Organizers’ Message

We are pleased to present this first workshop on embedded system education associated with the 2005 EMSOFT embedded software conference. This is one of the first workshops specifically devoted to systems and software and we believe this is an important field that deserves to be addressed. First it is a very strategic domain from an economic point of view. Some statistics show that a huge proportion of microprocessor production (98% according to some sources) is used in embedded system applications. Second the field is fragmented into many application domains, e.g. aerospace, automotive, railways, telecommunications, consumer electronics, which all have their own design methods and philosophy. Therefore, it is urgent that educators meet together and discuss what and how to teach in order to get both usefully trained students and researchers able to bridge the methodological gaps that exist between this large variety of application fields.

It is in this spirit that we organized the workshop. Two invited speakers from the University of California, Berkeley and the Royal Technology Institute of Stockholm will present some of their most advanced experiences in embedded system and software education. Multiple sessions focusing on embedded systems curricula and content; teaching experiences; and labs and platforms used in embedded systems education will be conducted during the workshop. We can reasonably expect the sessions to give matter to lively, animated and enriching discussions. We also anticipate that the panel discussion that will conclude the workshop will allow us to delve deeper into the subject and raise useful conclusions.

We feel this first workshop is a great success. However, there exist many venues for improving cooperation among interested parties and facilitating greater multidisciplinary interactions. There are many things that remain to be done on the subject and there is room for significant improvements that will be the task of future editions of the workshop and our successors in their organization.

Paul Caspi
Jeff Jackson
2005 Workshop on Embedded Systems Education

September 22nd, 2005, Jersey City, New Jersey, USA
(a satellite event of EMSOFT 2005)

Program

9:00 Opening

9:15 Invited Session

- Alberto San Giovanni-Vincentelli: Embedded System Education: A New Paradigm for Engineering Schools?

10:00 Coffee Break

10:30 Curricula and Contents

- Suehee Pak, Eunha Rho, Juno Chang and Moon Hae Kim: Demand-Driven Curriculum for Embedded System Software in Korea
- Masaki Yamamoto, Hiroyuki Tomiyama, Hiroaki Takada, Kiyoshi Agusa, Kenji Mase, Nobuo Kawaguchi, Shinya Honda and Nobuyuki Kaneko: NEXCESS: Nagoya University Extension Courses for Embedded Software Specialists
- Peter Marwedel: Towards laying common grounds for embedded system design education
- Jogesh K. Muppala: Experience with an Embedded Systems Software Course

12:30 Lunch

14:00 Invited session

- Martin Grimheden and Martin Torngren: How should embedded systems be taught? Experiences and snapshots from Swedish higher engineering education

14:45 Teaching Experiences

- Bettina Weiss, Günther Gridling and Markus Proske: A Case Study in Efficient Microcontroller Education
- Voin Legourski, Christian Trödhandl and Bettina Weiss: A System for Automatic Testing of Embedded Software in Undergraduate Study Exercises

15:45 Coffee Break

16:00 Labs and Platforms
• **Stephen A. Edwards**: *Experiences Teaching an FPGA-based Embedded Systems Class*
• **Kenneth G. Ricks, David J. Jackson and William A. Stapleton**: *An Evaluation of the VME Architecture for Use in Embedded Systems Education*
• **Falk Salewski, Dirk Wilking and Stefan Kowalewski**: *Diverse Hardware Platforms in Embedded Systems Lab Courses: A Way to Teach The Differences*

**17:30 Panel**: *Embedded Systems Education: Future Directions, Initiatives, and Cooperation*
# Table of Contents

**Organizers’ Message** ........................................................................................................................................ ii

**Program** .................................................................................................................................................... iii

Embedded System Education: A New Paradigm for Engineering Schools? ........................................ 1  
  *Alberto Luigi Sangiovanni-Vincentelli and Alessandro Pinto, University of California at Berkeley*

Demand-Driven Curriculum for Embedded System Software in Korea ............................................. 11  
  *Suehee Pak, Dongduk Women’s University; Eunha Rho, Sungkonghoe University; Juno Chang, Sangmyung University; and Moon Hae Kim, Konkuk University*

NEXCESS: Nagoya University Extension Courses for Embedded Software Specialists ............... 16  
  *Masaki Yamamoto, Hiroyuki Tomiyama, Hiroaki Takada, Kiyoshi Agusa, Kenji Mase, Nobuo Kawaguchi, Shinya Honda and Nobuyuki Kaneko, Nagoya University*

Towards Laying Common Grounds for Embedded System Design Education ............................. 21  
  *Peter Marwedel, University of Dortmund*

Experience with an Embedded Systems Software Course ............................................................. 25  
  *Jogesh K. Muppala, The Hong Kong University of Science and Technology*

How Should Embedded Systems be Taught? Experiences and Snapshots From Swedish Higher Engineering Education ................................................................. 30  
  *Martin Grimheden and Martin Törngren, Royal Institute of Technology*

A Case Study in Efficient Microcontroller Education ........................................................................ 36  
  *Bettina Weiss, Günther Gridling, and Markus Proske, Vienna University of Technology*

A System for Automatic Testing of Embedded Software in Undergraduate Study Exercises ...... 44  
  *Voin Legourski, Christian Trodhandl, and Bettina Weiss, Vienna University of Technology*

Experiences Teaching an FPGA-based Embedded Systems Class .................................................. 52  
  *Stephen A. Edwards, Columbia University*

An Evaluation of the VME Architecture for Use in Embedded Systems Education .................... 59  
  *Kenneth G. Ricks, David J. Jackson, and William A. Stapleton, The University of Alabama*

Diverse Hardware Platforms in Embedded Systems Lab Courses: A Way to Teach the Differences  
  *Falk Salewski, Dirk Wilking, and Stefan Kowalewski, Aachen University*
Abstract—Embedded systems are emerging as an essential component of modern electronic products. Embedded system design problems are posing challenges that involve entirely new skills for engineers. These skills are related to the combination of traditionally disjoint engineering disciplines. There is a shared concern that today’s educational systems are not providing the appropriate foundations for embedded systems. We believe a new education paradigm is needed.

We will argue this point using the example of an emerging curriculum on embedded systems at the University of California at Berkeley. This curriculum is the result of a distillation process of more than ten years of intense research work. We will present the considerations that are driving the curriculum development and we review our undergraduate and graduate program. In particular, we describe in detail a graduate class (EECS249: Design of Embedded Systems: Modeling, Validation and Synthesis) that has been taught for six years. A common feature of our education agenda is the search for fundamentals of embedded system science rather than embedded system design techniques, an approach that today is rather unique.

Index Terms—Embedded System Design, Education.

I. INTRODUCTION

Embedded systems have been a strong research area for the University of California at Berkeley. We will briefly review this intense research activity as a preamble to present the Berkeley effort in embedded system education that is intimately related to the research program.

The research activities on embedded systems at Berkeley can be cast in a matrix organization where vertical research areas cover application domains such as automotive, avionics, energy, industrial control, and horizontal areas cover enabling technologies such as Integrated Circuits, Sensors, Wireless Networks, Operating Systems, Embedded Software, Automatic Control, Design Methodologies and Tools. The important aspect of our approach is that the enabling technologies are explicitly linked to the vertical application areas and are geared towards the embedded system domain.

At Berkeley, we have traditionally based our research programs on a strong interaction with industry and collaboration among faculty in different disciplines; embedded system research is no exception.

Among embedded system application domains, automotive has been an area of interest for many years. The PATH project [41] of CALTRANS (California Transportation Department) has been a test bed to develop new concepts in control of distributed systems, modeling, tools and methodologies with a strong experimental part and an intense interaction with industry. The automotive emphasis of our design methodology work dates back to 1988 when a joint program on formal approaches to embedded controller design with Magneti Marelli for their Formula one robotized gear shift for Ferrari started. In the automotive domain, there has also been strong interdepartmental interaction between mechanical engineering and electrical engineering/computer science.

In US Universities, bottom-up aggregation of interests and approaches to education is more common than top-down planning. Hence, education initiatives in novel areas almost always start with advanced graduate course offerings to migrate towards coordinated graduate programs and eventually into undergraduate courses. Thus, it is no wonder that course offering in Berkeley on embedded systems has been strong for years in the advanced course series (the EE and CS 290 series) that are related to faculty research activities. One such course has migrated to a regular offering in the graduate program (EECS249: Embedded System Design: Modeling, Analysis and Synthesis), the main topic of this paper.

The guiding principle in our teaching and research agenda related to embedded systems is to bring closer together system theory and computer science. The two fields have drifted apart for years while we believe that the core of embedded systems intended as an engineering discipline lies in the marriage of the two approaches. While computer science traditionally deals with abstractions where the physical world has been carefully and artfully hidden to facilitate the development of application software, system theory deals with the physical foundations of engineering where quantities such as time, power and size play a fundamental role in the models upon which this theory is based. The issue then is how to harmonize the physical view of systems with the abstractions that have been so useful in developing the CS intellectual agenda. We argue that a novel system theory is needed that at the same time is computational and physical. The basis of this theory cannot be but a set of novel abstractions that partially expose the physical reality to the higher levels and methods to manipulate the abstractions and link them in a coherent whole. The research community is indeed developing some of the necessary results to build this novel system theory and we believe it is time to inject these findings in the teaching infrastructure so that students can be exposed to this new way of thinking that should advance the state of embedded system design to a point where reliable and secure distributed systems can be designed quickly, inexpensively and with no errors.

The paper presents the guiding principles we have followed
in our education effort and the set of courses offered that have direct relevance to the field of embedded system design. We list only the courses whose embedded system content is explicitly addressed. Otherwise, we may end up with a comprehensive list of all courses offered in engineering (except maybe a few) as today electronic system design is almost a synonym of embedded system design. In particular, we present first (Section 2) the graduate program: we zoom in on EECS249 and then we briefly review a set of advanced topical courses on embedded systems. In Section 3, we present an overview of our undergraduate program centered on a sophomore core course \(^1\) (EECS20N \([30], [32]\)) that has been now offered over the past five years. This course for EE and CS students addresses mathematical modeling of signals and systems from a computational perspective. This reflects an effort of Berkeley faculty to set new foundations for the education in electrical engineering that is based on fundamentals rather than application domains. In this section, we also offer a view on our programs for the near future to address specifically embedded systems at junior and senior level. In Section 4, we offer concluding remarks that could be of use for setting up a graduate or advanced undergraduate class in embedded system design in other institutions.

II. THE GRADUATE PROGRAM PART 1: EECS249 DESIGN OF EMBEDDED SYSTEMS: MODELS, VALIDATION AND SYNTHESIS

This course \([15]\) is part of the “regular” graduate program in EECS. It is taken by first year graduate students as well as by senior graduate students in EECS and other departments such as Mechanical Engineering, Nuclear Engineering and Civil Engineering.

The idea of the course is to emphasize the commonality among the variety of application fields and to use a design methodology as the unified framework where the topics of the course are embedded. In this way, the variety of the advanced courses offered in our curriculum benefits from the foundations laid out by EECS249. The course is rather unique as it aims at bringing together the behavioral aspects of embedded system design with implementation within a rigorous as possible mathematical framework. Behavior capture, verification and transformation are taught using the concepts pioneered by Edward Lee associated to models of computation. The implementation design is seen as a companion to the behavioral design as opposed to a more traditional academic view where implementation follows in a top-down fashion behavioral design. We adopt the view presented in \([45] [46]\) to provide the intellectual background. In this methodology, the design proceeds by refinement steps where the mapping of behavior into “architecture” is the step to move to the next level of abstraction. Using this view, embedded software design is the process of mapping a particular behavior on a computing platform. By the same token, the choice of a particular distributed architecture due to geographical distribution or to performance optimization is handled in a unified way. The choice of components including reconfigurable and programmable parts is the result of architectural space exploration where cost functions and constraints guide the search.

From this brief overview, it should be clear that our motivation is to bring out the fundamental issues and the formalization that enables verification and synthesis at a level that would not be possible otherwise. This particular aspect should be seen as the quest for a new system science that serves as a framework to teach design that transcends the traditional discipline boundaries.

Given the large scope of the course, it has a heavy load; four contact hours and two lab hours per week. The contact hours are broken into traditional lectures and discussion of papers presented by the students. The verification of the learning process is left to weekly homework that are a mix of exercises and of theory, and to a final project that is fairly advanced, so much so that often the project reports see the light in the community as conference or journal papers.

The contents and the organization of the class has been the result of a number of advanced courses in hybrid systems and system level design that date back to 1991, when the first such class was taught.

A. Organization of the class

The basic tenet of the methodology that forms the skeleton of the class is orthogonalization of concerns, and, in particular, separation of function and architecture, computation and communication. This methodology called platform-based design is presented as a paradigm that incorporates these principles and spans the entire design process, from system-level specification to detailed implementation. We place particular emphasis on the analysis and optimization of the highest levels of abstraction of the design where all the important algorithmic and architectural decisions are taken.

The course has four main parts that are summarized in table I.

**Introduction and Methodology.** After a presentation of the motivation for the class extracted from a variety of examples in different industrial domains, we introduce the methodology followed (Platform-based Design \([45]\)) and examples of its applications. We present many examples of embedded systems: cars, airplanes, smart buildings, elevators, and sensor networks. In the introductory lecture, we highlight commonalities among all the examples that we present to set the stage for the philosophy of the course aimed at defining the common methods that can be used across different application domains; the course is intended to solve “the embedded system design problem” rather than particular instances of it.

An entire lecture is devoted to an overview of the platform-based design principle. The method is justified by illustrating how it can be used to solve, or at least, to formalize, design problems that are common to the entire class of embedded systems.

**Function.** The notion of behavior is analyzed and the role of non-determinism in specification is explained. We present the basic models of computation that are needed to represent the behavior of most designs: Finite-State Machines, Synchronous

---

\(^1\) A core course is a required course for the educational programs offered by the Department
Languages, Data Flow Networks, Petri Nets, Discrete Event Systems and Hybrid Systems. For each model we present the computation, communication and coordination semantics with a particular emphasis on the properties that are guaranteed and verifiable. We outline the use of unified frameworks to compare the different models of computation and we present the Tagged-Signal Model (TSM) [31] and agent algebras [40] as unifying theories to allow the composition of different models of computation to describe a complete system. We introduce here the Ptolemy [44] and Metropolis [3] environments for analysis and simulation of heterogeneous specifications.

We then ventured in the presentation of the model of computation used in Metropolis: the Metropolis Meta Model (MMM). The MMM can be considered an abstract semantics since it can accommodate a variety of models of computation that are obtained by refinement of this model. We presented the additional constructs that are used in Metropolis to capture the specification of a design in a declarative style (a unique feature of Metropolis): the Language of Constraints (LOC) and a more conventional language for logical constraint specification, LTL [43]. We also presented the Ptolemy actor-oriented semantics and showed how this is another style for abstract semantics.

**Architecture and Mapping** We then introduce the notion of *architecture* as a set of components (interconnections for communication and building blocks that are capable of carrying out computation) that are providing *services* to the behavior that we wish to implement. *Optimal* architecture choice is presented as the selection of a particular set of computational blocks in a *library* of available components, and of their interconnection. The evaluation of the quality of a selected architecture is based on a *mapping* process of behavior to the computation blocks, including programmable components. The mapping process should be accompanied by an estimate of the performance of the design once mapped onto the architecture. Communication representation is illustrated. The representation of architectures in the Metropolis and Mescal [38] environments are presented.

We emphasize the communication aspect as one of the most important in architecture development. We introduce *communication-based design* as a design paradigm where computation is abstracted away and communication becomes the main objective of the design process. We present a formal definition of the communication problem using the tagged signal model framework that explains the communication phenomena as a restriction of the behaviors of the connected processes. We show a practical example of an architecture platform like Xilinx VirtexII Pro as a heterogeneous platform that can be used for fast prototyping.

Mapping functions to architectures is possibly the most relevant aspect of the Platform based design methodology taught in this class. The power of the MMM is evident here where the use of the same modeling constructs for function and architecture allows a particularly efficient way of performing mapping and analyzing the quality of the result.

By using the MMM notion of events, we show how the function netlist can be placed in correspondence with the architecture netlist by introducing the so-called mapping netlist. In this way, events in the functional netlist trigger events in the architecture netlist via the mapping netlist. We show how the mechanism can be exploited to change the mapping of function to architecture elements in a straightforward manner that does not require re-writing of the architectural and functional description. We present the scheduling problem as an essential part of the allocation of functionality to shared resources. In this respect, we review the fundamental results of the scheduling literature.

Then, we show how mapped functions can be simulated and how the performance of the mapping can be extracted. At this point, we introduce the notion of quantity managers as tools that compute quantities such as power and time used by architectural components when executing the part of the functionality mapped onto them. Finally, we present mechanisms to feed quantity managers information about the basic execution “costs” (e.g., power and time) and we show examples drawn from Xilinx programmable platforms [49] and from other platforms such as the Intel MXP5800.

**Synthesis and Verification.** We review the notions of verification and synthesis and present how verification is not a synonym of simulation but contains static analysis tools as well as formal verification. In particular, we discuss the notion of successive refinement as the process used in the PBD methodology to go from specification to final implementation. We demonstrate the use of the MMM to keep in the same environment both the more abstract and the more concrete representation to simplify the use of refinement verification techniques. We also show the simulation approach followed in the Metropolis environment to reduce or even eliminate the overhead that comes with the flexibility of maintaining both architecture and functionality present as separate entities of the design.

Then, we focus on the methods available in literature for software estimation, an important component of any verification methodology that mixes hardware and software implementations. The approach by Malik [34] is first introduced for static analysis of programs based on pipeline and cache modeling and integer linear programming followed by the abstract interpretation work of Wilhelm that yielded the well-known analysis program AbsInt [21].

We then move to the software synthesis problem and we present the model-based design approach where code is automatically generated from mathematical models. After reviewing shortly Real Time Workshop of MathWorks, we discuss a different way of generating code from models that follows the same paradigm used in hardware synthesis of optimizing the original description (software representation) before mapping it onto a given execution platform. In this case, we show that we have a “technology independent” phase followed by technology mapping. We show how Esterel [5] is compiled using this idea and using FSM optimization techniques based on MIS [7] originally developed at Berkeley for logic synthesis to generate implementation code. We then present another method to optimize the original description based on the Ordered Binary Decision Diagram [8] representation of programs. We show how to use the variable ordering methods developed to manipulate OBDDs in logic synthesis.
to generate efficient programs from Co-Design Finite State Machines [20].

We also present evaluation techniques to compute the time taken by synthesized programs to execute on a given platform that are used to guide the optimization search. These techniques are shown to be accurate when the software is automatically synthesized. We then move to the problem of optimizing code for data flow dominated applications with limited data dependent control. We show also how operating system features such as hardware-software communication mechanisms and scheduling policies can be synthesized from requirements [2], [23] and we point to the evolution of implementation platforms that can make the optimized “compilation” problem increasingly difficult.

Finally we present two industrial tools for automatic code generation: the real-time workshop (RTW) [36] by Mathworks and Targetlink [14] by dSPACE.

The last few lectures of the course are dedicated to placing the material presented in perspective. An application such as automotive control is used to show the complete flow from modeling the functionality with hybrid systems to mapping onto execution platforms that are modified according to the results of the analysis. We had in mind to use Metropolis and x-Giotto as well as the Xilinx back-end to demonstrate a complete design flow in action, but the parallel development of the tools needed did not provide us in time with the flow as we hoped to.

Another application space that we explored is wireless sensor networks where our view of design leads naturally to the definition of layers of abstraction that identify clearly the need of a “middleware” that can capture well the performance of a particular physical implementation of the network to offer the application programmer an abstraction that enables reuse across different implementations with appropriate abstract analysis of the effects of the physical network on the application program.

After the end of the course, the projects that are used to evaluate the students are presented in front of the entire class to open a wide range discussion among students, teaching assistant, mentors and faculty on the results and the ideas on how to improve the course.

B. Projects

The course is graded on the basis of a set of homework and on a final project. After the first week of class, a list of project proposals is given to the students. We rely on a group of highly qualified mentors form industry and academia that help the students in reviewing previous work and conducting the research to complete their projects on time. We push students to start as early as possible and we motivate them by mentioning the possibility of submitting for publication for the best reports.

Following the course organization, projects are divided in the ones that are related to functional description, architectural description, mapping, case studies and design methodologies.

The choices of students are often concentrated on the definition of formal models for describing a function and on the development of algorithms for verification and synthesis. The case studies were mainly developed by students coming from departments other than EECS that are interested in studying how the methodology can be applied to solve specific design problems. Given the short available time, it is not possible in general for students to develop and show the effectiveness of a complete design methodology for a specific application domain. This is the reason why this kind of projects are rarely taken even though a couple were chosen and had worthwhile results. Projects that were developed in the past years include Time-Based Communication Refinement for CFSMs, Heterochronous dataflow domain in Ptolemy II, Hardware extension to ECL and Extending CFSMs to allow multirate operations.

Few projects were given on architecture. This situation reflects the status of our research in the field that only recently has taken an important turn determined by the extensive introduction of the use of the MMM and of the characterization work carried out in conjunction with the Xilinx project. We expect that more projects on this subject will be proposed in the future particularly in the area of syntax and semantics of languages for architectural description and automatic verification and refinement tools for architectures.

A successful project was about a complete methodology for embedded software design and system integration of a rotorcraft UAV [26] whose results were published both as a conference and as invited paper in the Proceedings of the IEEE. This project was carried out by three students and two mentors coming from different knowledge domain. In the area of wireless sensor networks, a group of students applied the methodology for studying an ad-hoc sleeping disciplines for nodes [29]. This application domain is very relevant not only for its distributed nature but also for the severe energy consumption constraints on each node.

Two other projects that had success to the point of being published in primary conferences were related to chip architectures. One was about the synthesis of on-chip communication architectures [42] and automatic hardware/software partitioning for reconfigurable platforms [4].

Projects in less obvious application domains have also been offered. In particular, two years ago a project on the design of a real-time eye detection system was assigned to two students [50].

III. THE GRADUATE PROGRAM PART 2: ADVANCED GRADUATE COURSES

Advanced courses are an important feature of the Berkeley graduate program. These courses reflect very recent advances in the state of the art of a particular knowledge domain. They are topical courses; their content changes from year to year and can be taken by students multiple times. In general, they are taken by PhD students who are interested in research topics in the area covered by the course. Regular graduate courses are often derived from the advanced series after the content has jelled and there is enough interest in the student population. Since embedded systems are so important in our research agenda, there are several advanced courses that have a
TABLE I
CLASS SYLLABUS

<table>
<thead>
<tr>
<th>Course Section</th>
<th>Lectures</th>
<th>Discussions</th>
<th>Labs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>Definition of embedded systems, examples of applications, challenges, future applications</td>
<td>&quot;The tides of EDA&quot;</td>
<td>Introduction to the Metropolis Meta-Model language</td>
</tr>
<tr>
<td>Function</td>
<td>Finite State Machines, Co-design finite state machines, Kahn process networks and data flow, Tagged Signal Model, Agent Algebras, Petri nets, Synchronous languages and desynchronization</td>
<td>StateCharts, Data flow with firings, Desynchronization</td>
<td>Introduction to PtolemyII, Building a model of computation in Metropolis, Esterel.</td>
</tr>
<tr>
<td>Architecture</td>
<td>Performances, Architecture modeling, Modeling Concurrency: Scheduling, Interconnection architectures, Reconfigurable platforms, Programmable platforms, Fault tolerant architectures.</td>
<td>Formal modeling of processors, Rate monotonic hyperbolic bound, Interface synthesis.</td>
<td>Modeling architectures in Metropolis, Xilinx, PSoC.</td>
</tr>
<tr>
<td>Mapping</td>
<td>Mapping specification, Design Exploration, Software estimation and synthesis, Static analysis, Quasi static scheduling.</td>
<td>Synthesis of software from CFSMs specification</td>
<td>Virtual Component Co-design, Mathworks RTW, dSpace Target Link</td>
</tr>
</tbody>
</table>

The schedule of the class starts with an introduction to the emerging computing platforms composed of a large number of simple nodes communicating on wireless channels. Habitat monitoring applications are taken as representative of embedded networks of nodes that can sense and communicate. The design of the embedded network is driven by the application that sets the requirements to satisfy. After the introduction, one week is dedicated to the presentation of several platforms and operating systems that are currently available.

The following four weeks of the class give an overview of all the proposed protocols for wireless sensor networks: network discovery, topology information, aggregation, broadcast and routing. The design of all these protocols takes into account the constraints imposed by the application. The class puts a lot of emphasis on power consumption since nodes cannot be replaced.

The second part of the class gives directions for the implementation and deployment of these networks. Two weeks are devoted to the problems arising from the distributed nature of the applications. In particular, distributed storage of information and distributed control are presented as important research areas. Finally, privacy is considered as a potential problem since embedded networks have the capability of monitoring every objects in the environment in which they are embedded.

Students in the class are divided in groups and each of them works on a project. Possible topics range from programming models and simulations of large scale networks to new protocols for routing and localization. In Fall 2003, a considerable number of projects investigated the problem of programming the network starting form the description of the application.

2) Mobile Computing and Wireless Networking: The level of abstraction of the networks considered in this class is much higher than the one considered in the previous section. The focus in on new trends in mobile computing and integration of heterogeneous networks [12].

The class presents challenges in mobile computing where the end user is a person that uses a device to be constantly connected to the rest of the world. The requirements on the protocol implementation are derived by looking at the

direct relationship with the topics of this paper. These courses are labeled EE290 and CS294 followed by a letter indicating the area these courses belong to.

In Figure 1, we show a the backbone of a graduate program in embedded systems that traverses the courses presented in this paper. A well thought out course program in embedded systems should include domain-specific courses that provide the reference framework for the students to be productive in the outside world.

Fig. 1. A graduate program in embedded systems

A. Computer Science Courses

The Computer Science Division of our Department offers a graduate level class on embedded network and mobile computing platforms. The course is "CS294-1". As is always the case for advanced graduate classes that belong either to the EE290 series or the CS294 series, its content changes every semester.

In the last five years, this course has always been centered around applications that are embedded in the environment with which they interact.

1) Deeply Embedded Network Systems: This advanced class focuses on ubiquitous computing [11]. The course is based on the experience of researchers form different universities on wireless sensor networks.
issues that mobility brings up: routing, network registration and security. Some protocols to solve all these problems are presented.

A network node is an hand-held device that presents severe limitations in power consumption. This problem, which is presented as an important constraints, presents some commonality with wireless sensor networks of the deeply embedded networks class but it is not the only one. Connectivity and distributed information storage are also investigated and some solutions are presented.

Finally, some common platforms for this kind of systems like WLAN, UMTS, GSM and GPRS are presented.

B. Advanced Electrical Engineering Courses

The electrical engineering division of the EECS Department offers advanced courses that are focused on formal models for system level design and embedded software. Two classes are particularly relevant to our discussion on embedded systems.

1) EE290N: Concurrent models of computation for embedded software: This advanced class focuses on concurrent models of computation for embedded software development [17]. It can be seen as the extension to and deep analysis of the first five weeks of EECS249. It has been taught four different times with contents that are converging to a unified view so that there is a plan of making it a regular graduate course. Abstract semantics, concrete semantics and implementation of some of the most commonly used models are presented. Besides homework assignments, students are required to work on a project.

The first two lectures present the main differences between embedded software and software for standard applications. In particular, threads are formally defined and their limitations are underlined with a particular emphasis on the problems that arise when using this technique for developing concurrent programs. Then, languages for particular applications, like NesC [22] and Lustre [10], are taken as representative of domain specific models for embedded software.

Then Process Networks (PN) are introduced and an abstract semantics based on PN is presented together with the notion of partial ordering and prefix ordering on sequences of events. In this abstract settings, properties like monotonicity, continuity and determinacy of a process are described as properties of the input-output function on streams that characterize a process. The fixed point semantics is also introduced when multiple processes are connected together and loops are present in the corresponding functional graph. A concrete semantics is then presented and finally the concrete implementation of PN semantics, following the Kahn-McQueen execution semantics, is shown by using PtolemyII as a platform for implementing and composing models of computation. The problem of bounded execution (an execution that used a bounded amount of memory) is introduced and simulation techniques are presented.

After Process Networks, synchronous languages are introduced together with the notion of complete partial orders and the least fixed point semantics of synchronous programs. Dataflow models are extensively discussed: statically schedulable Data Flow, multidimensional and heterochronous data flow and boolean data flow.

The last part of the class introduces continuous time models and hybrid systems. The emphasis is on the semantics and the techniques that are used to solve systems of differential equations. In particular, problems like event detection and Zeno behaviors for hybrid systems are considered and the impact on the simulation engine are explained.

2) EE290O: Embedded Software Engineering: While the previous class explores the models that should be used in developing embedded software, this class focuses on a particular design flow. The class presents a model of computation, the Giotto model [24], and explains why it is suitable for a class of embedded software [18]. The class is divided in three parts: RTOS Concepts. A real time operating system is characterized by the services that it provides: environment communication services, event triggering services, software communication services and software scheduling. Tasks and threads are defined and a model for them is explained. A simple example of an RTOS is given. In their first homework, students have to implement an RTOS on the LEGO brick. The students now have a feeling of the RTOS abstraction level and the problems in modeling software at this level. The E-machine [25] is then described and its properties are formally explained: semantics of the E-machine, portability and predictability, deterministic behavior and logical execution time.

RTOS Scheduling. Some classic scheduling algorithms are presented. First, a task is modeled with execution time and deadline and the concept of preemption is explained. Then, early deadline first and rate monotonous scheduling are explained. The last part of this section is devoted to schedulability tests like rate monotonic analysis (RTA) and model based schedulability analysis (where tasks and schedulers are modeled as timed automata).

RT Communication. The last part of the class deals with real time communication. Messages are modeled with deadline and worst case latency. Two protocols are presented in details: Controller Area Network (CAN) and Time Triggered Protocol (TTP) [28]. The problem of fault tolerance is introduced and the solution proposed by the TTP protocol is explained.

3) EE290a: Concurrent System Architectures for Applications and Implementations: This experimental class was offered in the Spring 2002. The focus of the class is on models for concurrent applications, heterogeneous platforms and the mapping of the former to the latter [16]. This course can be considered as a follow-on to EECS249 with respect to Architecture and Mapping.

The first part of the class introduces the content of the course by giving examples of applications and platforms. Several models of computation like finite state machines, process networks, data flow, synchronous/reactive, communicating sequential processes and co-design finite state machines are introduced emphasizing the fact that each model is particularly suitable for a specific application domain. The Click [47] model of computation is explained for modeling the processing of streams of packets in routers.

The first example presented in the class is MPEG decoding together with an entire flow from specification (using a flavor
of Kahn process networks called YAPI [13]) to implementation. This example shows how the requirements of an application condition the selection of a model of computation and the underlying implementation platform. Several platforms are then presented: the Nexperia [39] platform by Philips for multimedia application and the Exp1200 [27] platform by Intel for network processing. For each platform, examples of what kind of application they target are given together with performances results.

The Giotto model of computation and the E-Machine are presented and an example of the software running on an autonomous helicopter is shown. Another example show the implementation of an IP router on a VirtexII Pro FPGA and the implication of using a multi-threaded implementation. The SCORE (Stream Computations Organized for Reconfigurable Execution) [9] project is also presented. In this project both function and architecture are described using Kahn process networks and the challenge is to find a schedule of processes for bounded execution. The architecture is composed of a set of processors that communicate over a shared network. Buffers are allocated on memory segments. The last lectures give an overview of multiprocessors platforms like RAW and IWar [6] and other programmable platforms like PSOC by Cypress and the Extensa Processor.

IV. THE GRADUATE PROGRAM PART 3:
CIVIL AND MECHANICAL ENGINEERING

These two Departments have traditionally been interested in embedded applications. They have some graduate courses where embedded topics are featured.

A. Mechanical Engineering 230: “Real Time Software for Mechanical System Control”

The Mechanical Engineering Department offers a class that teaches students how to control mechanical systems using embedded software. It is a lab oriented class in the sense that students are taught how to implement a controller in Java on an embedded processor.

Even if methodology is not really the focus of this class, controlling a mechanical system implies understanding the continuous dynamics governing it and the constraints that are imposed on the reaction time of the software controller. The class gives an overview of the real time control problems. Students are exposed to Java as a technology that allow the development of real time systems and how complicated control algorithms can be implemented on an embedded processor. This part takes one third of the entire class. This technology is applied to the control of motors.

Students learn real time programming through a set of labs. The first two labs are meant to teach students how to write a control algorithm for an embedded processor. The third lab show how the software world interfaces with a physical component like a motor. In the following lab sessions, students are guided in implementing a feedback control algorithm for a motor.

B. Civil and Mechanical Engineering 290I: “Civil Systems, Control and Information Management”

The possibility of sensing the environment and communicating over a dense network of tiny objects is of much interest for the civil engineering community. This course starts with an introductory lecture that motivates the use of embedded networks with several applications: automatic control of the BART (Bay Area Rapid Transportation) system, earthquakes monitoring and mesh stable formation flight of unmanned air vehicles.

The emphasis of the class is on the formal specification of complex networked systems. The Teja [48] environment is used as example of formal language to specify systems of users and resources. Syntax and semantics of the language are explained in the class and students are trained in using the environment with labs and homework.

V. THE UNDERGRADUATE PROGRAM

Our approach to embedded systems has been to marry the physical and the computational world to teach students how to reason critically about modeling and abstractions. Traditionally undergraduate EE courses have focused on continuous time and detailed modeling of the physical phenomena using partial and ordinary differential equations, while undergraduate CS courses have focused on discrete representations and computational abstractions. We noted that students then were “boxed” in this dichotomy and had problems linking the two worlds. The intellectual agenda was to teach students how to reason about the meaning of mathematical models, their limitations and power. EECS20N (Structure and Interpretation of Signals and Systems) was born with this idea in mind.

EECS 20N together with EECS 40 (Introduction to Micro-electronic Circuits), CS 61A (The Structure and Interpretation of Computer Programs), CS 61B (Data Structures), and CS 61C (Machine Structures) are mandatory courses for any Berkeley undergraduate EECS student. In addition to having an important role in the general undergraduate education, EECS20N (see [30], [32]) is also at the root of a system science program whose structure is shown in Figure 2. In this diagram, EE149, Hybrid and Embedded Systems, is an upper-division class that is under design and will provide the ideal follow-on to EECS20N in a curriculum that focuses on the foundations of embedded systems.

A. EECS20N: Structure and interpretation of signals and systems

a) Motivation and History: EECS20N [19] is a course that electrical engineering and computer science students take in their second year at Berkeley. Traditionally, our undergraduates were exposed to a rigorous approach to systems in classes offered in the junior and senior year dealing with circuit theory, communication systems and control. The contents of the courses were thought in terms of the application domain and exposed a certain degree of duplication, as ordinary differential equations and transforms such as Laplace and Fourier were common tools. In addition, the modeling techniques used were essentially continuous time ignoring for a large part the
discrete world. From conversations among the system faculty, the idea of revamping substantially the foundations of systems and signals emerged to provide a strong basis for embedded system design.

Traditional courses such as circuit theory (and the resulting emphasis on linear time-invariant systems) were no longer deemed core courses and a unified approach to the basics of signals and systems took form as the seed for launching an undergraduate program in systems. From conversations among the system faculty, the pre-requisite structure for undergraduate program in systems is inspired by the Tamagotchi virtual pet made by Bandai in Japan. Tamagotchi pets were popular in the late 1990s, and had the barrier to abstract reasoning using extensively visualization of system behavior. The description of their approach can be found in two papers [30], [32] from which this section is taken.

b) The Program of the Course: The themes of the course are:

- The connection between imperative (computational) and declarative (mathematical) descriptions of signals and systems.
- The use of sets and functions as a universal language for declarative descriptions of signals and systems.
- State machines and frequency domain analysis as complementary tools for designing and analyzing signals and systems.
- Early and frequent discussion of applications.

State machines were the means to introduce EE students to reason about digital abstractions, frequency domain analysis to introduce CS students to reason about the continuous time world. The use of applications is essential to keep the students interested and motivated in absorbing a material that otherwise may be too arid and abstract at an early stage of engineering education.

The introduction to the course motivates forthcoming material by illustrating how signals can be modeled abstractly as functions on sets. The emphasis is on characterizing the domain and the range, not on characterizing the function itself. The startup sequence of a voiceband data modem is used as an illustration, with a supporting applet that plays the very familiar sound of the startup handshake of V32.bis modem, and examines the waveform in both the time and frequency domain. Systems are described as functions that map functions (signals) into functions (signals). Again, the focus is not on how the function is defined, but rather on what is the domain and range. Block diagrams are defined as a visual syntax for composing functions.

The connection between imperative and declarative description is fundamental to set the framework for making the intellectual connection between the labs and the lecture material. The finite state machines are introduced and analyzed. Non-determinism and equivalence in state machines is explained. Equivalence is based on the notion of simulation, so simulation relations and bisimulation are defined for both deterministic and nondeterministic machines. These are used to explain that two state machines may be equivalent even if they have a different number of states, and that one state machine may be an abstraction of another, in that it has all input/output behaviors of the other (and then some). Composition of finite state machines and the feedback loop connection is introduced. The most useful concept to help subsequent material is that feedback loops with delays are always well formed.

The last part applies the theory to real examples that depend on the interests and expertise of the instructor, but we have specifically covered vehicle automation, with emphasis on feedback control systems for automated highways. The use of discrete magnets in the road and sensors on the vehicles provides a superb illustration of the risks of aliasing.

c) The Lab: A major objective of the course is to introduce applications early, well before the students have built up enough theory to fully analyze the applications. This helps to motivate the students to learn the theory. In the Lab, we emphasize the use of software to perform operations that could not possibly be done by hand, operations on real signals such as sounds and images.

While the mathematical treatment that dominates in the lecture and textbook is declarative, the labs focus on an imperative style, where signals and systems are constructed procedurally. Matlab and Simulink [37] are chosen as the basis for these exercises because they are widely used by practitioners in the field, and because they are capable of realizing interesting systems.

There are 11 lab exercises, each designed to be completed in one week. The exercises include: Arrays and Sound, Images, State Machines, Control Systems, Difference and Differential Equations, Spectrum, Comb Filters, Modulation and Demodulation, Sampling and Aliasing. Two of the weeks are quite interesting (State Machines and Control Systems) from the point of view of embedded systems and models of computation. The third lab uses Matlab as a low-level programming language to construct state machines according to a systematic design pattern that will allow for easy composition. The theme of the lab is establishing the correspondence between pictorial representations of finite automata, mathematical functions giving the state update, and software realizations. The main project in this lab exercise is to construct a virtual pet. This problem is inspired by the Tamagotchi virtual pet made by Bandai in Japan. Tamagotchi pets were popular in the late 1990s, and had behavior considerably more complex than that described in
this exercise. The students design an open-loop controller that keeps the virtual pet alive. This illustrates that systematically constructed state machines can be easily composed.

In the Control System lab, students modify the pet so that its behavior is nondeterministic. They are asked to construct a state machine that can be composed in a feedback arrangement such that it keeps the cat alive.

**B. Discussion and future directions**

EECS20N has reached a degree of maturity that will allow it to be taught by any faculty in the Department given the large amount of teaching material available. During the first years, the course was not among the favorites of the students given its generality and mathematical rigor. However, the fine-tuning of labs and lectures and of their interrelation has had a positive effect on the overall understanding of the material and the students now express their satisfaction with this approach. Having laid out the foundation of the work, we are now considering extending the present offering in the upper division. The EE149 class in gestation will emphasize three main aspects:

- The idea of introducing signals and systems as described here with the operational and the denotational view can be traced to the research work that led to the development of the Lee-Sangiovanni-Vincentelli (LSV) tagged signal model [31], a denotational framework to compare models of computation. While the concept of time is not as abstract as in the LSV model, the course does present the denotational view of systems along similar lines. EE149 will focus on the semantics of embedded systems and introduce a consistent family of models of computation with the relative applications, strengths and weaknesses.

- Using Matlab and Simulink as a virtual prototyping method and being exposed to both the digital and the analog abstraction, the students have an early exposure to hybrid systems [35] as an important domain for embedded systems where a physical plant described in the continuous-time domain is controlled by a digital controller. We will devote a substantial portion of the course to the discussion of the properties of hybrid models as paradigms for the future of large scale system monitoring and control.

- A substantial problem is the way in which Simulink combines discrete and continuous-time models. Simulink essentially embeds the discrete in the continuous domain, i.e., the numerical integration techniques determine the time advancement for both continuous and discrete models. Thus de facto Simulink implements a single model of computation: the synchronous reactive model where logical time is determined by the integration algorithm. This may make Simulink non ideal for teaching embedded systems where heterogeneous models of computation play a fundamental role. We will add to the Matlab/simulink environment, Ptolemy II as a design capture and verification tool to obviate this problem.

- Implementation platforms are important as they determine the real-time properties of the system as well as its cost, weight, size and power consumption. The course will present methods to capture implementation platforms and to map functionality to platforms.

- Applications will be emphasized as drivers and as test cases for the methods presented in the class. In particular, during the course, students will be asked to develop completely an embedded system in a particular application domain using the methods taught. This design work will be carried out in teams whose participants will have enough diversity as to cover all aspects of embedded system design.

Since many of these topics are covered in EECS249, the graduate class, we will borrow heavily from that experience while the material of EECS249 will evolve towards more advanced topics.

**VI. Conclusions**

We outlined the embedded system education program at the University of California at Berkeley. We stressed the importance of foundations in education as opposed to technicalities. Embedded systems are important enough to warrant a careful analysis of the mathematical bases upon which we can build a solid discipline that marries rigour with practical relevance. Given the present role of embedded systems in our research agenda and the traditional approach to education in the leading US Universities, the first courses to be developed are advanced graduate courses. The natural evolution is to solidify the teaching material to a point where regular graduate classes can be taught and finally move the contents to the undergraduate curriculum while the graduate courses adjust continuously to the advances in the field brought about by research. The flexibility of the US system allows to change fairly easily the curriculum and to strive for relevance to the ever changing society and cultural landscape.

While we believe that our program has achieved a set of important goals, we do realize that much remains to do. At the undergraduate level, we are planning to introduce an upper division class on hybrid and embedded software systems. At the graduate level, we are considering the addition of regular courses on theoretical foundations focusing on function description and manipulation as well as one on reconfigurable and programmable architectural platforms to follow the basic graduate course (EECS249) on embedded systems. We also believe that embedded system courses should be considered foundational courses for the entire college of engineering and we are working with our Dean and Department Chairpersons to address this issue.

The views presented here are for a large part shared with the Artist [1] Network of Excellence Education Team whose agenda is described in another paper of this special issue.

**VII. Acknowledgements**

We wish to acknowledge the long-time collaboration with Edward Lee, Tom Henzinger, Richard Newton, Jan Rabaey, Shankar Sastry and Pravin Varaija in the research and teaching agenda on embedded systems at Berkeley. The help and support of the Metropolis group is gratefully acknowledged.
Important impacts on our approach come from interactions with Albert Benveniste, Gerard Berry, Paul Caspi, Hugo DeMan, Luciano Lavagno, Joseph Sifakis and the ARTIST European Network of Excellence team, Alberto Ferrari and his colleagues of PARADES, and last but not least our industrial associates too numerous to be mentioned. Funding for this work came from CHESS NSF ITR, and the GSRC. Work (partially) done in the framework of the HYCON Network of Excellence, contract number FP6-IST-511368.

REFERENCES

[16] EE290A, “http://www-cad.eecs.berkeley.edu/respep/research/classes/ee290a/fall02/.”
[38] PATH, “http://www.path.berkeley.edu/”.
Demand-Driven Curriculum for Embedded System Software in Korea

Suehee Pak  
Dongduk Women's University  
23-1 Hawolgok-dong, Sungbuk-gu  
Seoul 136-714, Korea

Eunha Rho  
Sungkonghoe University  
1-1 Hang-dong, Kuro-gu  
Seoul 152-716, Korea

Juno Chang  
Sangmyung University  
7 Hongji-dong, Jongno-gu  
Seoul 110-743, Korea

Moon Hae Kim  
Konkuk University  
1 Hwayang-dong, Gwangjin-gu  
Seoul 143-701, Korea

mhkim@konkuk.ac.kr

ABSTRACT

This paper describes how Korean Government has designed and driven the innovation of undergraduate curriculums to meet increasing industrial demand for quality IT experts in the computer-software field. The computer-software field has been categorized into five specialized areas, tracks. The embedded system software area is one among them. Educational contents (including track curriculums, detailed syllabuses, and class materials) have been designed and constructed for the tracks. First, a curriculum development methodology is described, which is used to develop curriculums in the computer-software field. Next, artifacts for the embedded software system track produced in the process are described. By applying iterative and incremental principles of the methodology, all artifacts will be continuously updated and constantly upgraded as related technologies progress rapidly.

Keywords  
track, curriculum, industry demand, demand-driven course, detailed syllabus, class material, course development

1. INTRODUCTION

Despite the excessive supply of IT manpower in the market, many companies in Korea still find difficulty in recruiting manpower of quality satisfying their needs. This is so called ‘skill mismatch’ phenomenon, where maturity levels of skills of graduates from academic institutions are lagging far behind the required level in the spot industrial sectors. Therefore, the cooperation between universities and IT industry in producing IT manpower of quality is urgently called for to create the effective labor pool of supply and finally balance its supply and demand.

To facilitate the cooperation, Korean Government has been focusing on the training program of demand-driven IT workers as one of the major policy measures by applying the SCM (Supply Chain Management) model to human resource development [2].

The basic concept of SCM is to optimize whole supply networks to maximize total profit. Applying this concept to human resource development denotes that we have recognized a series of educational process and career path as a huge supply network so that optimization would be deployed into all of human resource development processes.

To reflect intensive and extensive technological changes and industrial needs from academia, a study is necessary to solve and act as an antidote against the imbalance problem between the skill factors demanded by industry and the curriculums of universities. As a sub-project for the SCM project, Korean Government launched a challenging project where curriculums and courses are designed and constructed, based on their own estimated demand for IT manpower.

According to IT industry, to solve the skill mismatch, the computer-software field needs to be subdivided into more specialized areas. For each specialized area, track in the project, educational contents are to be properly constructed and customized to the corresponding IT sector. The practical purpose of this project lies in producing students as IT experts equipped with specialized skills. As the first stage to innovate academic curriculum, computer-software field is categorized into five specialized tracks.

With five tracks to pursue as majors in computer-software field, four demand-driven courses for each track are designed and their detailed syllabuses and course materials are constructed.

In the following section, a methodology is suggested to develop demand-driven curriculums. In the next sections, the major artifacts of demand-driven curriculum...
development for the embedded system software track are described, which is followed by concluding remarks.

2. METHODOLOGY FOR CURRICULUM DEVELOPMENT

2.1 Curriculum Development as a Process of Change and Evolvement

Academic curriculums are to change and evolve over a period of time. Curriculum development as a process of change and evolvement is somewhat similar to software development.

Spiral model, originally proposed by Boehm [1], is an evolutionary software process model that combines the iterative and incremental nature of prototyping with the structural aspects of the linear sequential model.

Applying concepts and basic principles of the spiral model, Curriculum Spiral model for demand-driven curriculum development is derived and proposed. In the process, a curriculum is developed in a series of incremental releases. That is, a whole development process is divided into several, time-boxed small-scaled projects. This small-scaled project is called iteration stage and an iteration stage is again subdivided into four activities, called phases.

Figure 1 depicts the curriculum spiral model that performs 3 iteration stages, each of which contains four phases, and shows artifacts in each phase. In the rest of this section, each phase is described in terms of its activities and artifacts.

In the first phase in an iteration stage, Requirement Analysis phase, tracks are identified and selected by analyzing industrial demands and requirements, and also by collecting feedbacks from universities involved in the previous iteration stage. Partitioning tracks is to vertically decompose a certain generalized field. A track should be a highly cohesive subfield of computer-software field.

In the Design phase, a curriculum is designed and developed for each track. Several industrial demand-driven courses are included in each curriculum. A track curriculum includes track goals, a course tree, and preliminary design for demand-driven courses. A course tree includes demand-driven courses, prerequisite courses, basic computer science courses, and their associations.

In the Implementation phase, a detailed syllabus and class materials are constructed for each demand-driven course.

In the Realization phase, semester classes are lectured, based on the track curriculum, detailed syllabus, and class materials. As input to the requirement analysis phase of the next iteration stage, all feedbacks from the involved universities and industries are collected.

2.2 Human Infrastructure

The human infrastructure for curriculum development is classified in Figure 2.

As a coordinator, Comprehensive Manager coordinates all phases of a small-scaled project, that is, an iteration stage. Comprehensive Manager plans overall process, collects ideas, suggestions, and information from the rest of the participants and finalizes track decision, curriculum design and course materials. Comprehensive Manager identifies and organizes human infrastructure, assigns their roles and coordinates their communications and finally resolves their conflicts.

As a basic participant, Course Developer develops all artifacts from phases. Two or more Course Developers are assigned to each and every track. Course Developer constructs track curriculum and makes class materials.

Track Expert, another participant, consists of Academic Track Expert and Industrial Track Expert for each track. Track Expert is involved in the process of constructing artifacts. Course Developer works in close cooperation with Track Expert.

Consultant, as one of the stakeholders, consists of General Consultant and Track Consultant. Track Consultant consists of Academic Track Consultant and Industrial Track Consultant. Consultant reviews and monitors all artifacts constructed by Course Developer. Lecturer, another stakeholder, lectures using artifacts constructed by Course Developer.
3. TRACK: VERTICAL DECOMPOSITION OF COMPUTER-SOFTWARE FIELD

3.1 Necessity for Specialized Field Tracks

Applying industrial demands into academia, it should be considered what educational unit would be a suitable one for bearing the task. From the educational perspective, decision for the choice of an appropriate educational unit is one of the most important underlying factors to be considered. An undergraduate curriculum, especially for science and engineering majors, should be the proper combination of mathematics, understanding of principles, practices, application, and skills for tools [3].

Demand-driven curriculum needs to be balanced with five factors mentioned above, and categorized for more specialized area by possible career pursuits of students.

The concept of track is introduced, which has a demand-driven curriculum for a specialized area while satisfying the guideline of ABET, ACM, and IEEE/CS. ABET has ‘program’ as a unit of accreditation. ABET guideline indicates that each program should have its own objective, so track would have the objective of ‘satisfying demands from specific IT area’. In this paper, a track can be considered to be a proper educational unit (for undergraduate) composed of major issues reflecting industrial needs.

3.2 Five Tracks and Ongoing Changes

Selecting tracks is based on the work in [5], which suggested several specialized tracks in the field. In order to identify tracks, needs from IT industry experts are carefully examined. Requirements of universities are also considered. As a result, the computer-software field is divided into five specialized tracks as follows:

- Software Development (SD)
- System Integration (SI)
- Embedded System Software (EM)
- Multimedia and Game Software (MM)
- Business Information Technology (BI)

In the software development (SD) track, the goal is to educate skilled manpower with expertise in software development. It is recommended not to deal with fragmentary skills and detailed application skills, considering the limited time horizon of 4-year university education. Instead, the curriculum focuses on training the ability of problem solving and that of self-adapting to rapid changes in the practical IT field.

The goal of the system integration (SI) track is to train manpower with practical knowledge required in system development companies which make general business applications. In order to produce students who understand the whole process of system development, the curriculum is designed to focus on up-to-date technologies such as software modeling, software development process, and web service computing.

In the embedded system software (EM) track, students are trained to have practical knowledge required in system development companies which make general business applications. In order to produce students who understand the whole process of system development, the curriculum is designed to focus on up-to-date technologies such as software modeling, software development process, and web service computing.

The goal of the multimedia and game software (MM) track is to educate both multimedia software and game software developers. Students are trained to have basic abilities to have considerably high productivity in a short period of time after graduation. Multimedia software courses focus on understanding of the overall multimedia area, including moving picture processing and animation, and achieving the ability of integrating and applying various digital media. Game software courses deal with the concept and technology of computer game production, and train students to have the ability to collaborate in practical game production.

In the business information technology (BI) track, the goal is to train IT consultants and IT business managers who have the ability to apply and manage rapidly changing information technologies of management. The curriculum basically intends for students of computer-software departments. It integrates courses for overall knowledge of management and courses for information technologies, such as computers and communication.

Some of the tracks mentioned above are closely related, and some of them cover quite big areas. The number and
goals of tracks will be continuously updated and constantly upgraded as IT technologies progress rapidly. Tracks can be added, deleted or even merged. For example, information security track can be newly added on the track list.

3.3 Demand Survey of the Five Tracks
Surveys on industrial preferences for the five tracks have been conducted [4]. About 2,000 Korean companies are surveyed (about 1,000 IT companies and about 1,000 non-IT companies).

The result shows demand preferences for human resources in each track. Out of five tracks, SD and SI stand out in demand pressure from the industry. Table 1 shows preferences for the tracks in each IT profession. Table 2 shows preferences of the tracks in each technology.

<table>
<thead>
<tr>
<th>Table 1. Track preference in each profession (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Track</strong></td>
</tr>
<tr>
<td>IT consultant</td>
</tr>
<tr>
<td>Developer</td>
</tr>
<tr>
<td>System engineer</td>
</tr>
<tr>
<td>IT design / administrator</td>
</tr>
<tr>
<td>Computer information security engineer</td>
</tr>
<tr>
<td>Network design / administrator</td>
</tr>
<tr>
<td>Web engineer / development construction</td>
</tr>
<tr>
<td>Server / S/W developer and manager</td>
</tr>
<tr>
<td>System / S/W developer and manager</td>
</tr>
<tr>
<td>Web planner and designer</td>
</tr>
<tr>
<td>Game / animation / graphic designer</td>
</tr>
<tr>
<td>Visual reality / animation / graphic designer</td>
</tr>
<tr>
<td>System operation and administration</td>
</tr>
<tr>
<td>System administrator</td>
</tr>
<tr>
<td>Computer technology support / technician</td>
</tr>
<tr>
<td>Education</td>
</tr>
<tr>
<td>Business</td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>

The first group consists of demand-driven courses, the skills of which are analyzed as demanded by IT industry.

The second group consists of direct prerequisites of demand-driven courses.

The third group consists of common fundamental courses, which are classical computer science courses. These courses are included in all track curriculums in common.

The basic notation to represent designed courses and their associations are shown in Figure 3. Course tree for the embedded system software track is shown in Figure 4.

4.1 Track Curriculum Outline
Track curriculums are designed and constructed, with the main focus on the specialized skills demanded by IT industry. A track curriculum includes a course tree. Courses in the tree are grouped into three types.
Table 3. Specialized skills of demand-driven courses for the embedded system software track

<table>
<thead>
<tr>
<th>Demand-Driven Courses</th>
<th>Specialized Skills</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedded System Software I</td>
<td>Startup code Development, Memory/Clock/IO Control, Linux System Programming, Device Driver Development, and Assembly Language Programming</td>
</tr>
<tr>
<td>Embedded System Software II</td>
<td>RTOS concept, RTOS based Programming, and DMA/Timer/Interrupt Programming</td>
</tr>
<tr>
<td>Embedded Application Software</td>
<td>Embedded Network Programming, TCP/IP, Client-server Programming, and GUI programming</td>
</tr>
</tbody>
</table>

5. CONCLUSION AND FUTURE WORKS

IT education in academia has not properly accommodated the changing industrial demands. This paper presents research results to solve the imbalance between the skill factors demanded by industry and the curriculum of universities.

As a curriculum development methodology, Curriculum Spiral model has been suggested and applied.

As the first artifact of this work, five specialized areas in the computer-software field, called tracks, have been identified to reflect the demand from IT industry. The goal for each track has been established along with the demand-driven courses, which are major artifacts of the curriculum development process. Detailed syllabus and class materials have been designed and constructed as well.

Using the artifacts from this work, the governmental SCM project started in the beginning of 2004. Korean Government selected 39 universities (19 for the embedded system software track) from all over the country. Each university in the program selected one track and adopted the suggested track curriculum. At the point of yielding students equipped with specialized skills and expertise, meaningful feedbacks will be obtained to improve the curriculums. The feedbacks are expected from the parties involved.

As the Curriculum Spiral model explicitly suggests, curriculum development procedures will be iteratively and incrementally performed in order to promote the curriculum. First of all, changes are expected and inevitable in tracks. Proper classification and restructuring of tracks in computer-software field will lay the groundwork for developing a full-fledged demand-driven curriculum.

Secondly, track curriculums, detailed syllabuses, and class materials need to be maintained and elaborated according to feedbacks stemming from the universities and IT industry.

Finally, demand-driven courses need to be continuously updated and constantly upgraded as IT technologies progress rapidly.

Acknowledgements: Thanks to Korean Ministry of Information and Communication, and Institute of Information Technology Assessment for the support in development of demand-driven curriculums in the computer-software field.

6. REFERENCES


Abstract—In October 2004, Nagoya University has started an extension program on embedded software, called NEXCESS (Nagoya university EXtension Courses for Embedded Software Specialists). NEXCESS aims at education of engineers in industry, and is financially supported by the government. This paper describes the organization and course design of NEXCESS. Also, our experience of the first half year is reported.

Index Terms—Education, Embedded, Software

I. INTRODUCTION

Due to the ever increasing demand for the development of embedded software, it is an urgent issue to increase the number of embedded software engineers as well as to improve their skills. Since embedded software is usually developed under very limited hardware resources, specialized skills are required for the embedded software engineers. The complexity of embedded software has been growing, while the time-to-market pressure has been strengthened. Needless to say, the reliability or quality must not be compromised.

In Japan and many other countries, however, few universities provide sufficient classes on embedded software development. Students graduate from their universities without acquiring enough skills on embedded software even if they major in computer science or electronic engineering. In most companies, on the other hand, OJT (on-the-job training) is only the method for training unskilled freshmen. The OJT is seldom systematic, so they can hardly acquire wide knowledge.

In October 2004, Nagoya University has started an extension program on embedded software, called NEXCESS (Nagoya university EXtension Courses for Embedded Software Specialists) [1]. NEXCESS is targeted towards embedded software engineers in industry. We have designed eight courses on embedded software and developed teaching materials for the courses.

II. ORGANIZATION

The organization of NEXCESS is depicted in Fig. 1. NEXCESS is supported by Ministry of Education, Culture, Science and Technology, Japan. Masaki Yamamoto is with Information Technology Center, Nagoya University, Japan (e-mail: yamamoto@itc.nagoya-u.ac.jp). Hiroyuki Tomiyama is with Graduate School of Information Science, Nagoya University, Japan (e-mail: tomiyama@is.nagoya-u.ac.jp). Hiroaki Takada is with Graduate School of Information Science, Nagoya University, Japan (e-mail: hiro@ertl.jp). Kiyoshi Agusa is with Graduate School of Information Science, Nagoya University, Japan (e-mail: agusa@is.nagoya-u.ac.jp). Kenji Mase is with Information Technology Center, Nagoya University, Japan (e-mail: mase@nagoya-u.jp). Nobuo Kawaguchi is with Information Technology Center, Nagoya University, Japan (e-mail: kawaguchi@itc.nagoya-u.ac.jp). Shinya Honda is with Information Technology Center, Nagoya University, Japan (e-mail: honda@ertl.jp). Nobuyuki Kaneko is with Information Technology Center, Nagoya University, Japan (e-mail: nkaneko@agusa.i.is.nagoya-u.ac.jp).

This paper presents an outline of NEXCESS and also reports our experiences from October 2004 to March 2005.

The rest of this paper is organized as follows. Section 2 described the organization of NEXCESS. Section 3 presents how we have designed the courses. Section 4 reports our experience of the first six months.
Specifically, NPO TOPPERS Project [2] and NPO SESSAME [3] are primary contributors to NEXCESS. NEXCESS is financially supported by the Japanese government. An approximate total of 300 million yen is granted for five years from 2004 to 2009.

III. Designing Educational Courses

A. Classification of Technical Skills and Careers

When we designed extension courses, we first classified technical skills as follows.

(1) Introductory Level
An engineer at this level can accomplish his/her duties under directions by his/her superior. Typically, engineers with an experience of less than five years fall into this level.

(2) Intermediate Level
An engineer at this level can promote his/her projects by his/her own judgment. Typically, engineers with an experience of five to ten years fall into this level.

(3) Advanced Level
An engineer at this level can lead his/her company. Typically, engineers with an experience of more than ten years fall into this level.

Here, the number of years of experience is not an absolute criterion but just a rough one. It may vary depending on companies and individuals.

We also classified career paths for engineers as follows.

(1) Administrative Managers
Some engineers will become administrative manager who promote their company’s business through managing their teams.

(2) Technical Specialists
Some other engineers will become technical specialists who promote their company’s business through developing leading technology.

An advanced-level engineer understands his/her ability as an administrative manager and as a technical specialist. Thus, we assume that the career path is decided at the advanced level. Fig. 2 illustrates the relationship between the levels of technical skill and the career paths.

B. Designing Overall Courses

In many companies, an engineer at introductory level works for the programming and testing process. We planned the introductory engineer must study structure programming, embedded C programming and testing. These skill are useful in the programming and testing process.

In many company, an engineer at intermediate level works for design and requirement process with his subordinate. We planned the intermediate engineer must study the structure design, requirements analysis and project management.

And in many company, an engineer at advanced level works for the special technical field. We planned the advanced engineer must study real-time OS, C-based hardware design and other special technologies.

In 2004, eight courses have been developed. The courses are classified based on the levels of technical skills as follows.

(1) Introductory class (one course):
“Fundamentals of the embedded software development technology”

(2) Intermediate class (two courses)
01: “Design methodology and management of embedded software”
02: “Software design technology with a real-time OS”

(3) Advanced class (five courses)
01: “Internal structure of a real-time OS”
02: “C-based embedded hardware design”
03: “System control middleware and application”
04: “Software engineering for embedded systems”
05: “Ubiquitous interface and embedded software programming for image processing”

As mentioned in Section II, five professors and three researchers at Nagoya University are involved in NEXCESS. Each of the professors is responsible to one advanced course, while three researchers are primarily in charge of introductory and intermediate courses. Additionally, external lecturers from other universities, NPOs and companies are invited whenever necessary.

Terms of the courses are designed to be short so that engineers in industry can attend the NEXCESS courses without largely sacrificing their job. Introductory and intermediate courses are four days long, advanced-01 is three days, and the other advanced courses are two days.

The eight courses are independent of each other. A student does not have to attend all the courses. One can take one or more courses which he/she needs.

C. Designing Individual Courses

It is widely recognized that project-style education is very effective. However, it generally takes a long time (typically more than two weeks) and it is not easy to complete a realistic project within a few days. However, more than two weeks are
textbooks are based on presentation slides. Each textbook
LED, how to build program and how to write flash ROM.
microprocessor timer, how to get a switch, how to turn on a
student can learn how to initial a microprocessor, how to use
TMER program is only 500 lines C programming. And the
TMER start and second SW is for finish time length. The
time. This TMER has two SWs and two LEDs. First SW is for
making TMER. The TMER noti ces of termination at preset
discussion-based requirement analysis and code review.

For example at the introductory course the small workshop is
making TMER. The TMER notices of termination at preset
time. This TMER has two SWs and two LEDs. First SW is for
TMER start and second SW is for finish time length. The
TMER program is only 500 lines C programming. And the
student can learn how to initial a microprocessor, how to use
microprocessor timer, how to get a switch, how to turn on a
LED, how to build program and how to write flash ROM.

We have also developed textbooks for the courses. The
textbooks are based on presentation slides. Each textbook
consists of 196 to 652 pages.

IV. EXPERIENCE AND ANALYSIS

As mentioned above, NEXCESS is a five-year project
(exactly, four years and a half). This section describes and
analyzes our experience of the first-half-year.

A. Practice

We held nine courses from November 2004 to March 2005.
We held the introductory course two times. Each of the
intermediate and advanced classes was held once. The courses
were carried out weekdays from 9:30 to 17:00. Each day was
composed of four time periods each of which is 90 minutes
long.

The capacity of the introductory and intermediate classes is
30 persons, while that of advanced classes is 20.

Applications to the NEXCESS courses were received
through the Web site. We have received a total of 369
applications.

Since the applications largely exceeded our capacity, we
have selected 227 persons. Out of them, 13 persons cancelled,
so the number of persons who have completed their course was
214 in total.

For a project with microcontroller boards, we have prepared
one board for each learner. In addition to lecturers, four
teaching assistants (TAs) assisted the projects on the average.
All the TAs are graduate students in Nagoya University.

Through the courses, we requested the learners to answer a
detailed questionnaire via the Web site. We found that a
web-based questionnaire is more effective than a paper-based
one. A large number of detailed comments were given by the
learners.

All the courses were recorded with a digital video camera
which was synchronized with the presentation slides. After the
courses, we have created e-learning contents so that the
learners can review them at home or office through the Internet.
The e-learning contents were provided to the learners for two
months after the course.

B. The Number of Applicants

The number of applicants for each course is shown in Fig. 4.
The average numbers of applicants for the introductory,
intermediate and advanced courses were 67, 41.5 and 30.4,
respectively. Thus, applicants tend to decrease as the technical
level rises. This tendency is very natural since the scope of the
course inevitably narrow as its technical level arises.

Applicants for the second round of the introductory course
were largely decreased. This is mainly because the interval
between the two practices of the introductory course was close.

The number of applicants for advanced-01 was much larger
than that for the other advanced courses. As described in
Section 3, advanced-01 is a course on RTOS internals. Thus, it
is revealed that skilled embedded software engineers have a
strong interest in RTOSs.

C. Ages of Applicants

The average ages of applicants for the introductory,
intermediate and advanced courses were 28.5, 32 and 35.2,
respectively. If we assume that they join their company at 24
years old, the numbers of years of experience are 4.5, 8 and
11.2. These numbers are very close to our assumptions shown in
Section 3.1.
D. Types of Companies

Types of the companies to which applicants belong are summarized in Fig. 5. Consumer and industrial electronics manufactures account for about 60%. Specifically, there were a large number of applications to the introductory and intermediate courses from consumer electronics companies.

61.9% of applications were from the Tokai area where Nagoya University is located. 19% were from the Kansai area (including Osaka, Kobe, Kyoto and so on), and 15.2% were from the Kanto area (including Tokyo, Yokohama, Kawasaki and so on). We received some applications from the Kyushu (Fukuoka, etc.) and Tohoku (Sendai, etc.) areas.

E. Analysis of e-Learning Review

After the courses, we provided e-learning contents so that learners could review them at home and office. About 57% of the introductory course learners have accessed the e-learning contexts, while those of intermediate and advanced courses were 32% and 27%, respectively.

F. Evaluation by Learners

We requested the learners to rate the courses from 5 (highest) to 1 (lowest) in terms of interest, usefulness, easiness to understand and overall evaluation. The results are summarized in Fig. 6. The average score of overall evaluation is 4.4, which we think is a very high score.

G. Easiness to Understand

The score of easiness to understand is generally lower than that of interest. Especially, the difference between easiness and interest is more than 0.6 points at the second round of the introductory course, intermediate-02, advanced-03 and advanced-05. We have in depth analyzed comments by those who gave a low score to easiness. The analysis reveals that the low rating is mainly due to practices of programming. On the other hand, easiness of advanced-02 which also has a programming practice is 4.3, which is not low. Some learners of advanced-02 wrote that the programming was easy, so the practice needs to be increased.

If one feels difficulty in programming, he/she tends to give a low score to easiness. In general, giving a low score to easiness is not always negative because one can realize what he/she does not understand.

H. Usefulness

Fig. 6 shows that the score of usefulness is located between interest and easiness, and is a strong relationship with the overall score.

For advanced-03, 04 and 05, the score of usefulness is more than 0.5 point lower than that of interest. This is mainly because most learners were very conscious of how to apply what they learned to their daily work, and they felt that some part of the advanced courses are not directly applicable.

On the other hand, the contents of the introductory and intermediate courses are fundamental, so the score of usefulness is generally high.

I. How to Get to Know NEXCESS

44% of applicants were directed by their superiors. It should be noted that the percentage is up to 63% for the introductory courses. Since the courses were held weekdays, understanding and permission by their superior is necessary. However, 9% of applicants confessed that they did not obtain permission by their superior.

J. Reasons for Cancellation and Absence

Since the number of applications exceeded the capacity, we have selected attendees for most of the courses. However, 17 persons (five for intermediate courses and twelve for advanced courses) were cancelled or absent.

We have asked the reason to them, and received responses from seven persons. Four answered that it was because of some urgent work. Three out of them judged it by themselves, and one was directed by his superior or customer.

V. Summary

Education of engineers in industry as well as ordinary students is one of important responsibility of universities. Nagoya University has started an extension program on embedded software, named NEXCESS, with financial support by the government. From October 2004 to March 2005, we have held in total nine courses. The evaluations by the attendees were generally high.

NEXCESS is now in the second year. We are continuing to improve our courses. Especially, this year, we plan to open a new course for educating future educators so that they can educate young engineers in their companies.
ACKNOWLEDGMENT

We would like to thank NPO SESSAME, NPO TOPPERS Project, Professor Atusi Onisi (Ritumeikan University), Professor Masamitu Noro (Nanzan University), Professor Atusi Sawada (Kyoto University), Professor Atusi Yoshida (Wakayama University), ATR Media Information Science Laboratories, Fujitsu Prime Software Technology and Soliton Systems, Inc., for their supports.

REFERENCES

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 III. In section IV, we will discuss our experience with the course structure. Section V will provide a conclusion.

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).

![Design information flow](image)

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;

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\textsuperscript{\textregistered} 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 textbook 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

Experience with an Embedded Systems Software Course

Jogesh K. Muppala, Senior Member, IEEE

Abstract—In this paper we share our experience with designing and offering a senior undergraduate course on Embedded Systems Software in the Department of Computer Science at the Hong Kong University of Science and Technology. We give a detailed overview of the course, some reflections on our experience with the first offering of the course, followed by some discussion on the students’ views of the course.

I. INTRODUCTION

Embedded systems have been experiencing tremendous growth and deployment, especially with the availability of ever-expanding number of intelligent and complex devices with diverse capabilities. The added complexity in turn emphasizes the need for formal approaches in design and development of these devices. While embedded systems design draws upon knowledge from several traditional areas of study such as system specification, modeling and analysis; computer architecture and micro-architecture; as well as compilers and operating systems, it adds its own twists and new constraints on these areas that need to be addressed afresh [9]. Several universities have started offering courses on embedded system design. See for example the list maintained by Vahid [11]. While most of the courses concentrate on the hardware aspects of embedded systems, there is still paucity of courses dedicated especially to software aspects of embedded systems.

In this paper, we describe our experience in designing and offering a course specifically concentrating on the software requirements and design for embedded systems. We describe the details of the course including the list of topics, the hands-on laboratory exercises and some reflections on the experience with the first offering of the course. We also give a brief account of the students’ perception and opinions on the course. A related course on Embedded System Design is offered in the Department of Electrical Engineering. This latter course concentrated more on the hardware aspects of embedded systems including interfacing, system on a chip design and field programmable gate array approach to implementing custom embedded systems.

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

II. BACKGROUND

Most early embedded systems were implemented using custom-built hardware and software. However, with the growing complexity of embedded systems, system designers are increasingly resorting to the use of a commercial off the shelf embedded hardware, coupled with a real-time operating system (RTOS) kernel to provide a user-friendly API for software design, development and deployment. Thus traditional computer science areas such as compilers, operating systems, software engineering, modeling and simulation, and formal languages like UML are being increasingly used in the embedded system design and deployment process.

Many universities are beginning to offer courses on embedded systems [11]. Most such courses are hardware oriented, covering topics such as System on Chip (SoC) design, FPGA, and VLSI aspects of hardware design. Similarly most textbooks devoted to Embedded Systems tend to concentrated mostly on the hardware aspects of embedded systems. The increasing software component of embedded systems nowadays has generated interest in formal study of embedded software. We describe our experience in designing and offering one such course specifically designed to address all the aspects of embedded software.

Some universities typically offer courses on real-time systems where some of the topics that we include in our course are covered. Typically topics in real-time OS like scheduling are covered in such courses. Similarly many universities have courses on software engineering, but not necessarily with emphasis on embedded systems.

Many of the advanced topics are usually offered in graduate courses. However, these focus on current research topics in the area and tend to be more theoretical in nature. On the other hand, our course is targeted at undergraduate students. Hence the emphasis is more on practical topics.

III. THE COURSE

In this section we give the details of our course, including
the course structure and list of course topics, the hands-on laboratory exercises, and textbooks. We also reflect on our experience with the first offering of the course.

A. Course Topics and Structure

The course was designed with particular emphasis on the software aspects of embedded systems. The list of topics covered in the course is given in Table 1. The students were first given a comprehensive overview of embedded systems, their characteristics and design constraints. Three major areas that were emphasized in the course were: (i) embedded software development, (ii) real-time and embedded operating systems (RTOS), and (iii) embedded software engineering.

In embedded software development, 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 OS concentrated on reviewing the basics of operating systems, and also introducing the specific constraints imposed by real-time requirements. 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. Three full-fledged real-time OS viz., Windows CE, Embedded Linux and VxWorks were introduced. Three other OS, specifically targeted at the mobile environment, including Java 2 Micro Edition (J2ME), Symbian OS and Palm OS were covered. Although J2ME is not a true OS, but rather a portable virtual machine, it was given specific coverage because of its widespread usage.

Embedded software engineering was the most diffuse area to cover. 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.

Designing an undergraduate course on embedded systems software poses several interesting challenges. The course must be designed with a suitable balance of both theoretical and practical issues. The theoretical coverage gives the necessary foundation for the students in the area, while the practical training is essential to impart them some useful real-world skills and aid in retaining their interest in the course.

Table 1: List of Course Topics

<table>
<thead>
<tr>
<th>1. Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Introduction to Embedded Systems</td>
</tr>
<tr>
<td>• Examples of Embedded Systems</td>
</tr>
<tr>
<td>• Embedded System Characteristics</td>
</tr>
<tr>
<td>2. Embedded Systems Architecture</td>
</tr>
<tr>
<td>• Hardware Fundamentals: Processors, Memory, Bus, etc.</td>
</tr>
<tr>
<td>• Software: OS, Application Software</td>
</tr>
<tr>
<td>3. Embedded Software Development</td>
</tr>
<tr>
<td>• Hosts and Targets</td>
</tr>
<tr>
<td>4. Interrupts</td>
</tr>
<tr>
<td>• Introduction to Interrupts</td>
</tr>
<tr>
<td>• Interrupt Handlers and Interrupt Service Routines</td>
</tr>
<tr>
<td>5. Embedded Software Architectures</td>
</tr>
<tr>
<td>6. Real-Time Operating Systems (RTOS)</td>
</tr>
<tr>
<td>• Review of Operating Systems Basics</td>
</tr>
<tr>
<td>o Tasks, Processes and Threads</td>
</tr>
<tr>
<td>o Task Scheduling: Rate Monotonic Scheduling, Priority Inversion</td>
</tr>
<tr>
<td>o Task Synchronization and Coordination</td>
</tr>
<tr>
<td>o Intertask Communication</td>
</tr>
<tr>
<td>o Memory Management</td>
</tr>
<tr>
<td>7. Embedded Software Engineering</td>
</tr>
<tr>
<td>• Basics of Software Engineering</td>
</tr>
<tr>
<td>• Software Engineering Models</td>
</tr>
<tr>
<td>• Unified Modeling Language (UML)</td>
</tr>
<tr>
<td>• Software Testing</td>
</tr>
<tr>
<td>8. Testing and Debugging Embedded Systems</td>
</tr>
</tbody>
</table>

B. Textbooks and References

Embedded systems software development, as a field, is fairly recent with not a significant amount of formal techniques and approaches described in the literature. Hence we found that there is paucity of suitable books covering the complete set of topics that were chosen to be covered in our course. Thus we had to resort to using several different sources for gathering the material to be covered in the course.

Most formal textbooks available in the market are oriented towards the hardware aspects of embedded systems. See for example [12]. This book does give some software aspects, but mostly from the hardware perspective. Wolf [6] is another book that gives a more balanced hardware/software view, but still hardware oriented.

Simon [1] was one of the first books to concentrate more on
software aspects of embedded systems. It introduces the field with a running example, and some concepts are well explained. Kamal [2] is a recent book in this area which tries to give a balanced overview of both hardware and software. Lewis [7] also concentrates more on the software aspects, especially for a beginner in the area with limited programming experience. The treatment of the language aspects and memory management are very good.

Two books that introduce the area with several interesting working projects in different areas of embedded software development are [3][5]. These books are very useful for a practical approach to embedded software development, but need to be supplemented on the theoretical aspects by material from other sources.

Books on real-time systems [13][14][15][16][17] are good sources for information especially related to the real-time aspects including RTOS, scheduling etc. However they often do not present the topics from an embedded systems perspective. One book that gives a good overview of both the areas is [4], but it is not available in the international market. This book is designed more as a primer for those who already are into software development. Another good book that presents the real-time concepts with embedded systems in mind is [16].

In our course, we used [1][2] as the primary textbooks, and supplemented with material drawn from [3][4][5][6][7]. Also, materials about specific real-time OS were drawn from various websites dedicated to the specific RTOS. Details and links can be found on the course website [8].

C. Hands-on Laboratory Exercises

The hands-on laboratory component concentrated mainly on the use of several real-time OS and integrated development environments. We did not have a dedicated embedded systems laboratory set up by the time the course was offered. So a general purpose teaching laboratory equipped with standard PCs was used for the laboratory exercises. We are currently in the process of setting up a dedicated embedded systems laboratory where suitable embedded development kits and access to various RTOS and integrated development environments would be available.

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.

Students were also exposed to the μC/OS-II kernel, and the Symbian OS and Nokia Series 60 platform. The Java 2 Micro Edition (J2ME) platform was also introduced through Sun Java Wireless Tookit and NetBeans IDE 4.0.

The laboratory exercises were mainly aimed at exposing students to different environments for embedded software development and also prepare them for designing and implementing the course projects.

D. 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. The project topics ranged from a Bluetooth based positioning system implemented on Pocket PCs, a multi-player paper, rock and scissors game implemented using J2ME, an Internet based chat client implemented using J2ME, and a multimedia center implemented using J2ME. One group enhanced μC/OS-II by implementing support for addressing the priority inversion problem. They implemented support for the priority inheritance protocol.

E. Reflections on the Course

The first offering of our course provide a rich source of interesting experiences to reflect upon and make choices and adjustments for the future offerings of the course. We will first reflect on the three major areas that we covered in the course and some observations from our experience in teaching them. Embedded software development was one of the most unique aspects of this course that distinguishes it from other courses. This topic while quite practical has some interesting theoretical aspects to be covered. Emphasis was put on explaining cross-platform development including issues related to cross-compilation, host-based development and target deployment.

The RTOS section shared some overlap with a standard operating systems course. We leveraged on the fact that all the students have already had taken an specific course dedicated to operating systems earlier, and minimized the overlap by only providing a quick overview of the typical OS related material. Greater emphasis was put on real-time scheduling and memory management issues from an embedded systems perspective.

As already mentioned, Embedded Software Engineering posed the greatest challenge for teaching in this course. Software engineering is more geared towards large scale software projects. There is lack of suitable case studies illustrating the use of software engineering techniques in embedded software development. The relatively recent adoption of these techniques in the embedded software engineering field is partly the cause of this scarcity. Furthermore, the adoption of formal techniques like UML is fairly new in this area. The UML framework is too general to be adopted directly for embedded software. There has been
several recent efforts to introduce the concepts of real-time and timeliness into UML. But most of these are in the research stage. There is lack of suitable case studies or meaningful examples to illustrate the use of UML in embedded software development. We scoured the literature to find some simple examples, but this area needs to be explored further.

Several interesting issues arose before, while and after offering the course. Some of these issues arose from our own reflections, while others were in response to opinions expressed by our colleagues. Here we discuss some of these issues. The embedded community can perhaps address some of these issues.

One major issue was whether the embedded software techniques covered in this course merits a separate course, rather than being covered as part of existing courses on specific topics. For example, many of the RTOS concepts can easily be covered in a course on operating systems. Similarly, cross-compilation and cross-platform development can be covered in a compiler course. Similarly, embedded software engineering techniques can easily be covered in a software engineering course. While it is certainly useful to include discussion on embedded systems related issues in the specific courses, a detailed coverage may not be feasible. In our opinion, it is appropriate to have an integrated course where all the related topics are covered. Care must be taken to avoid significant overlap with the other courses. Where feasible, some of the related courses can be included as pre-requisites so that students already have sufficient background in specific topics.

Another major issue that we grappled with in designing this course is how to balance the course content between theoretical aspects and practical skills. Our approach was to give sufficient emphasis on practical skills through hands-on laboratory exercises, while ensuring that the related theoretical areas are also emphasized. Wherever feasible, the theoretical concepts were reinforced with related laboratory exercises. This area being relatively new is still deficient in generic techniques that can be covered. So a greater emphasis on specific platforms and solutions may be needed in the interim.

This brings us to another related issue, i.e., whether the course is best taught using a single platform (e.g., Windows CE, VxWorks, Embedded Linux), or whether more concentration should be given to the generic concepts and approaches. We adopted the latter approach, while still ensuring adequate coverage of specific platforms.

Another issue is how to balance the coverage between hardware and software. In particular, how much of the low-level techniques should be covered in the course. We opted to limit our hardware coverage to generic techniques and hardware platforms and not get into very specific hardware. A discussion on interfacing followed by detailed discussion on device driver design and implementation would be beneficial. In this area, complete isolation of the students from the underlying hardware is neither feasible nor desirable.

IV. Students

As already mentioned the course was designed as a senior undergraduate course. In the first offering of the course, most of the students were in their second year or third year (final year) of their undergraduate education at the university. 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.

A. Background

Not surprisingly, most of the students taking the course (almost 99%) were doing their bachelor’s degree in computer engineering. There were just a couple of students doing their bachelor’s in computer science, although the course was designed to attract students with both computer science and computer engineering background. Students in their final year of study made up approximately 60% while students in their second year made up the remaining 40% of the enrollment in the course.

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.

B. Student Feedback

Overall the course was well received by the students. 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 quite 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. They especially wanted to have experience with dedicated hardware and embedded development platforms, rather than the general purpose PC. This will be addressed in the future offerings of the course because we are setting up a dedicated Embedded Systems laboratory equipped with suitable hardware and software, including embedded development platforms.
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.
3. Most students expressed 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. Their preference was to concentrate on Windows CE, Embedded Linux and VxWorks.

4. Students also wanted more coverage on interfacing, especially with emphasis on device-driver development. Future offerings of the course will include more emphasis on these topics.

5. 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.

6. UML was not well appreciated because of the short time devoted to cover it. This is related to our observation earlier that lack of suitable case studies hampers the proper coverage of UML. This issue can be addressed by including meaningful case studies, especially demonstrating the application of UML in embedded software development.

V. CONCLUSIONS

In this paper we discussed our experience in designing and offering a senior undergraduate course on Embedded Systems Software. The course was described in detail and some reflections on our experience in teaching the course are included.

Our experience offers only one point of reference for academics interested in designing and offering similar courses. It is our hope that this experience sharing can contribute towards further sharing of views and experience from academics worldwide. In the longer term, a broader consensus on the teaching of Embedded Systems in the academia can thus be attained.

ACKNOWLEDGMENT

The author wishes to thank the Dept. of Computer Science at HKUST and especially the Dept. Head for encouraging the development of this course. We also thank Microsoft for supporting the course under the Windows Embedded Academic Program (WEMAP) by providing us access to Windows CE software.

REFERENCES

How should embedded systems be taught? Experiences and snapshots from Swedish higher engineering education

Martin Grimheden and Martin Törngren

Abstract—This paper presents results from a previously published analysis performed on the subject of embedded systems using a didactical approach together with some educational implications. To illustrate the didactical analysis, examples from embedded systems education at KTH in Sweden are given where an exemplifying selection and an interactive communication have been adopted in embedded systems related courses.

The exemplifying selection means that education in embedded systems would benefit from teaching ‘everything of something’ rather than ‘something of everything’, that is, depth rather than width. The interactive communication strongly motivates experimental work, hands-on approaches and problem based learning in general.

Index Terms—Embedded systems education, didactical analysis, engineering education.

I. INTRODUCTION

Even though embedded systems have been designed for more than 30 years, the academic subject of embedded systems is a new, relatively undefined subject, mostly regarded as an interdisciplinary field combining areas such as computer science, automatic control and electrical engineering. The scope of the subject is continuously discussed, with its implications on research and education.

The aim of this paper is to present results from an analysis performed on the subject of embedded systems using a didactical approach [9]. The purpose of this analysis is to help the establishment of the identity of the subject and to provide further arguments for why the subject ought to be taught in a certain way.

A. Context

The subject of embedded systems is mostly regarded differently among different academic institutions. An example of this is given if comparing the origin of the subject among different universities; in some cases the subject evolved from within departments of computer science, from departments of electrical engineering or from departments of automatic control, which in several cases gives a slight difference between the local definitions of the subject. As a consequence, some universities treat and teach embedded systems as a specialization of computer science, and some departments use the subject of embedded systems to promote education and research in automatic control.

In the examples presented in this paper embedded systems is described and analyzed according to the way embedded systems originally was implemented at the department of machine design, mechatronics lab, at the Royal Institute of Technology (KTH) in Stockholm, Sweden. In this case, embedded systems evolved from within the school of mechanical engineering, specifically the department of machine elements where the microcontroller was introduced as ‘a machine element’ similar to gears, bearings etc, but with a programmable functionality. Today, 30 years later, embedded systems at KTH is treated as a subject within machine design, with a heavy focus on product development, development of intelligent products.

B. From Education through Research to Industry

The purpose of teaching embedded systems at most universities is to provide the industry with competent staff. The purpose of performing research is to educate doctors in embedded systems, for the same purpose, as well as to create new knowledge in the area of embedded systems, for the industry as well as the common good of the society.

Within the ARTIST and ARTIST2 projects, work has been in progress for quite some time to identify and specify a curriculum for education in embedded systems [1, 2]. In the guidelines published by the ARTIST teams the technical competencies required by an embedded software engineer are well specified, and the need for practice is recognized as 'an essential component for a well-rounded education in embedded systems'. Practice, however, is defined as 'experimental laboratory work' and illustrated by the use of tools such as Matlab/Simulink, running code on real hardware platforms etc. The guidelines do not mention any complementary skills such as teamwork, communication, business, etc.

It is therefore of fundamental interest that the education provided by the universities is deemed relevant and useful by
the hiring industry, as well as by the industries of the future.
One main question then becomes how the universities can understand the needs and requirements of the current hiring industry, how to balance between fundamental knowledge and skills and short-term hot trends as well as predict the evolution of the subject and prepare students for life long learning.

C. Closing the loop: From Industrial experiences to Education

By tradition the academic practice is characterized by a narrow viewpoint and a systematic approach, while the industrial practice is characterized by the need for a holistic viewpoint and a, more or less, ad-hoc approach, as in Figure 1. The white arrows show the necessary path to fulfill a better balance in selection and communication (form and content). Industries are constantly struggling to move from an ad-hoc approach towards a more systematic approach, and in the universities there are certainly many initiatives in the direction of reaching a more holistic viewpoint.

One way to reach a more holistic viewpoint is to focus more on the communication, the form, and, for example, to teach in a more project organized and problem oriented way - both to increase student motivation and also to move towards a more functional approach and thereby hopefully reaching a higher level of understanding. A further motivational factor for the move towards these educational methods is that the project- and problem based educational methods can be seen as preparations for a future professional role as an embedded systems engineer, which require skills such as teamwork, communication etc.

Another example from teaching indicates synergetic effects when giving courses with participants both from industry and university. Industrial participants can contribute along the lines of Figure 1, with experiences that place theoretical pieces in a context, by reflections relating to tacit knowledge and with real-world engineering constraints that are seldom touched upon in the academic teaching [12].

II. A DIDACTICAL PERSPECTIVE ON EMBEDDED SYSTEMS EDUCATION

Didactics is a field of educational studies mostly referring to research aimed at investigating what's unique with a particular subject, and how the particular subject ought to be taught. In this paper we present results from previous attempts of performing a didactical approach to the subject of embedded systems [5, 9]. The didactical analysis is used to identify and describe the identity and legitimacy of the subject (what is the subject of embedded systems and why should embedded systems be taught?) [4]. We also use this analysis to provide insight into the questions of selection and communication (which material should be taught and how?).

A. The identity of embedded systems

According to the didactical analysis the identity of a subject could be described in relation to two extremes; as being either disciplinary or thematic, or somewhere in between. Most traditional subjects, with established structures and related organizations, are considered as disciplinary subjects. A closure has then been reached as to the contents and internal relations of the subjects and its domains. With subjects such as embedded systems this closure has not yet been reached. Every university and department considers and teaches the subject differently, and on most conferences the scope and content of the subject is discussed in depth, as is its preferred curriculum for example.

Instead of an established disciplinary identity, themes are used to describe embedded systems, and the themes vary amongst different universities. According to IEEE an embedded system is 'part of a large system and performs some of the requirements of that system; for example, a computer system used in an aircraft or rapid transit system' [10]. From this definition it follows that computer based systems embedded into products like for example TVs, telephones, toys and vehicles qualify as embedded systems, and that the characteristics of these products include their interactions with the environment. Examples of relevant aspects are dependability, performance and cost which implies key areas such as safety critical systems, real time systems etc. Such aspects, areas and applications can therefore be used to describe the subject of embedded systems, as themes of the subject.

However, even if most universities happily agree upon a number of themes, applications and products, discrepancies between how to approach these themes arises. Some universities focus on control aspects of embedded systems, some on developing methods within computer science to reach certain goals while some focus on more general product development of embedded systems.

B. The legitimacy of embedded systems

The legitimacy of a subject is defined as the relation
between the educational effort performed by the university, resulting in educated embedded systems engineers, and the demands put by the hiring industries on the graduated engineers. According to the didactical approach this legitimacy could be illustrated according to two extremes; as a formal legitimacy or a functional legitimacy. A formal legitimacy is defined as when the demands put by the surrounding actors on the university are expressed in terms of number of credits in specific subjects, which subjects the degree requires etc. In a functional legitimacy instead the actors requires skills and abilities, and expresses these in terms such as capabilities to design controllers etc.

In the case of embedded systems the legitimacy is classified as functional [9]. In Sweden, the hiring industry is primarily interested in functional skills, engineers capable of designing and implementing embedded systems. A study of 21 embedded systems companies in Sweden further motivates this; the companies studied put formal knowledge and traditional courses aside and instead ask for functional skills and complimentary skills such as teamwork abilities and presentation techniques [11].

C. The selection and communication of embedded systems

The results from the analysis, the thematic identity and the functional legitimacy, affects the final two dimensions or questions; the questions of selection and communication. These two questions deal with the what and how; what should be taught, and how.

A representative selection means that ‘something of everything’ is taught. This is often the case in, for example, first year courses in mathematics. The opposite is defined as an exemplifying selection where ‘everything of something’ is taught. This is often the case in problem based learning or in postgraduate education where the focus is on a narrow field instead of an overview. One example of teaching with a representative selection could be a course in, for example, microcontroller technology that focuses on giving general knowledge on the varieties of the existing microcontrollers, the differences between these, and more theoretical knowledge about microcontroller principles. If instead choosing an exemplifying selection, the course should focus on complete understanding of one single microcontroller. All effort should then be spent on learning this microcontroller, and to become an expert of understanding, programming and using this microcontroller. The basic idea with an exemplifying selection is that the knowledge and skills learned from this selection could be generalized and applied to similar problems and areas.

The final dimension, the question of communication, deals with the preferred method of communicating the subject. In an active communication both parties are active; teacher and student, but in an interactive communication the action by the teacher is dependant on the current status and knowledge level of the individual student or student team.

In Figure 2 the subject of embedded systems is mapped according to its identity, legitimacy, selection and communication. The exemplifying selection and interactive communication is a direct consequence of the identity and legitimacy, which also is emphasized in the Swedish study performed on 21 Swedish embedded systems companies [11]. In brief, the exemplifying selection is a consequence of the functional legitimacy; the companies want engineers with in-depth knowledge and functional skills rather than formal knowledge. The interactive communication also facilitates functional skills rather than formal knowledge since such a teaching approach focuses more on hands-on approaches and provides complimentary skills such as teamwork abilities and presentation techniques.

III. EXAMPLES FROM TEACHING EMBEDDED SYSTEMS WITH AN EXEMPLIFYING SELECTION AND AN INTERACTIVE COMMUNICATION

The aim of this section is to provide examples and experiences from teaching embedded systems at KTH with an exemplifying selection and an interactive communication. It is important to note though that the examples provided here should be considered in their respective context, as described
earlier in this paper. All courses below attract mainly mechanical, vehicle and management engineering students specializing in mechatronics or embedded systems.

A. Teaching embedded systems according to the CDIO initiative

The CDIO initiative was established in unison between KTH, MIT and a number of other Swedish universities to move engineering education further from a formal legitimacy towards a functional. The purpose of the CDIO initiative was to create a curriculum that trained students to Conceive, Design, Implement and Operate (C-D-I-O) a system; in this case an embedded system [3].

![Introduction, training, ability requested, unspecified]

Fig. 3. An illustration of the CDIO-concept applied to course design at KTH. The illustration shows how complimentary skills are integrated in traditional subject courses, and the three-step model where each subject or skill is introduced, trained and requested over time.

At KTH the CDIO idea is implemented mainly in a capstone course in the fourth and final year of the specialization in embedded systems, a complete problem based and project organized course. In this course, a team of 6 to 15 students are given a task, typically in terms of an industrial development project involving concept evaluation and prototype development, where a corporate sponsor provides the problem, motivation, relevance and funding to the project. The student team spends more than 50% of their time during three quarters of a year in a project that is organized in four phases (C-D-I-O). Complimentary skills such as teamwork skills, project management, economy, language skills etc are interwoven in the project as each student receives two responsibilities, one related to the product and one related to the process. Student responsibilities are cycled in each phase, and after the nine months each student will have practiced skills in embedded systems technology as well as team management, teamwork etc. For more information about the KTH capstone course, see for example [6].

In an evaluation performed by KTH all graduated mechatronics/embedded systems students were asked which course taken at KTH that was most important and useful [5].

B. The lab in your pocket

‘The lab in your pocket’ project was created to implement both an exemplifying selection and an interactive communication in a course in microcontroller technology. The basic ideas of ‘the lab in your pocket’ concept are the following:

1. Each student has constant access to his/her own set of laboratory equipment.
2. The laboratory equipment can be used at any location, at any time. The only requirement is access to an ordinary PC.
3. With the equipment, each student can perform all laboratory work within the course.
4. The equipment promotes open-ended solutions, meaning that all experiments are flexible enough to encourage creative solutions.
5. The total cost of all sets of equipment does not exceed the cost of the traditional laboratory facilities.

The laboratory equipment consists of one microcontroller, an Infineon C167CS, an I/O-board with a LCD-display, keyboard, buttons, LEDs, a DC motor etc as well as some sensors such as accelerometers and temperature sensors. Also, manuals, C-compilers and examples are provided for all students.

![Microcontroller modules, communication modules, interface modules, application modules, components, sensors, actuators, DC servo, AC servo, Pneumatics, IR, Ultrasonic]

Fig. 4. The Mecatronic Learning Concept, as developed by KTH. Modules for the ‘Lab in your pocket’ is chosen from these.
In an evaluation of this project we found that the participating students received considerably higher grades, that the students spent considerably more time on experimental work, and that the faculty spent considerably less time on supervision, all this as compared to a traditional experimental course [7]. As a consequence, the total course cost for the university was reduced, again as compared to the traditional course.

C. Results from project ‘Lab in your pocket’

For an evaluation 30 students constituting a focus group answered a set of questionnaires, and a selection of ten students was selected for interviews. In these questionnaires and interviews the students were asked to describe their approaches towards the experimental work; when, where and how the exercises and projects were done, the overall attitude towards the concept, how much time spent experimenting etc. Another comparison was made between the focus group and the reference group regarding the grades of the projects, the delivery time of the project results, and the overall view of the course. These results can be summarized into the following:

1. Overall course attitudes
   a. The focus group appreciated the exercises, the project and the teachers considerably more than the reference group (grade 4,41 compared to grade 3,8)
   b. The focus group attended fewer lectures than the reference group (53% compared to 68%)
   c. The focus group attended more exercises than the reference group (87% compared to 80%)

2. Results of the experimental work
   a. The focus group received considerably higher grades on the projects than the reference group (grade 4,7 compared to 3,9)
   b. In the focus group, only 6% compared to 43% in the reference group chose the projects with the lowest grade. 79% of the focus group chose the highest grades compared to 32% in the reference group.

3. Non-quantified measurements
   a. The average student in the focus group spent considerably more time doing experimental work than the average reference student. The reference group often worked against a deadline, while the focus group worked more spontaneous and at irregular hours.
   b. The faculty spent considerably less time supervising the focus group compared to the reference group. The reference group required constant supervision, while the focus group communicated via the educational web based platform, and collaboratively supported each other.

4. Economical results
   a. The cost of supervision can be reduced since the focus group required considerably less support from the faculty.

The cost of laboratory facilities can be reduced since the focus group did not require any facilities except the portable sets. In this experiment, the annual cost of keeping the laboratory exceeds the initial investment of portable sets.

D. Education as preparation for future work on a global market

The functional legitimacy gives at hand that the education ought to be a preparation for future work, and since most companies act on a global market there is a need to also encompass these global aspects into the education. In one example at KTH, this is done within a capstone course that is expanded to cover international aspects [8]. Several modes of international collaboration has been experimented with, either collaboration with a foreign corporate sponsor or collaboration with one or several international student teams.

Among other conclusions, the international collaboration in the studied capstone courses promoted [6]:

1. Improved disciplinary learning and other skills. The international collaboration creates access to

---

1 The grades vary from 1 (really bad) to 5 (excellent)

2 This data is based on the students’ own estimates, gathered in interviews.
more resources and gives new and different perspectives to problems. The collaboration also promotes general skills such as teamwork, team management and presentation techniques.

2. Awareness of cultural differences and different educational systems. The collaboration promotes this awareness, which is an important competence in a future career in a global company.

3. Enhanced motivation. The international collaboration itself is seen by several students as an interesting challenge.

In the above, examples can be found of functional legitimacy related to both the subject of embedded systems as well as to general skills related to working in a global company.

IV. CONCLUSIONS

The purpose of this paper is to argue for a view on embedded systems education where the results of the didactical analysis are put into focus. Embedded systems, in Sweden, is a subject with a thematic identity and a functional legitimacy, which means that the embedded systems companies are requesting engineers capable of conceiving, designing, implementing and operating embedded systems. The main idea of the didactical approach is not to specify a certain curriculum or a number of courses, but to start with the functionality – to agree upon the idea of primarily educating engineers capable of this rather than engineers with a certain number of credits in a certain number of courses.

Three examples are given from embedded systems education at KTH in Sweden, where the ambition has been to build on the foundation of this thematic identity and functional legitimacy, to create courses with an exemplifying selection and an interactive communication, meaning that the students should learn ‘everything of something’, with an experimental approach, or in a problem-oriented and project-organized setting.

REFERENCES

A Case Study in Efficient Microcontroller Education

Bettina Weiss, Günther Gridling, Markus Proske
Vienna University of Technology,
Embedded Computing Systems Group E182-2,
Treitlst. 3/2, 1040 Vienna, Austria
Email: {bw, gg, proske}@ecs.tuwien.ac.at

Abstract—Undergraduate education typically is characterized by a large number of students. Therefore, courses must be conducted efficiently and should not only focus on conveying the course material, but must also be oriented towards a maximum transfer of knowledge with a minimum amount of invested time on the instructor’s part. At the same time, courses should be flexible to accommodate different student needs.

In this paper[1], we identify the needs of a practical course in microcontroller programming with respect to course structure and grading, present our solutions, and discuss our experiences.

Index Terms—embedded systems education, automatic grading, distance learning, pedagogy

I. INTRODUCTION

Embedded systems education is a vital part of the computer engineering curriculum and has gained increasing importance in the last decade [1], [2]. However, it also suffers from problems that are specific to any hands-on course using hardware: To teach the practical skills involved, students must be assigned an appropriate workload, but lab resources are limited. Add to that the increasing demand for embedded systems engineers, which entails an increase in student numbers, and embedded systems education soon reaches its limits with respect to both lab resources and available personnel. It thus makes sense to invest a significant amount of time into the setup of a course, if the course can then be managed with less strain on the university’s resources.

The recently introduced bachelor study “Computer Engineering” at the Vienna University of Technology contains several courses which focus on programming embedded systems, among them the introductory course Microcontroller [3]. The Microcontroller course gives second-year computer engineering students their first experiences with microcontrollers and hardware near-programming. We expect students to be proficient in C programming and to have basic computer architecture and electrical engineering knowledge. The course is intentionally kept low-level, much like [4], and teaches students both the basics of microcontroller programming (without debugging tools) in Assembler and C and how to control external hardware. For most students, it is their first contact with microcontrollers, with hardware, and with Assembler programming.

When setting up this course, we faced the challenge of many (about 150) students in an 8 workstation lab. We also wanted to make the course more accessible to employed and handicapped students, potentially scalable to a large number of students, and suitable for distance learning, which further added to the challenge. This made it very demanding to balance the needs of the students, the demands of the curriculum, and the constraints of available resources. We are teaching the microcontroller course since 2003 and in each year experimented with different mechanisms to attain our goals. This year’s course had good student feedback and was conducted very efficiently from our point of view. So we can conclude that it is possible to find a satisfactory solution to this optimization problem, and we believe that our experiences, both positive and negative, gained from setting up this course can be of value to other instructors.

In the following sections, we will first identify the requirements on practical courses in Section II. In Section III we will present the course structure of the Microcontroller course and explain why we structured the course in this way. Section IV is dedicated to our grading scheme, which is essential to achieving our goals. In Section V, we give a discussion of our current course, its advantages and disadvantages, and student feedback. Section VI concludes the paper.

II. COURSE GOALS

When reviewing our expectations for the course, we identified three different and possibly conflicting areas of concern:

Financial Impact:
The course should be conducted efficiently, with a minimum amount of time and resource efforts. It should be scalable to a large number of students and should be suitable for distance learning. Since the time of a professor is most expensive, whereas tutors are comparatively cheap, the bulk of student supervision should fall to tutors.

Educational Impact:
The course should be successful, that is, it should have high student retention, a high passing rate, and convey both microcontroller programming skills and understanding of the issues involved.

Motivational Impact:
The course should motivate students and should re-

---

1This work is part of the SCDL “Seamless Campus: Distance Labs” project, which received support from the Austrian “FIT-IT Embedded Systems” initiative, funded by the Austrian Ministry for Traffic, Innovation and Technology (BMVIT) and managed by the Austrian Research Promotion Agency (FFG) under grant 808210. See http://www.ecs.tuwien.ac.at/Projects/SCDL/for further information.
result in high student satisfaction. Students should get interested enough to start their own private projects.

Naturally, these goals require different techniques, which in turn may have a positive or negative influence on the other objectives. For example, in our first try, we allowed students free access to the labs, gave them an exercise set that was completely voluntary, told them that we were available for questions by email or newsgroup at any time, and conducted one programming exam at the end of the term. So basically, we told them to exercise as much as they thought they needed and then come to the exam. This scored well on the financial axis, moderately well on the motivational axis, and pretty low on the educational axis, since most students put off doing the exercises until it was too late and then dropped the course. Even those that passed lacked some basic knowledge. This taught us that for each of our objectives, we must identify the techniques most beneficial to it, but at the same time must also account for their effects on the other areas. As a result, we went through the different pedagogical measures available to us and estimated their impact, see Table I. In the table, a + represents a positive impact, − a negative one, and values in brackets are setup costs that only occur once. The total column states the overall impact.

Due to limited space, we cannot discuss our reasoning behind the values, but bear in mind that the table represents our personal assessment and values, so other instructors may come to slightly different evaluations.

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>DIFFERENT TEACHING TECHNIQUES AND THEIR IMPACT IN DIFFERENT AREAS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fin.</td>
</tr>
<tr>
<td>autogradable exams</td>
<td>++</td>
</tr>
<tr>
<td>essay exams</td>
<td>−−</td>
</tr>
<tr>
<td>just 1 exam</td>
<td>++</td>
</tr>
<tr>
<td>several exams</td>
<td>−</td>
</tr>
<tr>
<td>drop result</td>
<td>−</td>
</tr>
<tr>
<td>flexible grading</td>
<td>−</td>
</tr>
<tr>
<td>voluntary homework</td>
<td>++</td>
</tr>
<tr>
<td>ungraded homework</td>
<td>++</td>
</tr>
<tr>
<td>graded homework</td>
<td>−−</td>
</tr>
<tr>
<td>flexible workload</td>
<td>−</td>
</tr>
<tr>
<td>extra work counts</td>
<td>−</td>
</tr>
<tr>
<td>open labs</td>
<td>++</td>
</tr>
<tr>
<td>closed labs</td>
<td>−−</td>
</tr>
<tr>
<td>supervision</td>
<td>−−</td>
</tr>
<tr>
<td>no supervision</td>
<td>++</td>
</tr>
<tr>
<td>group work</td>
<td>+</td>
</tr>
<tr>
<td>lectures</td>
<td>−−</td>
</tr>
<tr>
<td>electronic lectures</td>
<td>[++]</td>
</tr>
<tr>
<td>no lectures</td>
<td>+ +</td>
</tr>
<tr>
<td>script</td>
<td>−−</td>
</tr>
<tr>
<td>suppl. material</td>
<td>−−</td>
</tr>
</tbody>
</table>

With this table in mind, we then did a complete redesign of our course, employing the mechanisms indicated by bold numbers in the total column of the table, and had far better success with it while still keeping the workload of the professor acceptably low. We can thus conclude that it is possible to get good educational and motivational results with only moderate concessions on the financial part.

III. MICROCONTROLLER COURSE

This year’s course got very good student ratings. The structure we describe here already incorporates our experiences gained this year and will be first employed in the summer term of 2006. Since the changes intended for 2006 are minor and to a large extent supported by student and tutor feedback, we expect the course to have a similar (if not better) impact on students as this year’s course.

A. Educational Material

Since students should be able to work at home, we needed to get the lab to the students. Although it would have been more cost effective to use simulators like in [5], simulators cannot capture all the problems that arise when using real hardware [6]. Furthermore, working with real hardware instills into students the confidence that they know how to handle hardware, and often encourages them to start their own projects. Therefore, we decided to use real hardware in the course, which had to be cheap to be suitable for “mass” production. Unfortunately, we did not find any suitable existing hardware and hence proceeded to develop our own lab hardware.

One important lesson we learned about student hardware is that it is vital to let students connect the hardware themselves. In our first course, we used a commercial HCS12 evaluation board with fixed connections, and we ended up with a lot of students who simply did not understand how a LED and a microcontroller pin go together. They had just learned that “writing 0x0F to port X turns on LED0-3”. So our own hardware has no fixed connections, students must do all the wiring.

![Fig. 1. Basic microcontroller hardware](image)

We decided to use an Atmel microcontroller because they are common and fairly inexpensive, and because their RISC architecture is easy to understand and to program. In order
to be more flexible, we designed several separate boards: The microcontroller board only contains the microcontroller, an ATmega16, and its supportive logic (power jack and regulator, oscillator, reset button, programming connector) and single connectors to its I/O pins instead of a fixed connector like for example in [7]. As a side benefit, the microcontroller can be exchanged freely, and we are in fact working on several additional controller boards (HCS12, ARM, MSP430) to allow good students to broaden their knowledge by switching to a different controller during the course. The simple I/O board contains 8 LEDs, 8 buttons, 8 switches, a 6-digit numeric display, a 4×4 matrix keypad, 2 potentiometers, and a light sensor. Most of the exercises can be done with the controller board and the simple I/O board. For communication to the PC, we also offer an RS-232 board which contains a serial interface. All three boards are mounted on a wooden plate, see Figure 1, and can be connected via normal wires. For more challenging projects, we offer additional boards like a motor board with dc and stepper motors, see Figure 2, an interface board with serial, parallel, and CAN interfaces, or a chip-card reader.

Apart from the hardware, we also offer some instructional material. Although most of the course is lab work, there are some accompanying lectures that give students an introduction into the microcontroller and the hardware. To allow distance learning and to remove the lecture load from the instructor, lectures are not held in class, but will in the future be provided electronically as slides and/or video with audio track. A lecture script complements the electronic lectures. Finally, manuals for the hardware are available, and we provide sample programs to get students started.

B. Course Structure

Basically, our course employs a mixture of exams, light compulsory workload and flexible voluntary workload that allows different work styles while still yielding comparable grades.

The course is divided into three parts. The first part is intended as an introduction into the programming environment, the microcontroller architecture, the hardware, and assembly language. The second part is dedicated to learning how to program the basic features of the microcontroller like timers and the analog module, how to handle interrupts, and how to program typical hardware like matrix keypads and numeric displays. The third part is concerned with interfaces, motor control, and application programming. Both second and third part are in C. To ensure that the students learn the most important concepts, they have to do a couple (3-4) of compulsory exercises in each part. These exercises cover the microcontroller’s features in simple tasks designed to convey basic knowledge about how the feature works and how it is programmed. We decided to focus on simple tasks here because in our experience complex tasks cause students to concentrate on getting the program to work instead of on getting it right. To deepen their knowledge, students may select additional voluntary exercises from a significantly larger (20-30) exercise pool. Some of these exercises cover special details
of the microcontroller or the hardware in depth, others are applications. Depending on the quality of the work and the complexity of the exercise, students get a certain amount of “bonus” points per completed exercise, which are added to their course result.

Since there cannot be any guarantee that students do their homework all alone, each part is concluded by a programming exam (consisting of two exam tasks) and a theory exam. Attendance at all exams is not compulsory, but students must fulfill some minimum requirements concerning their exam results for a passing grade. If students come to all three exams of a type, their worst result on these exams will be dropped.

C. Lab Organization

Although closed labs seem to be more effective [8], they are hard on resources and not flexible w.r.t. working hours. Even though we concur with [8] that having a closed lab in the initial phases of the course would probably be a good thing, we cannot do this due to resource problems, except if we make students work in groups. However, after two years of experience with group work in a closed lab setting (3 persons/group), we cannot affirm the results of [9] that students learn more in group work. We suspect that the effect of group work depends on the tasks posed to the students: Large and difficult tasks that cannot be done by one member alone make for good teamwork and allow students to learn more in the team; simple tasks, however, are more conducive to one student doing the work and the others just watching. The student doing the work possibly benefits, but the onlookers will lack essential practical programming and debugging skills at the exam. Since we focus on simple tasks that capture the essentials, we observed that group work is detrimental to overall student performance and also brings the danger of students over-assessing their skills – after all, they put in a lot of hours watching and feel that it should suffice if they know how to do it in theory. For many students, the exams are real eye-openers in this regard.

So closed labs and group work are out. On the other hand, we also have experience with an open lab and voluntary individual work, which resulted in students putting off the work as long as possible and then getting into trouble with the exams due to lack of practice. Also, retention in this course was terrible, only about 30% of the students remained until the end of the course, because most failed to acquire the necessary skills in time. Furthermore, the lack in supervision resulted in exercise solutions that appeared to work, but were pretty convoluted and sometimes even wrong. Yet, students thought that their solutions were right and without feedback had no way of knowing otherwise. So just doing open labs was not a viable option either.

To get the best of both worlds, we decided on a mixture of open and closed labs: Students are expected to work at home, but each student is assigned to a supervised lab hour once a week. Attendance is not compulsory, but highly recommended. The lab supervisor (tutor), a senior student, is assigned 5 students per hour, and is there to answer their questions, help with debugging problems, check whether students are on schedule with their homework, and generally lend an ear in case of problems and keep up the students’ motivations. We found that the tutors have a tremendous influence on the motivation and performance of the students and are in fact the pivotal factor in determining the motivational impact of a course.

Since the lab hour is not for doing work, but for discussing problems and asking questions, it could easily be conducted in a distance learning setting via Webcam. The important thing here is that the student is encouraged to regularly report about his or her progress, and that the tutor keeps track about each student’s progress and motivation and can react to problems at an early stage.

D. Personnel

In order to minimize personnel costs and to provide scalability, the course uses a hierarchical system for student supervision: The course is managed by one assistant professor, one teaching assistant, and \( n/15 \) tutors, where \( n \) is the number of students in the course.\(^2\)

Tutors work for three hours a week and meet with 5 students per hour. In this time, they talk with their students about their progress, check their work, answer questions, and determine how well the students have mastered the material so far. They should also do a preliminary grading of their students’ homework (check whether the programs work and whether the protocols are in order). The work is very challenging for the tutors, so this job requires highly motivated students. It is important to be on the lookout for such students during the term, and to actively interest them in a tutor job for the following year.

The teaching assistant, who works for 18 hours per week, is responsible for the lab equipment, for organizing the exams, and for the final grading of the homework (after checking for plagiarisms). The TA is also the first address for tutors with questions or problems.

The assistant professor is responsible for the course contents and organization, for the course materials, for the exam contents, and for exam grading. Since we use electronic lectures which only have to be done once, after course setup is complete the course poses only a minimal load on the professor.

IV. Grading

For grading, we must ensure that diligent students can use homework to balance bad or missing exam results, while at the same time not allowing dishonest students to pass with somebody else’s work. To this aim, we do both theory and programming exams, and we require that all students should get at least two out of the six programming exam tasks right. Students who fail to meet this requirement do not get a passing grade, no matter how many points they attain. This ensures that students who pass the course have some basic microcontroller

\(^2\) This maximum allowable number of tutors is prescribed by the computer science faculty of the TU Vienna.
programming skills. For an excellent grade, students must also show that they have a grasp of the theory, so attaining at least one third of the theory points is a prerequisite for an excellent grade.

A. Programming Exams

Our programming exams consist of two tasks each. Students get two hours time, starting with a half hour design phase without computer, followed by the programming (and testing) phase on the actual lab hardware. To attain points for an exam task, students have to write a correct program. Since no partial credit is given, the tasks are very easy and generally consist of at most 20 lines of code. Students are provided with a framework, consisting of an Assembler or C skeleton program, possibly header files, additional object files, and a Makefile, and just have to add their code to the skeleton to complete the task. This allows us to place our exam tasks within more complex settings while still giving simple student tasks. For example, an exam task could be:

Task: Create a program which acts as a frequency measurement routine. Configure the Input Capture Function of Timer/Counter 1 to determine the frequency produced on pin PD7.

The frequency is in a range between 10 Hz and 99 Hz, so use prescaler 256. Display the ten’s place of the measured frequency value on the display, using the provided function display_result() .

Here, the students just have to set up the input capture feature of a particular timer and compute the frequency. All other functionality, like displaying a number or generating the input function that should be measured, is provided by the framework.

Tutors are available during the exams to help students with hardware trouble. Since hardware-near programming is prone to simple but hard-to-find bugs, the tutors are also allowed to help with debugging problems, but only if the student can demonstrate that he or she understands the material and just suffers from a blackout.

Right now, the correctness of the student programs is checked manually by the tutors. To help the tutors assess whether the register initializations of the timer and other components are correct, we add some functionality to the provided framework to regularly check the register configurations and put the results on the LEDs. Thus, the tutor can see at a glance whether the register initializations are correct, or whether there are missing and/or superfluous bits. In addition, we are currently working on an automatic external test system which can provide the microcontroller with specific test stimuli and verify its responses. Since the software check outputs its results on the LEDs, these results can easily be taken over by the external test system, which can thus perform a completely automatic correctness check. We expect to have this system working by next summer.

Doing the practical exams on a computer with the actual hardware has both advantages and drawbacks. On the positive side, students can try out their programs and have the opportunity to debug them. This in turn allows us to use an all-or-nothing grading, which definitely would not be accepted by students if this was just a paper exam. As another benefit, the exam results are instantly available, no later grading session is necessary. On the negative side, since students generally pass on their exam tasks to the colleagues that come after them, one either requires a lot of PCs or a lot of different exam groups. Since coming up with good yet simple exam tasks is quite hard, the first is preferable if the computers and hardware are available.

To allow a distance learning setting, we need to give students, who are sitting the exam at some external location, remote access to the automatic test system. We should also provide them with remote tutor help via Webcam. We are currently working on this problem in the SC DL project. Note that as soon as the exam can be conducted remotely, a large number of students can sit the exam concurrently. The persons supervising them do not have to be trained, only the tutors at the university giving the debugging help do.

B. Theory Exams

For our theory exams, we decided to use exams with yes/no questions instead of the traditional essay exams. Our decision was led by our desire for a fair grading scheme and by the fact that such exams can be graded automatically. Hence, we conduct our theory exams on computers, and the evaluation is done automatically as well. Originally, our exam questions were classical multiple choice questions in the form of a root (question) and several stems (answers) [10], but we gradually shifted away from this exam type and now use topics with several independent yes/no questions each. First of all, it broadens the range of material that can be covered with the exam (not every topic can be covered by a question with multiple plausible answers). As a nice side effect, we found that students like this kind of exam better as well.

Correcting the exams is done automatically. A correct answer is worth +1 points, a wrong answer gets −1, and no answer get 0 points. In consequence, if the maximum number of attainable points is \( N \), then a student can get between \(-N\) and \(+N\) points on the exam. A negative exam result is set to zero. Guessing is neither punished nor rewarded by our system: On the average, a student will neither gain nor lose with guessing. However, we strongly discourage students from guessing. At this time, we do not feel the need to actively punish guessing by making the penalty for a wrong answer any larger, but we could of course do so. However, since this also punishes students who did not guess but simply got the answer wrong, we decided to refrain from using this means to curb guessing as long as we get the impression that (most) students try not to guess anyway.

To improve the quality of our exams and to increase the students’ trust in our questions, we conduct a statistical evaluation of the exam papers prior to grading the exams, and we make students aware that we do this. For each question,
our statistic calculates the percentage of students who got the answer right, the percentage who got it wrong, and the percentage who abstained from answering. Questions with a high percentage of wrong answers or abstentions are carefully examined for their appropriateness, and are removed from the exam if we belatedly find them controversial. With this method, we have caught a fair share of ambiguous questions which slipped by our initial scrutiny but were identified by the number of students who had trouble with it. As a side benefit, this method also points out errors in the question database. We highly recommend this procedure for anyone who wants to do automatically evaluated exams. It does not cost much time, but makes the exams fairer and increases student satisfaction.

The theory exams can be graded very efficiently. Even with the additional effort of the statistic, we grade all exams (about 100) in a couple of minutes, and would have no problems at all to grade 1000+ exams either. A general advantage of our electronic exams is that they can be conducted anywhere where a computer and network are available. So it is no problem if such an exam is conducted at another university, or even at a high-school, as long as a trusted person is supervising the exam. As a drawback, one needs either many computers or a lot of different exam questions.

C. Homework

In addition to exams, students do some exercises from a larger set as homework. In order to get credit for their work, they have to submit a working program and a protocol describing their work and answering some questions about the subject.

The homework is the only part of the grading that does not scale well and is easily exploitable by dishonest students. However, it greatly increases the flexibility of the grading system, is beneficial to the student’s development of practical skills, and thus vastly improves student motivation and satisfaction. Therefore, we include graded homework despite all its problems.

To lessen our workload, we need homework that can be corrected mostly by tutors. Here, the microcontroller course benefits from its low-level approach: The exercises generally are not complicated. Thus, a well-trained tutor can check whether a program works, and can also give pointers if the solution is not programmed well. The same goes for the protocol: The tutor can check whether the answers to the questions are correct, and whether the protocol is well-written. This takes the bulk of the load off the instructors, who only have to verify the first assessment of the tutors to assign the grade.

For example, a simple homework task would be to explore the effects of a floating pin:

**Task:** Connect SW1 to pin PA4, set PA4 to input and do not activate the pull-up. In the main loop, display register bit PINA4 on LED0 and bit PORTA4 on LED1.

**Questions:** Turn SW1 to OFF. Wave your hand closely over the board, tap or wiggle the wire connecting SW1 to the controller. What do you see on the LEDs? Why?

Set SW1 to position ON. Repeat the actions of the previous question. What do you see on the LEDs now?

This exercise demonstrates the effects of a floating pin (when switch SW1 is turned OFF, the line floats; when it is ON, the line is grounded). Students observe the volatile nature of the power levels on the wire when they wave their hand over it or when they touch it, and are encouraged to speculate on the technical reasons for the observed effects. The exercise helps them identify floating problems, which occur quite frequently, in future programs.

Our most complex exercise task is to measure and plot the speed curve (acceleration and deceleration phases) of a dc motor for different PWM ratios. The students have to program the microcontroller to gather the data, send it over the serial interface to the PC, and create a 3D-plot with gnuplot. In their protocol, they have to interpret the resulting plot. This exercise forces students to put thought into program design, induces them to reuse previously written code, and thus gives the students a taste of complex application program development.

The major problem of homework is plagiarism. This can either take the form of copying (student A copies the work of student B, possibly slightly modified), or substitution (student A writes the homework for student B). Unfortunately, there is no sure-fire means to verify whether a student has done the work by himself. Even oral exams may not do the trick, since the student could have been tutored by the author of the code. Plagiarism detection mechanisms [11] can be used, but they too cannot identify cases of substitution. Therefore, our main means to catch such students is our requirement that students complete two exam tasks for a positive grade. This makes sure that students passing the course can write simple programs by themselves. A similar approach has been taken for example by [12]. Of course, this does not catch students who know how to program and are just too lazy to do the work by themselves. Therefore, we additionally check for similarities in the program codes and the protocols. Right now, we do this manually, which is extremely time-consuming. We thus strongly recommend to use an automatic tool for this purpose, and we are currently looking for suitable tools.

With our lab kits, homework is naturally suited for distance learning settings, since students can work anywhere. Electronic submission should not be a problem either, but in this case we should also provide some means for automated program verification, both to relieve the tutors from the load of having to check the correctness of all student programs and to allow the students to check the correctness of their programs prior to submission. To allow students to get feedback from their tutors on their submitted work, tutors should be available via Webcam to discuss the program and the protocol. So students
can submit their work anytime and can get feedback from their tutor in his or her next (virtual) lab hour.

V. DISCUSSION

This year’s course was on the whole well received by our students. Student favorites, in descending order, were the quality and friendliness of our tutors, the take-home lab kits, the interesting and diverse hardware, the possibility to get bonus points for additional work, the large exercise set with its varied exercises (and its availability from the start of the course), and the flexible grading system.

Students criticized the fact that we do programming exams, their all-or-nothing nature, and that we demand two completed programming exam tasks for a positive grade.

Suggestions for improving future courses included additional and more complex hardware, and to be able to take home all the hardware (the motor boards were only available in the lab this year).

We do of course try to improve criticized areas, but cannot do much about our programming exams. Neither do we really want to change our procedures. After all, our exam tasks are kept fairly simple to ensure that prepared students have sufficient time, and the tutors are allowed to help to prevent students from failing just because of a simple bug they were too nervous to find. Our impression is that the students take the exams too lightly and do not appreciate the fact that just getting a homework exercise to run with the help of the tutor and a lot of trial-and-error does not make them proficient microcontroller programmers. An unfortunate aspect of our on-target programming exams is that students are generally convinced that they were “just five more minutes” away from a correct solution, which makes them feel the lack of partial credit all the more\(^3\). We believe that we can overcome this problem with better exam preparation, and thus in the future will ask our tutors to assess the students’ current level of knowledge regularly.

Since the homework is beneficial to students, we allow it even though plagiarism may occur. Of course, we try to identify plagiarism, and in future courses will emphasize more why plagiarism is detrimental to performance. A study conducted on patterns of plagiarism [13] indicates that plagiarism is mostly done by weaker students, possibly because of problems with handling the work. We hope that regular supervision and support by the tutor can help weaker students overcome their problems without resorting to plagiarism.

To improve our motivational impact without too much financial backlash, we strove to shift as much supervision load as possible to tutors. It turns out that tutors can handle the bulk of the student supervision, as long as they in turn are well supported. Here, we still lack a few tools that would help tutors a great deal, foremost a tool to manage the student data (student name, progress, homework submissions, submitted voluntary work, …). We are currently working on such a tool in our SCDL project and expect it to be ready by next year. The tool should also be able to handle electronic submissions and check for plagiarism.

It is also important to prepare tutors properly for their work. They must of course be trained on the exercise material, but they must also be made aware of their social duties. For instance, they must understand why they should monitor the performance of their students and play an active role in their development. We must also call to their attention the problems of plagiarism and their obligation to try and prevent such things among their students. This is of particular importance because tutors, as senior students, are liable to have “helped” their own friends one time or another, and are thus vulnerable to turning a blind eye when encountering such activities among their own students. So care must be taken to explain to tutors why plagiarism hurts the person that plagiarizes.

We never really addressed these issues in former years, but always chose tutors that we believed were liable to have a good impact on students. By actively making tutors aware of their social role and explaining their impact on student motivation and grades, we hope to improve tutor motivation even more.

Since student satisfaction with our course is very good, student retention is quite high (70%). Of the students who remained in the course, 73% passed. These are acceptable values, but we hope to increase these numbers even more by improving the quality of support. The hardware in its current form is well suited to convey the course contents. To increase the fun aspect of our course, we plan to introduce a line-following robot into the third part. It will be used to teach motor control and power-efficient programming issues.

Although we are currently not conducting the course in a distance learning setting, we could easily do so. Due to the lab kits, homework can be done at home. All course material is contained on the Knoppix CD that is part of the lab kit, so students have all information they need at home. The supervised lab hour can easily be offered via Webcam. The theory exams are already conducted via Internet, and since the exam supervisor does not have to help students, any exam room and supervisor would suffice as long as it is guaranteed that students cannot get illegal help during the exam. The programming exams currently need skilled supervisors to help with debugging problems, but this might also be achievable with a Webcam, so as soon as our test system to automatically evaluate the correctness of student solutions is available, the programming exams can be conducted via Internet as well.

Although due to some hardware problems we do not yet have experience with the electronic lectures, some students already told us that they think it is a great idea. We will try to make each lecture as self-sustaining as possible, complete with indications on which other lectures it relies, to allow other courses to use our lectures as necessary. In turn, we believe it

\(^3\)It is interesting to note that in the cases where we followed up such claims, it always turned out that the student solution had one or more serious errors and was far from being complete.
would be useful to generate a faculty-wide database of such lectures, possibly even with exam or self-assessment questions about the material, so that instructors can easily assemble material students should brush up on for their course.

VI. CONCLUSION

The course in its current form is successful, student satisfaction is high. Our course structure has proved effective, students like the take-home lab kits, the flexible grading scheme, and the homework exercises. The course can easily be exported to other universities or can even be held in a distance learning setting.

To keep the course scalable and efficient, we employ a hierarchical structure for supervision, with 1 tutor per 15 students and 1 TA for all tutors. Lectures are provided electronically, so after the setup phase of the course, the professor only has to handle the course organization and assign the final grades.

Final scores are composed of programming exam results, theory results, light compulsory homework, and voluntary bonus homework. A certain minimum score on the programming exams is necessary to attain a positive grade to prevent unprepared students to pass by plagiarizing their homework.

There are still some open tasks which can make the course more self-supporting. First of all, we intend to expand our set of electronic lectures to include video introductions to the hardware, and generally introductions to all subjects students have problems with (e.g., C programming). Second, we will continue to develop our automatic test system, which is currently only available as a prototype, and work on the web cam approach so we can eventually conduct all exams remotely and without trained supervisors on site. We are also currently looking for suitable plagiarism detectors to strengthen the value of our homework exercises, which can in theory be exploited by lazy students to get a better grade. Finally, we need a suitable course management tool that also supports tutors in their work.

REFERENCES

A System for Automatic Testing of Embedded Software in Undergraduate Study Exercises

Voin Legourski, Christian Trödhandl, and Bettina Weiss
Vienna University of Technology,
Embedded Computing Systems Group E182-2,
Treitlstr. 3/2, 1040 Vienna, Austria
Email: legourski@yahoo.com, {troedhandl,bw}@ecs.tuwien.ac.at

Abstract—As student numbers in embedded systems lab courses increase, it becomes more and more time-consuming to verify the correctness of their homework and exam programs. Automatic verification can vastly improve the speed and quality of such tests. This paper describes a system that can carry out black-box tests to verify whether the embedded software running on a target system meets predefined requirements. To this aim, we employ a special test board using an ATmega128 microcontroller which is connected to both the target system and to a host computer. Tests can be selected and started remotely, the results are presented to the user on the host. Monitoring and control via Internet is also easily possible. A special meta-language is used to describe the correct behavior of the tested program, and this description is compiled and downloaded to the test system via a standard RS-232 interface, where the test is executed. The same interface is used to control the tests and for transfer of data and end results.

Index Terms—embedded software education, automatic testing, black-box testing

I. INTRODUCTION

Laboratory courses in embedded systems programming are characterized by tight resource constraints and a significant expenditure of time and effort per student. One of the major factors contributing to the time expenditure is the verification of student programs, be they homework or exam solutions. To facilitate this work, we proceeded to develop a test system which allows to verify the functionality of simple embedded software. In a first trial run, we plan to employ the test system in our introductory microcontroller course to automatically verify the correctness of student exam programs. The microcontroller lab course is held once every year and is attended by 120 to 150 undergraduate students. It teaches the basics of microcontroller programming, such as the usage of timers, a/d converters, UARTs, and so on in assembler and C language on simple 8 bit, 16 bit, or 32 bit microcontrollers, and is thus particularly well suited for a first trial. In the long run, the tool will be used to automatically verify student programs in several different embedded systems courses.

This work is part of the FIT-IT project SCDL “Seamless Campus: Distance Labs”1, which aims to develop a remote teaching concept for embedded systems laboratory courses. This concept incorporates the required web infrastructure and, depending on the requirements of the course, customized learning packages and remote workplaces. The distant goal of the project is to provide a remote learning environment that can be used for most of the embedded systems courses at the Vienna University of Technology. This will enable us to increase our training capacity and to better support working or handicapped students.

There are other projects similar to the SCDL project, with the goal to extend the education in embedded systems in a way to meet the requirements of the constantly growing embedded systems industry [1]. Such projects are described in [2], [3], [4]. In the past years, many systems have been developed which can test embedded software and hardware in different ways. For instance, testing by fault-injection is performed by the tools described in [5] and [6]. The object-oriented scenario-based test framework shown in [7] places its emphasis upon test re-usability in case of product families or new product versions. In contrast to our solution, which performs black-box functional tests, their approach requires knowledge of the underlying software structure of the tested system.

Our test system employs a special language to describe tests and expected results. There are several other projects that use special languages to test embedded systems. For example, SystemC is a language that defines abstract models of software and hardware components, and [8] explores the possibilities of using SystemC for testing embedded systems.

The system described in [9] is a generic test equipment for embedded software. It is based on the C++ interpreter CINT and adopts the C++ language and its features. Another design of a generic test and maintenance node for embedded system testing is the subject of [10]. It is built on the IEEE 1149.1 standard test bus and is designed to provide a cheap solution for the design-for-test and built-in-test overhead cost problem.

Our own test system [11] is executed on an ATmega128 microcontroller and is programmed and controlled via a Linux host computer. The basic idea of our system is to supply the tested target system with input signals on its I/O pins and to evaluate its responses in order to compare them with predefined patterns. Hence, we do black-box tests only and are mainly interested in the timing of the response signals.

1 The “Seamless Campus: Distance Labs” project received support from the Austrian “FIT-IT Embedded systems” initiative, funded by the Austrian Ministry for Traffic, Innovation and Technology (BMVIT) and managed by the Austrian Research Promotion Agency (FFG) under grant 808210. See http://www.ecs.tuwien.ac.at/Projects/SCDL/ for further information.
The tests and expected results are defined in a special meta-language, which in practice allows any type of program behavior to be tested. The test system performs and evaluates the tests automatically and provides reports to a human operator on a local or remote computer. New tests can easily be loaded to the system and executed.

Compared to the other works cited above, our approach corresponds well to the goals of an embedded software test system for the education. In contrast, [7], [9] are well suited for testing high-level functionalities and expect high-level communication of some kind in order to perform tests, and [8] is based on abstract modeling.

In the following paper, we first give an overview of our test system in Section II, then proceed to describe the system implementation in more detail in Section III. Section IV shows the system in action with a small example, Section V discusses the usefulness of the system, and Section VI lists some possible extensions. Section VII concludes the paper.

II. Test System Overview

Resources for undergraduate studies are rather limited – in obligatory courses, one or two faculty members have to handle up to 150 students, supported only by about ten tutors. Yet, embedded systems programming skills are best conveyed by letting students program a lot, so many such student programs have to be evaluated. In the Microcontroller laboratory course that will be used for a trial run, for example, students have to solve about 10-15 programming exercises during the term, may voluntarily submit up to another 15 programs to improve their grade, and additionally have to attend exams. The course offers three practical exams, and students need to do at least one of these exams to pass, but may also come to all three exams to further improve their grade. On the average, about 120 students attend these exams. Each exam consists of two simple programming tasks which the students have to solve in the given time. Students work on the hardware during these exams, so they can try out their solutions and debug their programs. Whenever a student believes he or she is finished, (s)he can submit the solution.

In our experience, much of the time spent per student in undergraduate embedded programming courses is invested into correcting homeworks and exams, so it makes sense to look into means of automating this process. Since our undergraduate courses generally tend to use simple exercises to teach the basic concepts, automatically verifying the correctness of programs appeared to be an achievable goal.

We wanted our test system to be versatile and to be useful to a wide variety of courses, ranging from FPGA programming over different microcontrollers to industrial automation with SPS hardware. So simulators or code instrumentation were no viable options. Instead, we focus on the interface of the control unit to the hardware, in a microcontroller’s case its I/O pins, and use black-box tests to verify program functionality and timing. Of course, these tests have their limitations, but they are the most flexible solution as far as the supported target system hardware is concerned.

Our test hardware consists of the test system itself, the tested board, which we will call the target system in the remainder of the paper, the host computer, and the connections between them. The whole structure is shown in Figure 1.

Currently, the test system is a universal ATmega128 board, which was developed in the Department of Computer Engineering at Vienna University of Technology. In addition to the microcontroller, a digital to analog converter is used in order to extend the functionality of the tests and to support analog test signals. There are four types of signals that can be connected to the target system board: Digital signals, analog signals, RS-232, and the TWI (I2C) interface.

The ATmega128 microcontroller executes a C program, the interpreter, whose main function is the interpretation and execution of the actual test program instructions, which are written in a special meta-language. The interpreter is also responsible for the communication with the host computer and the loading of new tests. Note that due to the interpreter, the test program itself is system independent. It can be executed on any hardware that provides the interpreter. So although we are currently using the ATmega128, we could easily change the system by porting the interpreter.

Two software tools are located on the host computer: The meta-language translator, which translates test descriptions into “executable” code, and the mc Programmer tool, which can download the tests to the test system and manipulate them. The results from the test evaluation as well as all messages are received at the host computer via the standard serial port in text format.

The language developed to describe the test behavior is referred to as “meta-language”. It is an assembler-type language whose purpose it is to facilitate the writing of test programs and to make them more understandable. It offers a set of instructions including arithmetical and logical operations, memory manipulations, compare and jump instructions, time measurement, analog and digital signals I/O, TWI and UART communication interfaces, and test control instructions. Currently, the meta-language is the top-level interface to the
test system.

III. IMPLEMENTATION

A. Hardware

As we already mentioned, the test system is implemented on a 5V Atmel ATmega128 microcontroller, which is driven by a 16 MHz clock. An external digital to analog converter, the DAC6574 of Texas Instruments, is connected via a TWI interface. Figure 2 shows a block schematic of the test system hardware and its available connections.

Fig. 2. Test system block schematic

32 digital inputs and outputs are available for the digital signals. Additionally, up to 14 other pins could be used for digital signals if their primary functions are not required. These are the analog inputs, the two UART interfaces, and the TWI interface. However, in order to prevent or at least minimize changes to the connections between the test system and the target system, it is recommended that the special function pins not be used for digital I/O.

In addition to the digital I/O, the ATmega128 has 8 analog input pins with 10 bit resolution. A resolution of 8 bit is also supported by the test system. The external d/a converter is controlled via the TWI (I^2C) serial bus of the ATmega128 and provides 4 analog outputs with 10 bit resolution. Again, an 8 bit resolution is supported as well. More converters could be attached to the bus as necessary, and the test system currently supports two DACs and thus 8 analog outputs. Note that the TWI interface can also be used to test TWI code on the target system, independently of the function of the DAC module.

One of the two UART interfaces of the test system is used for communication with the host computer. Over this interface, the user can control the tests, and data can be sent back to the host. The second UART interface can be used to test UART code running on the target system.

B. Connections

Our test system can dynamically assign the functionality of a pin. So if, for example, a certain pin PB0 of the target system is used to output a PWM signal in one test and utilized as an input from a switch in another, the test system can handle this change in functionality without requiring a change in connections. Hence, it is of no particular importance how the digital I/O pins of the target system are connected to the test system.

There could be one problem when connecting the test system to the target system, and this pertains to the analog pins of the target system, which may either be used as digital I/O or as analog I/O. In the case of digital I/O, they should be connected to digital I/O pins of the test system. If the target requires analog inputs, however, the pins must be connected to the outputs of the external d/a converter. Different tests may entail different functionality of these pins, thus necessitating a change in the connections.

In this version of the test system we do not have any built-in solution for this problem, so currently the user has to reconnect the pins if the change in functionality is required. In a future version, we will solve this problem with an external multiplexer, which can of course in turn be controlled by the test system, see Figure 3. This technique allows to dynamically switch between testing analog input functionality and digital I/O (or analog output). Of course, it requires one more pin to control the multiplexer, so it might be sensible to control several analog inputs/outputs of the test system simultaneously if the extra pins cannot be afforded.

Fig. 3. Connecting more than one test signal

Note that in most cases the problem will not occur anyway, since normally the connection between hardware and the control unit are fixed, so the pin functionalities will remain the same over all exercises and exam tasks that use this hardware. For courses that do not use such statically assigned connections, it may still be feasible to formulate program specifications that keep these problematic pins fixed. So this problem does not pose much of a limitation on the test system in its current form, and a couple of such multiplexers will be sufficient to support the few pins that for whatever reasons must remain flexible.

Another aspect we do not currently address is the problem of different power supplies for target and test system. If the target system happens to work with different voltage levels,
level converters must be employed to bridge the gap between the target and the 5V levels of the test system.

C. Software

The software of the test systems consists of the translator tool and the programmer tool on the host, and the interpreter on the test system itself, see Figure 4.

![Software structure of the test system](47)

The test description is written in the meta-language and is translated to a test program that can be executed by the interpreter on the test system. The programmer tool is responsible for uploading the test program to the test system, and for controlling the tests.

The test program itself consists of one or more phases, which can be individually controlled by the programmer. Each phase tests a particular aspect of the target program. Among the things that can be tested are the timing of signals, timing relations between two signals, signal states, or analog outputs. To perform the tests, the test system can generate diverse stimuli ranging from simple digital or analog signals over changing analog values to more complex actions like UART communication.

1) Meta-language and Translator: The meta-language is a full image of the instructions implemented on the test system, so it resembles an assembler language with an emphasis on the functionality required for the tests. Additionally, some higher-level constructs are implemented. Contrary to [9], our meta-language is built entirely on the test functionality and not on an existing programming language. It is also not object-oriented as are [7] and [9].

Each test description can contain up to three sections. The most commonly used section is the #PROGRAM part, which contains the sequence of instructions which form a test program. #VARIABLES contains pairs of addresses and sequences of values which are to be stored starting at the given address. The #PHASES section contains one or more groups of addresses and phase numbers.

Our meta-language also supports labels. Each label points to the address of the next instruction and does not need to be previously declared to be used in the code. Labels can for example be used in instructions which control the program flow, like jump instructions. It does not matter whether the instruction expects relative or absolute addresses. Another application where labels are useful is in the definition and subsequent call of subroutines.

Definitions can be used to associate a variable name with a number in the range of 0 to 255. The value of the definition cannot be changed once assigned, so it only allows the usage of definitions as constants.

As we have already indicated, our meta-language supports the definition and use of subroutines. They can be placed on any address, usually at the end of the test program. Each subroutine must start with a label and end with the return instruction. Parameters can be passed only implicitly.

The translator tool is used to transform programs in the meta-language into a hex-code that is ready to be transferred to the test system. The translator should not be referred to as a compiler, because no special high-level language constructions are translated. One of the basic features that the translator provides is the transformation of complex instruction parameters into a tightly packed form, which is thus not a responsibility of the test developer. The parameters are packed in spaces of less than 1 byte, so that one parameter byte can contain one, two or more values relevant to the instruction.

2) Programmer Tool: The mc parser programmer tool realizes the communication between the host PC and the test system. It can download test programs or variable sets to the system, and can start or stop a test. It accepts a hex-file as input. The hex-file consists of hex-numbers as they are to be uploaded into the memory of the test system, and of the directives (#PROGRAM, #VARIABLES, #PHASES) which tell the programmer what to do with the data following it. The hex-file must begin with a directive.

The upload to the test system is done through the serial port of the host computer.

3) Interpreter: The test system executes an instruction interpreter which interprets the instructions that describe the test behavior. All data required by the interpreter, such as the size of the last uploaded test program and a pointer to the currently executed instruction, are stored in global variables. The test description itself is stored in the microcontroller’s 4kB SRAM.

Every test description consists of several instructions. Most instructions are 2 bytes wide (1 byte for the instruction code, 1 byte for the parameters), with the exception of some extended instructions, which are 4 bytes in length and represent functionalities which either cannot be coded in two bytes or which are an extension of the corresponding 2 byte instruction for the purpose of optimization. Extensions perform a functionality
in one instruction which would otherwise take two or more two-byte instructions. This leads to faster program execution, because the instruction decoding time is roughly the same as the instruction execution time.

A set of local variables can be used for the purposes of the test process. They can be initialized during upload of the test program. The memory size for the variables is configurable and is currently set to 512 bytes. Paging is used in order to allow the addressing of all variables. All instructions which use 8 bit addresses refer to the active page, paging itself is controlled by special instructions.

The first 16 variables can also be addressed with a special 4-bit addressing mode. This type of addressing is provided so that the remaining 4 bits in the instruction word are free for other parameters. For example, the instruction test_bit 3, 13 (in hex: 0x06 0x3D) tests the 3rd bit of the variable at address 13 (the 14th variable). The result is implicitly stored in variable 0. The 16 variables are not affected by paging, so our example would have the same effect no matter which page is currently active. This addressing mode thus provides an easily reachable set of 16 variables and can be compared to the concept of registers in a microprocessor. Variable 0 is used as an accumulator, which means that it is used as an implicit parameter of some instructions or stores the results of an operation.

A special array contains values of user-defined timeouts. These timeouts define the test system reaction in case of instructions which depend on external signals, like for instance time measurement instructions or instructions that wait for a bit change. This prevents hang-ups in case of unexpected or missing signal patterns. We can provide timeouts in the range of 1 μs up to over a minute.

An important issue is the addressing of the digital and analog pins of the test system. To this aim, all of the microcontroller’s registers can be accessed by using the instructions load_acc_reg and store_acc_reg. The names of all I/O registers of the microcontroller are defined and can be used in the test descriptions. Instructions for manipulating separate bits of a register are also available. Thus, all digital I/O signals of the ATmega128 can be manipulated through their ports’ registers. The analog I/O pins as well as the special signals like TWI and UART can be manipulated through special instructions, depending on the function. Please refer to [11] for a detailed description.

The test system offers two types of time measurement: Common blocking mode realized through polling, and input capture mode, which works in parallel to the test execution. Time measurement through polling can be performed by every pin of the ATmega128. It blocks the test program execution until the measurement is finished or a timeout occurs, so it is for example not possible to check the timing of two such signals at the same time. Time measurement through input capture can only be performed by some pins of the ATmega128, but has the distinct advantage that the measurement is interrupt driven. Hence, it is possible to check the timing of two signals at the same time. In both cases, the time is measured in milliseconds, microseconds or in processor clock cycles.

The test system can also perform wait loops. Wait loops are for example useful when a time delay has to be implemented in the test program, or when the test needs an external event (signal change) to continue. Note that the value of the waiting period is stored in a variable and thus does not have to be set in advance, but can be dynamically computed during the test execution.

A variant of the wait loop is implemented by the wait-until-condition instructions. Depending on the parameters, these instructions wait for a positive or negative edge on a particular pin. The test execution is blocked until either the bit change takes place, or until a previously defined timeout occurs.

IV. Example

This section shows a practical example in the form of a simple program executed on an ATmega16 target system. The program should wait for the press of a button and then start to generate a PWM signal with a given period and ratio. A timing diagram of the desired behavior of the tested target system is shown in Figure 5.

![Fig. 5. Expected time behavior of the tested target](image)

The following connections are required between the target and the test system:

\[
\begin{align*}
\text{ATmega128} & \rightarrow \text{ATmega16} \\
\text{PA0} & \rightarrow \text{PA0} \\
\text{PB0} & \rightarrow \text{PB0}
\end{align*}
\]

The positive edge of PA0 is the start condition for the ATmega16. The output is on PB0. The width of the PWM signal is 240μs for LOW and 220μs for HIGH. The response signal on PB0 should be tested with an accuracy of 5μs.

To test the behavior of the target, we must write the test description using the meta-language. The program consists of several parts, which are described below.

The test program begins with the #PROGRAM directive, after which the addresses of some variables are declared. For instance, var2=2 shows that the name var2 refers to the variable at address 2.

```c
#define
// definitions
```
acc=0 // accumulator
var10=10
var2=2
var6_32=6
var6_32_LW=8
var0_32=6
var0_32_LW=8
pin0=0
low_duration=12
high_duration=4
accuracy=14

The first instructions initialize pins PA0 and PB0, which are used later on.

begin: // label; defines start of the program
  // INIT
  cbi_reg ADDR_DDRB pin0 // PB0 as input
  sbi_reg ADDR_PORTB pin0 // PB0 -> pull-up
  cbi_reg ADDR_PORTA pin0 // PA0 -> 0
  cbi_reg ADDR_DDRA pin0 // PA0 -> output
  // initialize timeout
  set_timeout 0 ms var10

Then the signal that is expected by the tested target program is produced, and the test system checks whether the response has come within the predefined response time. If not, the test program executes the jump instruction and outputs an error message.

// START
  // apply positive edge on PA0
  sbi_reg ADDR_PORTA 0 // sbi PA0
  wait_us_imm 50 // wait for about 50us (response time)
  // test for response
  load_acc_reg ADDR_PINB
  test_bit_var 0 acc // test bit 0 of acc
  if0_var acc // no response
  jump_rel_imm error_no_response

In the next section, the duration of the low pulse is measured. The first negative edge starts an internal counter and the measured value is obtained with the next positive edge. The measure_pulse_width instruction provides a resolution of the ATmega128 clock cycle. Two possible errors are checked here: Whether a timeout has occurred during the measurement, or whether the measured value does not match the requirements within the bounds set by the desired measurement accuracy. In case of an error a corresponding jump is performed.

// MEASURE duration of the LOW pulse
  use_timeout 0 // define use of timeout
  // pulse width in us, 0=LOW pulse, 1=one pulse, on
  // pin0; store into var6_32 (32 bits), of IO port PINB
  measure_pulse_width us 0 1 pin0 var6_32 ADDR_PINB
  // CHECK whether timeout has occurred
  if_flags flag_timeout
  jump_rel_imm errorTimeout
  memory_transfer 2 0 var6 var6_32_LW
    // save measured time in var6
  operation_var16_var sub var6_32_LW low_duration
    // var6_32_LW = low_duration
  comp_var16 var6_32_LW accuracy
    // compare difference to the stored accuracy
  if_flags
    jump_rel_imm error_low

Similar instructions are used to measure the duration of the high pulse. If the test has been successful, no jump has been performed and the next section is executed. It outputs a message to the computer and terminates the test.

// TESTING DONE! EVALUATE AND PRINT RESULTS

Now all that is left is the reaction for all error cases. Each one begins with a label and ends with a termination of the test. User messages are passed as well.

// PROGRAM TERMINATED WITH SUCCESS
  error_low: // similar to error_low

  error_timeout:
    sbi_reg ADDR_PORTA 0 // turn off PA0
    pc_printstr 0x80 // message "LOW pulse duration=
    pc_printdec16 var2 // print duration value
    pc_printstr 0x80 // message accuracy=
    pc_printdec16 accuracy
    pc_printstr 0x80 // message "US"
    pc_printstr 0x60 // new line
    stop_test 0 1 // if here - test success

This concludes the program part of the test description. The next section contains no instructions, but defines initial values for the program variables, as well as the user messages. After each #VARIABLES directive, we place first an address and then constants that should be stored at this location. This is done more than once, since each message begins at a different location.

// UPLOAD PREDEFINED CONSTANTS
#VARIABLES
0x0000 // start address 0
0 0 0 0
0 0 0 0
0 220 // high_duration must value
0 0 0 0
0 10 // var10 contains timeout
0 240 // low_duration must value
0 5 // accuracy
#VARIABLES
0x0210 // start address 0x210
17 'O' 'W' 'P' 'U' 'L' 'S' 'E' 'D' duration = 0
0 0 0 0
0 0 0 0
0 0 0 0
0 10 // var10 contains timeout
0 240 // low_duration must value
0 5 // accuracy
#VARIABLES
0x0550 // start address 0x550
13 10 0 // new line
#VARIABLES
In our test description, both automatic messages and texts defined by the test developer are returned via the serial interface. The following list shows the resulting output on the host PC in some important cases.

- No response within the predefined response time is recognized:
  
  ```plaintext
  STARTING COMPLETE TEST
  no response!
  TEST FAILED!
  ```

- The test is successful, the time properties of the measured signal meet the predefined expectations, the response time meets the requirements as well:
  
  ```plaintext
  STARTING COMPLETE TEST
  accuracy=+-5us
  TEST OK!
  ```

- The low pulse duration is not correct, considering the predefined accuracy:
  
  ```plaintext
  STARTING COMPLETE TEST
  LOW pulse duration=249
  must value=240
  accuracy=+-5us
  TEST FAILED!
  ```

- A timeout is reached while measuring the pulse width. This can be the case when the pulse is too long or if the signal does not change at all:
  
  ```plaintext
  STARTING COMPLETE TEST
  timeout!
  TEST FAILED!
  ```

This test description occupies 146 bytes of the memory of the test system. Additionally, 16 bytes are used for variables and constants, and 93 bytes contain the user-defined messages.

Of course, the tool is also capable of processing more sophisticated tests. By using conditional instructions, logical operations, and timeouts, various types of state machines could be implemented and therefore the complexity of the test scenario is only limited by the memory size.

### V. Discussion

The tool should facilitate program submission for both students and instructors: From the students’ point of view, the tool allows them to submit (and thus check) their programs at any time and get instant feedback. In the case of homework, this makes remote submissions with feedback possible. During exams, students can check the correctness of their programs at any time, without the need to call and wait for a supervisor. If the instructor desires, the tool can also give feedback on the nature of problems in case of failed submissions, although such features should probably be used with care to prevent students from employing a mindless trial-and-error strategy.

From the instructor’s point of view, the tool saves a lot of time otherwise spent on manually checking student programs, which generally is a tedious task. It also eliminates human error from the verification process, allows to conduct exams in a distance learning setting, i.e., at a remote location without qualified supervisors on site, and scales to a large number of students.

These features are bought with the additional effort necessary to generate automated tests. Here, exam programs reap the highest benefit from the tool, since they tend to be fairly simple and thus the correctness tests are easily constructed. As a rule of thumb, we estimate that the time it takes to write the test for an exam task is approximately equal to the time it takes to set up the task description and program its solution. So doing automated tests roughly doubles the time required to set up the exam. However, verifying a student solution by hand certainly takes at least ten times as long as doing it automatically, and this factor increases with increasing task complexity. Hence, even a moderate amount of students already makes automated testing economic. Add to that the time-independent advantages of no room for errors, getting rid of a tedious labor, and the advantages for students, and we feel that automated tests are justified even for smaller student numbers like 15-20 students.

Homework exercises pose more problems since they tend to be more complex. Furthermore, homework is generally not only about writing a correct program, but also about writing a good (well-designed) program. Since the test system does only a black-box test, it cannot be used to verify whether the student has employed special features of the target system, like using the timer to automatically generate a PWM signal, and it can of course not be used to evaluate higher-level issues like programming style or efficiency. Still, it does free the instructors from the rather tedious, uninteresting, and error-prone task of checking the correctness of a large number of programs and leaves more time to concentrate on high-level issues like programming style. Furthermore, the test system allows students to remotely submit their homework with instant feedback on the correctness, which will certainly be appreciated. So here it will certainly be worthwhile to investigate methods to generate test programs more easily, as we will elaborate in the next section.

As we have seen in the previous section, the result of a test is either that the test has completed successfully within the given constraints, or that an error has occurred. It is the responsibility of the test programmer to specify what output message the test system should print in case of errors, so detailed information about the nature of the failure is possible. Therefore, the test system can be used both for binary tests (program works/failed) and for more fine-grained evaluation and can thus also be used to assign partial credit (e.g., award k% of the full points if the program has successfully passed k% of the test).

### VI. Possible Extensions

As one can already see from the example, the test system is well capable of handling timing relations. However, setting up the test description is currently done by implementing a test program in the meta-language. Although not particularly
difficult, it is still time-consuming, so an obvious extension of the system would be to provide a high-level translator that can directly take timing diagrams as input and translate them into an appropriate test description. Such a tool would greatly facilitate the design of test descriptions. Of course, in case of a failed test, the system should still be able to tell the user which part of the test failed. But since we only check the timing of signals, it should be relatively easy to automatically generate test cases for all errors that can occur during a test.

As we have mentioned, the test system is intended for checking both exam programs and homework programs. Exam programs tend to be relatively simple, so the current memory space is more than sufficient for our purposes. If more complex homework is tested, however, it might be necessary to extend the test system’s SRAM. To allow completely automatic testing of diverse programs, we should also add some multiplexers for the analog pins to our target hardware.

It might sometimes be interesting not only to test the external signals, but also to verify internal information like register initializations of the target microcontroller. This can for example be useful to check whether a student has programmed the PWM generator of the target microcontroller as demanded in the exercise task, or is simply using a manually tuned busy-wait loop to achieve the same effect. If such additional checks are desired, the course instructor has to provide some additional (target-dependent) code which is linked to the student code and which verifies register initializations. This code could then use some free pins of the target system to communicate its results to our test system, which could in turn incorporate this additional information into its tests. No change in the test system is required to use such additional data provided by the target.

Of course, the meta-language can still be enhanced by adding complementary functionality to some instructions. For example, the duty cycle of a digital signal could be measured in one instruction and thus one would not have to measure the positive and negative pulse durations separately. Or, we could add some special types of comparison instructions which already compare with a certain accuracy. This would be useful when determining whether a measured time or an analog value is in the required interval.

VII. CONCLUSION AND FUTURE WORK

The test system described in this paper is intended to automatically check undergraduate student exam programs and homework exercises in the area of embedded systems programming for correctness, that is, in compliance with a given specification. To this aim, we do a black-box test on the I/O pins of the target system, for example a microcontroller. We provide the microcontroller with stimuli, thus simulating the behavior of hardware input elements like switches, and check the output responses of the microcontroller in both time and value domain against the program specification. The output of the system can be used both for simple binary grading (passed/failed), but also for assigning partial credit for partial functionality.

The tool will allow us to increase the cost efficiency of our teaching activities, since faculty staff is relieved from the monotonous task of correcting more than hundred program examples per exam. Additionally, the system enables us to increase the number of students and to devise courses where exams can be held at other locations than our own university. Students benefit from the tool as well, since they can submit their programs anytime and from any location, can expect fast and correct verification of their programs, and can also get feedback in case of errors. Therefore we are planning to try out this system in our next year’s microcontroller laboratory course, and in the long run intend to employ the tool in all undergraduate courses on embedded systems.

The system is currently available as a prototype. We are now working on enhancements of the basic functionality, most notably on a good and easy to use interface for developing test descriptions.

REFERENCES

Abstract—I describe a two-year-old embedded systems design course I teach at Columbia University. In it, the students learn low-level C programming and VHDL coding to design and implement a project of their own choosing. The students implement their projects using Xilinx FPGAs and tools running on Linux workstations.

The main challenges the students face are understanding and complying with complex and often poorly-documented interfaces and protocols, personal time management, and teamwork. While all real-world challenges, this class is often the first time the students encounter them, which makes the class quite challenging, but very practical.

In this paper, I describe the structure of the class, the configuration of our teaching laboratory, some of the more successful projects, and give suggestions to instructors wishing to implement the class elsewhere.

I. INTRODUCTION

Embedded system design is a challenging problem that represents the future of digital system design. Moore’s law and the relentless downward-spiraling cost of integrated circuits has made it possible to price very powerful computing artifacts at consumer levels, as the ubiquity of devices such as DVD players, digital cameras, and cell phones attest.

We in academia must follow this trend. As recently as fifteen years ago, wire-wrapping TTL parts was a sufficient introduction to state-of-the-art system design techniques; today’s systems are orders of magnitude more complex. Teaching students to handle this complexity is the central challenge.

In this paper, I describe an FPGA-based embedded systems course I developed and teach at Columbia University. Following the suggestion of Frank Vahid, whose book [1] I used for the first year, I created this course to replace a microprocessor system design course that until 2003, had students assemble systems using Z80-based trainers with breadboards.

I wanted the students to learn hardware/software codesign, specifically the design of microprocessor systems and their peripherals. Students taking the course have experience with Java and C, know basic digital design, and have taken computer organization, which should have familiarized them with assembly language, but few have any experience integrating these skills.

The class focuses on the design of a fairly complex hardware/software system, which the students implement on an FPGA board. This is consistent with the role of the course as a “capstone lab” in the Columbia Electrical Engineering and Computer Engineering programs. As I describe below, the first half of the class consists of cookbook-style lab assignments that teach the students the design tools. The second half of the class is devoted to the project.

As of September 2005, I have taught the class three times: twice at Columbia, and once at National Chaio Tung University (NCTU) in Hsinchu, Taiwan. At Columbia, the majority of students are fourth-year undergraduates completing an Electrical Engineering or Computer Engineering degree. The Taiwan group consisted of sixteen students, most of whom were Master’s-level, and many of whom had worked or were working full-time in the electronics industry.

II. HARDWARE

Just as embedded hardware/software systems can take many forms, there are many possible vehicles for teaching a practical embedded systems class. At one point, I considered having the students only build simulations, a practical approach used in computer architecture courses, but I wanted the students to experience real hardware.

Edwards is supported by an NSF CAREER award, an award from the SRC, and by New York State’s NYSTAR program.

1To emphasize their antiquity, these trainers were labeled “Copyright 1985.”
One standard approach is to use a microprocessor development board. Frank Vahid has taken this approach with his course at University of California, Riverside [1], using the 8051. While this is a practical option, it tends to lead to software-centric thinking that does not consider hardware/software trade-offs.

Modern field-programmable gate arrays (FPGAs) offer many advantages for instruction, including flexibility, fast reprogrammability, and the capacity to implement large, fast digital designs. The two leading FPGA companies—Xilinx and Altera—offer comparable technology. Xilinx, however, appears to have the superior university program, so I decided to use Xilinx FPGAs for this course.

Many FPGA development boards are available, but most are designed for industrial use (and budgets) and can cost as much as US $5000 per board—beyond our price range.

At Columbia, my TA and I selected a board built by XESS Corporation: the XSB–300E (Figure 1a), which sells for about $900. Centered around a Xilinx Spartan IIE FPGA—the XC2S300E—with a raw capacity of roughly 300k gates, the board also has a wide variety of peripheral chips, including video input and output; Ethernet; USB 2.0; a serial port; SRAM, DRAM, and flash memory; and an audio CODEC. The peripherals were particularly attractive; using these peripherals would be a focus of the projects.

In Taiwan, we used a newer, smaller board made by Digilent (the Spartan 3 starter kit board, Figure 1b). This was satisfactory, especially given its $120 price, but greatly limited the range of projects as its peripherals are limited to an 8-color VGA port, a PS/2 keyboard interface, and an RS-232 port. This particular board comes in a few different configurations. The most common has an XC3S200 part, but this is too small to accommodate the Microblaze soft processor core, which every project has used. Instead, we paid a bit more and got boards with the larger XC3S400 part.

### III. TEXTBOOKS

I have not found a satisfactory text for the class. In the first year, I used Vahid and Givargis [1], partially because Vahid had originally suggested the idea of the class to me and because the text embodies the philosophy of functions being implementable in either hardware or software. But the students and I found the book disappointing. It deliberately shies away from talking about any particular languages or platforms, making it useless as reference manual. The background material and ideas it contains are good, but the students do not find it terribly relevant to the project construction task.

Many texts are software-centric. Wolf [2], for example, discusses things such as the ARM instruction set and operating-system-level concepts such as processes. Simon [3] is even more focused on software, although a bit more practical than Wolf. Heath [4] is similar. Brown [5] has a more industrial focus and includes a large avionics example.

Many books are specific to a particular processor. Although practical, I find such an approach focuses too much on the idiosyncrasies of a particular instruction set. Books in this vein include Morton [6], which focuses on the Motorola 68HC11 series of microcontrollers. Lewis [7] targets the x86 architecture and ordinary PCs, which makes acquiring suitable hardware easy: most departments have a collection of old PCs that make suitable embedded targets. Barr [8] chooses an 80188-based board, although focuses mostly on C and C++. Pont’s book [9] should have been titled Embedded C on the 8051. Peatman [10], by contrast, makes it clear that he focuses on the PIC18F452 processor. Incidentally, is is the only book I know of that comes with a (bare) PC board.

There is another family of texts that are more concept-oriented and targeted at graduate students. These are even more abstract that Vahid and Givargis and would probably frustrate my students. Examples include Gajski et al. [11], Jantsch [12], Marwedel [13], and the volume edited by Jerraya et al [14]. Noergaard’s sprawling volume [15] tries to discuss just about everything, ranging from the difference between enhancement- and depletion-mode MOSFETs to HTTP methods. While a very interesting reference, it is too long to consider in its entirety in a single semester.

### IV. LAB ASSIGNMENTS

As I mentioned above, I divide the class into two: during the first half, the students follow cookbook lab assignments meant to teach them how to use the design tools. During the second half, they design and implement projects of their own devising.

Figure 2 lists the six labs I have given over the last two years and the main language they need to use in each. The goal of these labs was to get the students familiar with using the tools through a sort of tutorial style. I provided detailed explanations of what to do as well as collections of files as a starting point. The results were mixed.

In the spring of 2004, I made the mistake of trying to balance software and hardware labs, making half of them...
software-dominated; the rest hardware. While this does reflect the class focus I had in mind, it was not well-matched to the students' backgrounds, which were heavily software-centric. I found myself teaching experienced programmers who were able to complete the first three labs with almost no effort at all. But digital design with VHDL stumped them: they did not have any real experience designing digital circuits, despite having taken a beginning digital design class. They were also flummoxed by the odd syntax of VHDL. Many of them resorted to trying to write VHDL as if it were C.

In the spring of 2005, I made the labs more hardware-centric. Again, the first two gave the students experience in low-level C programming and some experience with the tools, but the rest of the labs were mostly about design with VHDL.

Most students, when introduced to synthesizable VHDL, treat it as a programming language, but it is more a textual form of coding schematics and state machines. VHDL “statements” such as if-then-else and assignments are deceptive: they only provide a way of decomposing a function and do not behave like the imperative versions the students are familiar with. To try to help them avoid this error, I tried to emphasize a particular design methodology.

Two concepts distinguish hardware from software: structure and timing. While software has structure in the form of functions and classes, the structure in hardware is at a block-diagram level, reflecting its concurrent nature. Similarly, the software programming style is to ignore performance concerns until absolutely necessary and only concentrate on functionality, a technique that does not work in hardware. As a result, I teach the students a three-step hardware design process: draw a block diagram, such as the one for the text-mode video controller in Figure 3, draw a timing diagram (e.g., Figure 4), and then code it in VHDL. To get them started, the third lab in 2005 had them do this in reverse: we provided them with a clearly written VHDL description and asked them to draw the block diagram and a timing diagram for it.

The biggest challenge the students faced while doing the projects in 2004 was dealing with existing protocols such as the bus protocol spoken by the Microblaze soft processor (i.e., the OPB protocol) or the protocol spoken by the audio codec. To try to address this, the three hardware labs in 2005 were protocol-centric. The first was fairly easy: building a controller that would sum the contents of a small on-chip memory. Students had to understand the (very simple) interface to the memory, design a simple data path with controlling state machine, and understand how to use the VHDL simulator.

The second hardware lab involved interfacing with the OPB. I had students design and implement a simple memory-mapped peripheral that performed complex multiplication. We supplied the students with a combinational (one-cycle) $8 \times 8$ multiplier and asked them to construct a simple datapath and controller that used the multiplier four times to compute the product of two complex numbers.

The final hardware lab was the most complicated, although still far from what the students would have to do while implementing their projects: an interface for an off-chip static RAM part. This is a typical problem: interfacing one protocol with another—in this case, the protocol of the OPB and the protocol of the static RAM. To keep things simple, I had them only map half of each 32-bit processor word to match the 16-bit width of the SRAM chip. This would allow them to store data in the memory, but not execute code from it since the processor needs all 32 bits.

These labs definitely worked better in the second year, but there remains room for improvement. While all groups managed to complete the labs successfully, many seemed to forget the lessons they taught when doing the project. I often found myself answering questions with “we did that in lab four,” which was disappointing. Furthermore, it remains the case that the students need more practice at hardware design and debugging the mess they have created.

At Columbia, the course spans a normal fourteen-week semester; in Taiwan, it was condensed to a single month in which the class met daily. To accommodate the tight schedule, I omitted the sixth lab assignment and scaled down the scope of the projects.
V. THE PROJECT

In my experience, students prefer working on projects of their own devising rather than what I could supply. As an example, when I taught the compilers class at Columbia for the first time, I had the students implement the Tiger language from Appel’s Modern Compilers book, and the students hated it. The next year, I had the students design and implement a language of their own devising—a much more difficult procedure, but the students clearly enjoyed it far more.

For the projects, I break the class up into groups of between two and six students. A size of three seems optimal—any smaller and the project becomes too simple, any larger and the group starts to lose its cohesion and ability to communicate.

Overall, about 80% of groups have completed the project, meaning they have something working at the end that closely resembles what they set out to do. The remaining 20% have difficulties with group dynamics (e.g., the members hate each other), technical difficulties (one group spent their time trying to communicate with the USB controller, to no avail), or are just incompetent. The good students at Columbia are excellent; the bad students are awful.

I ask the project groups for four deliverables: a two-paragraph project proposal, a project design document, a demonstration on “75% day,” and the final demonstration and report. Such deadlines are absolutely necessary to keep the students moving as otherwise they would work on other classes’ shorter deadlines. Even so, four deadlines seem like it may not be enough; I plan to add a “50% day” next year.

I expect a reasonable project to incorporate both software (C) and custom hardware (VHDL) and interface with at least one of the on-board (but off-chip) peripherals on the XESS board. Interfacing with one of the peripherals is relatively straightforward, but using two or more is difficult because of the odd shared bus structure of the XSB–300E board, which connects all of the peripherals to a common set of pins on the FPGA. Thus, to communicate with multiple peripherals, the students must build a controller that behaves differently depending on the peripheral being accessed (each has very different timing requirements), yet does so through a common set of pins. Without question, this is one of the most awkward aspects of the XSB–300E board.

Students usually start by proposing overly ambitious projects (at least in the US—the Taiwanese students were much more realistic, but this may have been because many were professionals). My teaching assistants and I have had to curtail countless proposals that incorporate MPEG encoding, a complete TCP/IP implementation, or other systems that are orders of magnitude more difficult than beginning students could realistically implement in half a term.

Below, I describe the majority of the successful projects students have completed over the last two years. Broadly, they fall into four classes: video, audio, networking, and “other.” The majority of projects focus on one of these areas, but some of the more ambitious and successful projects incorporate, say, both video and networking.

A. Video Projects

I am a fan of video-centric projects, having built some as an undergraduate. They have certain advantages: they are visually satisfying when they work; they can often be debugged by inspecting the displayed image; they have substantial, but not insurmountable, real-time requirements; and VGA-style video signals are a simple protocol whose central idea (a raster image) is fundamental. To support video development, each workstation in our lab has two flat-panel displays: one connected to a Linux workstation; the other connected to the XSB–300E board and its video DAC.

Video games Simple video games make for excellent projects. Students have implemented games inspired by Pac-man, Scorched Earth, and a 3D maze game. For the Pac-man-like maze game, students designed and implemented a custom video generator capable of drawing sprites over a character display, much like the original Namco arcade game. The game logic, implemented in C, was primitive, but I was more concerned with their implementation of the custom hardware and the hardware/software interface.

Scorched Earth is an artillery game in which players take turns lobbing shells at their opponents’ tanks over a 2D terrain. The students implemented custom graphics hardware that superimposed sprites for the tanks and shells over a terrain generator (each column has a height that corresponds to the line at which the sky ends and the ground begins) and a character generator for displaying the current score, gun inclination, and so forth. Again, the game logic was simple but successful. This project was the star of 2005; most students wanted to play it.

One group implemented a 3D maze game. I had them create a custom video controller that contained two numbers for each X coordinate: one that corresponded to the line at which the sky begins and the wall starts; the other that holds the line at which the wall ends and the floor begins. The students used a primitive raycasting technique to determine these numbers: from the player’s position, they sent out a ray that goes until it hits a wall. The distance the ray travels indicates the size of the wall at that column (more precisely, it is the reciprocal of the distance). The students found it challenging to do this calculation using fixed-point arithmetic (the Microblaze does not have a hardware floating-point unit), but were ultimately able to achieve nearly a 20 fps frame rate.

Since we used the simpler Digilent Spartan 3 Starter Kit Board (Figure 1b) at NCTU in Taiwan, the range of projects the students could build was greatly restricted. I suggested they build simple video games and most groups did.

Chess Rather than spend time on the algorithm for playing chess, this group built a two-player chess game that could display the chessboard, let a player select a piece to move, show where it could be moved, and move it. They even implemented such complicated rules as pawn promotion and castling. As with most of the NCTU projects, this group adapted the video display code I provided (based on Figures 3 and 4) to display the board and pieces.
The four other NCTU video game projects were Tetris, Sokoban, a scrolling maze game loosely patterned on the Namco game Rally-X, and a two-player snake game. Each group modified the text-mode video controller code I had provided, adding color and changing the size of the characters.

Video Effects Processor Modern FPGAs have enough processing power to perform limited real-time image processing. One group put this to use by building a framebuffer with the ability to distort its output. Rather than simply reading the contents of memory in sequence for adjacent pixels, they added the ability to change the starting point and memory stride for each line. Such a set-up was able to transform, say, a rectangular picture into a triangular one, and be able to modify this distortion on-the-fly. The group had originally intended to perform this distortion on real-time video (the XSB–300E has a Philips video decoder chip), but ran out of time and only displayed the static contents of memory.

Digital Picture Frame This project, which was unsuccessful in 2005, decodes JPEG images and displays them on the screen. The easiest way is to perform most of the computation in software and only use hardware for the framebuffer. The XSB board includes a Compact Flash interface (a parallel bus protocol), so in theory it would be possible to read and display files from a digital camera, but no group has been successful at being able to read from a CF card.

One group attempted to port the independent JPEG library onto the Microblaze in the process of performing this project, but ran into serious size and complexity problems. In the future, I will advise any group that undertakes this project to write their JPEG decoding code from scratch and not worry about making it support all JPEG variants. There is also an obvious opportunity for hardware acceleration (i.e., the inner loop of the DCT).

Another group, this one in Taiwan, also attempted the digital picture frame project, this time with greater success. First, they implemented a frame buffer that used external SRAM (there is only 32K of on-chip block RAM on the XC3S400 used on the Digilent board) and had to grapple with the usual problem of simultaneous access from both the video system and the processor. They took the simplest route and added a mode bit that would blank the display and enable to processor to access it. Next, they took a small JPEG library written by Pierre Guerrier and made it compile on the Microblaze. Finally, they added a mechanism for copying data from the RS232 port into memory in preparation for decompressing and displaying it. Unfortunately, little of this worked completely at the end of the class. The JPEG library, for example, just barely fit in the 16K of on-chip memory they were using.

Video Input Projects These are quite a bit more challenging than video generation projects. First of all, the video DAC is a much simpler chip than the Philips SAA7114H video decoder, which has a 140-page manual. Second, it is much easier to generate a signal than to understand one, especially when it comes from the real-world.

One group implemented a stereo depth extractor using the video decoding capabilities of the XSB–300E board. They pointed a video camera at a mirror nearly parallel to the centerline of the camera to generate a split image from two slightly different vantage points. They then looked for the two brightest spots on the image and used the difference in their location to compute the 3D location of the spot. To test this, they placed the camera in a black cardboard box and shined a laser pointer at a movable target. Although clearly not at the cutting edge of computer vision, the group was able to get interesting results.

Robot with Vision Perhaps the most unique project to date, this incorporated the XSB–300E board as the controller for a mobile robot built using Lego Mindstorms. In the end able to follow a black line drawn on a white piece of paper, the most unique aspect of this project was the successful use of video as vision. The group mounted a small video camera to the Mindstorms robot and fed the input to the XSB–300E board. The board decoded the video, posterize it to one bit per pixel, divided the image into nine rectangular regions, and used the relative number of black pixels in each region to decide whether to turn left, right, or to go straight.

Software running on the Microblaze analyzed the heavily decimated video input signal and transmitted simple commands through a serial port to an IR tower and the robot itself, whose controller was running a very simple program that took simple direction commands.

B. Audio Projects

Like video projects, audio projects engage the senses and therefore share some of the thrills of success and easy debugging of video projects. Compared to video, however, audio is nearly three orders of magnitude slower and one-dimensional, making it much easier to manipulate and presenting many more opportunities for elaborate signal processing.

The audio CODEC on the XSB–300E (an AKM AK4565: 50 kHz, 20 bits/sample, stereo) has a synchronous serial interface with a fairly simple protocol, although its configuration protocol, which goes through a separate synchronous serial interface that is, unfortunately, connected to the low-order data bits on the XSB–300E peripheral bus, was difficult for most of the students.

MIDI Synthesizer One of the most successful projects of 2004, a MIDI synthesizer leads to a nice combination of hardware and software. While it would be possible to perform the sound synthesis in software, its real-time requirements are sufficiently demanding and its computational complexity makes it simple enough to do in hardware. This group implemented both the standard FM synthesis algorithm and the Karplus-Strong plucked instrument algorithm. Both sounded remarkably good.

MIDI is an asynchronous serial protocol like RS-232, but at an unusual bit rate and generally transmitted through a current-loop designed to be terminated with an optoisolator to avoid noise. The group built a simple MIDI-to-RS-232 level

---

2Their source was http://www.es.ele.tue.nl/~minnoc/c_prog/djpeg_orig/
converter and used the soft UART core supplied by Xilinx to receive the protocol.

The MIDI protocol consists mostly of note-on and note-off messages. While fairly simple, managing polyphony with a finite number of oscillators is much easier to do in software, which the group did. Thus, the MIDI protocol was decoded in software and the synthesis was done in hardware.

Sound Effects Synthesizers FPGAs now have more than enough processing power to perform fairly complex audio-band signal processing. At least two groups have taken advantage of this by implementing various sound effect generators. One, for example, was designed to implement various effects, such as phasing and distortion, that worked well with input from an electric guitar. The algorithms for such filters are fairly straightforward; the groups implemented them in hardware and placed their parameters under software control.

Audio Spectrum Analyzer I was a little surprised by the speed of the FPGA for this project: one group implemented a real-time 1024-point FFT running at audio speeds. The majority of the algorithm was in software; they only used hardware to accelerate complex multiplication. Coupled with a nifty graphic equalizer-like display, this was an impressive project.

Pitch Detection Detecting the fundamental pitch of an audio signal, such as a voice, is a fairly interesting problem with applications to singer training. Two groups have attempted projects along these lines, with varying results. One group did the obvious and performed an FFT on audio input samples, but found that the linear bin arrangement of the FFT made for rather imprecise measurement at low frequencies. Another group attempted to implement an algorithm based on autocorrelation and got so far as to build a prototype in Matlab, but did not complete the project because of group dynamics.

C. Networking Projects

The XSB–300E contains an NE2000-compatible Ethernet chip, and a number of projects have used it for network communication. By sheer numbers, students seem to prefer audio and video projects, but the successful network projects have been quite impressive.

Internet Camera This project, which combined video and networking components, spoke the most protocols of any project to date. The students used the Philips video decoder chip to sample real-time video, decimate its resolution and frame rate, packetize it, and send it as UDP packets over Ethernet. On the receiving end—a standard Apple laptop—a simple Java program of their own devising received the packets and displayed them. I joked that the project amounted to a very expensive wire, but it was actually one of the most technically challenging projects that exemplified good engineering: it made something quite complicated look effortless.

Internet Audio In 2005, two groups built projects that communicated audio over the Internet. One built an Internet radio broadcaster that took audio in through the CODEC, packetized it, and sent it out via RTP. They connected an iPod and an Ethernet cable to the board and were able to listen to the iPod through the speakers on the Linux-based workstation running a standard mplayer program. Like the Internet camera project, this one went to pains to speak many standard protocols and ultimately made something very complicated look simple.

A similar project took on an even more complex protocol: SIP. Designed for Internet telephony, SIP is a standard protocol for establishing Voice-over-IP telephone calls. I was amazed when this group was able to hook up the board to the campus Ethernet network and make a friend’s VoIP phone ring. The final result was a little disappointing—the audio quality was poor, which I attributed to some sloppy programming somewhere—but overall the project was a success.

D. Unique Projects

All the projects I described above used the XSB–300E board and its FPGA at the center. By design, in this class I have not focused on the electrical and physical challenges of embedded system design, but a number of groups decided to tackle these problems, too. The result has been a largely successful group of unique projects.

Automotive Projects Columbia participates in the Society of Automotive Engineers’ Formula SAE competition, in which student groups design, fabricate, and race small cars built around motorcycle engines. While largely targeted at mechanical and automotive engineers, two groups in my class have done projects related to this effort.

The first, in 2004, built a vehicle telemetry system that gathered data from various sensors on the car (e.g., tachometer, oil pressure) and sent it through a wireless link. The group purchased the wireless transmitters and receivers, but built a small processor system on the car centered around a PIC microcontroller. This was a particularly challenging project for this class because it involved a number of analog signal conditioning circuits. While ultimately mostly successful, the group did have the problem of relying on the car itself—which was often unable to start—to demonstrate their project.

The 2005 group built a digital dashboard and controller for an automatic shifter. They worked with a pair of mechanical engineers who designed the mount and levers for a large (25A) solenoid that moved the mechanical gear shift lever (the gears on this motorcycle engine could be selected by moving a lever—designed to be operated by foot—up and down). The group also built a dashboard with LEDs displaying engine RPMs, the current gear, and a suggestion about whether to shift up or down. As is typical of beginning electrical engineering students, their fabrication skills were lacking, resulting in a tangled mess of wires and cold solder joints, exactly the sort of problems using an exclusively FPGA-based approach avoids. Alas, this group, too, was stymied by the engine not starting when they tried to demonstrate it to me, although it had been working earlier.

Scrabble Timer When the class started, an inventor happened to approach me about building a prototype of a timer, much like a chess timer, for the game of Scrabble—he had aims of selling the design to a large board game manufacturer.
The students who worked on such independent projects, information they needed from tutorials and web references. Programming in class, but the students were able to glean instruction from me. I do not discuss microcontroller understanding this class.”

I have put all the class materials on the web, including all slides, handouts, lab assignments, lab template files, and students’ project files and reports. All can be found at http://www.cs.columbia.edu/~sedwards/classes.html.

I have not been able to address to my satisfaction the balance between practical knowledge and fundamental understanding. There is a plethora of practical knowledge the students need to be able to execute their projects, ranging from VHDL coding styles to how to get the Xilinx tools to work, but I feel the class is overly demanding of knowledge of such trivia. Frustratingly, the students actively complain that I spend too little time lecturing about such details. While this is very realistic and class is meant to be a capstone class in which students bring together the knowledge they have gained during their careers as undergraduates, it still frustrates me that I feel they are learning trivia that will be out-of-date in a year.

VI. Evaluation

As with all Columbia classes, students were asked to fill out a standard course evaluation form. The results are summarized in Figure 5. While the results are positive overall, and the ratings improved uniformly in the second year, the main negative complaints were that the course required a lot of work (which is certainly true) and that my lectures were not that relevant to the labs and project. Specifically, they wanted much more instruction on how to use the CAD tools, which are very complicated. I had attempted to address this issue through detailed lab writeups, which described the operation of the tools in detail, but some students obviously prefer to be shown something.

VII. Conclusions

The embedded systems class I have described remains a work-in-progress, but has been fairly successful. The word-of-mouth among students has been excellent, to the point where it has clearly siphoned off students from other, competing classes. An informal poll suggests students prefer the mix of hardware and software and the ability to choose their projects.

I have not been able to address to my satisfaction the balance between practical knowledge and fundamental understanding. There is a plethora of practical knowledge the students need to be able to execute their projects, ranging from VHDL coding styles to how to get the Xilinx tools to work, but I feel the class is overly demanding of knowledge of such trivia. Frustratingly, the students actively complain that I spend too little time lecturing about such details. While this is very realistic and class is meant to be a capstone class in which students bring together the knowledge they have gained during their careers as undergraduates, it still frustrates me that I feel they are learning trivia that will be out-of-date in a year.

The more fundamental ideas I see in practical embedded system design—the balance between top-down and bottom-up design necessary to build high-performance systems, the ability to debug, the ability to seek and find the information you need, and the ability to understand and reverse-engineer poorly-written documentation—are subtle, difficult to convey, and not the sort of thing you can easily ask on a test. My hope is that these more subtle ideas will enter in the students’ thinking during the process of implementing these complex projects, because it is unlikely they will learn them from a lecture or a book.

References


<table>
<thead>
<tr>
<th>Question</th>
<th>2004</th>
<th>2005</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount Learned (out of five)</td>
<td>3.72</td>
<td>4.04</td>
</tr>
<tr>
<td>Appropriateness of Workload</td>
<td>3.33</td>
<td>3.64</td>
</tr>
<tr>
<td>Overall Quality</td>
<td>3.74</td>
<td>3.89</td>
</tr>
</tbody>
</table>

Selected comments from 2005:

“Tough class but learned a great deal. Recommended.”

“I’d like to see a lecture that goes into more detail about the way that the various files definitions and programs are used to create the hardware. We end up learning it in pieces but a more detailed overview would be useful since the tools are a key component of understanding this class.”

“The lectures didn’t seem to serve as much help for the assignments and project.”

Fig. 5. Course evaluation results for the Columbia classes. Numbers are averages, with 0=poor and 5=excellent.
Abstract—The VMEbus is an IEEE standard architecture upon which many embedded and real-time systems are built. It has existed for nearly 25 years and has been extensively used for military, industrial, and aerospace applications. This paper describes the general characteristics of the VMEbus architecture, specifically relating these characteristics to aspects of embedded systems education included as components of the IEEE/ACM CE2004 computer engineering model curriculum. Portions of this model curriculum are currently being implemented at universities across the country as part of an increasing effort to address the need for embedded systems education. This evaluation will help to identify the strengths and weaknesses of this architecture as a general-purpose embedded systems educational tool.

Index Terms—Computer architecture, computer engineering education, educational technology, embedded systems.

I. INTRODUCTION

Embedded systems are becoming more and more common as feature size and power requirements of electronic components decrease and overall performance improves. As a result, more and more graduates are entering careers working directly with the design and implementation of embedded systems [1]. In order to prepare graduates for the challenges in this field, many universities are incorporating embedded systems concepts into their computer engineering curriculum. However, embedded systems education is difficult to generalize due to the following:

- Embedded systems education incorporates a broad range of concepts from many disciplines;
- Different programs have different goals associated with their embedded systems curricula;
- There exists a diverse range of embedded applications;
- There are many different technologies used in embedded systems;
- There is a wide range of design constraints imposed on embedded systems; and
- There are many different embedded systems architectures and platforms in use today.

Because of these characteristics, embedded systems education is fragmented and customized from one university to another. In an attempt to provide curricular guidelines for education in this field, the IEEE/ACM CE2004 computer engineering model curriculum for embedded systems was created [2]. This model provides a comprehensive list of recommended topics that are important for embedded systems education. Universities are challenged to integrate these topics into the classroom and to create laboratory platforms capable of supporting these topics.

In this paper, we evaluate a popular embedded systems architecture to determine its suitability as a general-purpose laboratory platform for embedded systems education. This architecture, the VMEbus, was developed nearly 25 years ago and has been used extensively in industrial, military, aerospace, communication, and control applications. Despite its age, it is still one of the most popular architectures for embedded and real-time systems accounting for 38% of the embedded systems market in 2002 [3,4]. The emphasis here will be to evaluate the VMEbus architecture against the model curriculum as opposed to comparing various embedded architectures.

The remainder of this paper is organized as follows. Section II provides a general description of the VMEbus architecture. Section III evaluates how the architecture supports many of the components of the model curriculum. Conclusions are presented in section IV.

II. THE VMEBUS ARCHITECTURE

A. Background

The VMEbus is a system architecture consisting of the electrical specifications for a communication bus and the mechanical specifications describing the backplane, bus connectors, board sizes and formats, as well as card cages, racks, and enclosures. VME stands for Versa Module
Eurocard and refers to the VERSAbus, the predecessor electrical data bus standard, and the Eurocard, a predecessor mechanical format for computer components. The VME specification was completed in 1982 and is now part of the ANSI/IEEE STD. 1014-1987. This is an open architecture allowing third parties to freely develop VMEbus products [5, 6].

B. General VMEbus Architecture

The VMEbus architecture is a shared system bus architecture. The system bus resides on a backplane. The backplane has slots where processor modules, memory modules, or I/O modules connect to the system bus [6, 7]. The backplane and all of the modules connected to it reside in an enclosure that serves to protect the components, provide structural support for the system, and house utility components such as power supplies and cooling fans. Figure 1 shows one possible system configuration. The modular design of this architecture provides great flexibility. All the components are available commercially-off-the-shelf (COTS).

C. System Bus Characteristics

A functional block diagram of the VME system bus is given in Figure 2, and the basic system bus characteristics are summarized in Table 1. It is beyond the scope of this paper to describe all of the specific characteristics and the timing behavior of the bus. However, Figure 2 and Table 1 provide a broad summary of the basic bus characteristics that will aid the reader during this evaluation. Also, these characteristics are those of the basic VME32 specification. Many extensions to this specification have been created, but for this discussion, the basic design will be used.

![Figure 1. Diagram of the general components and configuration of the VMEbus architecture [8].](image)

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Master/Slave</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfer Mechanism</td>
<td>Asynchronous (no central synchronization clock)</td>
</tr>
<tr>
<td></td>
<td>Fully handshaked</td>
</tr>
<tr>
<td></td>
<td>Non-multiplexed</td>
</tr>
<tr>
<td></td>
<td>Multiplexed for 64-bit transfers</td>
</tr>
<tr>
<td>Addressing Range</td>
<td>16-bit (short I/O)</td>
</tr>
<tr>
<td></td>
<td>24-bit (standard I/O)</td>
</tr>
<tr>
<td></td>
<td>32-bit (extended I/O)</td>
</tr>
<tr>
<td></td>
<td>64-bit (long I/O)</td>
</tr>
<tr>
<td></td>
<td>Address range is selected dynamically</td>
</tr>
<tr>
<td>Datapath Width</td>
<td>8, 16, 24, 32, 64-bit</td>
</tr>
<tr>
<td></td>
<td>Datapath width selected dynamically</td>
</tr>
<tr>
<td>Unaligned Data Transfers</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Compatible with most popular processors</td>
</tr>
<tr>
<td>Error Detection</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Using BERR* (optional)</td>
</tr>
<tr>
<td>Data Transfer Rate</td>
<td>0-80 Mbytes/sec</td>
</tr>
<tr>
<td>Interrupts</td>
<td>7 levels</td>
</tr>
<tr>
<td></td>
<td>Priority interrupt system with STATUS/ID</td>
</tr>
<tr>
<td>Multiprocessing Capability</td>
<td>1-21 processors</td>
</tr>
<tr>
<td></td>
<td>Flexible bus arbitration</td>
</tr>
<tr>
<td>System Diagnostic Capability</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Using SYSFAIL* (optional)</td>
</tr>
<tr>
<td>Mechanical Standard</td>
<td>Single height (160 x 100 mm eurocard)</td>
</tr>
<tr>
<td></td>
<td>Double height (160 x 233 mm eurocard)</td>
</tr>
<tr>
<td></td>
<td>DIN 603-2 connectors</td>
</tr>
</tbody>
</table>
III. THE VMEBUS SUPPORT FOR THE MODEL CURRICULUM

To evaluate the capability of the VMEbus architecture for use as a general-purpose embedded systems educational platform, its characteristics and capabilities will be discussed as they apply to each embedded systems component of the IEEE/ACM model curriculum for computer engineering [2]. The rationale is that if the architecture can address the concepts in the model curriculum it should be able to support the diverse embedded systems curricula being taught across the country.

The embedded systems portion of the model curriculum is broken into eleven areas of which seven are core topics and four are considered to be elective material. In the following subsections, each of these areas is addressed. Each section begins with a list of the topics and learning outcomes from the model for a particular area that can be supported by the VMEbus architecture. How the architecture supports these topics and learning outcomes for each area is then presented.

A. CE-ESY0 History and Overview (core)

**Topics**
- Highlight some people that influenced or contributed to the area of embedded systems;
- Contrast between an embedded system and other computer systems [2].

**Learning Outcomes**
- Identify some contributors to embedded systems and relate their achievements to the knowledge area;
- Describe the meaning of an embedded system;
- Explain the reasons for the importance of embedded systems [2].

The VMEbus architecture holds its own place in the history and evolution of embedded systems. It was developed specifically for embedded applications. It is one of the most frequently used architectures for real-time applications [3] and accounts for 38% of the embedded systems market [4]. Also, the fundamental components of the VMEbus architecture are designed and configured differently than those used in desktop computer systems. The form factor, component
placement, component interaction, and overall system design is different. This makes it easy to introduce embedded systems, compare and contrast them to general-purpose computing systems, and to justify their significance.

**B. CE-ESY1 Embedded Microcontrollers (core)**

**Topics**
- Structure of the basic computer system: CPU, memory, I/O devices on a bus;
- CPU families used in microcontrollers: 4-bit, 8-bit, 16-32-bit;
- Basic I/O devices: timers/counters, GPIO, A/D, D/A;
- Polled I/O vs. interrupt-driven I/O;
- Interrupt structures: vectored and prioritized interrupts;
- Direct memory Access (DMA) transfers;
- Memory management units;
- Memory hierarchies and caches [2].

**Learning Outcomes**
- Understand the CPU in the context of a complete system with I/O and memory;
- Understand how the CPU talks to the outside world through devices;
- Understand how memory system design (caches, memory management) affect program design and performance [2].

The VMEbus architecture design is based upon the most basic of system structures. The modular design and interchangeable characteristics of the system components gives students real system design and integration experience reinforcing the first learning outcome. Also, the asynchronous bus protocol and variable width data transfers make the architecture compatible with many different processor families. This is a key concept since a large percentage of embedded systems rely on technology other than 32-bit processors [9]. The ability to connect a logic analyzer directly to the bus signals also provides a means for students to analyze bus and system timing, to study the asynchronous handshaking protocol in depth, and to investigate bus arbitration and bus saturation.

Since the architecture was originally designed for I/O intensive applications, there are many COTS I/O devices that can be used to teach the I/O topics and achieve the second learning outcome. The asynchronous system bus design allows the architecture to support I/O peripherals of various speeds using either polled I/O or an interrupt-driven design. The VME system bus supports 7 levels of prioritized interrupts and can handle vectored interrupts.

The VME specification also provides flexible memory systems with I/O and memory; the program translation process: compilation, assembly, linking; fundamental concepts of assembly language and linking: labels, address management; compilation tasks: mapping variable to memory, managing data structures, translating control structures, and translating expressions; what can/cannot be controlled through the compiler; when writing assembly language makes sense [2].

The VMEbus architecture supports these topics and learning outcomes through the wide variety of operating systems and software development environments available for use with the architecture. The variety of software available for this architecture is unparalleled. For example, there are at least 103 operating systems known to be ported to this architecture [11]. Compilers and assemblers for many different high-level languages (HLLs), such as C, C++, and FORTRAN, and assembly languages are available depending upon the system components used. This makes it convenient to examine high-level code and its associated lower-level constructs. Applications can be implemented in both a HLL and an assembly language for performance comparisons. Also, the processor modules can be used without an operating system (OS), which is often the case for embedded systems. In this case, students are exposed to rudimentary development environments possibly even entering the binary representation of each instruction directly into memory manually. Students can be exposed to both locally hosted development environments and cross-compiled environments.

**C. CE-ESY2 Embedded Programs (core)**

**Topics**
- The program translation process: compilation, assembly, linking;
- Fundamental concepts of assembly language and linking: labels, address management;
- Compilation tasks: mapping variable to memory, managing data structures, translating control structures, and translating expressions;
- What can/cannot be controlled through the compiler; when writing assembly language makes sense [2].

**Learning outcomes**
- Understand how high-level language programs convert into executable code;
- Know the capabilities and limits of compilers [2].

The VMEbus architecture supports these topics and learning outcomes through the wide variety of operating systems and software development environments available for use with the architecture. The variety of software available for this architecture is unparalleled. For example, there are at least 103 operating systems known to be ported to this architecture [11]. Compilers and assemblers for many different high-level languages (HLLs), such as C, C++, and FORTRAN, and assembly languages are available depending upon the system components used. This makes it convenient to examine high-level code and its associated lower-level constructs. Applications can be implemented in both a HLL and an assembly language for performance comparisons. Also, the processor modules can be used without an operating system (OS), which is often the case for embedded systems. In this case, students are exposed to rudimentary development environments possibly even entering the binary representation of each instruction directly into memory manually. Students can be exposed to both locally hosted development environments and cross-compiled environments.

**D. CE-ESY3 Real-Time Operating Systems (core)**

**Topics**
- Context switching;
- Real-time scheduling concepts;
- Interprocess communication mechanisms [2].

**Learning outcomes**
- Distinguish a real-time operating system (RTOS) from a workstation/server OS;
- Distinguish real-time scheduling from traditional OS scheduling;
- Understand major real-time scheduling policies;
- Understand interprocess communication mechanisms [2].

The VMEbus architecture is one of the most popular
architectures for real-time computing and has been for many years [3]. Many different RTOSs have been ported to many different processor families for this architecture. Some of these RTOSs include Real-Time Linux, VxWorks, QNX, and OS-9. Many of these RTOSs support the POSIX real-time extensions for interprocess communication. Various scheduling strategies and timing issues can be investigated. A VMEbus system using an RTOS can be configured to address an exact subset of these topics and learning outcomes to accommodate different educational strategies. In addition, multiple processor modules in one VMEbus system can be configured to execute different O/Ss. In this way, direct comparisons can be made between an RTOS and a non-real-time O/S executing on the exact same architecture.

E. CE-ESY4 Low Power Computing (core)

Topics
- Sources of energy consumption;
- Instruction-level strategies for power management: function unit management;
- Memory system power consumption: caches, off-chip memory;
- Power consumption with multiple processes;
- System-level power management: deterministic, probabilistic methods [2].

Learning outcomes
- Understand why low-power computing is important;
- Identify sources of energy consumption;
- Identify possibly remedies for energy consumption at various levels of design abstraction [2].

There is no specific support in the VMEbus specification for low power computing. A typical VMEbus enclosure is not remotely powered, thus eliminating the need for strict energy management strategies such as those seen in handheld and remotely deployed embedded systems. However, this does not prevent one from monitoring energy consumption of the system modules such as memory and comparing and contrasting energy usage in single and multiple processor systems to achieve the learning outcomes. Additionally, the asynchronous bus protocol will support both high and low speed processors in the same system. Since reducing clock speed is one primary power management technique, this architecture can provide a platform to make direct comparisons of power consumption for system components having different clock speeds. Finally, custom low power circuitry can be developed on prototyping system modules and interfaced to the VMEbus system for control and monitoring. Controlling custom low power components within an existing system architecture can also help to achieve the learning outcomes suggested in this area of the model curriculum.

F. CE-ESY5 Reliable System Design (core)

Topics
- Transient vs. permanent failures in hardware;
- Sources of errors from software;
- The role of design verification in reliable system design;
- Fault-tolerant techniques [2];

Learning outcomes
- Understand the variety of sources of faults in embedded computing systems;
- Identify strategies to find problems;
- Identify strategies to minimize the effects of problems [2].

To address these concepts and learning outcomes, students using the VMEbus architecture can be exposed to a wide variety of problems ranging from low-level errors in the fundamental bus transactions to debugging software written in high-level languages. Fault tolerance using redundancy can be introduced by including redundant system components. Students have access to both hardware and software on these systems requiring failure strategies and debugging in both domains.

G. CE-ESY6 Design Methodologies (core)

Topics
- Multi-person design projects;
- Designing on-time and on-budget;
- Design reviews;
- Tracking error rates and sources;
- Change management [2].

Learning outcomes
- Understand why real-world projects are not the same as class projects;
- Identify important goals of the methodology;
- Understand the importance of design tracking and documentation [2].

The VMEbus architecture actually supports these design concepts and learning outcomes better than many other educational technologies. The reason for this is that the VMEbus architecture supports design at various levels of abstraction. For example, given a set of modules, students can make meaningful system-level design decisions including the number of processors, the design of the I/O system, and the configuration of memory (distributed, shared, hierarchy characteristics, etc.). Low-level design can be incorporated into a project by having students design logic components capable of interfacing to the VME system bus. Real-world concepts can be addressed by incorporating VMEbus component documentation into projects, thus exposing students to typical technical documentation they are likely to see as a practicing engineer. This tends to reinforce the importance for students to produce accurate documentation of their own systems. The fact that the VMEbus is a real embedded systems architecture can motivate students since they see the project as more than an academic exercise.

H. CE-ESY7 Tool Support (elective)

Topics
- Compilers and programming environments;
• Logic analyzers;
• RTOS tools;
• Power analysis;
• Software management tools;
• Project management tools [2].

Learning outcomes
• Understand the role of hardware and software tools in system design;
• Understand how to use tools to support the methodology [2].

The tools and environments available to VMEbus designers are plentiful and varied in design, objective, and scope. There are many different software tools such as compilers, assemblers, debuggers, development environments, and real-time analysis tools. There are also custom function libraries and drivers tailored to particular system components available from various vendors. For locally hosted systems, software tool availability is dependent upon the OS executing on the processor module. But, this is typically not a limitation since, as previously mentioned, at least 103 OSs have been ported to this architecture [11]. There are also special purpose logic analyzers designed specifically to analyze VME system bus timing. Individual institutions can select particular OSs and tools to tailor their VMEbus system to accommodate their particular curricular goals and objectives. Finally, something that is not directly listed in the model curriculum but does represent a major concept is the development of device drivers [12]. If desired, students can develop drivers for various I/O peripherals using this architecture.

I. CE-ESY8 Embedded Multiprocessors (elective)

Topics
• Importance of multiprocessors as in performance, power, and cost;
• Hardware/software partitioning for single-bus systems;
• More general architectures;
• Platform field programmable gate arrays (FPGAs) as multiprocessors [2].

Learning outcomes
• Understand the use of multiple processors in embedded systems;
• Identify trade-offs between CPUs and hardwired logic in multiprocessors;
• Understand basic design techniques [2].

With the increasing complexity of embedded systems, multiprocessing is becoming a requirement in order to meet performance specifications. This is especially true for real-time applications. One of the strengths of the VME architecture is its multiprocessing capabilities [4]. The modular design of the components makes it easy to create various multiprocessor configurations. For example, all processors can be identical creating a homogeneous system, or each processor can be different, even executing different OSs, to produce a heterogeneous system. Memory configurations are also flexible ranging from completely shared to completely distributed. Thus, the VMEbus architecture can easily be used to teach students general multiprocessor architectures and basic design techniques. FPGA devices integrated into a VMEbus module are available and can be used to teach hardware/software co-design [13], partitioning, trade-off analysis and performance analysis. The use of reconfigurable hardware technology for embedded systems education is common [14], but incorporating it into the VMEbus architecture creates a robust educational platform where this hardware can be integrated with real-time system components. Multiprocessor architectures can be used to teach other topical areas in the model curriculum as well including bus characteristics such as arbitration and saturation. Bus saturation is a key design consideration within multiprocessor architectures. Also, memory characteristics such as mutual exclusion and interprocess communication, as discussed in section III D, are reinforced.

J. CE-ESY9 Networked Embedded Systems (elective)

Topics
• Why networked embedded systems;
• Example networked embedded systems;
• The OSI reference model;
• Types of network fabrics;
• Network performance analysis;
• Basic principles of the Internet protocol;
• Internet-enabled embedded systems [2].

Learning outcomes
• Understand why networks are components of embedded systems;
• Identify roles of hardware and software in networked embedded systems;
• Compare networks designed for embedded computing with Internet networking [2].

The VMEbus architecture can provide insight into networking concepts in several different ways. Since the architecture is common in industrial control applications, it is normal to have VMEbus enclosures physically distributed throughout the factory yet logically connected to promote sharing of resources and provide for centralized monitoring and control. There are many different products available that can logically connect distributed VMEbus systems. Some of these products include reflective memory (shared memory), fiber optics, and various standardized communication protocols like the TCP/IP and the MIL-STD-1553 bus standard. With so many choices, institutions can investigate concepts such as shared memory, dedicated local-area-networks, shared wide-area-networks, different protocols, and network bandwidth and congestion. Comparisons can be made between dedicated networks and Internet networking. Various customized networked systems can be created to achieve the specified learning outcomes.
A/D and D/A conversion are important components of I/O processing and are addressed in that context in section III B. The main topic in this section deals with mixed-signal design. There is no direct support for this within the VMEbus specification. However, mixed-signal design fits well within the VMEbus architecture which was designed for I/O intensive applications. Specifically, interfacing the digital and analog components of the system can be addressed using bus modules which include prototyping areas for custom design. In this way, students can be tasked with interfacing analog components to an existing digital system outside the normal I/O context. The real-time aspects of the architecture provide unlimited possibilities for investigating signal processing with real-time constraints.

IV. CONCLUSIONS

Embedded systems education is rapidly being implemented in electrical and computer engineering programs across the nation. But, it is unlikely that institutions of higher learning will implement a common embedded systems curriculum in the near future. The broad nature of the field and different educational objectives have led to highly customized curricula supported by equally customized laboratory platforms. The IEEE/ACM model curriculum attempts to provide guidelines for the concepts required in this field. Laboratory platforms capable of supporting these concepts can be applied as general-purpose educational tools across many curricula.

In this paper, the VMEbus architecture is evaluated against the model curriculum to determine its suitability as a general-purpose educational platform in this field. This architecture is shown to support many of the concepts in the model curriculum addressing many different educational objectives and learning outcomes. Specifically, there is strong support for topics including historical and introductory concepts, system design, I/O, real-time systems, multiprocessing, memory configurations, networked embedded systems, and software tool support. Other benefits of the use of this architecture include exposing students to a real-world embedded architecture, the wide range of COTS components available supporting various system configurations, and its longevity in the field.

Like all other architectures, it is not perfect. The VMEbus architecture does not address one key concept from the model curriculum, low-power computing. Also, size constraints are not addressed outside the context of the VME form factor. Although size is not specifically part of the model curriculum, system-on-a-chip implementations are critical for many embedded applications including handheld consumer electronics. Finally, VME system components are relatively expensive. This is a serious consideration for educational use.

Despite these weaknesses, the authors feel that the VME architecture represents a powerful, capable embedded systems architecture that is appropriate for general-purpose embedded systems education based upon its strong support for most of the concepts in the model curriculum.

REFERENCES


[11] Portions of this FAQ have been reprinted (with permission) from The VMEbus Handbook, 4th Edition by Wade D. Peterson. VITA 1997. For more details the user is directed to the handbook, or the VMEbus specification(s). Other items have been reprinted from the VITA Journal (with permission) VMEbus FAQ's article series by John Rynearson. Available online: http://www.vita.com/vmefaq/


Diverse Hardware Platforms in Embedded Systems
Lab Courses: A Way to Teach the Differences

Falk Salewski, Dirk Wilking, Stefan Kowalewski

Abstract—Traditional methods for teaching the design of embedded systems usually deal with either a hardware or a software view of the system. In computer science it is mostly the software view. The hardware issues taught mostly deal with CPU based systems only and seldom with reconfigurable hardware. We recommend having a more general view at embedded systems in the way that it is always a programmable hardware platform (CPU based or reconfigurable hardware) which has to be programmed in a suitable programming language. In this context we offer a lab course where students should get familiar with different hardware platforms used in embedded systems. They should solve the same task both with a CPLD and a microcontroller each in order to clarify the differences between the two implementations. In this paper our experiences in this field of embedded systems education are described as well as our plans to continue.

Index Terms—Computer science education, Lab course, Real-time and embedded systems

I. INTRODUCTION

The number of embedded systems is increasing. These days, already more than 98% of all microprocessors are found within embedded systems [3]. Such systems integrate hardware and software components and require developers with skills in both subjects. In this respect an interesting aspect is, which types of hardware are applied preferably in embedded systems. This information should form the basis for the education of hardware and software for embedded systems. Aside from many versions of CPU based systems like microprocessors, microcontrollers (MCU) and digital signal processors (DSP) also two other groups of hardware can be found. The first group consists of reconfigurable hardware as complex programmable logic devices (CPLD) and field programmable logic arrays (FPGA); application specific integrated circuits (ASIC) form the second group. In most embedded system design courses for computer science students only the design of CPU based systems is taught [1], [2]. However, reconfigurable hardware is a good option to improve many embedded applications [4], [8]; an option which students will probably never choose if they do not get in touch with it before. According to the need of education in this field [1], [2], [3] we aim to teach our students the general ideas of different hardware and their corresponding software. In the lab course presented in this paper the differences between CPU based systems and reconfigurable hardware are focused on.

Since reconfigurable hardware (CPLD/FPGA) has blurred the gap between hardware and software programming [1] it has become harder to distinguish between these two areas. This is our reason to recommend dealing with all kinds of "programmable hardware" like hardware whose behavior is defined by software. On the one hand, an MCU is used, which is programmed in classical software (C in this case) as a representative of CPU based hardware. On the other hand, we use a CPLD, which is programmed in a suitable hardware description language (VHDL in this case), as a representative of reconfigurable hardware.

The basic ideas of the differences between these two hardware approaches should be taught in a lecture. This basic knowledge should include the possibilities to implement sequential and parallel structures, possibilities of interfacing internal and external hardware modules, limitations according to the amount of memory and the number of logic cells etc. However, if the students lack the practical experience, most of them would not consider designing a system containing reconfigurable hardware. Thus, we recommend a lab course to become more familiar with these embedded systems. Since we focus on the differences of hardware platforms and their according software, a single task is given that has to be solved successively with the CPLD and the MCU. Thus, a suitable task had to be found which is representative for the field of embedded systems and which could be solved on the basis of both types of hardware. We will deal with this topic in the following chapter. Another important aspect regarding the lab-structuring is how much a priori knowledge can be assumed, which issues can be taught in a short introduction as well as how much the students can teach themselves by using the according material. It is also important which hardware and software is necessary during the lab and how the lab course should be structured. These points will be discussed in chapter 3. Chapter 4 then states the experiences we gained in the first run of our lab course and the most interesting results are presented in chapter 5. Then, chapter 6 states some advantages of this kind of a lab course and lab courses in general. We conclude with chapter 7 and present our plans for future work.

II. ONE TASK FOR DIFFERENT IMPLEMENTATIONS

The task chosen for the lab course should allow presenting as many properties of embedded systems as possible. In our opinion the most important properties are the need to interface with the environment (real time requirements, concurrency) and with external peripherals (e.g. bus driver, memory chips, engine driver). Other important properties are
the restricted resources in memory/logic cells, space, cost and power consumption. The task we chose for our lab course is the implementation of the speed measurement for an automotive prototype vehicle we are designing at our institute for research and teaching purposes [6]. The task comprises three major sub-tasks: first, the actual speed measurement, second, a measurement data processing and third, communication via CAN bus.

The sensor signal for the speed measurement is rectangularly shaped with a frequency proportional to the speed of the according wheel. Four of these signals have to be evaluated concurrently in order to gain new data for all four wheels in a short interval. Maximum values for delay and accuracy are given.

The measured values have to be processed according to a given table in order to fit into a given format of the CAN message. Furthermore an error message should be calculated.

The measured and processed values of the actual speed values have to be sent to the CAN bus as soon as new data is available in a single CAN message. For an easy access to the CAN bus an external stand alone CAN controller (SJA1000) is used which has to be initialized before.

Since this task is a combination of parallel and sequential tasks we believe that it is suited for both implementations. Details of the task can be found in [7].

III. THE LAB COURSE

According to the fact that hands-on work significantly improves a students learning and subsequent retention of material [5], all student’s should have access to suitable development boards. For a lab course consisting of 24 students twelve sets of hardware were needed (groups of two). Since all students should work on both hardware platforms they were split up in two main groups. One group started with MCUs and the other one with CPLDs. This procedure allowed working with six MCU and six CPLD boards. For the CPLD the Xilinx CoolrunnerII CPLD Starter Kit was chosen, for the MCU we used a development board based on the ATMELe ATmega16 8-bit RISC MCU. Both decisions were made according to the following reasons: low price, free development environment and good support with tutorials, news groups etc.

The development environments used are based on freeware only for the following reasons: First we would not be able to afford the high number of licenses. The second and even more important point was, that students this way could use the environment at home in order to become familiar with it. The simulator, which was available in both cases, even allows the debugging of example implementations without having the actual hardware.

For the access to the CAN bus additional boards with CAN controllers and the according bus interface were needed. We designed boards, which can be connected to the MCU (5V device) and to the CPLD (3,3V device), and provided 6 of them. We also provided 6 simple frequency synthesizers for testing the velocity measurement (simulating the sensor signals). Therefore, two groups had to share one CAN board and one frequency synthesizer for debugging.

For the participation in this lab course we recommend visiting the lectures Introduction to Embedded Systems and Embedded Software Design offered by our institute. The first lecture deals with the basic properties of embedded systems and provides an introduction for microcontrollers and programmable logic controllers. We are planning to add basics of Programmable Logic Design in the future. The second lecture deals with development processes and methods for software for embedded systems. This includes requirements engineering (functional and non-functional requirements) and architecture design and analysis. In addition to this lectures we offered a two day introductory course in the week before the actual lab course had started. On the first day, the students were introduced to the programming of MCUs in C. On the second day, an introduction was given in the programming of CPLDs in VHDL. Since most of the students were not familiar with
VHDL, an introduction to the basics of this language was given as well. At the end of each day the same example (traffic light with user interface) had to be implemented on the actual hardware in order to give a first impression of the differences. During the introductory course we tried to clarify the important properties and techniques for MCUs (e.g. Interrupts, I/O ports, internal peripherals like timer, hardware specific C commands, debugging) and CPLDs (e.g. parallel/sequential structures, I/O ports, VHDL, debugging). Data sheets for all hardware, the introduction slides and a document with VHDL basics were handed out to each group on a CD.

The actual lab course took place on 13 appointments for 3h/week. We had to offer more appointments for some students who had not completed their task after the 13 dates yet; however some of the students also finished earlier. Since there had been no dedicated room available for the lab course, the hardware equipment and all cabling had to be set up for each day. Therefore, for installation and unistallation about 15min each were needed.

During the lab course all CAN boards were connected to a single bus, which in turn was connected to a PC based CAN bus monitor. On the monitor, the students could check if they were sending their messages correctly. The general function of the velocity measurement could be tested with the simple frequency synthesizers. For testing the accuracy of the sent values a waveform generator was available. The successful communication between the MCU/CPLD and the CAN controller could be checked by a special LED on the CAN board.

The students had to organize their work on their own and to find the necessary information in the according data sheets. However, at least one instructor was available all the time to answer upcoming questions as well as to help if necessary. Most of the answers given to one group during or after the lab were made available via Email to the other students the following day.

At the end of each implementation an acceptance test was done in order to check the basic functionality. Each group also had to hand in a documentation of their final versions (CPLD and MCU).

IV. EXPERIENCE WITH LAB COURSE

The first lab course has been finished successfully with 26 students by now. Almost all of the students managed to pass the final acceptance test with both of their hardware implementations. During the introductory course most of the students learned very enthusiastically and showed strong motivation for teaching themselves the necessary additional knowledge. This knowledge included how to work with the integrated timers (MCU), the hardware specific parts of the programming language C (MCU) and the basics of the programming language VHDL (CPLD). For both implementations the access to the external CAN controller (SJA1000) and its correct use had to be understood.

According to the high number of possibilities in the field of the SJA1000 initialization many problems have occurred in this context. Wrong initialization was hard to detect since students had difficulties in separating the problems they had in the software part of the communication (wrong values in initialization, incorrect timing in write/read function, incorrect order of write/read accesses) and the hardware part of the communication (problems with bidirectional communication, incorrect jumper settings, incorrect connections in general). Thus we are planning to provide more detailed information about the use of the SJA1000 and its initialization in the next lab.

One of the major problems that occurred with the implementation on the CPLD were the restricted resources. Many students had problems fitting their design on the chip. In most of the cases this problem could be solved by adapting the calculations done in the design (reducing size of variables, simplifying calculations). However, in some cases the problem remained and thus we are planning to use bigger FPGAs (much more logic available) for the next term. We also realized that some students were not able to implement a useful combination of parallel and sequential structures successfully in VHDL. Thus, more information how to solve these problems has to be given in the introductory course next term.

Another problem concerned the CPU time needed for a compile run (synthesize, translate and fit) for the CPLDs. It turned out to be much higher than the one needed for the MCUs (minutes instead of seconds). Thus, we will try to improve the performance of our computers for the next term.

According to the longer compile runs and the less familiar software and hardware the groups which started with CPLDs needed longer to finish their first implementation. Thus, we had to provide some extra time for this implementation. This problem was solved by shifting the date of changing the groups by one week and by offering an extra date. However, we are planning to use more hardware sets in the next term to avoid the problem according to the need of changing the hardware in the middle of the course.

At the end of the course, all of the students had to fill out two questionnaires in order to evaluate the lab course. The first one is part of the university teaching evaluation and the second has been developed at our institute to improve the contents of the lab course itself. Some of the results are presented in the following chapter.

V. RESULTS

In this chapter, the most interesting results of the evaluation will be presented. One of these results concerns the question of how much the students have improved their knowledge in the field of programming MCUs, CPLDs and interfacing external peripherals. According to the results presented in figure 1 most improvements could have been gained in CPLD and peripheral programming. However, a small number of students stated that they had learned only very little in this field. Another interesting point is the amount of external help the students needed in order to solve their task (figure 3). In the case of the CPLD significantly more external help was needed. This result corresponds well with figure 4 in which the amount of code
developed at home is presented. In the case of the MCU about 76% of the students never have programmed at home, in the case of the CPLD it only has been 32%. Figure 5 provides some additional information about the CPLD programming done at home with respect to the fact if the groups started with the MCU or the CPLD.

Those who started with the CPLD did much more CPLD programming at home than the group who started with the MCU. This could be interpreted in two different ways. First, the groups who had just started could have been more motivated and have thus worked more at home. Second, the first steps to solve the given task (understand the general ideas of CAN communication, speed measurement, etc.) were harder to realize in the case of the CPLD. In the next term an according question will be added in the questionnaire to clarify this point.

Another aspect we will have to look closer at in the next term is presented in figure 6. According to this figure some students (20%) have to improve their ability to distribute the work within their group. Finally, the students were asked how useful the knowledge gained in this lab course is to solve future tasks in this field. The result presented in figure 7 represents in our opinion a good result for this first lab course. Since many results indicated that the programming of the CPLD needed more effort we will try to reduce this difference by giving a more detailed introductory course in the next term.

VI. ADVANTAGES OF DIVERSE PLATFORMS IN EMBEDDED SYSTEMS LAB COURSES

Lab courses always involve a lot of work before and during the actual course, especially when a lot of hardware is used. However, we believe that they provide the best way to teach the practical aspects of embedded systems. Especially the use of diverse platforms turned out to be an ideal way to clarify the differences between the different platforms used in embedded systems. In our view, the results gained from this first lab course confirm this position. Additionally, a lab course could be used for collecting empirical data on many aspects of the embedded systems domain. In our case all versions of software compiled during the lab course were collected. We are planning to analyze these data in order to receive information about the differences between CPLDs and MCUs according to non-functional qualities as reliability, safety, maintainability and changeability.

VII. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

In this paper, a way to teach embedded system design with focus on the difference of diverse hardware platforms like CPLDs and MCUs and their according languages was proposed. In this context, the same task had to be solved on a CPLD and a MCU each by every group in order to demonstrate the differences. The biggest problem that occurred during the lab was the inappropriate use of parallel and sequential structures, the limited resources in the case of the CPLD and the problems in understanding the data sheet of the SJA1000. We are sure to meet these problems in the next term by improving our introductory course and by using a larger device. We also stated that lab courses like the one presented in this paper are well suited for the collection of empirical data in the field of embedded systems.

B. Future Works

As mentioned above, the lab course for the next term will be improved by using different hardware and by extending the introductory course in the fields where the problems occurred. We are also planning to add some small additional tasks as a simple user interface. This way, we hope to improve our
intention in clarifying the differences between reconfigurable hardware and CPU based systems. We also plan to implement a web page including the FAQs from the first lab course. A further lab course is planned for the summer term 2006 based on Atmel’s FPSLIC Chip. According to the fact that this chip includes an 8bit MCU and a 40k FPGA we hope to demonstrate not only the differences between CPLDs and MCUs but also possible approaches of HW/SW Co-Design in the way that a single task is implemented on a combination of both hardware. We plan to let the students solve a modified task on three different hardware platforms. The first one would be an MCU, the second an FPGA and the third a combination of both (FPSLIC Chip).

C. Acknowledgments

We thank XILINX (www.xilinx.com) for providing us the Spartan3 FPGA development boards we are going to use in the next term. We thank ATMEL (www.atmel.com) for providing us the FPSLIC development boards we plan to use in summer term 2006.

REFERENCES