the suggested course. Therefore, the structure of the course has been designed such that "plug-in's" can be easily added. Such "plug-in's" provide more detailed information which the presenters might want to focus on. In fact, we have designed some plug-in's ourselves. These include the following:

- Detailed description of UML diagrams;
- Computation of invariants of Petri nets;
- Proof of optimality of rate monotonic scheduling;
- D-algorithm for gate-level testing.

These plug-ins are available in "more in-depth" sections on the slides.

A lab is an indispensable part of the course. Due to the broad coverage of topics, the lab cannot and should not offer hands-on experience to tools in all of the above six areas. We propose to use a mixture of theoretical assignments and a limited set of tools to be used. The following list includes examples for each of the above areas:

1. Search for definitions, characterization of embedded systems, implications of the definition of reliability and maintainability;
2. Using tools for the hierarchical description of finite state machines like StateMate or StateFlow (hands-on experience), proofs concerning the depth of SDL FIFO buffers (simple examples), designing and showing properties of Petri nets, working with UML diagrams;
3. Using LEGO® Mindstorm robots as an example of hardware in the loop (hands-on experience);
4. Solving scheduling problems, simple proofs in scheduling theory;
5. Generation of integer programming models for scratch pad allocation, hardware/software partitioning and dynamic voltage scaling, using the SCE system on a chip (SoC) design environment based on SpecC [14], experimenting with program optimizations like loop tiling and loop unrolling;
6. Generation of examples of test cases, manual test compression of test responses, writing self-test programs for processors.

B. Prerequisites

The proposed course requires the students to have gained experiences in the following areas (see fig. 2):

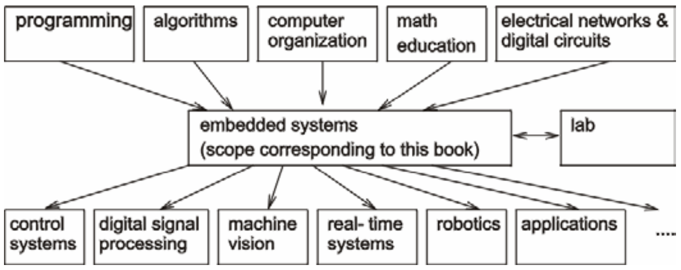

Fig. 2: Prerequisites and follow-up courses

- Basic programming skills and the knowledge of fundamental algorithms like topological sorting.
- A basic understanding of computer arithmetic, computer structures and computer organization as well as the implementation of higher level languages by assembly programs. This requirement can be met by attending a course based on the introductory book by Hennessy and Patterson [15]. The fundamentals of finite state machines must be known.
- Math education, including linear algebra and probability theory may be required for the discussion of certain more specialized material. Integrals should be known as a result of high-school education.
- A basic understanding of electronic circuits is necessary for the section on embedded system hardware. This includes the capability to understand digital logic (especially CMOS logic), Kirchhoff's laws and operational amplifiers. It is assumed that physical terms such as currents, voltages, power, energy, and electrical fields are also known (typically, the level reached at good high-schools is sufficient).
- Basic understanding of operating systems, including memory maps, the use of interrupts, system calls, real-time clocks and timers, mutual exclusion and task synchronisation.

Obviously, the list of prerequisites is not very long. Typically, the proposed course can be taught rather early in the curriculum. For CS students, all prerequisites are available in the fourth term at our University. Due to some other constraints, the fifth term is the first term in which students can actually enrol themselves into the course (and a large amount of students do).

For computer engineering (CE) or information technology (IT) students, all prerequisites should be available in the fourth term as well.

For electrical engineering (EE) students, programming skills, algorithm knowledge and knowledge about operating systems may be missing. However, it would make sense to add these courses for EE students who would like to specialize in embedded systems.

C. *Suggested follow-up courses*

Due to the rather broad coverage of embedded systems in the suggested course, it is recommended to extend the students knowledge in more specialized courses. Such more advanced courses could include the following topics (see also fig. 2):

- Control theory;
- Digital signal processing and wireless communication;
- Machine vision;
- Real-time systems, advanced scheduling algorithms, scheduling theory;
- Robotics;
- Courses on selected application areas (automotive, telecom, consumer market, industrial control);
- A large application project;
- Presentations by the students on selected advanced topics;
- Electronic design automation (EDA) and hardware design, SystemC, using field programmable gate arrays (FPGAs), hardware synthesis algorithms, placement, routing;
- Formal verification of embedded systems, equivalence checking, model checking, theorem proving.

These courses should include hands-on experiences wherever possible. It is suggested to include a major project in the educational program. At Dortmund, such projects are organized such that students have to work in teams of up to 12 students (so-called project groups). In a typical CS program, only a certain percentage of the students will select an embedded systems project. Those who do, benefit from the described course. Skills resulting from such a project should be comparable to those resulting from the corresponding courses at Berkeley [11].

Obviously, some of these courses do already exist at many universities. Preceding these courses by an embedded systems course as outlined should improve the motivation of the students and put the more specialized material into perspective.

IV. Experiences

The proposed course has evolved during the past ten years. During the last two years, the course has been taught from the published textbook and slides. Two types of classes were involved:

1. Each winter term, the course is taught in German to about 100 students. While the majority of the students are going for a diploma degree in computer science, the course has also been opened for students going for a diploma degree in information technology offered by the department of electrical engineering and information technology.
2. Each summer term, about 3/4 of the material is covered in a shortened course that is given in English. Students in this course are going for a master's degree in automation and robotics, are guest students from various other countries or are going for the same degree as the students in the winter course, but would like to improve their foreign language skills. This course is typically attended by about 40 students.

Both courses include a lab, mid-terms and finals. Labs comprise theoretical and practical work. Theoretical work consists of solving assignments, e.g. on real-time scheduling. Practical work involves programming Lego Mindstorm robots and using hierarchical state diagram specification techniques.

A dominating observation for all the courses following the structure outlined is the large motivation and enthusiasm with which the courses were received. Students consistently reported that the courses opened a new area for them.

While all courses followed the structure, no “plug-in's” were available in the very first iteration. This resulted in requests for some more in detailed coverage of certain areas. This is taken into account by adding special “in-depth” sections to the course and also to the slides. These sections cover, for example, proofs for the optimality of rate-monotonic scheduling. The introduction of these sections clearly improved the quality of the course, as no such questions popped up in the second winter term and as they were appreciated by colleagues. Future improvements will extend this direction and will cover, for example, the mathematical notions for reliability evaluation.

Furthermore, it was found that good slides offered by colleagues could be easily integrated into our own set of slides as the structure of the course was adequate.

Based on the experiences made so far, it was also decided to spend additional effort on preparing a set of standard assignments. Another experience concerns the overlap for CS students: hierarchical state diagrams and UML are covered in software engineering courses. Hence, too much emphasis on these techniques has to be avoided.

The last iteration of the German course was complemented by a follow-up course on electronic design automation (EDA). About 40% of the students of the embedded systems courses enrolled for the EDA course. The EDA course included a more detailed coverage of SystemC, FPGA programming and EDA algorithms (each about 1/3 of the course). The section on FPGA programming brought the students in contact with hardware circuits (in addition to the ones that are used in the LEGO® mindstorms). This course included about 60 hours (at 45 minutes) of lectures and 30 hours of labs. During the course, it was realized that the ES course had laid excellent foundations for this more advanced course. A number of topics could be discussed at a more detailed level, since the fundamentals were already known. The deductive approach turned out to work well in practice. Based on this experience, it was decided to offer this sequence of courses on a regular basis.

More advanced topics are typically also covered in seminars (presentations by students). Each student has to attend a course exclusively based on such presentations. For example, we have offered such “seminar” courses on security in embedded systems, and on reliability modelling in embedded systems. Again, the broad knowledge provided in the described embedded system course laid excellent foundations for the presentations.

V. Conclusion

In the paper, we have proposed the structure of a standard course on embedded systems that can be taught rather early in a computer science, computer engineering or electrical engineering curriculum. The course has been well-received by a large number of students and the corresponding text book has been picked-up by a number of departments. It is suggested to introduce a course following the structure outlined above into the CS, CE and EE education.

References

[1] ARTIST network of excellence: Guidelines for a Graduate Curriculum on Embedded Software and Systems, *http://www.artist-embedded.org /Education/Education.pdf*, 2003

[2] Jack G. Ganssle: Programming Embedded Systems, *Academic Press*, 1992

[3] Stuart R. Ball: Embedded Microprocessor Systems – Real world designs, *Newnes*, 1996

[4] Stuart R. Ball: Debugging Embedded Microprocessor Systems, *Newnes*, 1998

[5] Michael Barr: Programming Embedded Systems, *O'Reilly*, 1999

[6] Jack G. Ganssle: The Art of Designing Embedded Systems, *Newnes*, 2000

[7] Bruno Bouyssounouse and Joseph Sifakis: Embedded Systems Design: The ARTIST Roadmap for Research and Development, Springer *Lecture Notes in Computer Science, Vol*. 3436, 2005

[8] Axel Jantsch: Modeling Embedded Systems and SoCs: Concurrency and Time in Models of Computation, *Morgan Kaufman*, 2003

[9] Frank Vahid: Embedded System Design, *John Wiley & Sons*, 2002

[10] Wayne Wolf: Computers as Components, *Morgan Kaufman Publishers*, 2001

[11] Alberto L. Sangiovanni-Vincentelli and Alessandro Pinto: An Overview of Embedded System Design at Berkeley, *ACM Transactions on Embedded Computing Systems* (to appear), available *at http://www-cad.eecs. berkeley.edu/~apinto/Data/publications/TECS-Education.pdf*

[12] Peter Marwedel: Embedded System Design, *Kluwer Academic Publishers*, 2003 and *Springer*, 2005

[13] Peter Marwedel: Home page for the embedded system design book, *http://ls12-www.cs.uni-dortmund.de/~marwedel/kluwer-es-book.html*

[14] Samar Abdi, Junyu Peng, Haobo Yu, Dongwan Shin, Andreas Gerstlauer, Rainer Doemer, Daniel Gajski: System-on-Chip Environment, SCE Version 2.2.0 Beta Tutorial, *CECS Technical Report # 03-41, http://www.cecs.uci.edu/~cad/sce.html*

[15] John A. Hennessy and David A. Patterson: Computer Organization – The hardware/software interface, *Morgan Kaufman Publishers*, 1995