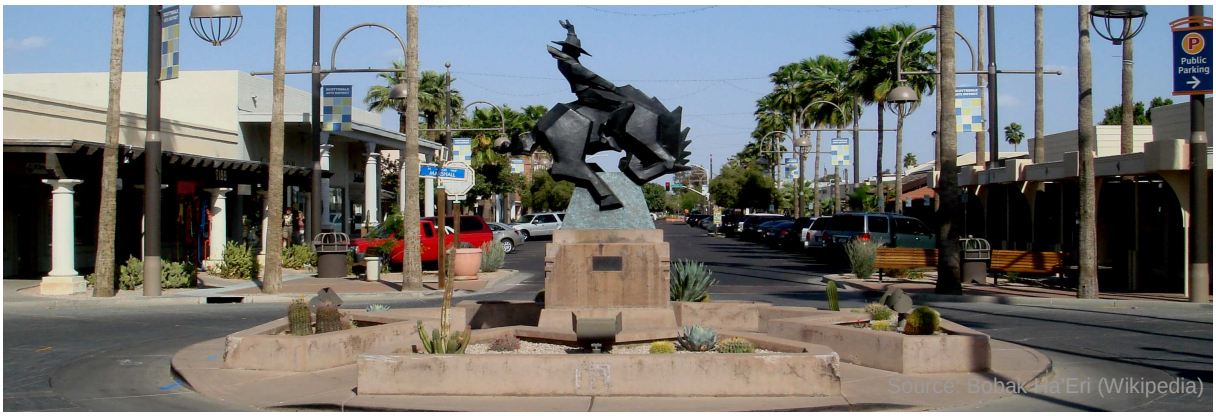


Source: Bobak Ha'Eri (Wikipedia)

Proceedings

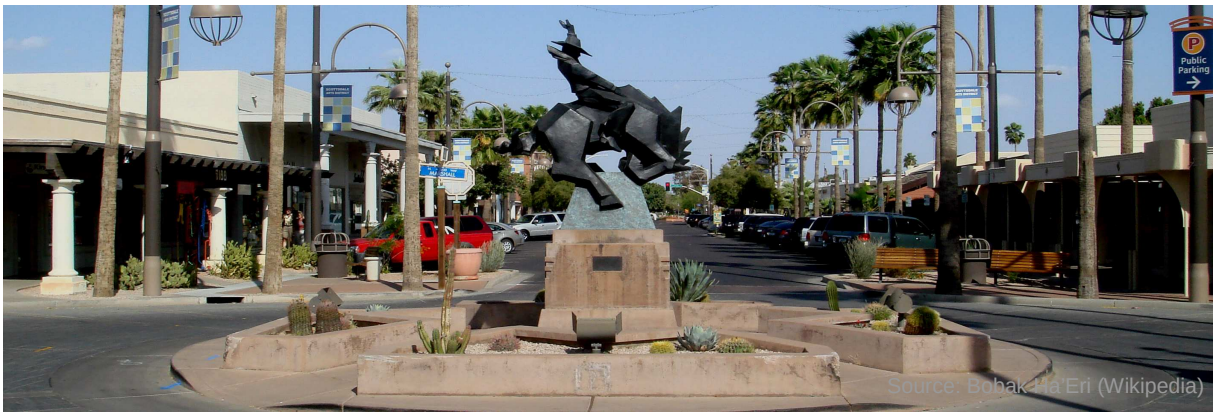
2010 Workshop on Embedded Systems Education

WESE 2010

Editors
Peter Marwedel
Jeff Jackson
Kenneth G. Ricks

Scottsdale, AZ, USA

October 28th, 2010



Organizer's Message

Embedded systems (ES) is a multidisciplinary field, requiring skills from control and signal processing theory, electronics, computer engineering and science, telecommunication, etc., as well as application domain knowledge. As ES designs grow more complex and the time to market diminishes, quality ES education becomes more and more important. This sixth workshop on the subject aims to bring researchers, educators, and industrial representatives together to assess needs and share design, research, and experiences in embedded systems education.

The program of this workshop promises to be a very interesting one incorporating many topics from both academia and industry tied together with the underlying theme of ES education. The program includes both presentations and discussion sessions to help stimulate and disseminate ideas among the participants.

Due to the high importance of ES education and the good quality of the selected papers, we are making these proceedings accessible via the ACM digital library. This availability should help the contributions to become visible electronically.

The papers have been selected by the program committee comprising well-known specialists from all over the world. We would like to thank the program committee members for their time and efforts in reviewing the submitted papers. Without their efforts the workshop would not have been possible. We would like to thank the authors for their manuscript submissions and their timely response to implement improvements suggested by the reviewers. We would like to also thank the ESWEEK organizers and their support for the WESE2010 workshop. Finally, we would like to thank the European ArtistDesign network of excellence for the support.

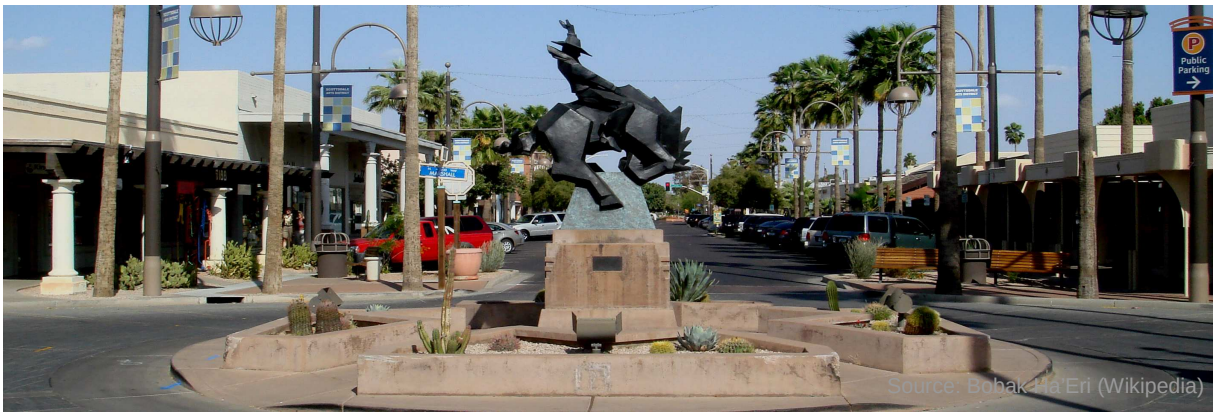
There exist many opportunities for international cooperation in the community of embedded systems researchers and educators and we look forward to those efforts.

Dortmund (Germany) and Tuscaloosa, Alabama (US), Oct. 2010

Peter Marwedel

Jeff Jackson

Kenneth G. Ricks



Source: Bobak Ha'Eri (Wikipedia)

Organizers

Jeff Jackson, The University of Alabama, US

Peter Marwedel, TU Dortmund, Germany

Kenneth Ricks, The University of Alabama, US

International Program Committee

Alex Dean, North Carolina State University, USA

Martin Grimheden, Royal Institute of Technology, Sweden

Yann-Hang Lee, Arizona State University, USA

Sin Ming Loo, Boise State University, USA

Jogesh K. Muppala, The Hong Kong University of Science and Technology, Hong Kong

Falk Salewski, Lacroix Electronics, Willich, Germany

Stewart Tansley, Microsoft Corporation, USA

Martin Törngren, Royal Institute of Technology, Sweden

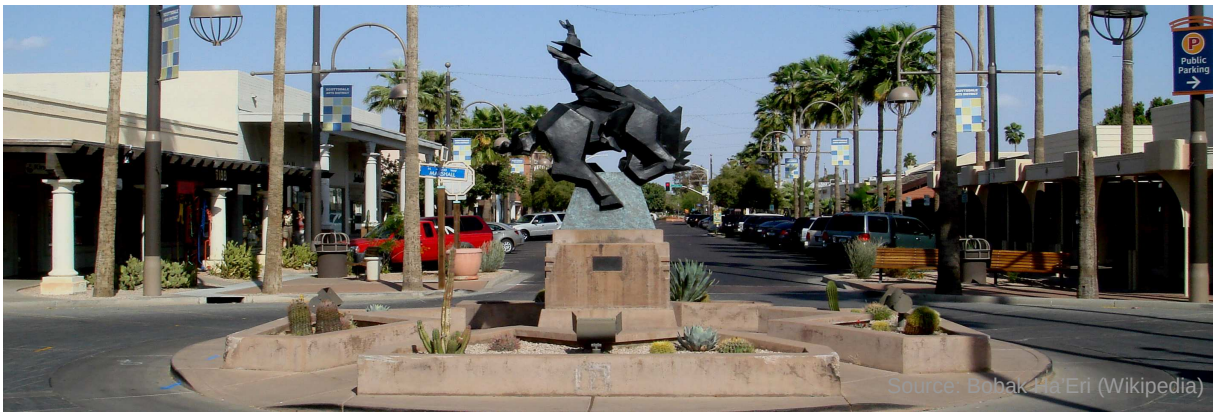
Mariagiovanna Sami, ALaRI, Lugano, Switzerland

Chi-Sheng (Daniel) Shih, National Taiwan University, Taiwan

Shiao-Li Tsao, National Chiao Tung University, Taiwan

Frank Vahid, University of California Riverside, USA

Marilyn Wolf, Georgia Tech, USA



Program October 28th, 2010

08:30 hrs Kenneth Ricks, Bruno Bouyssounouse: **Opening**

08:40 hrs Edward A. Lee, Sanjit A. Seshia (UC Berkeley, US)
Morning Keynote: **An Introductory Textbook on Cyber-Physical Systems**

09:30 hrs Break

10:00 hrs Sin Ming Loo, Josh Kiepert, Michael Pook, Jim Hall, Derek Klein, Vikram Patel, Carl Lee, Arlen Planting (Boise State University, Boise, USA): **From Scratch to System: A Hands-on Introductory Embedded Systems Course**

10:30 hrs Alexander G. Dean (NCSU, Raleigh, USA): **Teaching Optimization of Time and Energy in Embedded Systems**

11:00 hrs Bruno Bouyssounouse (IMAG, France): **Common Technical Baseline - A common reference of concepts and definitions across all industrial sectors**

11:30 hrs Meng-Ting Wang (NTHU), Po-Chun Huang (NTU), Jenq-Kuen Lee (NTHU), Shang-Hong Lai (NTHU), Roger Jang (NTHU), Chun-Fa Chang (NTNU), Chih-Wei Liu (NCTU), Tei-Wei Kuo (NTU), Steve Liao (Google) (Taiwan, ROC): **Support of Android Lab Modules for Embedded System Curriculum**

12:00 hrs **Lunch**

13:00 hrs Philip Koopman (CMU, Pittsburgh, USA)
Afternoon Keynote: **Risk Areas In Embedded Software Industry Projects**

13:50 hrs Discussion: **How to teach cyber-physical systems?**

14:30 hrs Samia Bouzefrane (CNAM, Paris, France): **The Embedded and Mobile Systems Master at the CNAM of Paris**

15:00 hrs Matthew H. Netkow (SAVO, Chicago, USA), Dennis Brylow (Marquette U., Milwaukee, USA): **Xest: An Automated Framework for Regression Testing of Embedded Software**

15:30 hrs **Break**

16.00 hrs André Stollenwerk, Andreas Derks, Stefan Kowalewski (RWTH Aachen, Germany), Falk Salewski (Lacroix Electronics, Willich, Germany): **A Modular, Robust and Open Source Microcontroller Platform for Broad Educational Usage**

16:30 hrs Discussion: **What have we learned today?**

17:00 hrs **Close**

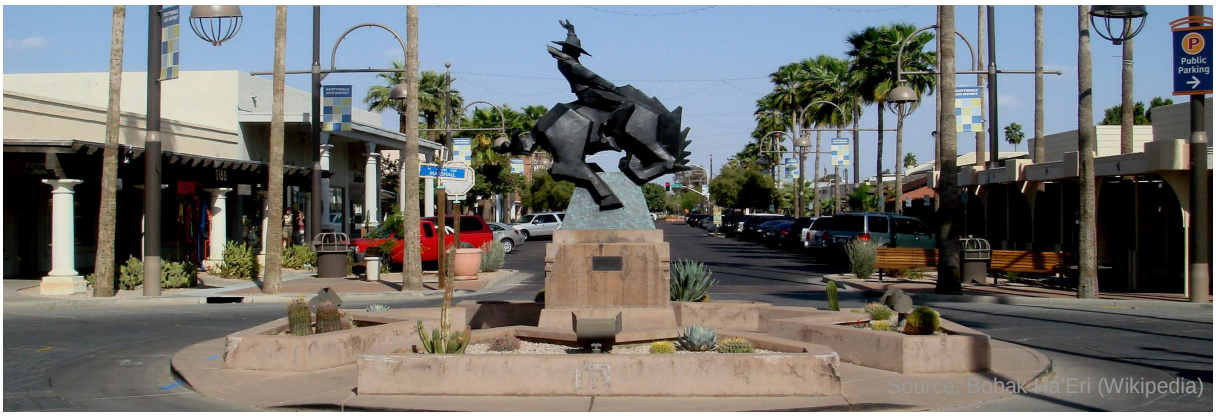
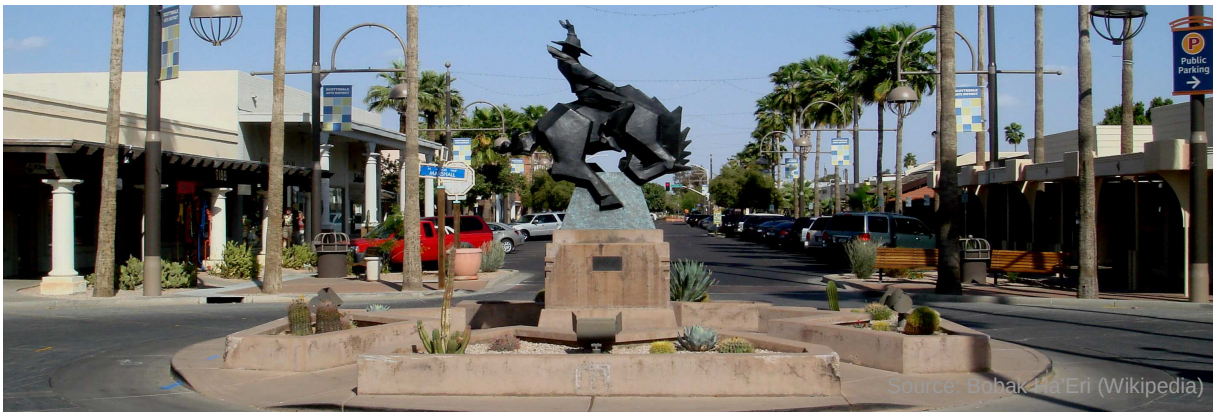


Table of Contents



An Introductory Textbook on Cyber-Physical Systems *

Edward A. Lee
EECS Department
University of California, Berkeley
Berkeley, CA, USA
eal@eecs.berkeley.edu

Sanjit A. Seshia
EECS Department
University of California, Berkeley
Berkeley, CA, USA
sseshia@eecs.berkeley.edu

ABSTRACT

We introduce a textbook that strives to identify and introduce the durable intellectual ideas of embedded systems as a technology and as a subject of study. The emphasis is on modeling, design, and analysis of cyber-physical systems, which integrate computing, networking, and physical processes. The book is intended for students at the advanced undergraduate level or the introductory graduate level, and for practicing engineers and computer scientists who wish to understand the engineering principles of embedded systems. It is also an experiment in publishing. The book is available free in electronic form, in the form of PDF file designed specifically for on-line reading. Specifically, the layout is optimized for medium-sized screens, particularly the iPad and forthcoming tablets. Extensive use of hyperlinks and color enhance the online reading experience. A print version will be available through a print-on-demand service, enabling rapid evolution and immediate correction of errors. See http://LeeSeshia.org.

1. INTRODUCTION

The most visible use of computers and software is processing information for human consumption. We use them to write books (like the one presented here), search for information on the web, communicate via email, and keep track of financial data. The vast majority of computers in use, however, are much less visible. They run the engine, brakes, seatbelts, airbag, and audio system in your car. They digitally encode your voice and construct a radio signal to send it from your cell phone to a base station. They control your microwave oven, refrigerator, and dishwasher. They run printers ranging from desktop inkjet printers to large industrial high-volume printers. They command robots on a factory floor, power generation in a power plant, processes in a chemical plant, and traffic lights in a city. They search for microbes in biological samples, construct images of the inside of a human body, and measure vital signs. They process radio signals from space looking for supernovae and for extraterrestrial intelligence. They bring toys to life, enabling them to react to human touch and to sounds. They control aircraft and trains. These less visible computers are called **embedded systems**, and the software they run is called **embedded software**.

Despite this widespread prevalence of embedded systems, computer science has, throughout its relatively short history, focused primarily on information processing. Only recently have embedded systems received much attention from researchers. And only recently has the community recognized that the engineering techniques required to design and analyze these systems are distinct. Although embedded systems have been in use since the 1970s, for most of their history they were seen simply as small computers. The principal engineering problem was understood to be one of coping with limited resources (limited processing power, limited energy sources, small memories, etc.). As such, the engineering challenge was to optimize the designs. Since all designs benefit from optimization, the discipline was not distinct from anything else in computer science. It just had to be more aggressive about applying the same optimization techniques.

Recently, the community has come to understand that the principal challenges in embedded systems stem from their interaction with physical processes, and not from their limited resources. The term **cyber-physical systems** (**CPS**) was coined by Helen Gill at the National Science Foundation in the U.S. to refer to the integration of computation with physical processes. In CPS, embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. The design of such systems, therefore, requires understanding the joint dynamics of computers, software, networks, and physical processes. It is this

*This work was supported in part by NSF CAREER grant #0644436, an Alfred P. Sloan Research Fellowship, and the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #CCR-0225610 (ITR), #0720882 (CSR-EHS: PRET), #0647591 (CSR-SGER), #0931843 (ActionWebs) and #0720841 (CSR-CPS)), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312 and AF-TRUST #FA9550-06-1-0244), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyc), and the following companies: Agilent, Bosch, National Instruments, Thales, and Toyota.

Workshop on Embedded Systems Education October 28, 2010, Scottsdale, USA.

study of *joint* dynamics that sets this discipline apart.

When studying CPS, certain key problems emerge that are rare in so-called general-purpose computing. For example, in general-purpose software, the time it takes to perform a task is a *performance* issue, not a *correctness* issue. It is not incorrect to take longer to perform a task. It is merely less convenient and therefore less valuable. In CPS, the time it takes to perform a task may be critical to correct functioning of the system. In the physical world, as opposed to the cyber world, the passage of time is inexorable.

In CPS, moreover, many things happen at once. Physical processes are compositions of many things going on at once, unlike software processes, which are deeply rooted in sequential steps. Abelson and Sussman [1] describe computing as "procedural epistemology," knowledge through procedure. In the physical world, by contrast, processes are rarely procedural. Physical processes are compositions of many concurrent processes. Measuring and controlling the dynamics of these processes by orchestrating actions that influence the processes are the main tasks of embedded systems. Consequently, concurrency is intrinsic in CPS. Many of the technical challenges in designing and analyzing embedded software stem from the need to bridge an intrinsically sequential semantics with an intrinsically concurrent physical world.

Today, getting computers to work together with physical processes requires technically intricate, low-level design. Embedded software designers are forced to struggle with interrupt controllers, memory architectures, assembly-level programming (to exploit specialized instructions or to precisely control timing), device driver design, network interfaces, and scheduling strategies, rather than focusing on specifying desired behavior. The sheer mass and complexity of these technologies tempts us to focus an introductory course on mastering them. But a better introductory course would focus on how to model and design the joint dynamics of software, networks, and physical processes. Such a course would present the technologies only as today's (rather primitive) means of accomplishing those joint dynamics. The book presented here is our attempt at a textbook for such a course.

2. RELATED BOOKS

Most texts on embedded systems focus on the collection of technologies needed to get computers to interact with physical systems [2, 3, 4, 9, 14, 16, 19, 22, 23]. Others focus on adaptations of computer-science techniques (like programming languages, operating systems, networking, etc.) to dealing with technical problems in embedded systems [5, 6, 18]. While these implementation technologies are (today) necessary for system designers to get embedded systems working, they do not form the intellectual core of the discipline. The intellectual core is instead in models and abstractions that conjoin computation and physical dynamics.

A few textbooks offer efforts in this direction. Jantsch [8] focuses on concurrent models of computation, Marwedel [12] focuses on models of software and hardware behavior, and Sriram and Bhattacharyya [20] focus on dataflow models of signal processing behavior and their mapping onto programmable DSPs. These are excellent starting points. Models and concurrency (such as dataflow) and abstract models of software (such as Statecharts) provide a better starting point than imperative programming languages (like C), interrupts and threads, and architectural annoyances that a designer must work around (like caches). These texts, however, are not suitable for an introductory course. They are either too specialized or too advanced or both. This book is our attempt to provide an introductory text that follows the spirit of focusing on models and their relationship to realizations of systems.

The variety of textbooks on embedded systems that have appeared in recent years is surprising, often reflecting the perspective of a more established discipline that has migrated into embedded systems, such as VLSI design, control systems, signal processing, robotics, real-time systems, or software engineering. Some of these books complement the one we present nicely. We strongly recommend them to the reader who wishes to broaden his or her understanding of the subject.

Specifically, Patterson and Hennessey [17], although not focused on embedded processors, is the canonical reference for computer architecture, and a must-read for anyone interested embedded processor architectures. Sriram and Bhattacharyya [20] focus on signal processing applications, such as wireless communications and digital media, and give particularly good coverage to dataflow programming methodologies. Wolf [23] gives an excellent overview of hardware design techniques and microprocessor architectures and their implications for embedded software design. Mishra and Dutt [13] give a view of embedded architectures based on architecture description languages (ADLs). Oshana [15] specializes in DSP processors from Texas Instruments, giving an overview of architectural approaches and a sense of assembly-level programming.

Focused more on software, Buttazzo [5] is an excellent overview of scheduling techniques for real-time software. Liu [11] gives one of the best treatments yet of techniques for handling sporadic real-time events in software. Edwards [6] gives a good overview of domain-specific higher-level programming languages used in some embedded system designs. Pottie and Kaiser [18] give a good overview of networking technologies, particularly wireless, for embedded systems.

No single textbook can comprehensively cover the breadth of technologies available to the embedded systems engineer. We have found useful information in many of the books that focus primarily on today's design techniques [2, 3, 4, 7, 9, 14, 16, 19].

3. THEME OF THE BOOK

The major theme of our book is on models and their relationship to realizations of systems. The models we study are primarily about dynamics, the evolution of a system state in time. We do not address structural models, which represent static information about the construction of a system, although these too are important to embedded system design.

Working with models has a major advantage. Models can have formal properties. We can say definitive things about models. For example, we can assert that a model is deterministic, meaning that given the same inputs it will always produce the same outputs. No such absolute assertion is possible with any physical realization of a system. If our model is a good abstraction of the physical system (meaning that it omits only inessential details), then the definitive assertion about the model gives us confidence in the physical realization of the system. Such confidence is enormously valuable, particularly for embedded systems where malfunc-

tions can threaten human lives. Studying models of systems gives us insight into how those systems will behave in the physical world.

Our focus is on the interplay of software and hardware with the physical environment in which they operate. This requires explicit modeling of the temporal dynamics of software and networks and explicit specification of concurrency properties intrinsic to the application. The fact that the implementation technologies have not yet caught up with this perspective should not cause us to teach the wrong engineering approach. We should teach design and modeling as it should be, and enrich this with a *critical* presentation of how to (partially) accomplish our objectives with today's technology. Embedded systems technologies today, therefore, should not be presented dispassionately as a collection of facts and tricks, as they are in many of the above cited books, but rather as stepping stones towards a sound design practice. The focus should be on what that sound design practice is, and on how today's technologies both impede and achieve it.

Stankovic et al. [21] support this view, stating that "existing technology for RTES [real-time embedded systems] design does not effectively support the development of reliable and robust embedded systems." They cite a need to "raise the level of programming abstraction." We argue that raising the level of abstraction is insufficient. We have to also fundamentally change the abstractions that are used. Timing properties of software, for example, cannot be effectively introduced at higher levels of abstraction if they are entirely absent from the lower levels of abstraction on which these are built.

We require robust and predictable designs with repeatable temporal dynamics [10]. We must do this by building abstractions that appropriately reflect the realities of cyber-physical systems. The result will be CPS designs that can be much more sophisticated, including more adaptive control logic, evolvability over time, improved safety and reliability, all without suffering from the brittleness of today's designs, where small changes have big consequences.

In addition to dealing with temporal dynamics, CPS designs invariably face challenging concurrency issues. Because software is so deeply rooted in sequential abstractions, concurrency mechanisms such as interrupts and multitasking, using semaphores and mutual exclusion, loom large. We therefore devote considerable effort in this book to developing a critical understanding of threads, message passing, deadlock avoidance, race conditions, and data determinism.

4. ORGANIZATION OF THE BOOK

As shown in Figure 1, this book is divided into three major parts, focused on **modeling**, **design**, and **analysis**. Modeling is the process of gaining a deeper understanding of a system through imitation. Models imitate the system and reflect properties of the system. Models specify **what** a system does. Design is the structured creation of artifacts. It specifies **how** a system does what it does. Analysis is the process of gaining a deeper understanding of a system through dissection. It specifies **why** a system does what it does (or fails to do what a model says it should do).

The three parts of the book are relatively independent of one another and are largely meant to be read concurrently. Strong dependencies between chapters are shown with arrows in black. Weak dependencies are shown in grey. When there is a weak dependency from chapter i to chapter j, then j may mostly be read without reading i, at most requiring skipping some examples or specialized analysis techniques. A systematic reading of the text can be accomplished in seven segments, shown with dashed outlines. Each segment includes two chapters, so complete coverage of the text is possible in a 14 week semester, assuming each of the seven modules takes two weeks. We now briefly describe the three main parts.

4.1 Modeling

This part of the book focuses on models of dynamic behavior. It begins with a light coverage of the modeling of physical dynamics, specifically focusing on continuous dynamics in time. It then talks about discrete dynamics, using state machines as the principal formalism. It then combines the two with a discussion of hybrid systems. The fourth chapter focuses on concurrent composition of state machines, emphasizing that the semantics of composition is a critical issue that designers must grapple with. The fifth chapter gives an overview of concurrent models of computation, including many of those used in design tools that practitioners frequently leverage, such as Simulink and LabVIEW.

In this part of the book, we define a **system** to be simply a combination of parts that is considered a whole. A **physical system** is one realized in matter, in contrast to a conceptual or **logical system** such as software and algorithms. The **dynamics** of a system is its evolution in time: how its state changes. A **model** of a physical system is a description of certain aspects of the system that is intended to yield insight into properties of the system. In this text, models have mathematical properties that enable systematic analysis. The model imitates properties of the system, and hence yields insight into that system.

A model is itself a system. It is important to avoid confusing a model and the system that it models. These are two distinct artifacts. A model of a system is said to have high **fidelity** if it accurately describes properties of the system. It is said to **abstract** the system if omits details. Models of physical systems inevitably *do* omit details, so they are always abstractions of the system. A major goal of this text is to develop an understanding of how to use models, of how to leverage their strengths and respect their weaknesses.

A cyber-physical system (CPS) is a system composed of physical subsystems together with computing and networking. Models of cyber-physical systems must include all three parts. The models will typically need to represent both **static properties** (those that do not change during the operation of the system) and dynamics.

Each of the modeling techniques described in this part of the book is an enormous subject, much bigger than one chapter, or even one book. In fact, such models are the focus of many branches of engineering, physics, chemistry, and biology. Our approach is aimed at engineers. We assume some background in mathematical modeling of dynamics (calculus courses that give some examples from physics are sufficient), and then focus on how to compose diverse models. This will form the core of the cyber-physical system problem, since joint modeling of the cyber side, which is logical and conceptual, with the physical side, which is embodied in matter, is the core of the problem. We therefore make no attempt to be comprehensive, but rather pick a few modeling techniques that are widely used by engineers and well

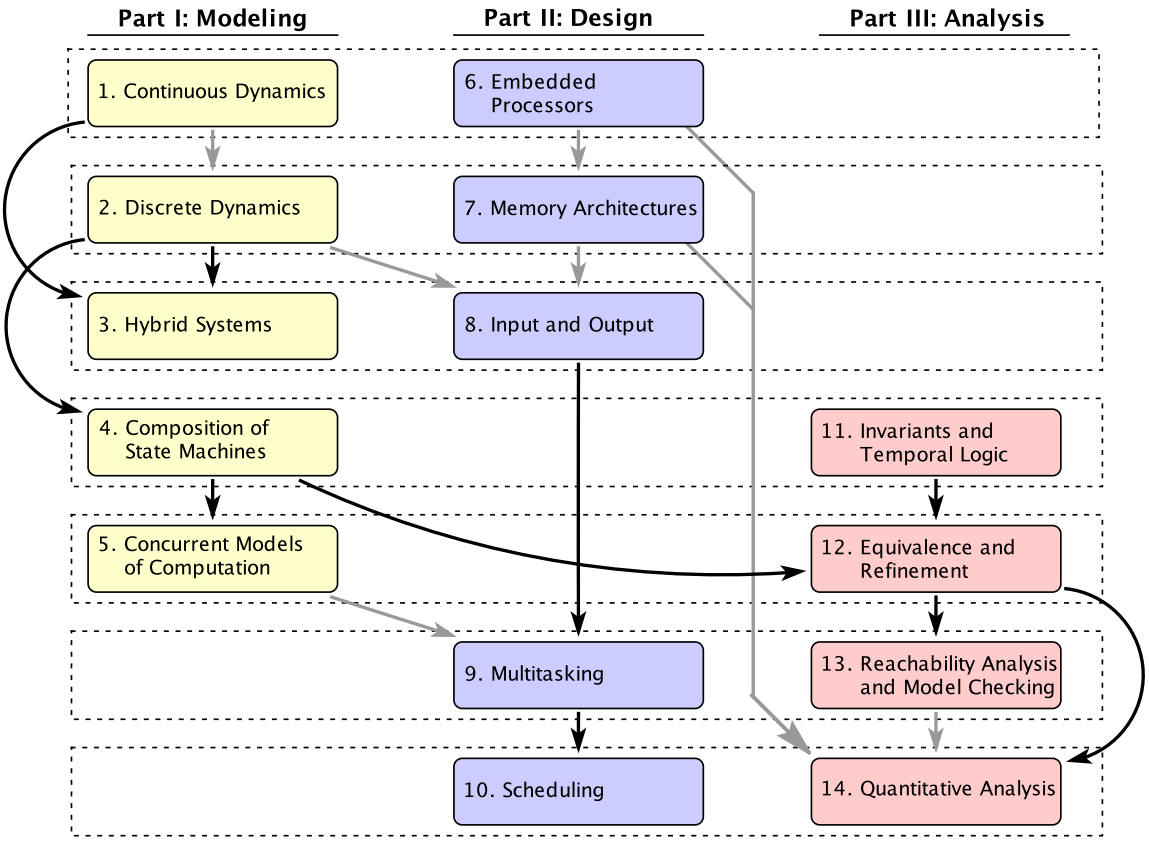


Figure 1: Map of the book with strong and weak dependencies between chapters.

understood, review them, and then compose them to form a cyber-physical whole.

4.2 Design

The second part of the book has a very different flavor, reflecting the intrinsic heterogeneity of the subject. This part focuses on the design of embedded systems, with emphasis on the role they play *within* a CPS. Chapter 6 discusses processor architectures, with emphasis on specialized properties most suited to embedded systems. Chapter 7 describes memory architectures, including abstractions such as memory models in programming languages, physical properties such memory technologies, and architectural properties such as memory hierarchy (caches, scratchpads, etc.). The emphasis is on how memory architecture affects dynamics. Chapter 8 is about the interface between the software world and the physical world. It discusses input/output mechanisms in software and computer architectures, and the digital/analog interface, including sampling. Chapter 9 introduces the notions that underly operating systems, with particularly emphasis on multitasking. The emphasis is on the pitfalls of using low-level mechanisms such as threads, with a hope of convincing the reader that there is real value in using the modeling techniques covered in the first part of the book. Chapter 10 introduces real-time scheduling, covering many of the classic results in the area.

In all chapters in this part, we particularly focus on the mechanisms that provide concurrency and control over timing, because these issues loom large in the design of cyber-physical systems. When deployed in a product, embedded processors typically have a dedicated function. They control an automotive engine or measure ice thickness in the Arctic. They are not asked to perform arbitrary functions with user-defined software. Consequently, the processors, memory architectures, I/O mechanisms, and operating systems can be more specialized. Making them more specialized can bring enormous benefits. For example, they may consume far less energy, and consequently be usable with small batteries for long periods of time. Or they may include specialized hardware to perform operations that would be costly to perform on general-purpose hardware, such as image analysis. Our goal in this part is to enable the reader to *critically* evaluate the numerous available technology offerings.

One of the goals in this part of the book is to teach students to implement systems while *thinking across traditional abstraction layers* — e.g., hardware *and* software, computation *and* physical processes. While such cross-layer thinking is valuable in implementing systems in general, it is particularly essential in embedded systems given their heterogeneous nature. For example, a programmer implementing a control algorithm expressed in terms of real-valued quantities must have a solid understanding of computer arithmetic (e.g., of fixed-point representations) in order to create a reliable implementation. Similarly, an implementor of automotive software that must satisfy real-time constraints must be aware of processor features – such as pipelining and caching – that can affect the execution time of tasks and hence the real-time behavior of the system. Likewise, an implementor of interrupt-driven or multi-threaded software must understand the level of atomicity provided by the

underlying software-hardware platform and use appropriate synchronization constructs to ensure correctness. Rather than doing an exhaustive survey of different implementation methods and platforms, this part of the book seeks to give the reader an appreciation for such cross-layer topics, and uses homework exercises to facilitate a deeper understanding of them.

4.3 Analysis

Every system must be designed to meet certain requirements. For embedded systems, which are often intended for use in safety-critical, everyday applications, it is essential to certify that the system meets its requirements. Such system requirements are also called **properties** or **specifications**. The need for specifications is aptly captured by the following quotation (paraphrased from [24]):

> "A design without specifications cannot be right or wrong, it can only be surprising!"

This part of the book focuses on precise specifications of properties, on techniques for comparing specifications, and on techniques for analyzing specifications and the resulting designs. Reflecting the emphasis on dynamics in the text, Chapter 11 focuses on temporal logics, which provide precise descriptions of dynamic properties of systems. These descriptions are treated as models. Chapter 12 focuses on the relationships between models. Is one model an abstraction of another? Is it equivalent in some sense? Specifically, that chapter introduces type systems, as a way of comparing static properties of models, and language containment and simulation relations as a way of comparing dynamic properties of models. Chapter 13 focuses on techniques for analyzing the large number of possible dynamic behaviors that a model may exhibit, with emphasis on model checking as a technique for exploring such behaviors. Chapter 14 is about analyzing quantitative properties of embedded software, such as finding bounds on resources consumed by programs. It focuses particularly on execution time analysis, with some introduction to others such as energy and memory usage.

In present engineering practice, it is common to have system requirements stated in a natural language such as English. It is important to precisely state requirements to avoid ambiguities inherent in natural languages. The goal of this part of the book to help replace descriptive techniques with *formal* ones, which we believe are less error prone.

Importantly, formal specifications also enable the use of automatic techniques for formal verification of both models and implementations. This part of the book introduces readers to the basics of formal verification, including notions of equivalence and refinement checking, as well as reachability analysis and model checking. In discussing these verification methods, we take a *user's view* of them, discussing, for example, how model checking can be applied to find subtle errors in concurrent software, or how reachability analysis can be used in computing a control strategy for a robot to achieve a particular task.

4.4 Missing Sections

Version 1.0 of the book will not be complete. It is arguable, in fact, that complete coverage of embedded systems in the context of CPS is impossible. Specific topics that we cover in the undergraduate Embedded Systems course at Berkeley[1] and hope to include in version 2.0 or later versions of the book include sensors and actuators, networking, fault tolerance, simulation techniques, control systems, and hardware/software codesign. If you are an author interested in contributing, please contact us at authors@LeeSeshia.org.

4.5 Intended Audience

This book is intended for students at the advanced undergraduate level or the introductory graduate level, and for practicing engineers and computer scientists who wish to understand the engineering principles of embedded systems. We assume that the reader has some exposure to machine structures (e.g., should know what an ALU is), computer programming (we use C throughout the text), basic discrete mathematics and algorithms (e.g., graph traversal through depth-first or breadth-first search), and at least an appreciation for signals and systems (what it means to sample a continuous-time signal, for example).

5. PUBLICATION STRATEGY

Recent advances in technology are fundamentally changing the technical publishing industry. Almost every aspect of how academics, teachers, and intellectuals communicate is in flux, and we believe that the landscape in the 21st century will be very different from that of the 20th century. In response to this, the book we introduce here is an experiment in publishing. With apologies to the many hard-working men and women in the traditional publishing industry, we hope that if this approach is successful, that it will be followed by other authors.

The book is available free in electronic form from the website http://LeeSeshia.org. In recognition that there is real value in a tangible manifestation on paper, something you can thumb through, something that can live on a bookshelf to remind you of its existence, the book will also be available in print form. Our plan is to use a print-on-demand service, which has the advantages of dramatically reduced cost to the reader and the ability to quickly and frequently update the version of the book to correct errors and discuss new technologies.

The reduced revenue stream that results from this publication strategy, of course, has disadvantages. But prior experience of these authors indicate that it is rare for authors of technical books to even earn minimum wage for their efforts. Remuneration is clearly not the main goal. The main goals seem to be communication, education, and impact. We believe that our strategy enhances all three goals.

Consider for example the opportunities afforded by a focus on on-line dissemination. The electronic version is a PDF file designed specifically for on-line reading. The layout is optimized for medium-sized screens, particularly the iPad and forthcoming tablets. Extensive use of hyperlinks and color enhance the online reading experience. Links to external websites are included directly at the relevant point.

We attempted to adapt the book to e-book formats, which, in theory, enable reading on various sized screens, attempting to take best advantage of the available screen. However, like HTML documents, e-book book formats are a reflow technology, where page layout is recomputed on the fly. The results are highly dependent on the screen size and prove ludicrous on many screens and suboptimal on all. As a conse-

[1] See http://chess.eecs.berkeley.edu/eecs149/

quence, we have opted for controlling the layout, and we do not recommend attempting to read the book on an iPhone.

As of this writing, a preliminary version of the book is available at http://LeeSeshia.org. We plan a complete release of version 1.0 by the end of summer, 2010, with an associated print-on-demand version available in the Fall.

6. ACKNOWLEDGEMENTS

The authors gratefully acknowledge contributions and helpful suggestions on the text from Elaine Cheong, Gage Eads, Stephen Edwards, Shanna-Shaye Forbes, Jeff Jensen, Wenchao Li, Isaac Liu, Slobodan Matic, Steve Neuendorffer, Minxue Pan, Hiren Patel, Jan Reineke, Chris Shaver, Stavros Tripakis, Pravin Varaiya, Maarten Wiggers, and the students in UC Berkeley's EECS 149 class, particularly Ned Bass and Dan Lynch.

7. REFERENCES

[1] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, second edition, 1996.

[2] M. Barr and A. Massa. *Programming Embedded Systems*. O'Reilly, 2nd edition, 2006.

[3] A. S. Berger. *Embedded Systems Design: An Introduction to Processes, Tools, & Techniques*. CMP Books, 2002.

[4] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time POSIX*. Addison-Wesley, 3d edition, 2001.

[5] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, second edition, 2005.

[6] S. A. Edwards. *Languages for Digital Embedded Systems*. Kluwer Academic Publishers, 2000.

[7] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner. *Embedded System Design - Modeling, Synthesis, and Verification*. Springer, 2009.

[8] A. Jantsch. *Modeling Embedded Systems and SoCs - Concurrency and Time in Models of Computation*. Morgan Kaufmann, 2003.

[9] R. Kamal. *Embedded Systems: Architecture, Programming, and Design*. McGraw Hill, 2008.

[10] E. A. Lee. Computing needs time. Technical Report UCB/EECS-2009-30, EECS Department, University of California, Berkeley, February 18 2009.

[11] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[12] P. Marwedel. *Embedded System Design*. Kluwer Academic Publishers, 2003.

[13] P. Mishra and N. D. Dutt. *Functional Verification of Programmable Embedded Processors - A Top-down Approach*. Springer, 2005.

[14] T. Noergaard. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Elsevier, 2005.

[15] R. Oshana. *DSP Software Development Techniques for Embedded and Real-Time Systems*. Embedded Technology Series. Elsevier, 2006.

[16] J. S. Parab, V. G. Shelake, R. K. Kamat, and G. M. Naik. *Exploring C for Microcontrollers*. Springer, 2007.

[17] D. A. Patterson and J. L. Hennessey. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2nd edition, 1996.

[18] G. Pottie and W. Kaiser. *Principles of Embedded Networked Systems Design*. Cambridge University Press, 2005.

[19] D. E. Simon. *An Embedded Software Primer*. Addison-Wesley, 2006.

[20] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC press, 2nd edition, 2009.

[21] J. A. Stankovic, I. Lee, A. Mok, and R. Rajkumar. Opportunities and obligations for physical computing systems. *Computer*, pages 23–31, 2005.

[22] J. W. Valvano. *Embedded Microcomputer Systems - Real Time Interfacing*. Thomson, 2nd edition, 2007.

[23] W. Wolf. *Computers as Components: Principles of Embedded Computer Systems Design*. Morgan Kaufman, 2000.

[24] W. Young, W. Boebert, and R. Kain. Proving a computer system secure. *Scientific Honeyweller*, 6(2):18–27, July 1985.

From Scratch to System: A Hands-on Introductory Embedded Systems Course

Sin Ming Loo, Josh Kiepert, Michael Pook, Jim Hall, Derek Klein, Vikram Patel, Carl Lee, Arlen Planting
Electrical and Computer Engineering Department
Boise State University
Boise, Idaho 83725
208-426-5679
smloo@boisestate.edu

Abstract
This paper describes a hands-on introductory embedded systems course, which continues from the first microprocessor course. Instead of using an off-the-shelf microcontroller development board, it shows how students can build one from scratch and add components when required for assignments or as the need arises. The course begins with wiring a microcontroller system from scratch, continues through interfacing to various sensors, and culminates in a final project. The course also focuses on embedded systems code layering concepts and enforces their usage. Lectures on practical analog interfacing circuits, such as op-amp circuitry, were presented. There were two written tests and seven hands-on laboratory assignments. The course reviews indicated students like this approach tremendously.

Categories and Subject Descriptors
K.3 [Computers and Education]: Computers and Education - General

General Terms
Experimentation

Keywords
Microcontroller, Curriculum

1. Introduction
Microcontroller courses have traditionally been taught using an off-the-shelf development board. Currently, there are many microcontroller development boards in the marketplace, with costs ranging from a few dollars to a few hundred dollars. They are built with various functionality and capabilities. Some boards have additional areas for prototyping, extra connectors, and different means to provide power to the board. Others boards are more limited, usually due to a lower cost.

It is important to note that the development board is not the only cost associated with the development platform. The device programmer and development toolchain are very important and costly parts of the development platform. For some, the microprocessor can be programmed through USB because the programming circuitry has been built onto the board. For others, programmers will be necessary. These vary in price but can be quite expensive.

Because this is an embedded system course, students usually prefer to have the option to work whenever they can find the time wherever that may be. This makes purchasing hardware the most desirable option for the student. Additionally, they would like to have the development toolchain installed on their personal computer and their own device programmer. These factors need to be kept in mind when selecting a development platform. Fortunately, the toolchain is usually very cost effective. Typically, there is a student version or an open source version available. For some toolchains, the Integrated Development Environment (IDE) is proprietary. Interestingly, for many, it is an Eclipse based IDE [1]. Once Eclipse is installed, a compiler can be installed as a plug-in to the IDE. The Eclipse IDE is used in the junior Microprocessor course. The embedded systems course picks up from where the junior microprocessor course left off.

In the previous version of the embedded systems course, class time was divided into two sections: lecture and an end-of-semester project. Typically, the lectures concentrated on the features and capabilities of particular microcontrollers, assembly only programming (no C instruction), and device-to-device communication protocols. The course had few opportunities for hands-on experience. The bulk of the hands-on experience came from the end-of-semester project. The scope of the project was proposed by each group of students and negotiated to an acceptable difficulty level for the allotted time (~4 weeks). It was time to update the course to leverage the new technology and development tools available.

The approach to the new embedded systems course is to let the students work on various project-based laboratory assignments. This new paradigm is a hands-on based approach with lecture as backup when a need arises. The new format has been offered three times. In the first two offerings, an off-the-shelf microcontroller development board was used as the teaching platform for most of the laboratory assignments. . For both of these offerings, one of the laboratory assignments involved wiring a microcontroller system from scratch, using a Microchip 18F4620, wall power supply, linear regulator, LEDs, clock oscillator, and UART/RS232 level shifter on a breadboard. Students also needed to write test code to verify their system was wired correctly. At the completion of this assignment, students commented on how useful it had been and offered many constructive insights for improving their understanding. The comments usually ended with "it is very simple to wire up the system, having done it personally greatly increased my confidence for later assignments." With these comments in mind, the third offering was re-designed with the system building as the starting point.

The remainder of this paper describes this third offering of the embedded system course at Boise State University. Section 2 outlines BSU's electrical engineering curriculum and computer engineering courses. Sections 3 and 4 discuss the course goals and details. Section 5 discusses the course outcome and reviews. A summary and conclusion is presented in Sections 6.

2. Existing Situation

Boise State University has an ABET-accredited electrical engineering program with computer engineering as an option. One of the core courses offered at Boise State University for students specializing in computer engineering is a Microprocessors course. The students take Microprocessors after they have taken Introduction to Computer Science (an introduction to basic software skills and object oriented programming with Java) and Digital Systems (the sophomore digital logic course).

The Microprocessors course at Boise State University covers microprocessor architectures, software development toolchains, and low-level hardware interfaces with emphasis on 16-bit and 32-bit microprocessor systems. Some of the covered topics include [2]:

- Machine language
- Assembly
- C programming and associated techniques
- Instruction set
- Addressing modes
- Memory systems
- I/O interfacing
- Handling interrupts
- PWM/ADC

The course also covers practical applications in data acquisition, control, and interfacing. This course was reported to be a favorite of many students, largely because of the interesting devices (such as the magnetic card reader) that could be played with by the end of the course. The intent was to retain and potentially enhance this characteristic of the Microprocessors course with the changes implemented in the embedded systems course.

Since the microprocessors course (lecture and lab) is a requisite for both electrical and computer engineering (ECE) and computer science students, the course must endeavor to address the dissimilar interests and needs of students in both disciplines. In addition to those specializing in computer engineering, the ECE group includes students interested primarily in other areas such as integrated circuits, communication and signal processing, control systems, power and energy systems, etc. Most computer science students are more interested in hardware with an operating system. Therefore it is important to achieve a balance in the course that will adequately teach electrical engineering and computer science students the needed fundamentals of microprocessors, while also providing the computer engineering students a solid foundation for later courses.

A course titled “Embedded and Portable Computing Systems”, specifically addressing embedded design with the PIC microcontroller, has been offered at BSU. This was primarily a hardware-oriented senior/graduate level course utilizing assembly language only. Students taking this class received no C programming instruction. This paper describes how this course has been re-designed with a project-oriented approach.

3. Goals

The goals of the updated course were to teach the design of a complete system from the ground-up (including coding and interfacing techniques). Using this approach students see how a microcontroller system comes together. With the basic system operational, one can optimize as necessary for different applications. The objective is to learn skills that can be applied as an embedded systems engineer – designing the optimal hardware (for a project.

The course would include:

- Introduction of microcontrollers (specifics of memory, devices, and how they relate to programming)
- How coding hierarchy and language can affect the ease of transferring code to other platforms (portability)
- Advanced devices (I^2C, SPI, USB, UART) from hardware and software perspectives
- Analog interfacing circuitry to digital systems
- Advanced time management issues and usage of state machine constructs in order to manage time
- Proper coding techniques including advanced hierarchical design

4. Course Details

The semester began with an introduction and discussion of the motivations of the course. The first challenge was a pre-semester quiz. Students were told what to review for the pre-semester quiz. The main objective was to gauge the students’ programming skills. This also served to check the email submission process (details in a later paragraph). The quiz included two problems on determining the outputs of C programs. The first problem (see Figure 1) was shifting and masking of bits for a look-up-table. The second problem (see Figure 2) was on structures, unions, and pointers. It included a *struct* declaration initialized to some values. Students were to determine what the outputs of the functions were as produced by *printf()*.

There were two other problems in this quiz, both were programming problems. They were assigned as take home problems due the following day (due within 24 hours). The objective of these problems was to determine the students’ coding skills and coding styles in a no “pressure” environment (it is assumed that a 24 hour time constraint was not a stress factor!).

```
#include <stdio.h>

char HEXCHAR[] = "0123456789abcdef";

int main()
{
      unsigned int val = 0xa5b4c3d2;
    char             ch;

    ch = HEXCHAR[ (val>>17) & 0xf ];
      printf ( "==>%c\n", ch );

      return 0;
}
```

Figure 1: Code for pre-semester quiz Problem 1

```
#include <stdio.h>

typedef unsigned int    ui32;
typedef unsigned short  ui16;
typedef unsigned char   ui8;

struct abc {
    union {
        ui32    word;
        struct  {
            ui8 b0;
            ui8 b1;
            ui8 b2;
            ui8 b3;
        } bytes;
    }    a;
    ui8  b;
    union {
        ui16    half;
        struct {
            ui32 n0 : 4;
            ui32 n1 : 4;
            ui32 n2 : 4;
            ui32 n3 : 4;
        } nibbles;
    }    c;
};

int main(void)
{
    struct abc x[] = {
        {{0xa1a2a3a4}, 'a', {0x1234}},
        {{0xb1b2b3b4}, 'b', {0x5678}},
        {{0xc1c2c3c4}, 'c', {0x90ab}},
        {{0xd1d2d3d4}, 'd', {0xcdef}},
        {{0xa5b5c5d5}, 'e', {0x1111}},
    };

    struct abc *px = &x[0];

    printf ( "a. 0x%x\n", (px+2)->a.bytes.b2      );
    printf ( "b. 0x%x\n", (*px).a.bytes.b0        );
    printf ( "c. 0x%x\n", px->c.nibbles.n3        );

    px++;
    printf ( "d. 0x%x\n", x[2].c.nibbles.n0       );
    printf ( "e. 0x%x\n", (x[4].a.word >> 12) & 0xff  );
    printf ( "f. %c\n",   px->b );

    return 0;
}}
```

Figure 2: Code for pre-semester quiz Problem 2

The course met twice a week for one hour and fifteen minutes each. For most weeks, one meeting was assigned to lecture. The other meeting occurred in the lab. It was stressed that lecture did not have to take the entire time slot. It was more important to deliver the materials efficiently and clearly, rather than filling in the time with unimportant materials. For example, there is no point in discussing a datasheet in detail during a lecture. It is more efficient to spend just a few minutes highlighting the important items. Preferably, assign it as reading material and inform the students that they will be quizzed. When there was time left over in a lecture session, the instructor and students met in the lab to discuss their coding/design and to provide help when it was necessary. For the lab meeting, there was no step-by-step instruction of what one needed to accomplish. Having been told of the assignment, it was up to the students to come up with the approaches to get it accomplished. The instructor reviewed the approaches and made suggestions where necessary to help students complete the assignment.

The course was taught with very little use of paper. An email account was set up where the assignments (that involved writing or source code submission) were to be emailed. Submissions were accepted as PDF or Microsoft Word files. On only two occasions was paper used when it was not convenient to go paperless: for the pre-semester and mid-semester tests.

The early assignments were well defined. It was described to the students what one needs to achieve (if it is not working, 40% to 50% of assignment grade will be deducted). As part of the assignment, the students were told what they needed to show during the assignment demonstration. Each assignment submission had two parts. Any missing part resulted in a zero grade for that assignment. Part 1 was the assignment demonstration. If it was a non-working demonstration, one opportunity was given to re-work (usually due by the next day). Part 2 was the submission of source code and readme files. Each source code file was required to have proper header information including the name of the coder, purpose of the file, date, and revision information. The readme file was a description of what the code was about. The student was given the choice to make the readme file a short paragraph or a comprehensive description depending on the requirements of the assignment.

Towards the third quarter of the semester, a test was given. The test included questions on pull-up and pull-down resistors, technical details of some of the assignments, voltage conversion system design, op-amp interface circuit design, and analog to digital conversion accuracy and resolution.

For the instruction of coding styles and source code generation, the use of layering, structures, pointers, and header files was enforced [3]. The main (main.c) file was to be kept very short with the details included in the respective C and H files. This resulted in a clean main C file containing no clutter, with details distributed into device or function specific C and H files. In this manner the students were instructed on the generation of readable/maintainable code.

The remainder of this section discusses details of the assignments. There were a total of two writing assignments and seven laboratory assignments. These were individual assignments.

4.1 Writing and Coding Assignments

Writing Assignment 1: Microcontroller Summary

This was a microcontroller research assignment. The objective was for students to realize that there are many microcontrollers in the marketplace, each with a different amount of resources and capabilities. Basically, this was a microcontroller data sheet reading assignment. The students needed to find ten microcontrollers (from at least five different vendors) and generate a table with device features (i.e. the number of ADC channels, the amount of on-chip memory, etc).

Hands-on Assignment 1: Microchip from Scratch

In this assignment, students wired up a microcontroller system with output to a terminal though RS232. The students were also required to write code to blink the LEDs and display messages to a terminal program (e.g. Hyperterminal or TeraTerm). For this

assignment, the students were instructed to bring an unused wall power supply (e.g. old cell phone charger, at least capable of supplying 100 mA at 6V out) and were supplied with all other components needed to build the system. The Microchip PIC18F4620 was used as the microcontroller. Since a breadboard was used for wiring, the DIP IC package was used. The surface mount version is also available for use if one decided to fabricate a printed-circuit board. The breadboard wiring method was chosen as a starting point to eliminate additional complications (e.g. PCB generation, soldering, etc.) that would impede the learning of how a microcontroller system is put together.

The students were supplied with DC-DC converter, capacitors, resistors, LEDs, RS232 level converter, oscillator, RS232 cable, PIC18F2620, wires, and breadboard. In the lecture, the students were shown how to wire-up the system (no handout was given). A C program was also needed as part of this assignment. The C program was required to blink the LEDs and send a message to the terminal program using the PIC's integrated UART. This program was to run in a continuous loop. The hands-on assignments that followed built upon this initial system.

Writing Assignment 2: Microcontroller Write-up

This was a follow-up assignment to Hands-on Assignment 1. The objective of this assignment was to assess what the students gained, whether the students understood the role of each component in the system in Hands-on Assignment 1.

Hands-on Assignment 2: Analog to Digital Converter

This was an analog to digital conversion (ADC) assignment. The objective was to learn how to use the ADC that is integrated in the PIC18F4620. Students used the bench-top power supply to generate voltages for the ADC. The converted digital value obtained from the ADC was then scaled to the proper voltage as supplied by the power supply. The PIC18F4620 UART was used to display the value to a terminal program.

Hands-on Assignment 3: Thermistor

This assignment furthered the use of the ADC. Students were given thermistors and op-amps. The assignment objective was to add the thermistor to the microcontroller board. One needed to design an interface circuit to connect the thermistor to the microcontroller, then write code to convert the measured voltage to a temperature. The PIC UART was used to display the value to a terminal program.

Hands-on Assignment 4: ADC Chip with SPI

This was an SPI programming assignment. The Microchip C18 SPI API routines were used to give students the experience of integrating their program with another set of code. Students were given an ADC chip with an SPI interface, and were instructed to interface the ADC chip to the microcontroller. The ADC was to convert the voltage to a digital value and send the value to the microcontroller through SPI for processing. As before, the PIC UART was used to display the value to a terminal program.

Hands-on Assignment 5: Printed-Circuit Board Design

In this assignment, the students used Eagle PCB software to layout a board using the system as built in Hands-on Assignment 1. Some students decided to have their boards made by board house. Some students decided to use the in-house two-lay PCB milling machine.

Hands-on Assignment 5: Embedded Menu

For a system based on an 8-bit microcontroller, a computationally intensive menu system is out of the question. This assignment was designed to show how a simple and low-overhead menu system could be coded to setup and control system options. Skeleton source code was provided to the students which provided the basic structure that the students needed to use to write their embedded menu program. The PIC UART was used to interface with the terminal program for both displaying the menu and receiving user input.

Hands-on Assignment 7: I2C

This assignment is similar to Hands-on Assignment 4, but with an I^2C based digital-to-analog converter. Students coded three programs to output a sine waveform, a square waveform, and a triangular waveform. A mixed-signal oscilloscope was used to view the output waveforms.

Project: Independent Project

For the Independent Project, each student discussed and proposed a project to the instructor for approval. The student then used the knowledge and experience gained in the course to implement their project. The final project grade consisted of the project implementation, presentation, and demonstration. This was 30% of the course grade.

5. Course Outcome

Final lab projects were undertaken to consolidate and demonstrate the knowledge gained in the lecture and lab portions of the course. The final projects successfully demonstrated the students' grasp of the knowledge presented in the course. There were ten students signed up for the course in the third offering with one dropping by the midpoint of the semester. A wide range of projects were attempted:

- Power usage monitor
- Remote temperature sensing
- Clapping detector
- RV battery charger and monitor
- Wireless Homework Uploader
- Air Filter Pressure Detector
- Frequency Detector
- Remote Control Code Decoder
- Autonomous Micromouse

All projects were completed within the time frame provided with minimal assistance from the instructor. Based on student evaluations of the course, the updated format of the course was considered successful. In the course review, students were asked to answer four questions:

- What do you like about the structure of this class?
- Will you recommend this course to a friend?

- What do you NOT like?
- What other topics we could have added?

For the first question, most students commented positively on the ratio of labs to lectures. “The lectures give some preliminary information, and then the labs let you get your hands dirty. This seems to be very effective for learning, especially when you must struggle with a lab.” One major theme from the review has been on developing the PIC microcontroller system from scratch. “Doing the "PIC from scratch" on a prototype board was much better than using a purchased development board. Now, I feel confident with acquiring just a microcontroller chip and making something useful with it. I also learned how to use clock modules, design practical op amp circuits, and some things to avoid when using an ADC.” The reviews indicated that as long as someone is interested in embedded systems, students recommended taking the course.

Some suggested the addition of sensor calibration, interrupts, and more lectures. The sensor calibration and lectures will be evaluated in the next offering. As for the interrupt discussion, due to time, no formal assignment was given. Instead, the majority of the students were provided with materials on how to use the PIC interrupt.

The pre-semester test had an average of 80%, with seven out of ten students scoring 80% and above. The third quarter semester test had an average of 78%, with six out of nine students scoring 78% and above. Using the rule 70% of the students scored at least 70% and above, the grades showed the course achieved the major objectives. It should be noted that the assignments and the project were 70% of the grade.

6. Conclusion

To date, this course has been taught three times and refined based on experiences in the first and second offerings. The revised course placed emphasis on the hands-on skills. It instilled a good foundation in embedded systems design, which could be built upon by later advanced embedded systems courses.

This course was taught without a textbook. Reference materials were provided in-class or through email. One of the references is [4]. Publications and articles from trade magazines were also used. This has encouraged students to look for their own references. However, in a few cases, providing some basic materials in class instead of having students search for it themselves proved to be more beneficial. Potentially, some students might waste a lot of time looking for useless information.

The updated embedded systems course has been fruitful for students and. Students learned that they are able to research the assignment, prototype the hardware, and code the software necessary to make the hardware useful.

9. References

[1] Eclipse IDE, http://www.eclipse.org/, Visisted: July 19, 2010.

[2] Sin Ming Loo, Arlen Planting, “Use of Discrete and Soft Processors in Introductory Microprocessors and Embedded Systems Curriculum,” SIGBED Review, Volume 6, Number 1, January 2009, Special Issue from the Workshops on Embedded System Education (WESE) in 2007 and 2008.

[3] Michael Pook, Sin Ming Loo, Arlen Planting, Josh Kiepert, Derek Klein, “Coding Practices for Embedded Systems,” 2010 ASEE Annual Conference, June 20-23, 2010.

[4] John B. Peatman, Coin-Cell-Powered Embedded Design,” Available at http://www.lulu.com, 2008.

Teaching Optimization of Time and Energy in Embedded Systems

Alexander G. Dean
Center for Efficient, Secure and Reliable Computing
Dept. of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC, USA
(919) 513-4021
Alex_Dean@ncsu.edu

ABSTRACT
The graduate Embedded Systems Design class in the ECE Department at NCSU has evolved over the past ten years. This paper describes how the course prepares students for profiling and optimizing run-time and energy performance, porting code across architectures, and also provides student feedback.

Categories and Subject Descriptors
TBD

General Terms
Measurement, Performance, Design, Experimentation.

Keywords
Embedded systems class, run-time optimization, energy, M16C.

1. INTRODUCTION
The graduate Embedded Systems Design class in the ECE Department at NCSU was first offered in fall 2000. It has evolved over the past ten years based on instructor experience, especially due to perspectives gained from performing design reviews deeply-embedded systems in industry.

The course teaches feedback-driven performance optimization of run-time and energy use. The fundamental message is "avoid all run-time work possible, and do what remains as efficiently as possible." This is coupled by encouraging "outside the box" thinking; the whole system is capable of optimization, not just discrete and disparate pieces. Embedded systems differ from general-purpose computers in that the designer has control over multiple aspects of the system: This multidimensional design space typically offers many opportunities for optimization.

This whole-system view is coupled with an analytical, performance-driven approach. After each optimization we evaluate the performance impact, giving the students hands-on experience in determining what works well and what does not.

A common experience in industry is porting existing code to a new processor architecture. To give students experience in this domain we have a code porting project involving interfacing the microcontroller with either a microSD card or a color graphical TFT LCD.

This paper reports on the course as taught in the Spring 2010 semester at North Carolina State University in Raleigh. Enrollment was 82 students. Materials for the course are available from the instructor at the email listed above.

This paper is organized as follows. Section 2 briefly presents an overview of the course. Section 3 describes the target embedded computing platform. Section 4 describes optimization activities with performance analysis support. Section 5 describes code porting activities. Section 6 presents student feedback. Section 6 presents conclusions and reflections.

2. COURSE OVERVIEW

Table 1. Schedule of Lectures and Course Activities

Introduction to Embedded Systems and the QSK62P Plus
Using the LCD Analog and General-Purpose Digital Interfacing
Project #1: Analog and Digital Interfacing
Using Interrupts in C
Timers and Event Counters
Project #2: Using Timers
Polled and Interrupt-driven Serial Communication
Project #3: Serial Communications
Performance Analysis: Time Measurement and Profiling
Program Optimization
Fixed Point Math

Project #4: Run-Time Performance Optimization
Low Power and Low Energy Design and Analysis
Project #5: Energy Optimization
Understanding Software and System Impacts on Power Guest Speaker from Industry
Sharing the CPU: Scheduler Alternatives
RTC: Non-preemptive Dynamic Scheduler
Developing RTOS-Based Applications: Concepts, Task Management and Sharing Data Safely
Real-Time Scheduling Theory and Analysis

3. TARGET PLATFORM

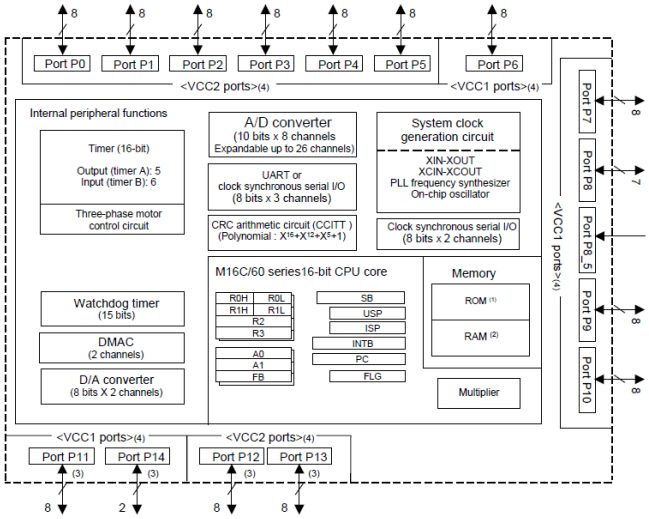


Figure 1. M16C/62P Group processor overview (Courtesy Renesas).

We use an embedded processor (M30620FCP) from the Renesas M16C architecture's 62P group[1]. This is a 16-bit CISC architecture with a 1 Megabyte address space. As shown in Figure 1, it features extensive peripherals for embedded interfacing, including digital I/O, five USARTs for serial communication, a 10-bit successive approximation analog to digital converter (with extensive input multiplexing support), dual 8-bit digital to analog converters (DACs), five 16-bit timers with pulse-width modulation (PWM) support, dual direct memory access controller (DMACs), multiple clock options, watchdog timer, and other useful peripherals. The processor runs at up to 24 MHz, with voltages from 2.7 V to over 5 V. Onboard memory is 10KB SRAM and 64KB flash ROM; this can be expanded off-chip to up to 1 MB.

The M16C architecture was chosen in part for the good tool support. The Windows-based HEW integrated development environment includes a C compiler with no optimizations disabled. Instead, application code size is limited to 64KB after a 30 day evaluation period. This size limit has not posed a problem in the over six years we have used M16C architecture processors, although we focus on systems without software libraries. Adding a networking stack, full graphic LCD support (e.g. with windowing) and FAT file system would likely use close to the 64KB available in the free tool

HEW also includes other useful tools, including static stack depth analysis support.

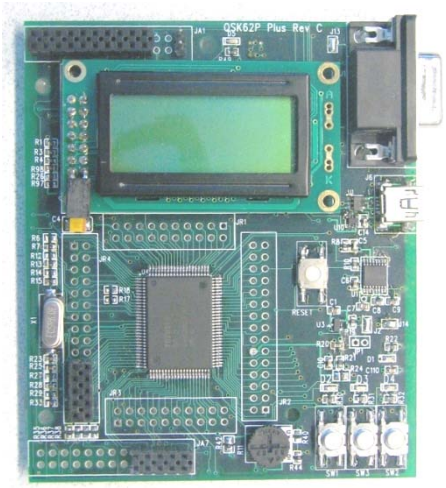


Figure 2. QSK62P Plus Quick Start Kit (QSK)

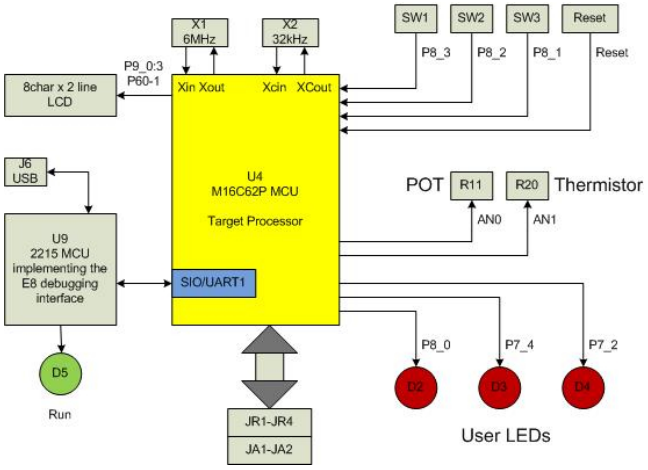


Figure 3. Block diagram of QSK62P Plus (Courtesy BNS Solutions).

The MCU is mounted on a Quick-Start Kit (QSK62P Plus from BNS Education[1]). Shown in Figure 2 and Figure 3, this board provides a 2x8 character LCD, potentiometer for generating an analog input voltage, three switches and LEDs for user applications, RS232 to logic level shifter, pads for a DB9 connector, thermistor, 6 MHz crystal, 32 kHz crystal, pads for expansion headers. On the back of the board is a debug controller microcontroller which communicates with a PC via USB. The board is powered by 5V from the USB connector, but only the LCD runs at 5V. The remainder of the board runs at 3.3V supplied by an onboard fixed linear regulator.

4. OPTIMIZING FOR TIME AND ENERGY

4.1 Motivation

Time and energy are limited resources for many embedded systems. Time may be limited due to deadline or throughput requirements interacting with high processor loads or constrained clock speeds. Energy may be limited due to powering by a battery or a renewable energy source.

Embedded systems differ from general-purpose computers in that the designer has control over multiple aspects of the system: application program, scheduler/operating system, compiler, communication behavior, user interface design and hardware design. This multidimensional design space typically offers many opportunities for optimization. This fosters "outside the box thinking" by viewing the whole system as capable of optimization, not just discrete and disparate pieces. Brainstorming discussions in class further promote this perspective.

The course teaches performance optimization with an example in lecture and two projects (one in run-time, one in energy use). Several fundamental messages are presented.

Be a slacker -- avoid all work possible.

- Avoid unnecessary conversions (type, units, etc.)
- Reuse results of computations
- Eliminate extra run-time work by precomputing data at compile time when possible

Do whatever remains as efficiently as possible.

- Start with an efficient algorithm and problem representation, and then work your way down into the details.
- Avoid double-precision math. Use fixed-point, single-precision float, or look-up tables.
- Use low-complexity algorithms.

Measure the impact of your optimization.

The whole-system view is coupled with a *rigorous, analytical, performance-driven approach*. After each optimization the performance impact is quantified, giving the students hands-on experience in determining what works well and what does not.

4.2 Run-Time Optimization

4.2.1 Performance Analysis: Profiling

The first step in optimizing any characteristic of a system is determining which part is most to blame. We use statistical program counter sampling as a simple mechanism to rapidly determine program a program's execution time profile across different functions. A timer peripheral is configured to periodically interrupt the processor at a configurable frequency. The timer's interrupt service routine examines the return address on the call stack and identifies the corresponding function or other code region using a linear search through a look-up table created from debug information at compile time. The ISR increments the corresponding function's count variable and then returns.

The students are provided with a gawk script which processes the map file and generates a region table.

An interesting side-lesson the students learn is that profiling incurs overhead. As they optimize the program more, the relative profiling overhead increases.

4.2.2 Project

In this project students optimize code which calculates the distance and bearing from an arbitrary position on the surface of the earth to the nearest of 163 weather and sea state monitoring station of NOAA's National Data Buoy Center[1]. The locations of these stations are stored in an array in the MCU's flash ROM. Locations are represented as latitude and longitude coordinates. Code to perform these calculations (using spherical geometry) is provided to the students as a starting point[4].

The students change and enhance the code in and called by Find_Nearest_Waypoint. Their grade depends in part on speed-up. The students are encouraged to read the specifications and determine which features in the sample code are gratuitous time-wasters which can be deleted. Several low-hanging fruit are provided.

Table 2. Example of Run-Time Optimizations

Optimization	Run-Time (seconds)	Speed-Up (x)	
		Incr.	Cumul.
Base program	19.78	n/a	n/a
Delete debug printf to LCD during search	17.17	1.15	1.15
Eliminate bearing computation during search	7.75	2.22	2.55
Replace PI/180 with constant	7.52	1.03	2.63
Precalculate radians, sine and cosine for fixed locations	4.15	1.81	4.77
Force compiler to treat doubles as single floats	0.11	37.73	179.82
Remove arcos and multiplication	0.04	2.75	494.50
Sort fixed location table by latitude, start at approx. position in table	0.061	0.66	324.26

Students achieved a broad range of performance optimizations, as shown in Figure 4. Maximum speed-up was 7,204.2x, while the geometric mean was 666.5x.

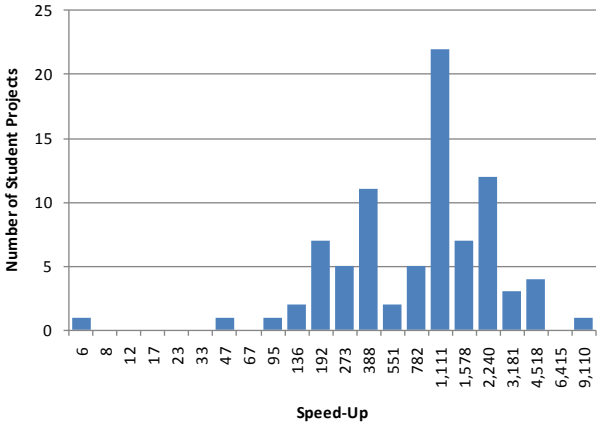


Figure 4. Distribution of Speed-Ups for Class

4.3 Energy Optimization

Battery capacity and thermal limits make energy and power optimization important for many computing systems, whether embedded or general-purpose. Students intuitively understand the importance of this due to their plethora of battery operated devices (laptops, phones, media players, etc.).

A motivational example is presented for the students in the form of an embedded industrial sensor with extremely tight power consumption constraints. The processor was an 8-bit microcontroller. The processor clock had to be scaled down from the rated 20 MHz to under 1 MHz. This in turn limited the available computation cycles, forcing the code to be optimized as much as possible. The design team determined that the 32-bit floating point math offered adequate accuracy for sensor data linearization but was too slow. So they rewrote the floating point math library provided by the compiler vendor to use a 24-bit representation.

Students are given background (in lecture and reading materials) on power and energy use of digital CMOS circuits. This focuses on the dependence of static and dynamic power on operating voltage and switching frequency. Students are then introduced to the power and energy saving features for the M16C-series processor as designed into the QSK62P+ board. Processor clock speed can range from 32 kHz to 24 MHz, using a combination of external crystals and a 4x phase-locked loop for frequency multiplication. The processor offers different power modes such as active, idle and stop. The clocks for different peripherals can be disabled (gated) or divided down to lower frequencies. Peripherals can be disabled (e.g. resistor chain for ADC). External and internal oscillators can be disabled or run in low power modes. These features are detailed in an application note on the subject for M16C processors.

4.3.1 Performance Analysis: Energy Use

We use a simple method to determine actual energy use: we measure circuit run-time when powerd by a supercapacitor. This eliminates the need to fabricate complex circuits to multiply current and voltage to find power, and then integrate power over time to determine energy.

Students use a 1 Farad supercapacitor charged to 4.7 V. The energy used during a period of time (e.g. computations, idle time) is determined based initial (V_0) and final (V_1) supercapacitor voltages:

$$E = \frac{C}{2}\left(V_1^2 - V_0^2\right) \quad (1)$$

The students construct the circuit in Figure 5 to power the QSK and charge the supercapacitor. The total current drawn from the USB port must be limited, but a simple series resistor would lead to excessive charging times. The circuit and associated control software keep the current to under 120 mA yet reduce charge time by reducing the resistance as the capacitor charges (as sensed by the microcontroller's analog to digital converter at node VCap_Div_2). Resistor R1 sets initial charging current. The current is then increased by asserting signals Stage1 and Stage2 which switch in R2 and R5 in parallel. Diode D1 allows discharge to bypass the resistors and can be a Schottky diode to reduce forward voltage drop.

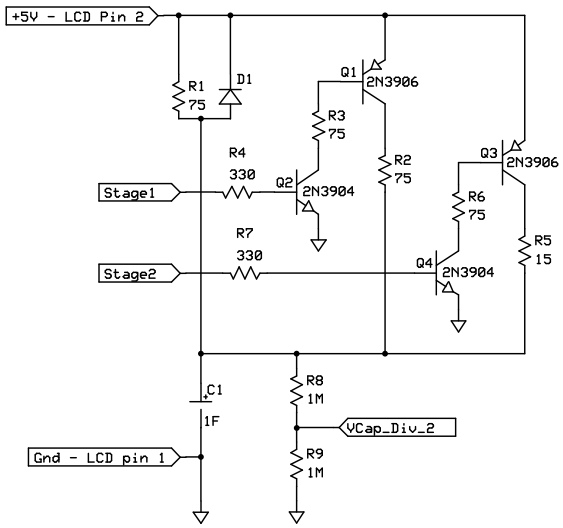


Figure 5. Supercapacitor charger and monitor circuit.

An interesting complication in this application is that as the supply voltage falls, portions of the board will stop working. First, the LCD requires 4.3 V to be visible. The 0.6 V drop across D1 makes the LCD unreadable when running off the supercapacitor, so students must use other means to evaluate program operation (e.g. LEDs, serial port). Second, the microcontroller and LEDs operate at down to 2.7 V. A supply voltage supervisor IC on the QSK (MCP120T-2701) holds the processor in reset when the supply voltage BOARD_VCC falls below 2.7V.

4.3.2 Project

In this project the students write and optimize code which periodically measures a thermistor's temperature and sends the formatted data out the serial port. Once a second the thermistor is sampled ten times. These samples are averaged and converted to a text string (e.g. "77.2 F") which is sent out the serial port at 57600 baud. Students optimize the code so that it uses less energy by (1) using the MCU's power- and energy-saving features, and (2) by optimizing the code run time by avoiding floating-point math (e.g. by using fixed-point math or a look-up table). The grading depend in part on how long the system operates and is curved based on the results of the entire class.

The initial non-optimized program runs for roughly 3 seconds with the 1 F capacitor. Students applied various optimizations; Table 3 presents one student's experiences.

Table 3. Example of Energy Optimizations

Optimization	Run-Time (seconds)	Energy Savings	
		Incr.	Cumul.
Base program	3	n/a	n/a
Switch CPU to wait mode when idle	100	33.3x	33.3x
Disconnect QSK power LED	150	1.5x	50x
Set all unused I/O ports to inputs	180	1.2x	60x
Convert ADC and UART code from polled to	200	1.1x	67x

interrupt-driven			
Disable peripheral clocks while processor is in wait mode after UART transmission finishes	250	1.25x	83x
Disable PLL when CPU is idle	275	1.1x	92x
Place CPU in wait mode during ADC and UART operation	280	1.02x	93x
Use UART transmission completion interrupt	290	1.04x	97x
Use 32 kHz clock instead of 24 MHz main clock when idle	300	1.03x	100x

Students achieved a broad range of performance optimizations, as shown in Figure 6.

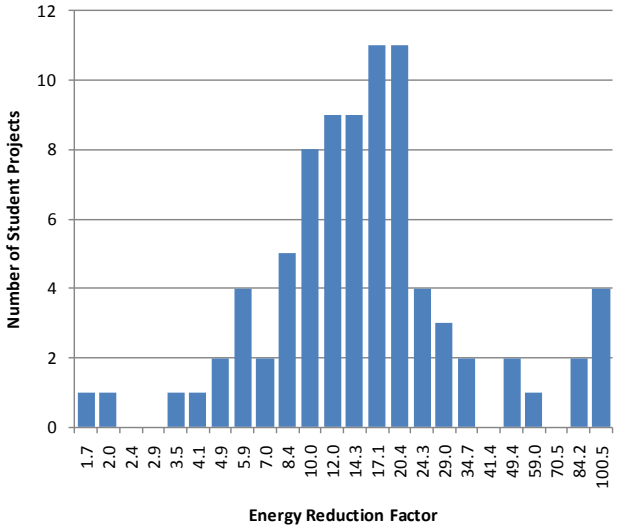


Figure 6. Maximum energy reduction was 100x, while the geometric mean was 14.3x.

.

5. PORTING CODE

A common experience in industry is porting existing code to a new processor architecture. This may be due to a processor upgrade or reuse of code from a different project within the company or open source software. To give students experience in this domain we have a code porting project which gives reverse-engineering experience for both hardware and software. The students choose between porting code to interface with either a microSD card or a color graphical TFT LCD.

5.1 MicroSD FAT File System Porting

The first option involves porting code to support microSD cards with the FAT file system. A serial communication interface (SPI) is used to interface with the microSD card, though there are faster parallel modes available.

The students must write a program which lists all the files in a directory, displays the contents of a selected text file, and creates a text file. We use the Embedded File System Library [5] (EFSL), which targets the AVR 8-bit architecture and the ARM7 32-bit architecture. In order to port EFSL successfully, the students must add SPI communication support (by configuring a UART to operate in synchronous mode) and change the source code slightly, and modify compiler settings.

The hardware interface between the MicroSD card and the MCU is quite simple. There are power (3.3V) and ground and four SPI signals (clock, chip select, data in and data out).

A challenge which students must address is that there is a large array (512 bytes) declared as an automatic variable. Its size exceeds the 255 byte limit for the default compiler settings. The students either declare the variable as static or change a compiler option which supports additional addressing modes, removing this restriction.

Students then added C Standard I/O support to the system, using the EFSL API. This involved creating fopen and fclose functions which return FILE object pointers. The remaining stdio functions can then be used without modification.

Table 4. Example of MicroSD File Read and Write Speeds

Operation	Base time (us)	Time per Byte (us)
Read – direct EFSL	8718	4.35
Read – C Standard I/O	8871	127.9
Write – direct EFSL	8041	4.01
Write – C Standard I/O	7975	149.1

Students then evaluated the speed of read and write operations of both the raw EFSL and the C Standard I/O interfaces. Reads and writes of data blocks of various sizes were timed and analyzed, as shown in Table 4. The C Stdio interface is much slower than EFSL because it performs reads and writes individual bytes rather than performing block operations.

5.2 TFT LCD Driver Porting

The second option involves porting code to support a 320x240 pixel, 18-bit per pixel color thin-film transistor LCD[6], which is controlled by an SD2119 controller IC [7]. The controller supports several interface options: 8 bit parallel modes, 18 bit parallel, and serial SPI. The display module includes a four-wire resistive touch screen. Students must create an application to fill the entire screen with a single color, set a pixel to a specified color, and draw a line between given endpoints. Students must also solder a two fine pitch surface-mount connectors to interface with the LCD and touchscreen. The vendor (CrystalFontz) provides C source code targeting the AVR architecture. Students port this code to the M16C architecture, add graphic rendering code (line drawing and screen fill) and then measure the performance of the system. Some students optimized the performance of their rendering code using the same techniques in the run-time optimization project.

6. STUDENT FEEDBACK

Students were encouraged but not required to submit anonymous feedback on the course. 15 students responded, with results presented below. Some students made multiple suggestions, so counts do not total 15.

Table 5. Student Feedback on Successful Aspects of Course

Count	What worked best in the course?
8	Projects
1	Using chalkboard
1	Simple enough projects, built up to final project
1	In-class examples
1	Range of freedom in projects, flexibility in how to achieve results
1	Teaching style
1	Everything
1	Enthusiasm of instructor

Table 5 shows what the students liked most about the course -- the projects! Another factor was the use of the chalkboard rather than powerpoint slides whenever possible. This kept the students more involved in the class.

Table 6. Student Feedback on Problematic Aspects of Course

Count	What worked worst in the course, and how would you fix it?
2	Embedded coding style should be taught
2	Too much pre-written code
2	Cover less introductory material, assume students will be able to read manual
2	Should include project based on RTOS
1	Complex hardware diagrams drawn on chalkboard were hard to copy
1	Introductory projects were too simple
1	Compare with other MCUs
1	Should include description of MCU's memory structure
1	Too much material covered in course
1	Didn't like complexity of soldering/wiring, wanted to emphasize coding instead
1	Homeworks not needed
1	More strict project grading (too many A+ grades!)
1	Don't do in-class programming exercises - just explain in class and then make them homework
1	Improve mechanics of electronic component distribution
1	Less code demonstration
1	More focus on embedded system design
1	More hands-on interaction by students.
1	Use a higher-end processor (Beagleboard with Linux)
1	Talk slower, leave time for students to transcribe

Responses about how to improve the course are much more disparate.

Some students suggested reducing the amount of time spent on introductory microcontroller material, and modifying the overly-easy introductory projects. These components were built into the course because the majority of the students had no experience with the M16C architecture and development toolchain. A point to take away from this is that it is appropriate to challenge the students to come up to a new processor and toolchain without excessive handholding.

Two students requested an RTOS-based project. RTOS coverage in the class consisted of lecture and hands-on demonstrations. An RTOS-based project was planned but dropped after the schedule slipped excessively.

Another student recommended using a higher performance processor running an OS (e.g., Beagleboard running Linux). This reflects the extremely broad spectrum of embedded systems in the filed.

7. CONCLUSIONS

The course was effective in teaching optimization and porting techniques. Students gained an in-depth understanding of these topics through their hands-on projects. The competitive nature of the optimization projects pushed students to excel, leading to some truly remarkable improvements.

8. ACKNOWLEDGMENTS

The author thanks Renesas and BNS for equipment donations and other support. This work was funded in part by NSF CSR-EHS award 0720797 and ERC program award number EEC-08212121. The instructor is thankful for the contributions of the students in ECE 561 and ECE 492-13, as well as teaching assistants Josh Fisher and Hayden Stewart.

9. REFERENCES

[1] M16C/62P Group (M16C/62P, M16C/62PT) Hardware Manual

[2] www.bnssolutions.com/QSK

[3] http://www.ndbc.noaa.gov/ and http://www.ndbc.noaa.gov/cman.php

[4] Chris Veness. http://www.movable-type.co.uk/scripts/latlong.html.

[5] Embedded File System Library, http://efsl.be/

[6] http://www.crystalfontz.com/products/320240f/datasheets/1936/CFAF320240FTTS.pdf

[7] http://www.crystalfontz.com/controllers/SSD2119.pdf

Support of Android Lab Modules for Embedded System Curriculum*

Meng-Ting Wang[1], Po-Chun Huang[2], Jenq-Kuen Lee[1], Shang-Hong Lai[1], Roger Jang[1], Chun-Fa Chang[3], Chih-Wei Liu[4], Tei-Wei Kuo[2] and Steve Liao[5]

[1] National Tsing-Hua University, Hsin-Chu, Taiwan. [2] National Taiwan University, Taipei, Taiwan.
[3] National Taiwan Normal University, Taipei, Taiwan. [4] National Chiao-Tung University, Hsin-Chu, Taiwan. [5] Google Inc.

ABSTRACT

Technologies for handheld devices with open-platforms have made rapid progresses recently which gives rise to the necessities of bringing embedded system education and training material up to date. Android system plays a leading role among all of the open-platforms for embedded systems and makes impacts on daily usages of mobile devices. In this paper, we present our experience of incorporating Android-based lab modules in embedded system courses. Our lab modules include system software labs and embedded application labs. The Android embedded application lab modules contain computer vision, audio signal processing and speech recognitions, and 3D graphics materials. Lab modules for Android systems in embedded system software cover topics on embedded compiler, HW/SW co-design, and power optimization. We also illustrate how these laboratory modules can be integrated into embedded system curriculum. Feedbacks from students show that these laboratory modules are interesting to students and give them essential training of adopting Android components for embedded software development.

Categories and Subject Descriptors

K.3 [**Computing Milieux**]: Computers and Education;
D.4 [**Operating Systems**]: Software;
C.5 [**Computer System Implementation**]: Computer Systems Organization

General Terms

Embedded Operating System, Multicore System, Open-platform

1. INTRODUCTION

*The correspondence author is Jenq-Kuen Lee. Author's address: Department of Computer Science, National Tsing Hua University, Hsin-Chu 30013, Taiwan; email: jklee@cs.nthu.edu.tw.

As technologies for handheld devices with open platforms have made rapid progresses recently, open-platforms such as Android and MeeGo are getting momentum. Mobile devices with microphone and speaker, video camera, touch screen, GPS, etc, are served as sensors for experiencing with augmented reality in human life. The prosperity of embedded system applications on open-platforms give rise to the necessities of bringing traditional materials of embedded system education up to date to help the students learn to design mobile applications and gain experience of embedded software development for handheld devices. This paper presents our experience of incorporating laboratory modules based on an open-platform, Android, in devising innovative embedded system courses.

We coordinate with Embedded Software Consortium (ESW Consortium) to develop the Android-based embedded system curriculum. The ESW Consortium is funded by the Ministry of Education, Taiwan in 2004 with a mission of embedded system course development, promotion, and evaluation for computer science and electrical engineering programs in almost all universities in Taiwan. We used a two-phase scheme for Android-based embedded system curriculum development which is shown in Figure 1. The two-phase plan including the course development phase and promotion phase. During the development phase, the Android-based embedded system curriculum is devised by responsible course development team members and underwent course trial runs in the developer's universities respectively. All of the lab modules are further peer-reviewed to make sure that they can be conducted successfully. In the promotion phase, curriculum promotion workshops are held to help spread the education of embedded system development techniques for Android platforms. After the curriculum development reaches a certain level of maturity, the resulting course materials would be contributed to a web repository and donated to the Google Code University with Creative Commons License.

In this paper, we present our Android-based lab modules for embedded application and system software development. Lab modules for Android in embedded applications include computer vision, audio signal processing and speech recognitions, and 3D graphics materials. These are lab modules to illustrate the design flow for writing applications such as augmented reality for embedded devices. The first lab

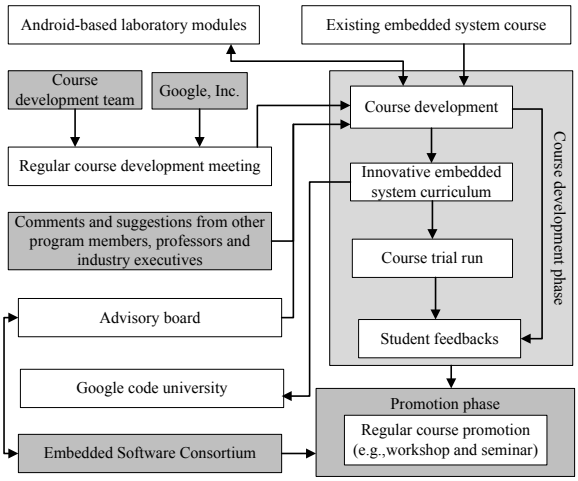


Figure 1: A two-phase Android-based curriculum development

module introduced is the embedded computer vision module focusing on the development of a face recognition, mining, and synthesis system on Android platform. Second, the course module related to audio signal processing and recognition aims to help the students become familiar with the design flow of implementing speech technologies within Android application framework. In the 3D graphics module, we focus on the rendering of high-quality lighting effects, which is as known as global illumination, on Android platforms. Next, lab modules for Android systems in embedded system software cover the topics on embedded compiler, HW/SW co-design, and power optimization. Due to the fact that the progresses of open-source platforms mainly rely on software communities, our system software lab modules are designed in a way to explain the design flow of further embedded system optimization so that the students can gain the abilities to make contributions to the open-source platforms. In the embedded compiler lab modules, the students learn how to optimize garbage collection algorithm for Android Dalvik virtual machine (Dalvik VM). Another lab module covering code size optimization topics teaches the students to configure and customize the Android systems libraries. Next, the HW/SW co-design module employs a hardware-accelerated multi-tasking coprocessor interface as a case study for HW/SW co-design flow for the Android system. Then, the power optimization lab module, instructs the students in energy consumption reduction for applications. In addition, we also illustrate how these laboratory modules can be integrated into embedded system curriculum. Feedbacks from students show that these laboratory modules are interesting to students and give them essential training of adopting Android components for embedded software development. Initial tryout of teaching and feedbacks from students show that these lab modules are interesting to the students and certainly give essential training to students of adopting Android components for embedded software development.

In Section 2, we give a brief introduction to Android platform framework and the software development flow within the Android framework. Next, Section 3 presents our Android lab modules. Section 4 discusses the use of these lab modules for existing embedded system curriculum. Next, course evaluation and comments from the students after trial runs of the embedded system courses with our Android-based lab modules are analyzed in Section 5. Finally, Section 6 concludes this work.

2. BACKGROUND WITH ANDROID FRAMEWORK

Android is an open-source mobile phone platform developed by Google and Open Handset Alliance. Within the Android framework, software designers can customize the mobile phones with their innovative ideas according to user experience and preference [15]. Android can be regarded as a software stack consisting of five layers, the bottommost Linux kernel, middleware libraries, Android runtime, application framework, and the applications. The Linux-based kernel layer provides system services such as memory management, process management, and device drivers which enable the access of the underlying resources for system software. This layer serves as the Hardware Abstraction Layer (HAL) to reduce the complexity of access of the hardware device resources. Next, the middleware library layer provides a set of libraries with various functionalities such as the Open Graphic Library (Open GL) [10], standard C library(libc), and the media framework. A majority of the middleware core libraries are written in C/ C++ language while Android applications are developed in JAVA. Therefore, applications access the resource through Java Native Interface (JNI). The next layer is the application framework layer which simplifies the reuse of applications by defining a common Application Programming Interfaces (APIs) for all kinds of applications, for example, the on-screen Graphical User Interface (GUI) components and the application-specific data convention. Finally, the topmost applications layer contains a set of core application which are written in JAVA. To make the Android software development more convenient, the Android system are equipped with the Android Software Development toolkit (Android SDK) [11] which contains tools to help the developer set up the development environment, profile the program, and manage the user interface of the software, such as Ant build tools, Android Virtual Device Manager, and Android emulator. In addition, Android Native Development Toolkit (Android NDK) is provided for developers so that they can build native code for performance-critical programs. During the development, programmer can verify their design using the Android emulator and the Android Debugger (ADB) provided in Android development toolkits.

Within the Android application framework, each application works on a dedicated Dalvik VM. Thus, the Android applications are basically developed in Java language within the Android application framework so that they can be converted in to Dalvik bytecode. In our Android-based application lab modules, the students are also guided to access the related C/C++ libraries through JNI for efficient development of the Android application. Android components such as libc, content providers, view system, package manager, and resource manager are utilized for software development in the lab modules; each lab module is related to a certain number of specific Android system components. This will help train students to develop their own applications on

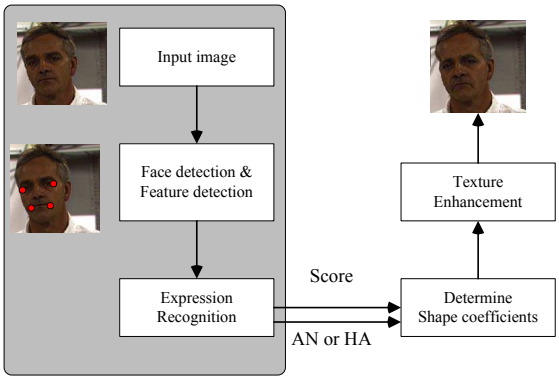


Figure 2: The RMS system operation flow

Android phones. Computer vision lab module adopts the display driver, camera driver, and the media framework of Android system. The audio signal lab module employs both audio driver and media framework to support the speech recognition operation and 3D Graphics lab module makes use of API in OpenGL and SGL. Similarly, system lab modules are designed to guide students to be able to further optimize system performance and power. The infrastructures of Android system software are revealed by lab modules including garbage collection(GC) optimization for Dalvik VM and code size optimization for tuning GCC options for Android applications or systems. The HW/SW co-design lab module utilizes the binder driver and the Android runtime modules to deal with inter process communication (IPC). The dynamic power management lab module references power management middleware and the Dalvik VM to carry out system optimization idea.

3. EMBEDDED ANDROID LAB MODULES

Android encourages application development for customizing mobile devices by providing practical APIs and middleware libraries. We design a set of lab modules for students with the aim to give them hands-on experience on Android platforms. Our lab modules include system software labs and embedded application labs. We will describe each of them in this section.

3.1 Embedded Android Application Lab Modules

Android application lab modules include three major topics, embedded computer vision applications, speech technologies, and 3D Graphics and Game APIs. These are lab modules to illustrate the design flow for writing innovative applications for mobile phones under Android platforms.

A. Embedded Computer Vision Applications on Android Platforms

• Related Android Component

Android SDK, Android NDK, Display Driver, Camera Driver, Media Framework

• Lab Outline

In embedded computer vision applications lab modules, we set the goal to develop Android applications using the lightweight Recognition, Mining, and Synthesis schemes (RMS), which is developed in C++ language. The face RMS system will focus on the eigenface-based face detector, facial interest point detector, facial expression recognition by using Support Vector Machine (SVM), and facial expression exaggeration by image deformation warping from expression optical flow analysis [5]. Figure 2 depicts the RSM operations, from input image parsing, face and feature detection in the input image, and recognition of the facial expression. The facial expression recognition scores the image to determine the shape coefficients in order to conduct further texture enhancement [16]. Moreover, face image mining and imposing map information on street images are also applications students can experiment with this lab module. In this lab, the students first learn to develop the RMS system on a PC, and then port the system onto the Android system to increase the portability of the applications. The RMS system can be applied to both face image mining and street scene matching. Once the face object and the street scene is matched with instance within the database, informations can be imposed to the input photo. For instance, the 3D viewing direction and the location information can be added to screen scene pictures.

B. Speech Technologies for Embedded Systems

• Related Android Component

Android SDK, Android NDK, Audio Driver, Media Framework

• Lab Outline

The Speech Technologies for Embedded Systems module aims to assist the students in getting familiar with the design manipulation of implement speech technologies under Android application framework. The students are expected to work on projects in groups. The projects require the students to come up with innovative speech and audio applications on Android platforms which can be well exemplified by speech interface for name dialing, query-by-singing interface for song database search, and voice-activated personal verification systems. The development of these applications in labs needs the support from the Speech Recognition library written in C and C++ language which we prepared before running the class [7]. The library was turned into shared object files using the Android NDK. Through JNI, the students can access the shared object files to manage the operation sequence form reading data from the specified wave file, conducting speech recognition, and displaying the result in the labs.

C. 3D Graphics and Game APIs on Android Platforms

• Related Android Component

Android SDK, Android NDK, OpenGL, SGL

• Lab Outline

The photo realistic rendering of 3D scenes is a focus of this lab module. Issues like various options to display the scenes and the method to increase the frame rates on thin clients such as mobile devices are discussed. We introduce the 3D Graphics and Game APIs lab modules on Android platform to give students tutorial of the technologies mentioned above. Learning from development of 3D games, the students would gain the ability to design the 3D graphics and gaming APIs for the Android platforms in these lab modules. In addition, certain lab modules are designed to demonstrate the workload partition of the traditional 3D rendering for cloud computing models which include handheld devices as

clients. A typical example is to employ cloud computing techniques for pre-computed radiance transfer (PRT) which is used in 3D graphics for dynamic lighting changing [14]. In this case, heavy pre-computation could be done in the cloud while the final rendering is done on the client side. Students learn to manage the client side final rendering operations on Android platforms from this lab module.

3.2 Embedded Android System Software Lab Modules

In addition to the applications, lab modules for Android systems in embedded system software cover the topics on embedded compiler, HW/SW co-design, and power optimization. The software lab modules are designed in a way to explain the design flow of embedded system optimization so that the students can gain the abilities to make contributions to the open-source platforms. These lab modules are descrbed as follows.

D. Garbage Collection Optimization of Dalvik VM

• Related Android Component

Android SDK, Android Open Source Project (source code), Core Libraries, Dalvik VM

• Lab Outline

Among various system optimizations methods, garbage collection (GC) is a important part of memory management deallocating the occupied memory space of objects which are no longer in use. In this lab module, we illustrate ways for possible enhancements with GC in Dalvik VM. For example, as Dalvik bytecode is with registers of no type information, Dalvik VM suffers from the penalty with GC. For example, when a memory space is not used anymore while a register contains a non-pointer data whose value happens to be the address of the memory space, the system can not conduct GC precisely to reclaim the space because it has the misunderstanding that the memory space is still in use. We illustrate how to devise a binary-code analyzer for performing type-precise scan and live reference analysis for GC in Dalvik VM [8]. With understandings of internal designs of GC in Dalvik, students will be able to devise further optimizations schemes with GC under Dalvik framework. Figure 3 illustrates the flow of our GC optimization scheme in this example with lab module. The system obtains register data types of methods included in the Dalvik executable files after the executable files undergo static data flow analysis. With the register data type information, Dalvik VM is able to run precise GC operation during run time.

E. Embedded Compiler Code Size Optimization for Android

• Related Android Component

Android SDK, Android emulator, ADB, Android Virtual Device, ARM EABI

• Lab Outline

Subject to the limited storage of embedded systems, code size has been a critical issue. Compilers are equipped with numbers of options that can reduce the resulting binary code size [3]. Options like"-O2","-O3" and"-Os" are actually combinations of several options with different influences on the binary code, such as frame pointer elimination, variable visibilities, and function section adjustments. However, interactions between different options cause uncertainties code size. We design this lab module to teach the students these concepts and develop a code size optimization tool for the lab.

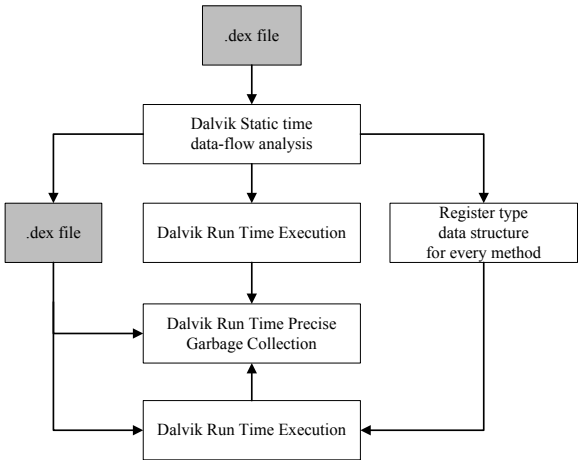


Figure 3: Garbage collection optimization for Dalvik VM

This tool gives suggestions on compiler option combinations which lead to optimal or suboptimal binary code size with performance concerns.The optimization flow of the tool is shown in Figure 4. To compile the input program under consideration of code size and performance, the tool compares the results of compilations with different option combinations. The genetic algorithm is employed to search within different option combinations for the best result [4]. The search begins forming a baseline set which is determined in experiments during development of the tool. The baseline set contains the options which makes major impact on code size for benchmarks such as EEMBC and MiBench. In Android code size optimization lab modules, the students can get hands-on experiences of the tool, verify the result binary using Android emulator, and then discuss on the results of distinct option combinations. Besides, the students can learn to tune the parameters in genetic algorithm to reach a balance between code size reduction ratio and the searching time.

F. HW/SW Co-Design Flow on Android Multi-core

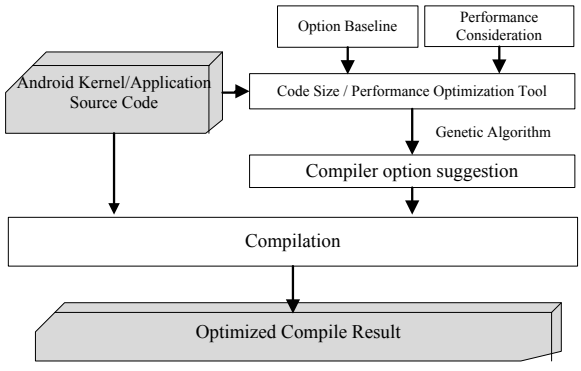


Figure 4: Code size optimization for Android system software and applications

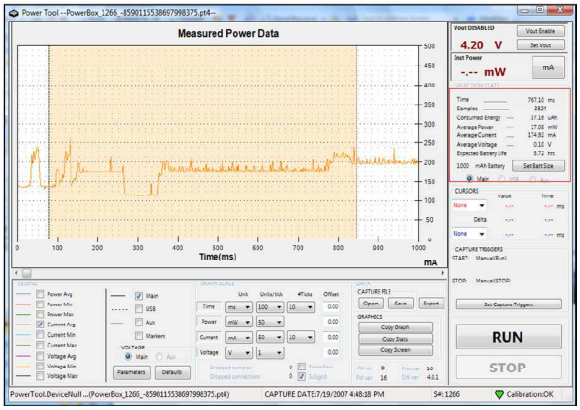


Figure 5: PowerBox, the power management tool

Platforms

• Related Android Component

Android SDK, Android NDK, Linux Device Driver, Binder (IPC) Driver, ARM Developer Suite v1.2

• Lab Outline

This lab module was designed with the major goal to teach the students how to follow HW/SW co-design flow in the presence of the Android platform. Through experiments, students will learn and develop a low-power design flow in a HW/SW co-design mechanism. After the training, students would be familiar with ARM SoC platform [18]; to be more specific, ARM Versatile board and the Android system as our target platform [2]. The students would fist requir to set up the configuration in the Android source code with the Versatile hardware information and then develop a JPEG encoding program in the system. There are two ways to implement the operation, either spread the workload of the MPU to other DSPs or simply running the operations on the master processor. Further, the students are guided to refine the program following the HW/SW co-design flow to make the operations fast and with low-power consumption.

G. Dynamic Power Management and Voltage Scaling on Android Platforms

• Related Android Component

Android SDK, Power Management, Core Libraries, Dalvik VM

• Lab Outline

For mobile phones, power management is critical in the light of battery life of the device. Also, as the emerging of many applications that rely on heavy data communication, the knowledge about dynamic power management of the communication modules (such as 3G and WiFi) is required [6]. The goal of this lab module is to maximize the lifespan of the cell phone under the given periodic task sets [12]. In the lab, the power management tool, PowerBox shown in Figure 5, is adopted to monitor the instantaneous current of the device. Multi-core issues are considered by the students in the lab. Students are requested to design and implement intelligent energy-efficient policies, and then demonstrate the effectiveness of their dynamic voltage scaling (DVS) designs. This lab module contains two project assignments from which students learn to turn on and turn off the backlight of mobile phones in an on-demand mechanism with the supports from Android power management libraries and also try to design dynamic backlight scaling algorithms with real-time and energy-efficient considerations.

4. ANDROID-BASED EMBEDDED SYSTEM CURRICULUM

In this section, we illustrate how these laboratory modules can be integrated into existing courses in embedded system curriculum. These courses are courses offered either in Taiwan Embedded Software Consortium (ESW Consortium) or in the department the professor developed this lab module. Table 1 gives a summary of these courses. We will briefly describe below each course and their uses of Android lab modules developed.

4.1 Embedded Computer Vision Course(ES-Y09-N1)

Major topics of computer vision course are about the theory supporting image information abstraction in artificial systems. Contents of computer vision course cover the knowledge ranging over geometric operations and image processing techniques like the eigen-face detection approach, segmentation, object recognition. In addition to conventional materials for embedded computer vision course, this course also gives the students computer vision application examples like biological vision and the image Recognition, Mining, and Synthesis schemes(RMS).

We use the RMS lab module to introduce practical applications in computer vision field to the students. The face RMS system consists of face detection, facial expression recognition, and facial expression exaggeration components; we explain algorithms corresponding to each of the three operations to the students and then teach them to implement the applications in C++ programs on a PC platform first and port onto the Android platform afterwards. Students are trained with two face-related RMS implementations on Android-ready multi-core platforms to gain knowledge about the optical flow of images. For the facial expression exaggeration, we focus on the image warping technique to synthesize facial expression for a face image. For the face image warping, we use the optical flow fields computed from the training face images as the warping function. The students will learn how the optical flow is computed from images and how it can be used to warp images. “Facial expression exaggeration” experiments also emphasize the details of input face images or the synthesis of cartoonized face images. The other application is “automatic background substitution”, which is to segment the face and upper body region from the background region and then replace the background image. The implementation in the labs is corresponding to the course topics about face detection, facial expression recognition, and exaggerated facial expression synthesis.

4.2 Audio Signal Processing Course(ES-Y09-N2)

This course covers basic methodologies in audio signal processing and recognition, including fast Fourier transform (FFT), Mel-frequency cepstral coefficients (MFCC), Gaussian mixture model (GMM), dynamic time warping (DTW), and hidden Markov model (HMM) [17]. Specific applications based on these methodologies are also introduced, such

Table 1: Embedded system curriculum with corresponding Android lab modules

Course Category	Course Number	Course Name	Incorporated Android Lab Module
Application courses	ES-Y09-N1	Embedded Computer Vision	• Embedded Computer Vision Applications on Android Platforms
	ES-Y09-N2	Audio Signal Processing	• Speech Technologies for Embedded Systems
	ES-Y09-N3	3D Graphics	• 3D Graphics and Game APIs on Android Platforms
System software courses	ES-Y05-2	Embedded Compiler	• Garbage Collection Optimization for Dalvik VM • Embedded Compiler Code Size Optimization for Android
	ES-Y08-3	HW/SW co-design	• HW/SW Co-Design Flow on Android Multi-core Platforms
	ES-Y03-1	Real-Time Systems	• Dynamic Power Management and Voltage Scaling on Android Platforms

as endpoint detection, pitch tracking, speaker recognition, query by singing or humming, query by tapping, speech recognition, speech assessment, text-to-speech synthesis. We also prepare elaborating course website with abounding online resource [1].
In the incorporated lab module, the students are guided to develop speech application program interface for name dialing, query-by-singing for song database search, and voice-activated personal verification systems. The laboratories can help the students map abstract course topics about audio signal processing to practical implementation and evolve the abilities to build up useful audio signal processing software using the Android components.

4.3 3D Graphics Course(ES-Y09-N3)

In 3D Graphics course, there are three major topics covered: general 3D techniques, state-of-the-art of GPU-based lighting effects and 3D graphics on Android. At the beginning of this course, Open GL is introduced to the students since the OpenGL API plays a key role in many 3D graphics implementation. Next, this course discusses common 3D graphic issues like global illumination, texture mapping, programmable shaders, 3D graphics pipeline, and rasterization. Examples of OpenGL program illustrate these concepts to the students.

The cooperating lab module of this course, 3D graphics and game APIs on Android platforms, help the students fully understand the GPU-based lighting effects includes environment lighting, indirect lighting, refraction and caustics, and volumetric effects. The rendering of high-quality lighting effects on mobile platforms (as known as global illumination) is one of the key points throughout the 3D Graphics course and our 3D graphics lab is dedicated to this theme. With concept of global illumination and the current GPU-based implementation of various 3D effects, students are trained to develop an OpenGL program to keep trace of the execution path all the way from the Java code, JNI, native C code to the low-level source code of Android OpenGL.

4.4 Embedded Compiler Course(ES-Y05-2)

Embedded compiler course introduces the system optimization schemes of embedded compilers. It includes data dependence analysis, data flow equation, classical loop optimizations, compilers for embedded DSP processors, compiler for low-power, VM, GC, and JIT, compiler for code size reduction, and performance analysis. GC lab module can be used when GC and VM section are discussed. In addition, Lab module for code size reduction can be used when one discusses the section related to compiler for code size. For the GC lab module, GC is introduced as a form of automatic memory management which makes the memory utilization more efficient. Algorithms to solve GC problems like memory leakage, dangling pointer, and double free bugs are discussed. As an example, the GC algorithm of Dalvik, "mark and sweep", is introduced to the students. Students are expected to understand how GC works and then devise schemes to further optimize the GC in Dalvik environments.

For the yopics on code size reduction, we include classic code compression and compaction methods study and introduction of existing code size optimization schemes like procedure abstraction, variable visibility handling, and dual-width instruction set architecture. To establish background knowledge for students, a certain number of examples of code size reduction scheme are employed to illustrate principles for code size optimization. As one of the most common used code size reduction scheme, the mechanism of optimization options for code size reduction provided by GCC is an emphasized topic in this embedded compiler course. In the code size optimization lab module, students can observe the code size reduction resulted from GCC code size optimization options. Further discussion about experiment result in the lab can inspire the students the operations and effects of compiler options. Moreover, the code size reduction tool used in the lab adopts genetic algorithm to search for the best optimization option combination, so the students can also learn to design an instance of genetic algorithm to help embedded system optimization.

4.5 HW/SW Co-design Course(ES-Y08-3)

This course mainly focus on the framework of HW/SW co-design of embedded systems. Process management and interaction between procedures are introduced to the students first. Students learn about how tasks divergence impact the operations in embedded systems and the controller utilization of the reactive procedures. Heterogeneous multi-core system on chip issues are our major concern in this course. With the basic knowledge of the interaction between hardware and software in embedded systems, the students can gain insights into how software provides features and flexibility to an application, how the hardware is designed under consideration of performance, and most important of all, how to join the design flow of hardware and software for a better application design.

In this course, we introduce how HW/SW co-design helps with the enhancement of task management of a RISC architecture combined with a DSP processor since that tasks divergence in most embedded systems, especially for heterogeneous multi-core system, sometimes causes inefficient

task management and inter-processor communication (IPC) overhead [9]. To help the students learn from practical instance, we introduce the framework of Android to the students first. Then in the incorporating lab module, HW/SW co-design Flow on Android Multi-core Platforms, the students are guided to program with performance concerns for heterogeneous dual-core platforms using the hardware accelerated streaming remote procedure calls, which can be seen as a multi-tasking coprocessor interface. The students can gain the ability to handle the whole HW/SW co-design flow and the stream controller programming for multi-DSP environment from the course and the lab module.

4.6 Real-Time System Course(ES-Y03-1)

The limited resource of mobile devices poses the necessity of system and application programs to reduce the energy consumption of applications [13]. In real-time system course, we first give a brief introduction of embedded system architecture and the underlying hardware. The students can gain basic knowledge of peripherals, memory, interfacing, hardware architecture thorough the lectures. In addition, topics about essential components and features of embedded real-time operating system are covered by this real-time systems course.

The students learn the scheduling policies, hardware resource control and inter procedure communication concepts from existing embedded real-time system examples.In particular, we managed to let the students learn how to design and implement real-time energy-efficient policies adapting to different behavior type of applications. DPM and DVS techniques are the major topic in this course. We use the dynamic power management(DPM) and voltage scaling (DVS) on Android platforms lab module to help the students fully understand the concepts of real-time system DVS. Through the lab, Android power management libraries and associated Android device drivers assist the students in implementation of subjects in this course.

5. EDUCATIONAL EFFECTS AND ACHIEVEMENTS

5.1 Statistical Data Analysis of Course Evaluation

After the course development reached a certain level of maturity, trial runs of the newly-devised courses were held respectively in National Taiwan University, National Chiao-Tung University, National Tsing-Hua University, and National Taiwan Normal University. In order to evaluate the educational results of the Android-based embedded system curriculum, when the course trials were completed, we conducted an appraisal survey about the educational results and achievements through questionnaires for students.

According to the feedbacks received form the 3D graphic course, embedded compiler course, and real-time system course, 82% of the students stated that after taking the courses, they understood the framework of mobile platform systems. 60% of the total strongly agreed on that the Android SDK and the NDK certainly offered a user-friendly Android application development environment by providing most of the necessary Android components for the laboratories, whereas 18% of the students thought handling the operations through the SDK and NDK are not conveient enough. Overall, 75% of the total said that they made a great effort to implement the lab requirement and learned from the well-prepared course materials. All of the students are willing to do further studies about the Android SDK/NDK. More than 65% of them showed great enthusiasm for more researches about software development for mobile devices, which is exactly one major objective of incorporating Android-based laboratories into embedded system curriculum. In other words, according to the careful analysis of obtainable resulting data shows that the lab modules we designed for embedded system courses enhancement certainly offer competent support for devising innovative embedded system curriculum to help students to get hands-on experience of Android.

5.2 Feedbacks from Students

After the course trial run, we receive comments from the students taking the 3D Graphics and the real-time system course which shows that the Android-based lab modules substantially aid embedded systems education by connecting the theoretical knowledge and the practical implementation. The goals of the lab modules is achieved; the lab modules certainly help the students evolve the ability to independently develop applications and system software with the supports form related Android components. Some instances of the feedback comments are list bellow.

Comments For 3D Graphics Course

• The most different part of develop gaming application for mobile phone is that the operation were designed on the account of IO interfaces provided by the device, e.g., touch screen and vibration , rather than the standard IO. I think this is the main factor in making the course an outstanding one.

• In this course, my group came up with a pinball game. The 3D pinball game application can be partitioned into 4 modules, UI, dynamic object module, data center and render engine. The UI implementation was relatively simple due to supports form Android SDK while it took our great efforts to design the render engine part which needed 3D graphics techniques like the texture mapping and illumination estimation.

• To develop the 3D game for the term project, we adopted Eclipse IDE because it can cooperate with Android SDK seamlessly. Eclipse not only simplified the access of resources in the SDK (such as UI layout and the application service management), but also provided a handy interface for the Android Virtual Device management. The development was quite straight forward once the Eclipse and the Android SDK were settled down.

Comments For Real-Time System Course

• The Android architecture made me wonder if the software stack complexity affects the performance of the Android system and in the lab, I realized that optimizations are necessary indeed. In the term project of this course, we tried to change the processor clock rates by using clock-rate-tuning functionality provided by the Kernel layer of Android in or-

der to lower the power consumption. The Android-based labs bridged my knowledge about the real-time system and the practical implementations.

• At the very first beginning, we tried to derive performance optimization by decreasing the delay of file system access caused by Android software stack overhead but it turned out to be in vain. We found out that directly access of the file system in Linux environment only contributed 10% performance improvement in this case. Although our first method is not successful, we learned that workload distribution to other processor would be a better way to optimize the system. Spreading massive and repetitive computation from the application on the MPU to the DSP led to a large improvement in out experiments. To carry out the implementation on Versatile platform, the streaming IPC interface which we learned to use from the classes is very helpful.

• Honestly, we struggled to build the Android SDK environment when working on the term project at once. Our suggestion for this course is to simplify this process by providing a VM settled with the environment so that we can get to the core implementation of HW/SW co-design easier.

• I learned to use ARM tools which are provided in the ARM Developer Suite to build the Linux kernel for Android and implemented the DPM to carry out an energy-efficient plan of tuning power levels of the peripheral devices on a smart phone. After the training, I know how to employ Android components for application development to customize the mobile devices with new ideas.

• The experiments in classes certainly made me learn to speed up operations by spreading workload to other DSP form the lab on heterogeneous platform consisting of a MPU and DSP units. And the implementation in the lab also made me understand how the MPU prompt the DSP operations by invoking device driver. We managed to deal with inter-processor communication and DSP task scheduling. My greatest gain from the course is the ability to use system call to access functionalities of the hardware.

6. CONCLUSION

We developed a series of Android-based lab modules involving various topics from Android applications to Android system software. All of these Android-based lab modules were also integrated into the related embedded system courses in order to bring up the embedded system education. A careful analysis of the evaluation of trial-run course and comments from the students leads to the conclusion that Android-based lab modules substantially aid embedded systems education by connecting the theoretical knowledge and the practical implementation. The overall result shows that Android-based lab modules offer competent support for devising innovative embedded system curriculum.

7. REFERENCES

[1] Audio signal processing and recognition. http://mirlab.org/jang/books/audioSignalProcessing.

[2] K. Barr. *Summarizing multiprocessor program execution with versatile, microarchitecture-independent snapshots.* PhD thesis, Citeseer, 2006.

[3] J. Cavazos, G. Fursin, F. Agakov, E. Bonilla, M. O'Boyle, and O. Temam. Rapidly selecting good compiler optimizations using performance counters. In *Proceedings of the International Symposium on Code Generation and Optimization*, pages 185–197. IEEE Computer Society, 2007.

[4] K. Cooper, P. Schielke, and D. Subramanian. Optimizing for reduced code space using genetic algorithms. *ACM SIGPLAN Notices*, 34(7):1–9, 1999.

[5] C. Hsieh, S. Lai, and Y. Chen. Expression-invariant face recognition with constrained optical flow warping. Institute of Electrical and Electronics Engineers, 2009.

[6] C. Hung, J. Chen, and T. Kuo. Energy-efficient real-time task scheduling for a DVS system with a non-DVS processing element. In *27th IEEE International Real-Time Systems Symposium, 2006. RTSS'06*, pages 303–312, 2006.

[7] J.-S. R. J. Jiang-Chun Chen. Trues: Tone recognition using extended segments. In *ACM Transactions on Asian Language Information Processing*, 2008.

[8] D. Jung, S. Bae, J. Lee, S. Moon, and J. Park. Supporting precise garbage collection in Java Bytecode-to-C ahead-of-time compiler for embedded systems. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, page 42. ACM, 2006.

[9] Y. Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, and J. Stockwood. Hardware-software co-design of embedded reconfigurable architectures. In *Proceedings of the 37th Annual Design Automation Conference*, page 512. ACM, 2000.

[10] K. A. Mark Segal. *The OpenGL Graphics System: A Specification.*

[11] R. Meier. *Professional Android Application Development.*

[12] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of the 1998 international symposium on Low power electronics and design*, pages 76–81. ACM, 1998.

[13] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, page 102. ACM, 2001.

[14] P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 527–536. ACM, 2002.

[15] B. Speckmann. *The Android mobile platform.* PhD thesis, Eastern Michigan University, 2008.

[16] M. Turk and A. Pentland. Face recognition using eigenfaces. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume 591, pages 586–591, 1991.

[17] H. Wang, Y. Qian, F. Soong, J. Zhou, and J. Han. A multi-space distribution (MSD) approach to speech recognition of tonal languages. In *Ninth International Conference on Spoken Language Processing.* ISCA, 2006.

[18] H.-C. C. Y.-C. Liao, C.-C. Lin and C.-W. Liu. Self-compensation technique for simplified belief-propagation algorithm. In *IEEE Trans. Signal Process*, volume 55, pages 3061–3072, June 2007.

[19] Y. You, C. Huang, and J. Lee. Compilation for compact power-gating controls. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 12(4):51, 2007.

Risk Areas In Embedded Software Industry Projects

Philip Koopman
Carnegie Mellon University
ECE Dept., HH A-308
Pittsburgh, PA 15213 USA
+1 (412) 268-5225
Koopman@cmu.edu

ABSTRACT
A powerful way to understand where gaps are in the expertise of embedded system designers is to look at what goes wrong in real industry projects. In this paper we summarize the "red flag" issues found in approximately 90 design reviews of embedded system products conducted over a ten year period across a variety of embedded system industries. The problems found can be roughly categorized into the areas of process, requirements, architecture, design, implementation, verification/validation, dependability, project management, and people. A few problem areas, such as watchdog timers and real time scheduling, are standard embedded education topics. But many areas, such as peer reviews, requirements, SQA, and user interface design might be worthy of increased attention in texts and education programs.

Categories and Subject Descriptors
J.7 [**Computer Applications**]: Computers in other systems – *industrial control, process control, real time.*

General Terms
Management, Documentation, Performance, Design, Economics, Reliability, Security, Human Factors, Verification.

Keywords
Embedded system education, software engineering, industry experience, design reviews, real time systems, software process.

1. INTRODUCTION
Most embedded education approaches stem from some attempt to create an overarching set of principles, list key topics, and adopt a particular teaching philosophy. That's a great basis from which to start. But, an interesting question is, what might that approach be missing?

In this paper we look at the problems and risks encountered by practicing embedded system designers. If they are making omissions or mistakes that materially affect the quality of their product or introduce undue risks to product success, then those areas seem reasonable to consider as potentially in-scope for embedded system education.

In this paper we identify 43 areas that were significant risk items for real products in a variety of embedded system applications. The items were identified over the course of more than 90 design reviews conducted by the author, spanning approximately the past 10 years. While the data points are self-selected and are vulnerable to reviewer bias, they nonetheless provide insight into what sorts of skill gaps and problem areas are present in embedded software projects.

WESE'10, October 24, 2010, Scottsdale, AZ, USA.

2. BACKGROUND
The basis of this paper is a retrospective study of design reviews conducted by the author for a variety of embedded system companies. The companies are not identified to protect all parties involved, but most are divisions of large corporations or similar business entities which specialize in embedded systems. Such development groups would be expected to have mature and well organized procedures for designing and supporting moderate to large scale product deployments. A few reviews were of prototypes, but in all cases the developers were skilled, experienced, and tasked with designing real commercial products.

2.1 Product Types Included
The product types that were the subject of reviews generally include the following areas. This list is intended to give an idea of scope of the findings and does not necessarily include every single product:

- Transportation
  - Automotive control
  - Train control
  - Navigation
- Chemical processing
  - Metering and flow control
  - Chemical analysis
  - Process automation
- Buildings
  - Heating, Ventilation, and Cooling (HVAC)
  - Lighting and building security
  - Elevator and related transportation systems
  - Utility services
- Telecommunication systems and data centers
  - Power regulation, switching, and backup
  - HVAC
- Manufacturing
  - Motion control
  - Inspection
  - Robotics
  - Monitoring and equipment maintenance

- Underlying technology
  - Embedded real time control networks
  - Safety critical system design
  - Security

Some embedded application areas are absent from this data, such as consumer electronics and large military combat systems (although the author has had product experience with these areas in the past). There is no reason to believe our results are solely limited to the listed domains, but it seems plausible that concerns would vary depending upon which market segment a product is in.

Code size ranged from a few hundred bytes to about a million lines of code. Development team sizes ranged from one part-time developer to teams of up to 25 developers. (In many cases an overall system had many more developers, but only a specific subsystem was the topic of a review.) Most projects were in assembly language, C, or C++, but other languages were occasionally used. Developers most often followed Waterfall or Vee development approaches, but some products used Spiral, Incremental, or Agile approaches. Most reviews were of US-based teams, with a handful of reviews in Europe and Asia. Perhaps one fifth of development teams used development partners or remote team members in India or China. In most cases remote developers participated in reviews either in person or via conference call.

Systems were about evenly divided between small microcontrollers and bigger CPUs that ran some sort of RTOS. A very few systems used DSPs or FPGAs, and none used custom or domain-specific silicon. Product volume ranged from prototypes to hundreds of thousands of units per year, although most reviews were for products in the 1,000 to 10,000 units per year range.

The results of some design reviews beyond those in the data set were excluded due to contractual obligations or because they did not result in formal review reports. But the missing data would have been unlikely to materially change the outcome of this study.

2.2 Design Review Process

A typical design review involves the steps of setting up the review engagement, learning some domain background, obtaining as many project documents as possible, selectively reviewing documents, setting a meeting agenda, traveling to hold an on-site review visit, and generating a written report. A minority of reviews to examine very specific areas or answer narrow questions were done electronically, with no visit.

On-site visits typically lasted one day or, in some cases, two days. The amount of information available before the visit ranged from essentially nothing to thousands of pages of design information (often including complete source code listings). The degree to which developers self-identified problems before a visit varied, but most problems were identified by the reviewer without hints from the review team. More importantly, in almost all cases the review team accepted the problems identified as valid feedback. (This is not to say that every recommendation was necessarily carried out. But, for the most part, teams agreed that the areas identified as critical risks were in fact significant issues that materially affected the likelihood of project or product success.)

A reasonable fraction of the design reviews, especially initially, were carried out by two independent reviewers in parallel, with a shared visit and jointly issued report. More recent reviews were largely single-reviewer events in large part due to economic constraints.

Most reviews were performed at about the time the product was ready to start acceptance testing or be released. In cases where a problem was identified, an attempt was made to trace back to a reasonable root cause. For example, a bug-prone module might be identified as having implementation problems, design problems, architecture problems, or requirement problems depending on which stage in the design process was the most effective place to have avoided the bug (and other similar bugs).

The primary output of each review engagement is list of recommendations, including “red flag” issues that present significant and immediate risks to the success of the project or product. Other, less pressing, “yellow flag” risks and reviewer observations are also listed in review reports. Values were not assigned to all topics in every review due to lack of time. Rather, emphasis was placed on areas that seemed to the reviewer and the developers to be the most likely place to be sources of major risk. The study we present here is solely concerned with the red flag issues.

Over time, reviews became more formal and repeatable as a list of typical problem areas and review questions was developed using the input of a number of experts over the first few years of conducting reviews. This list was formally used for perhaps a third of the reviews, and the general knowledge of what was in this list informed most of the other reviews. The list presented in this paper does not strictly conform to the items in that proprietary checklist, but is similar in nature. The checklist has approximately three times as many topics as the red flag list below. In other words, two thirds of the entries on the checklist are worth asking about, but have failed to generate any red flag in a decade of performing reviews.

There is no way to know how many significant risks were missed because reviewers didn’t think to ask the right questions. However, the chance of this happening was reduced by initially by the use of multiple reviewers, and in later years by the use of a comprehensive checklist-based triage process as just described.

2.3 Background of designers

The design teams reviewed varied in technical background significantly. Many team members had degrees and experience in mechanical or electrical (non-electronic) engineering. A number had electronic and computer engineering degrees. A few had computer science or (rarely) software engineering degrees. For the most part, senior developers started as domain specialists and picked up embedded computing on the job. Junior developers were more likely to have had software training of some sort, but in most cases had more of an electrical or computer engineering background.

Over the years, there has been a trend for many design teams to advance to a higher level of software process sophistication (for example, many teams progressed to SEI Capability Maturity Model [6] Level 2 or above). This is in large part due to a concerted effort by some corporations to improve software quality. It is also in part due to hiring of developers with formal

software training into embedded system product teams. But, high process maturity is not universal across embedded projects. In particular, each company seems to find its own way up the software learning curve as it introduces the first non-trivial computing capability into its products and attempts to write software using domain experts who have little or no formal software training.

3. RISK AREAS IDENTIFIED

The following risk areas are identified as red flag (significant) risks in one or more reviews in this study. They are grouped and organized to provide some structure in terms of typical development process stages and activities. However, the ordering does not connote any severity, priority, or frequency. A typical item in this list was a red flag in a few reviews and a yellow flag in several more.

The examples and likely consequences of risk areas given are generally representative of the risks actually seen without revealing company- or product-specific information. (To the degree that examples or statements are true about any particular product, they are typically true of many different products that were reviewed.)

3.1 Development Process

#1. Informal development process

The process used to create embedded software is ad hoc, and not written down. The steps vary from project to project and developer to developer. This can result in uneven overall software quality.

#2. Not enough paper

Too few steps of development result in a paper trail. For example, test results may not be written down. Among other things, this can require re-doing tasks such as testing to make sure they were fully and correctly performed.

#3. No written requirements

Software requirements are not written down or are too informal. They may only address changes for a new product version without any written document stating old version requirements. This can lead to misunderstandings about intended product functions and difficulty in designing adequate tests.

#4. Requirements with poor measurability

Software requirements can't be tested due to missing or subjective measurement criteria. As a result, it is difficult to know whether a requirement such as "product shall be user friendly" has been met.

#5. Requirements omit extra-functional aspects

Product requirements may state hardware processing speed and hardware reliability, but omit software response times, software reliability, and other non-functional requirements. Implementing and testing these undefined aspects is left at the discretion of developers and might not meet market needs.

#6. High requirements churn

Functionality required of the product changes so fast the software developers can't keep up. This is likely to lead to missed deadlines and can result in developer burnout.

#7. No SQA function

Nobody is formally assigned to perform an SQA function, so there is a risk that processes (however light or heavy they might be) aren't being followed effectively regardless of the good intentions of the development team. Software Quality Assurance (SQA) is, in essence, ensuring that the developers are following the development process they are supposed to be following. If SQA is ineffective, it is possible (and in our experience likely) that some time spent on testing, design reviews, and other techniques to improve quality is also ineffective.

#8. No mechanism to capture technical and non-technical project lessons learned

There is no methodical effort to identify technical, process, and management problems encountered during the course of the project so that the causes of these problems can be corrected. As a result, mistakes made on one project are repeated in future projects.

3.2 Architecture

#9. No defined software architecture

There is no picture showing the system's software architecture. (Many such pictures might be useful depending upon the context – but often there is no picture at all.) Ill defined architectures often lead to poor designs and poor quality code.

#10. No message dictionary for embedded network

There is no listing of the messages, payloads, timing, and other information for messages being sent on an embedded real time network such as CAN. As a result, there is no basis for analysis of real time network performance and optimization of message traffic.

#11. Poor modularity of code

The design has poorly chosen interfaces and poorly decomposed functionality, resulting in high coupling, poor cohesion, and overly long modules. In particular, interrupt service routines are often too big and mask interrupts for too long. The result is often increased risk of software defects due to increased complexity.

3.3 Design

#12. Design is skipped or is created after code is written

Developers create the design (usually in their heads) as they are writing the code instead of designing each module before that module is implemented. The design might be written down after code is written, but usually there is no written design. As a result, the structure of the implementation is messier than it ought to be.

#13. Flowcharts are used when statecharts would be more appropriate

Flowcharts are used to represent designs for functions that are inherently state-based or modal and would be better represented using a state machine design abstraction. Associated code usually has deeply nested, repetitive "if" condition clauses to determine what state the system is in rather than having an explicit state

variable used to control a case statement structure in the implementation. The result is code that is significantly more bug prone and more difficult to understand than a state-machine based design.

#14. No real time schedule analysis

There is no methodical approach to real time scheduling. Typically an ad hoc approach to real time scheduling is used, frequently featuring conditional execution of some tasks depending upon system load. Testing rather than an analytic approach is used to ensure real time deadlines will be met. Often there is no sure way to know if worst case timing has been experienced during such testing, and there is risk that deadlines will be missed during system operation.

#15. No methodical approach to user interface design

The user interface does not follow established principles (e.g., [5]), and is likely to make using the product difficult or error-prone. The interface might not take into account the needs of users in different demographic groups (e.g., users who are colorblind, hearing impaired, wearing gloves, or who have trouble grasping small objects due to arthritis).

3.4 Implementation

#16. Inconsistent coding style

Coding style varies dramatically across the code base, usually in part due to lack of a written coding style guideline. Code comments vary significantly in frequency, level of detail, and type of content. This makes it more difficult to understand and maintain the code.

#17. Resources too full

Memory or CPU resources are overly full, leading to risk of missing real time deadlines and significantly increased development costs. An extreme (but not infrequent) example is to have zero bytes of program and data memory left over on a small processor. Significant developer time and energy can be spent squeezing software and data to fit rather than developing new functionality.

#18. Too much assembly language

Assembly language is used for most or all of the code when an adequate high level language compiler is available. Sometimes this is due to lack of big enough hardware resources to execute compiled code. But more often it is due to developer preference, reuse of previous project code, or a need to economize on purchasing development tools. Assembly language software is usually more expensive to develop and more bug-prone than high level language code.

#19. Too many global variables

Global variables are used instead of parameters for passing information among software modules. The result is often code that has poor modularity and is brittle to changes.

#20. Ignoring compiler warnings

Programs compile with ignored warnings and/or the compilers used do not have robust warning capability. A static analysis tool is not used to make up for poor compiler warning capabilities. The result can be that software defects which could have been caught by the compiler must be found via testing, or miss detection entirely. If assembly language is used extensively, it may contain the types of bugs that a good static analysis tool would have caught in a high level language.

#21. Inadequate concurrency management

Mutexes or other appropriate concurrent data access approaches aren't being used. This leads to potential race conditions and can result in tricky timing bugs.

#22. Use of home-made RTOS

An in-house developed and maintained RTOS is being used instead of a commercial (or free third party) operating system. While the result is sometimes technically excellent, it also commits the company to maintaining RTOS development skills as a core competency, which may not be the best strategic use of limited resources.

3.5 Verification & Validation

#23. No peer reviews

Code and other documents are not subject to a methodical peer review, or undergo ineffective peer reviews. As a result, most bugs are found late in the development cycle when it is more expensive to fix them.

#24. No test plan

Testing is ad hoc, and not according to a defined plan. Typically there is no defined criterion for how much testing is enough. This can result in poor test coverage or an inconsistent depth of testing.

#25. No defect tracking

Defects and other issues are not being put into a bug tracking system. This can result in losing track of outstanding bugs and poor prioritization of bug-fixing activities.

#26. No stress testing

There is no specific stress testing to ensure that real time scheduling and other aspects of the design can handle worst case expected operating conditions. As a result, products may fail when used for demanding applications.

3.6 Dependability

#27. Insufficient consideration of reliability/availability

There is no defined dependability goal or approach for the system, especially with respect to software. In most cases there is no requirement that specifies what dependability means in the context of the application (e.g., is a crash and fast reboot OK, or is it a catastrophic event for typical customer?). As a result, the degree of dependability is not being actively managed.

#28. Insufficient consideration of security

There is no statement of requirements and intentional design approach for ensuring adequate security, especially for network-connected devices. The resulting system may be compromised, with unforeseen consequences.

#29. Insufficient consideration of safety

In some systems that have modest safety considerations no safety analysis has been done. In systems that are more overtly safety critical (but for which there is no mandated safety certification),

the safety approach falls short of recommended practices. The result is exposure to unforeseen legal liability and reputation loss.

#30. No or incorrect use of watchdog timers

Watch dog timers are turned off or are serviced in a way that defeats their intended role in the system. For example, a watchdog might be kicked by an interrupt subroutine that is triggered by a timer regardless of the status of the rest of the software system. Systems with ineffective watchdog timers may not reset themselves after a software timing fault.

#31. Insufficient consideration of system reset approach

System resets might not ensure a safe state during reboots that occur when the system is already in operation, resulting in unsafe transient actuator commands.

#32. Neither instrumentation nor error logs

There is no run-time instrumentation to record anomalous operating conditions, nor are there error logs to record events such as software crashes. This makes it difficult to diagnose problems in units returned for service.

#33. No software update plan

There is no plan for distributing patches or software updates, especially for systems which do not have continuous Internet access. This can be an especially significant problem if the security strategy ends up requiring regular patch deployment. Updating software may require technician visits, equipment replacement, or other expensive and inconvenient measures.

#34. No IP protection plan

There is no plan to protect intellectual property of the product from code extraction, reverse engineering, or hardware/software cloning. (Protection strategies can be legal as well as technical.) As a result, competitors may find it excessively easy to successfully extract and sell products with exact software images or extracted proprietary software technology.

3.7 Project Management

#35. No version control

Sometimes source code is not under version control. More commonly, the source code is under version control but associated tools, libraries, and other support software components are not. As a result, it may be difficult or impossible to recreate and modify old software versions to fix bugs.

#36. No backward compatibility and version management plan

There is no plan for dealing with backward compatibility with old products, product migration, or installations with mixed old and new product versions. The result may be incompatibilities with fielded equipment or a combinatorial explosion of multi-component compatibility testing scenarios necessary for system validation.

#37. Use of cheap tools (software components, etc.) instead of good ones

Developers have inadequate or substandard tools (for example, free demo compilers instead of paid-for full-featured compilers) because tool costs are can't be reckoned against savings in developer time in the cost accounting system being used. As a result, developers spend significant time creating or modifying tools to avoid spending money.

#38. Schedule not taken seriously

The software development schedule is externally imposed on an arbitrary basis or otherwise not grounded in reality, and so is not taken seriously by anyone. As a result, developers may burn out or simply feel they have no stake in schedules.

#39. Presumption in project management that software is free

Project managers and/or customers (and sometimes developers) make decisions that presume software costs virtually nothing to develop or change. This is one contributing cause of requirements churn.

#40. No risk mitigation for problems with external tools and components

External tools, software components, and vendors are a critical part of the system development plan, and no strategy is in place to deal with unexpected bugs, personnel turnover, or business failure of partners.

#41. Disaster recovery not tested

Backups and disaster recovery plans may be in place but untested. As a result, it may be that data won't be recoverable when a real problem occurs.

3.8 People

#42. High turnover and developer overload

Developers have a high turnover rate. This is especially prevalent with work outsourced to India and China. As a result, code quality and style varies. Lack of a robust paper trail makes it difficult to continue development. Often more important is that replacement developers may lack the domain experience necessary for understanding the details of system requirements.

#43. No training for managing outsource relationships

Engineers who are responsible for interacting with outsource partners do not have adequate time and skills to do so, especially for multi-cultural partnering. This can lead to significant ineffectiveness or even failure of such relationships.

4. ANALYSIS

4.1 Projects Don't Need To Be Perfect

It is important to point out that not every project needs to get everything on the preceding list perfect. Indeed, many projects had only one or two red flags out of that list, and most had fewer than five red flags. (Some – a very few – had zero red flags.) By the same token, most projects had many yellow flags, indicating there were areas that could be improved over time.

It is also important to note that the red flag areas were based on risk specific to a particular domain and product. A development team could totally ignore many or most items on the above list and, so long as that approach didn't create a significant risk of product or project failure, that wasn't a red flag. For example, having the watchdog timer turned off is likely to be a red flag on unattended equipment with 24x7 operational requirements, but might not be an issue on a non-critical hand-operated device that is power cycled before each use.

In other words, items were red flags because they were a big deal in the context of that particular product, *not* because they were on a list of best practices that had to be done regardless of project tradeoffs.

That having been said, identification of red flag issues was at the discretion of the reviewer with feedback from the developers being reviewed, and therefore somewhat subjective.

4.2 Back to Basics – But Less Than Expected

Perhaps surprisingly, there are only a very few risk areas that are almost universally accepted as embedded system core topics. Real time scheduling, watchdog timers, and concurrency management are likely to be on a typical embedded system educator's list of desirable technical topics for either a first or second course in the area. But most of the problem areas aren't like that. Many of the items are things omitted by typical embedded system texts and courses.

That doesn't mean core technical areas don't matter. We believe it is important to give embedded system designers a principled understanding of core engineering principles and underlying technology. But these results suggest that informally trained embedded designers (who have nonetheless been formally schooled in a certain way of thinking about technical problems in general) tend to find ways to fill in basic technical areas on their own, even if they didn't have technology-specific training. So, apparently, self-teaching with a book in one hand and a development board in the other works to an extent. But it doesn't seem to work when you get beyond the technical basics.

Most risk areas seemed uncorrelated with developer backgrounds, but there were a few areas in which team members' formal educational background affected likely risk areas. For example, developers with formal software training were more likely to use a version control system. However, differences were not as widespread as might be expected. In part, this is because non-software engineers are trained to follow a methodical development approach (such as creating written requirements and formal test plans) for non-software aspects of the system, and that approach carried over to software. But, developers without formal embedded training are more likely to have gaps in the more advanced embedded-specific areas such as concurrency control and real time scheduling since they are beyond the scope of most introductory programming texts (and even many introductory embedded system texts).

4.3 Knowing You Have A Problem

Most of the problem areas might be characterized as having the property that they are the result of a gap in the developer's understanding or the software process being used. In other words, developers didn't realize (or didn't have time) to look for some types of problems. Basic functionality for a desired system was usually there at the time of the design review. For example, everyone had figured out by the design review how to use an A/D converter well enough to get acceptable sample quality. And they had found and fixed whatever bugs they were likely to find with their testing approach. The risks tended to come more from having a high risk of undetected bugs, missing chances to have avoided big problems that surfaced late in the project, and missing chances to avoid project schedule or cost problems.

On the whole, smart motivated developers can figure out most of the technology and fix most problems if they have a way to know what's broken. The biggest risks come when they don't realize something in their technology or development process is broken, or when they attempt ad hoc solutions to difficult problems because they don't know more robust solution approaches are available.

4.4 Weak Process Hurts

A surprise is that a significant fraction of the problem areas ended up being software process problems instead of technology problems. While many educators are technologists at heart, the fact of the matter is that poor software process is a huge problem impeding the success of embedded system development efforts. (It's hard to have a good product with bad technology. But it's also hard to succeed with an ineffective development process.)

The lack of process content in most developer degree programs is deeply ingrained, and has various sources. But it is really hurting embedded developers, and is a critical skill they must currently pick up once in industry.

4.5 Embedded Software Problems Are Only A Little Special

Most of the red flag areas would not be out of place in a list of IT project risks. We are, after all, talking about software and some practices are good ideas regardless of the domain. However, the ways to mitigate risks are often different for embedded applications than for desktop applications.

4.6 Five Forebodes Failure

One of the informal observations made across the course of these reviews was that developer teams with exactly 5 primary contributors usually fail. Invariably these teams had previously completed a project with 3 or 4 members, and increased the team size to tackle a more complex project without making any changes in their software process.

While this is an anecdotal result, projects that grow past 4 developers in size should seriously consider switching to a heavier weight software process (more paper, more formality, more methodical rigor). Smaller teams still seem to benefit from good process, but basically can get away with informality with less risk than larger teams working on more complex projects.

5. EDUCATIONAL IMPLICATIONS

5.1 Formal Education Doesn't Affect Risk Areas Much

Embedded system software development is often performed by engineers with no formal training in that area. As mentioned previously, the surprising part is not that such developers have gaps, but rather that they seem to do a pretty good job of filling in the gaps in basic technology areas all on their own. In other words, there isn't much difference in the risks areas identified in projects being performed by computer-trained embedded system engineers vs. non-computer trained domain experts.

The gaps that were identified are largely either in a few system integration areas or in the broader area of software development process. Most computer engineers (and even many computer scientists) receive little software process training. Thus, most

embedded system engineers don't see formal educational material that would fill these gaps.

We believe that plugging the gaps in embedded system projects isn't likely to be solved by having more engineers take existing embedded system college courses. The problem is really that these topics aren't being taught to (nor packaged for learning by) embedded designers. Rather, this data suggests that it might be useful to rethink the core skills that should be taught in embedded system courses and included in texts.

5.2 Our Course Approach

Informal awareness of the types of topics that cause problems in industry embedded projects has been guiding our graduate and undergraduate course content choice for a number of years. But, until we performed this study, we were operating on gut feel instead of data. As a result of this analysis we have updated a two-course embedded systems sequence to address most of the risk areas.

18-348 Embedded System Engineering is mostly taught to third- and fourth-year Electrical and Computer Engineering (ECE) undergraduate students. The syllabus might superficially appear to be an introduction to microcontrollers course using 16-bit CPUs. But, portions of lectures, homework assignments, and lab assignments have been crafted to instill an understanding of the basics of methodical software process. For example, every assignment has formally written requirements, and many assignments require documented peer reviews, designs, test plans, defined acceptance tests, and so on. These are lightweight approaches to instill awareness rather than rigorous treatment of process topics, largely because undergraduates lack the world-view and experience to appreciate and learn about process topics. Technology topics from this list taught at this level are concrete, technical, or linked directly to implementation: #11 modularity, #12 design before implement, #13 statecharts, #14 real time scheduling, #16 coding style, #19 globals, #20 compiler warnings, #21 concurrency, and #30 watchdog timers from the list previously given.

18-649 Distributed Embedded Systems is taught to ECE Masters Degree students, usually in their first year of graduate school, and to fourth-year undergraduates as a follow-on to 18-348. Course lectures are divided into three parts: one third cover software process, one third cover embedded networking and distributed systems, and one third cover dependable and critical system design. Most of the remaining risk areas not covered by 18-348 are covered in this course, with the coverage increasing over time as lectures are modified to correspond to the risk area list. A semester-long course project is used to demonstrate the execution of process methods and (for students who are at a point that they are ready to learn the lesson) instill the value of having a lightweight but complete process for software development.

5.3 Industry Training

Outreach to industry is problematic since embedded developers are geographically scattered and often not local to the usual high technology cities. (Most of the design reviews were in the US Midwest, where companies build more embedded systems than computer systems. Only a few were near high-tech cities.) Despite advances in distance education, traditional university-run courses aren't doing a good job of reaching most of them.

To address this audience I have written a book that covers most of the topics in this list [4]. Each chapter gives a summary of issues and concrete solution approaches for risk areas. The hope is that having a book available will help solve some problems, and at the very least make it possible for developers to know where their gaps are so they can address them before they suffer a dramatic project failure. This book was adopted as the text for 18-649 starting in 2010.

5.4 Related Work

There has been little formalized work on attempting to analyze the needs of industry with regard to embedded systems. [2] is based in part on an analysis that takes into account industry surveys, and suggests that embedded system education should be more cross-disciplinary and more representative of embedded industry experiences. These are important observations and worthy goals. Our results extend these observations by reporting problems that even experienced industry designers aren't able to resolve on a consistent basis.

A number of embedded system educators emphasize some of the areas on our risk list, most typically the areas we identify for inclusion in 18-348 as well as distributed system and dependability topics. Examples include [1], [7], and previous courses at our institution [3]. Other curriculum proposals include an explicit software engineering courses (e.g., [8]). No doubt there are some other degree programs that address most or all of these areas in one way or another (for example, our institution has an interdisciplinary Master of Science in Information Technology – Embedded Software Engineering degree [9] that requires both graduate embedded system technical courses and graduate software engineering courses). But such programs are not the norm. Our belief is that software process concepts should be central and integrated throughout the embedded curriculum rather than an optional or distinct course module.

Embedded system courses almost universally use hands-on project content as a way for students to get a feel for system integration issues. This certainly gives students experience in how difficult complex projects can be and gives them a chance to test their fundamental technical skills. However, we have found that even engineers who have been through a large number of industry design projects have gaps. Thus, we believe that merely experiencing a design project without guidance and reflection upon solid principles and these specific risk areas is not enough to fill these gaps. It is difficult to self-teach ways to fix problems when you don't ever realize you got things wrong.

6. CONCLUSIONS

This paper identifies 43 areas that were identified as red flag risk areas across reviews of 90 industry embedded system projects in the past decade. The most striking aspect of the list is that, by and large, even self-trained developers are not at huge risk of missing the basics of embedded systems. Rather, most risks are either complex system integration skills (e.g., concurrency management) or software development process issues (e.g., requirements problems or inadequate test plans).

While many of the areas identified might not seem specific to embedded systems, they are the risk areas that are actually affecting real embedded projects. Embedded educators should take notice and take steps to ensure that our future courses and

degree programs address most of these areas, preferably in a way that teaches the skills most useful in an embedded system context.

7. ACKNOWLEDGMENTS

The author wishes to thank all the design teams that have been through the design review process with him. It's never fun having an outsider come in to tell you all the mistakes you made, and I appreciate the openness and willingness to discuss things shown by so many developers over the years.

8. REFERENCES

[1] Capsi, P., et al., 2005, "Guidelines for a graduate curriculum on embedded software and systems," *ACM TECS*, vol. 4, no. 3, pp. 587-611, August 2005.

[2] Grimheden, G.; Torngren, M., 2005, "What is embedded systems and how should it be taught?---results from a didactic analysis," *ACM TECS*, vol. 4, no. 3, pp. 633-651, August 2005.

[3] Koopman, P., et al., 2005, "Undergraduate embedded system education at Carnegie Mellon," *ACM TECS*, vol. 4, no. 3, pp. 500-528, August 2005.

[4] Koopman, P., 2010. *Better Embedded System Software*, Drumnadrochit Press, Wilmington.

[5] Nielsen, J., 1993. *Usability Engineering,* AP Professional, Boston.

[6] Paulk, Mark C., et al., 1995. *The Capability Maturity Model: Guidelines for Improving the Software Process.* Addison Wesley, Boston.

[7] Sangiovanni-Vincentelli, A.; Pinto, A., 2005, "An overview of embedded system design education at Berkeley," *ACM TECS*, vol. 4, no. 3, pp. 472-499, August 2005.

[8] Seviora, R., 2005, "A curriculum for embedded system engineering," *ACM TECS*, vol. 4, no. 3, pp. 569-586, August 2005.

[9] MSIT-ESE program web page, Carnegie Mellon University, accessed July 30, 2010. http://mse.isri.cmu.edu/software-engineering/web1-Programs/MSIT-ESE/index.html

Discussion: How to teach cyber-physical systems?

Abstract:

Cyber-physical systems refer to those computing systems that are tightly coupled with some physical process. These systems provide engineering challenges due to their limited resources and this close interaction with physical processes. Such systems often require low-level design techniques as well as a complete understanding of timeliness and concurrency. Educating students about cyber-physical systems is itself a challenge due to the broad array of topics required. During this discussion, WESE 2010 participants are encouraged to consider ways to integrate the study of cyber-physical systems into the classroom.

The Embedded and Mobile Systems Master at the CNAM of Paris

Samia Bouzefrane
Associate-Professor
Computer Science Department
Conservatoire National des Arts et Métiers, 292 rue Saint Martin
75141, Paris Cédex 03, FRANCE
samia.bouzefrane@cnam.fr

ABSTRACT
In this paper, we present the "*Emebdded and Mobile Systems*" Master provided by the CNAM teachers since September 2005. We describe the courses given and the evolution of the Master in adequacy with the industry and the research worlds. This paper reports our experience in teaching embedded and mobile systems.

Keywords
Embedded systems, mobile systems, education, Masters.

1. INTRODUCTION
The Computer Science department of the Conservatoire national des Arts et Métiers (CNAM) [1] decided to develop new axes of teaching and research by creating the Master of Embedded and Mobile Systems (EMS) that started on September 2005. The aim of this training is to provide high-quality courses in modern and well-demanded technologic themes related to embedded and mobile systems.

The embedded and mobile data processing is taking more and more place in our daily life, such as: the car, the mobile telephony, the electronic money, etc. The concept of system is taken in a large sense since it gathers traditional aspects of operating systems and all the elements involved in application development, such as for example of the dedicated executives, the services, the distributed or centralized applications, the communication networks. The design of such systems must thus take into account different and multiple problems: footprint memory, energy consumption, safety constraints, geo-localization, deployment, maintenance, verification, etc. These very diverse characteristics have an impact on the design and the development of solutions for embedded and mobile systems.

The objective of this master is to train students to master various dimensions that involve in the design and the development of embedded and mobile systems.

Section 2 gives an overview of the missions of the CNAM and describes briefly the research activities of the CEDRIC Laboratory. Section 3 details the functioning and the specificity of the Master EMS before describing in Section 4 the different courses that are taught, as well as the internship that is mandatory to obtain the Master degree. Section 5 presents some results through curves that show the success rate and the number of EMS'degree students that find a job in embedded and mobile systems per year. Section 6 concludes the paper by decribing briefly the new master SEMS that will start on September 2011 and that is the evolution of the master SEM..

2. THE CNAM AND THE CEDRIC LAB
The CNAM (see Figure 1) is a scientific, cultural and professional corporation. Since its foundation by Henri Grégoire in 1794, it is dedicated to the training of adults along their life. Placed under the supervision of the high education ministry, it fulfils three missions: the professional training for adults, the technological research and the innovation, the diffusion of the scientific and technical culture. The CNAM is based for its development on its network made up of 29 regional centers and several teaching centers around the world (e.g. Lebanon, Germany, Benin, Spain, Greece, Hungary, Romania, Morocco, etc). It is organized around two schools: Industrial Sciences and Information Technlogies School (SITI) and Management and Society School. SITI includes 7 departments and 10 research laboratories. The CNAM research is based on a multi-field activity and on the engagement of the companies: with researchers including 128 professors and 239 associate-professors; and approximately 2000 external contributors, where 330 are Ph-D students. The CNAM, with its Musée des Arts et Métiers [2], is a major actor of diffusion of the scientific and technique culture: 350 events and conferences are open to all; 50 000 participants in all France; 200 000 visitors of the Musée.

Figure 1: The CNAM

The CEDRIC (Centre d'Etudes et De Recherche en Informatique du CNAM) Lab [3] gathers the research activities in computer science at CNAM. CEDRIC members belong to the Computer Science department, and are researchers and teachers in computer science or mathematics.

The CEDRIC research activities are organised along five areas:

- Reliable systems: Certified Design and Programming
- Information systems and Databases

- Interactive media and Mobility
- Combinatorial Optimization
- Statistical methods for data-mining and learning

The CEDRIC performs fundamental and applied research, is linked to the main french industrial and public institutions around new information technologies, and cooperates with many labs in France, or abroad. The lab is funded by fundamental French research contracts (ANR, IST, ...). The lab contributed to the creation of the "pôle de compétitivité Cap Digital" and is a member of this "pôle". It is represented in the steering committee of the Video games theme of this "pôle". It is a member of the "pôle Systém@tic" as well and is active within the Security-Defence and open source groups of this "pôle".

The CEDRIC lab is deeply concerned with teaching activities. CEDRIC's members were active is the achievement of the European LMD (Licence-Master-Doctorate) reform: the CNAM now offers high-level masters in all of the lab's research areas, in collaboration with other universities (Paris VI, Paris I, La Rochelle, Poitiers). Some of our best students continue with doctoral studies at CEDRIC or in industrial companies (Cifre, Fongecif fellowhips). Finally, the CEDRIC Lab plays an important role in the ENJMIN (Ecole Nationale Supérieure du Jeu et des Médias Interactifs) [4] a national school on interactive medias and games which opened at Angouleme in 2004.

On June 30th, 2009, the lab includes hundred people among of them 47 permanent researchers, around 50 Ph.D.students and 2 administrative staff. The Cedric lab is labeled by the French minister of research, technology and education as an "Equipe d'Accueil" (EA 1395) since its creation. The CEDRIC lab is the largest CNAM laboratory and one of the wellknown computer science laboratories in the region of Paris and around.

3. THE EMS MASTER

Among the Computer Science Masters proposed by the CNAM, the Master EMS (Embedded and Mobile Systems) has been created on 2005 by Pierre Paradinas that has been appointed as the head of the chair of Embedded Systems when this chair was created on 2003. When Pierre Paradinas became on January 2007 the technological development director at the INRIA [5], the professor Eric Gressier directed the Master EMS and created a research team called Embedded and Mobile Systems for ambient intelligence [6]. The major researchers of this team are teachers in the EMS' Master. From September 2008 to August 2010, I have been the responsible of this Master.

3.1 The Master objective

The application domains dealt with within our training cover a large spectrum of data processing from classic embedded data processing (industrial data processing, avionics, process control, electronic money, mobile telephony) to ambiant data processing (the Internet of things, ubiquitous games, intelligent house, energy management). Embedded and mobile systems are omnipresent nowadays: mobile telephony 3G+, smart cards, RFID tags, contactless communication, sensor networks, geo-localization, etc. The topics approached by the Master training address a promising industrial perimeter from the point of view of job creation. The objective of this master is to train the students with the various dimensions involved in the design and the development of embedded and mobile systems.

3.2 The lecturers

The pedagogical team consists of members belonging to CEDRIC laboratory and more particularly to the "Interactive media and Mobility" team, and members from industry.

In fact, this team is reinforced by outside contributors coming from professional environments in relation with the Master. Here is a list of some companies with which we work: Gemalto, Oberthur Tecknologies, Esterel Technologies, Dassault Systems Communication, Trialog, EDF R&D, Renault, Peugeot, Hippocad, RATP, etc. This list is not exhaustive and through our trainees, we tie each year of new labour relations which enable us to enrich our research activities and or teaching by concrete applications.

3.3 The students

The students are titular of a diploma of engineer or a Master degree level 1 in Computer Science or Electronics. We accommodate students coming from various French universities, schools of engineers (ECE [9], ENSIIE [10], ISEP[11]) or foreign universities.

Table 1. The list of courses of EMS Master

Course Title	Code	Responsible	Credits
Networks for embedded and mobile systems	RSEM*	S . Boumerdassi	6
Platform for embedded and mobile systems	PFSEM*	S. Bouzefrane	6
Programming embedded and mobile systems	PSEM*	E. Gressier	6
Advanced Architecture	AA*	F. Anceau	3
Data management for embedded and mobile systems	GDEM*	S. Bouzefrane	3
Programming real-time systems	TRA	S. Bouzefrane	3
Security	SEC	N. Pioch	3
Synchronous Languages	LS	J.F. Susini	3
Program verification	VERI	J.F. Susini	3
Safte functioning	SdF	D. Delahaye	3
Architecture of on-line games	AJL	E. Gressier	3
Networks and Quality of Service	RQoS	F. Sailhan	3
Internship	*	EMS'Master responsible	18

4. COURSES DESCRIPTION

The Master EMS level 2 represents a set of 60 credits, where 5 courses are mandatory and some others are optional, as described on the web site of the master [7]. The internship represents 18 credits. The students have one day free per week assigned to homework. Students have to pass exams, to prepare exposés and projects. The following sub-sections detail all the courses given Table 1. Each course code followed with a star is mandatory course.

4.1 RSEM course

The objective of this course is to present in-depth principal wireless and mobile networks. It also approaches the future orientations of this kind of technology.

After a presentation of the principal problems and the challenges in relation with mobility, the course deals with various architectures and standards of wireless and mobile networks. In the mobile networks part, architectures of GSM until the 4G are presented. The wireless part develops all 802 standards.

4.2 PFSEM course

The PFSEM course objective [8] is to specify, design, carry out and deploy applications dedicated to smart cards. This course makes it possible to understand the general functioning of contact/contactless smart cards, depending on the application domains (banking cards, health cards, SIM cards, etc.). It allows to explain all the development process of an application that has to be carried out on a smart card, but also to understand the protocols involved in the communication between a smart card and a reader such standard protocols (ISO/IEC 7816, ISO/IEC 14443) of the smart cards, or many others related to banking such as EMV standard, or SIM Toolkit standards (see Figure 2). The course investigates also NFC and RFID technologies. It lets the students to program applications for smart cards by using Java Card technology (Java Card 2.2, Java Card 3.0, SIM Toolkit Application), Java Card RMI, .NET for smart cards. Lessons on contactless cards through RFID and NFC technologies are presented together with the development and the deployment of the applications with those standards. Professionals known in this field are invited to enrich the course by a series of conferences to address the relevant aspects in the field.

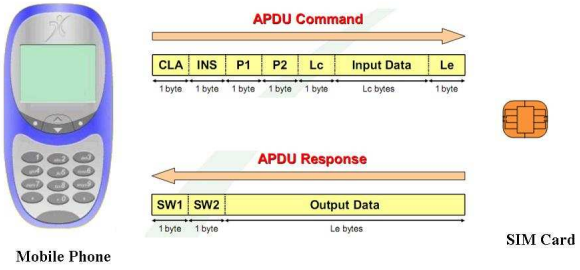


Figure 2: Protocol exchange between a cell phone and a SIM card

4.3 PSEM course

This course aims at offering an exhaustive panorama as regards design, development and test of embedded systems for mobile telephony. One of the posted objectives is that to master the programming of portable applications for cell phones. It is also a question of understanding how are designed the platforms and their impact on the design of the operating systems, the applications and the user interface.

To the students is proposed a deep analysis on the concrete and the theoretical aspects having milked with the material and software platforms for mobile telephony. This teaching is organized around the following axes:

- Definition of the material structure of the mobile phones,

- Functionalities and implementation of the embedded operating systems on wireless phones (embedded Linux, Symbian, Windows Mobile, etc.),

- Radio Communication, implementation of the communication protocols for mobile telephony and presentation of their APIs,

- Design of multimedia applications within programming environments such as Java ME or Objective C for the iphone,

- Basic Concepts as regards user interface.

4.4 AA course

The objective of this course is to train students with the comprehension of the subjacent constraints of hardware devices (size, energy, physical security, etc.), and with the novel methods in the design of hardware components.

The AA course consists of several parts:

- Micro-electronics (evolution of the integrated circuits, test methods),

- Elements of VHDL,

- Structure of the embedded processors (processors CISC, RISC, languages and systems, the example of processor ARM7, VLIW),

- The Systems One Circuit (SoC): the SoC industry, the integrated buses, design methods, economic and commercial constraints, electronic and dimensional constraints,

- Techniques of reduction of consumption: the race with the power and its recent stop, clock industry at variable frequency, phases shift,etc.

4.5 GDEM course

The acquired knowledge concerns the mechanisms used for an efficient data management in embedded and/or mobile systems. The content of this course is related to the principal concepts of data management in different environments such as in:

- A classical DBMS (Data Base Management System),

- The real-time context,

- The sensor networks: the example of TinyDB,

- Mobile databases,

- The smart cards: the example of PicoDB,

- The mobile phones.

4.6 TRA course

The aim of the TRA course is to understand the architecture and the characteristics of a real-time system in order to develop

applications in this type of environments. The course is composed of different parts:

- Introduction to the real-time issues,

- The architecture of a real-time system,

- Synchronization mechanisms (using C and Java programming languages),

- The scheduling problems and the resources management in real-time systems,

- RTSJ: the specification for real-time Java (jRate is used as an implementation of real-time Java to develop some basic applications),

- BOSSA a platform that integrates real-time schedulers under Linux.

4.7 SEC course

This course deals with a large spectrum of the security issues regarding the embedded systems and aims to familiarize with the security management policies. This course approaches the security policies and the standards often met and/or imposed in various application domains. The course is supplemented by specific conferences around:

- Hidden channels and countermeasures in the field of smart cards

- Intrusion detection in the embedded domains with a particular attention to the intrusion detection in the sensor networks,

- Security issues in the RFID domain, etc.

4.8 LS course

The LS course introduces the reactive systems and the synchronous languages in the context of embedded systems.

The content includes the following aspects : introduction to the synchronous languages, reactive systems and embedded systems, principal synchronous formalisms, automat-based representation (the machines of Moore/Mealy), representation in sequential circuits, the principle of the synchronous observers and Model Checking, the Esterel language, checking in Esterel (Xeve), the Lustre language and the networks of Kahn, other synchronous formalisms such as: Signal, Polychrony, Synchronous Lucid, SynDEx, ForSyDe, synchronous distribution of code, GALS systems.

4.9 VERI course

This course provides a good acknowledgement of analysis and verification of concurrent programs, network protocols and/or distributed algorithms. It details the methods of checking by "model checking" applied to states/transitions systems, automats and Petri nets; the problem of the state space *explosion* and the solutions used to avoid this problem (partial order, compression of states, abstraction, etc.)

Presentation of formalisms and tools: from UML to SDL, Promela and coloured Petri nets: modelling, analyses and checking with Design CPN or Helena, Spin, Tau (Telelogic) or ObjectGeode (Verilog).

4.10 SdF course

The objective of this course is to focus on the needs and techniques of the reliability. The following aspects are emphasized: concepts and means of the reliability like diagnosis, redundancy, codes' errors and correctors, reconfiguration. Other notions concern architectures' validation: specification of properties, modelling of architectures, properties checking, assessment, avoidance of execution errors, development process of sure systems, as well as tools for reliability.

4.11 AJL course

The aim of AJL course is to understand the concepts and the solutions of the technical architectures dedicated to multi-players and ubiquitous video games. After a short presentation of the principles of the video games, and the massively multi-player games, the course focuses on a description of the architectural elements of the on-line multi-players games: algorithmic, services, problem of the cheating, etc.

In a second part, students work on scientific papers related to ubiquitous games. Lastly, the course ends with a practical work which allows to work with a multi-player game on mobile phones using a GASP platform with Java language.

4.12 RQoS course

The goal is to know the various aspects of Quality of Service (QoS) in the networks and applications which can influence the design of a network game. After a short recall of QoS concepts, at the network's level as well as at the application's level, mechanisms of QoS management are seen by the students in various technologies: WIFI, Internet, mobile telephony. The impact on the applications of the QoS provided or required will be used as a main idea within the course. Certain specific protocols of the multimedia applications are studied, in particular RTP/RTCP, the multicast-IP with PIM, (its various alternatives). Architectures of video streaming (numerical TV on Internet) or audio (radio broadcast), the co-operative applications are presented. The course ends with the presentation of the recent research works on QoS in the network and multimedia contexts. For example, new versions of TCP, the TCP-friendly approaches, multimedia applications in peer to peer (P2P).

4.13 The internship

At the end of the Master's degree, the training is supplemented by 6 months internship. This internship could be carried out in a laboratory or in a company. In case the internship is done within a laboratory, the delivered diploma leads the student to pursue a thesis in a laboratory. Otherwise, if the internship is done within a company (in general within a service of R&D of a wide company or a specialized enterprise in embedded and/or critical systems), the delivered diploma is professional-oriented. The trainee is followed up during his internship by a member of the pedagogical team. At the end of the internship, the student must submit a report describing his/her work and builds up a defence. The mark obtained counts for 18 credits in the final mark obtained for the Master. The internship may let the student to prepare a thesis in the company (CIFRE) or to lead to recruitment.

The student obtains his Master degree if he/she only obtains a global mark higher or equal to 10/20. A minimal mark of 10/20 is

required for mandatory courses, while a minimal mark of 8/20 is required for optional courses. Exceptionally, and based on the decision of the pedagogical team, the Master level 2 may be prepared over 2 years.

5. MASTER RESULTS

The internships we propose for students are completely coherent with the research topics of the SEMpIA research team, since these internships belong to research projects of the team. In addition, experimentation works carried out within the courses allow a better understanding of the embedded and mobile themes. These courses are also taught during the evening to lifelong learners.

In this section, we present some results that may be used to evaluate the EMS Master quality. Thus, curves C1 and C2 of figure 3 show the Master evolution in terms of the number of admitted students and the number of success. Curve C2 does not show the number of students that succeed on September 2010, because when this paper was written, this information was not available. The evolution of curve C1 can be interpreted as the growing interest of the students and their motivation to embedded and mobile systems, a domain where hiring is relatively important. Indeed, we note the case of students that take the EMS courses but that are recruited before starting their internship. For example, among 34 students of this year, 5 students had job opportunities as developer engineers instead of internships. Moreover, 80% of the students that obtain the Master degree work in the embedded and mobile systems (avionics, house automation, smart cards, mobile telephony, military transport, etc). Each year, one or two students start a PhD thesis as shown in curve C3.

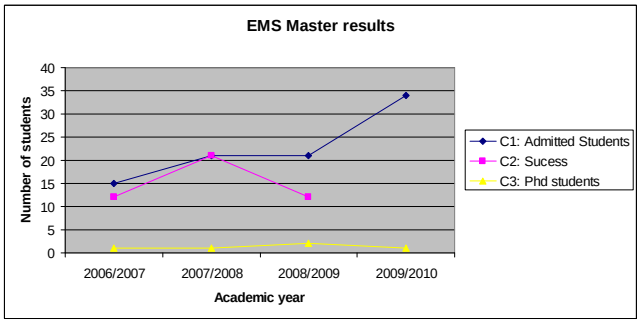


Figure 3: Statistics concerning EMS students

6. CONCLUSION: TOWARDS SEMS MASTER

The success of the EMS' Master encouraged the pedagogical team to propose an enhanced Master for September 2011 that will replace EMS and that is called SEMS (Secure Embedded and Mobile Systems). This is motivated by the following observation:

The request of experts, developers, architect designers but also scientists in the field of secured embedded and mobile systems is in full expansion. The training is proposed to better apprehend the various aspects of the design of architectures which are viewed under two complementary angles:

- Execution environments constrained in terms of resources (CPU, memory, energy, connectivity, etc.)

- Security and reliability (rigorous design, test, checking, proof of programs, etc).

The Master SEMS keeps some courses of the EMS' Master that are considered as the heart of the Master EMS like PFSEM, RSEM or SEC, enriches some other courses like VERI and creates new courses such as a course on Network Senses for example.

7. ACKNOWLEDGMENTS

Thanks to Pierre Paradinas which was the creator of the Master EMS when he was the professor of the embedded systems chair at the CNAM. Thanks also to the current pedagogical team (Eric Gressier-Soudan, François Anceau, Samia Bouzefrane, Jean-Ferdinand Susini, Selma Boumerdassi, Françoise Sailhan, Pierre Courtieu, Nicolas Pioch and David Delahaye) that takes care of the good progress of the Master and worked hard to elaborate the Master SEMS. Special thanks to Pamela Ingrassia, our Master secretary.

8. REFERENCES

[1] CNAM: http://www.cnam.fr

[2] Musée des Arts et Métiers : http://www.arts-et-metiers.net/

[3] CEDRIC Lab: http://cedric.cnam.fr

[4] ENJMIN : http://www.enjmin.fr

[5] Institut National de la Recherche en Informatique et en Automatique, http://www.inria.fr

[6] SEMpIA research team, http://cedric.cnam.fr/AfficheEquipe.php?id=14&lang=fr

[7] EMS Master description, http://deptinfo.cnam.fr/master/spip.php?rubrique16

[8] PFSEM course, http://cedric.cnam.fr/~bouzefra/cours_pfsem09-10.html

[9] ECE, école centrale d'électronique, http://www.ece.fr

[10] ENSIIE, école nationale supérieure pour l'industrie et l'entreprise, http://www.ensiie.fr/

[11] ISEP, http://www.isep.fr/

Xest: An Automated Framework for Regression Testing of Embedded Software

Matthew H. Netkow
Solution Architecture Team
The SAVO Group
Chicago, Illinois, USA
matt@dotnetkow.com

Dennis Brylow
MSCS Department
Marquette University
1313 W. Wisconsin Ave.,
Milwaukee, WI 53226
brylow@mscs.mu.edu

ABSTRACT

As embedded systems permeate an ever-widening circle of safety- and mission-critical applications, robust testing of embedded software remains of paramount importance. Yet narrow I/O channels, scarce memory and processor resources, real-time and interrupt-driven behavior, and low-level source languages make state-of-the-art validation techniques much more difficult in an embedded context. For students, for whom testing is often already a secondary concern, the challenges in methodical testing of embedded systems can appear insurmountable. We present the Xinu External Suite Tester (XEST) framework, a tool for automated, parallelized regression testing of embedded software kernels running directly on real embedded hardware. We discuss the requirements for such a system, and evaluate its power as both a quality control mechanism in an actively developing system and as an assessment tool for students in conjunction with the Embedded Xinu experimental laboratory.

Keywords

Embedded systems; Testing; Xinu; Nexos

1. INTRODUCTION

Testing of embedded system software remains challenging for several key reasons. Embedded systems typically have narrower communication channels and tighter resource restrictions for testing overhead than comparable non-embedded systems. Embedded software is by and large still written in C and assembly, languages that resist modern static analysis and model checking approaches. Reactive and real-time embedded systems have dense, interrupt-driven control flows that baffle analysis tools and cause explosively large state spaces. Simulations are common, but few modern embedded processors have industrial-strength, cycle-accurate simulators available, and simulation of the entire system additionally requires accurate models of peripheral components and ambient environments.

As embedded systems permeate an ever-widening circle of safety- and mission-critical applications, robust testing of embedded software remains of paramount importance. In this paper we present the Xinu External Suite Tester (XEST) framework, a tool for automated, parallelized regression testing of embedded software kernels running directly on real embedded hardware. Our implementation works with the Embedded Xinu experimental embedded systems laboratory, but exists as a separate tool that can be applied in many contexts.

The contributions of this paper are twofold:

1. We present Xest, a testing framework designed to provide a simple, intuitive mechanism to emphasize test-driven development in embedded system courses, and
2. We evaluate the effectiveness of Xest in the context of an existing embedded operating system course.

In the next sections of the paper, we provide a review of prior and related work in the area of embedded software validation, concluding that there is a clear need for a new tool of this type. Next, we provide background on the Embedded Xinu laboratory infrastructure and the applicable prerequisites for Xest to be useful. We discuss capabilities and design decisions inherent in our prototype implementation, and give examples of its usefulness. Finally, we evaluate Xest in the context of both an active kernel development project and as a driver for teaching test-driven development in our embedded operating systems course.

1.1 Prior and Related Work

Software validation in general, and analysis and testing of embedded systems in particular, have been rich areas of research in recent years.

Specific properties of embedded systems have proven to be amenable to static analysis techniques that model and check system source code. Subproblems like worst-case execution time (WCET) analysis have been addressed through automatic derivation of loop bounds and detection of infeasible paths by abstract execution [13]. Similarly, worst-case stack size can be estimated [19]. Many additional properties can be validated through the extensive research of the model checking community; embedded systems remain a difficult target because reactive and real-time embedded systems have dense, interrupt-driven control flows that baffle analysis tools and cause explosively large state spaces [6].

Embedded software is by and large still written in C and assembly, languages that resist modern static analysis and

model checking approaches. Many alternatives have been proposed that limit the expressiveness of the language in return for enhanced checking and domain-specific features, like the nesC language [12]. Such programming language innovations present an important path forward, but face an uphill battle in gaining wide acceptance in the industrial world. For the foreseeable future, testing of embedded systems remains an important part of mainstream practice.

Simulations of embedded systems can be an important cost-effective and time-saving alternative to testing alone. However, few modern embedded processors have industrial-strength, cycle-accurate simulators available. High quality research simulators exist for platforms ranging from the Motorola Coldfire [10] to the Atmel AVR [23], but major embedded architectures like the MIPS and ARM processor families remain poorly represented. Berg et al. [1] have presented guidelines for processors to be more readily analysis and simulation-friendly, but it remains to be seen how this advice will be taken to heart by the industry.

Proper simulation is much more than a matter of modeling processor behavior, however. Embedded simulation often relies on careful modeling of peripherals, environment, or parallel subsystems. Some narrow domains have seen progress in this area, such as sensor networks, which have both fine-grained network simulators [18] and simulators that scale to thousands of nodes [15]. General-purpose tools that are widely-applicable beyond a narrow domain remain elusive. In short, cycle-accurate simulators can be extremely helpful, but often aren't scalable, don't model peripherals, or most frequently do not exist.

Generation of proper testcases has been another rich area of research for decades. The WISE tool [8] delivers automated test generation for worst-case complexity. Complexity has an important correlation with timing requirements in embedded systems, but like many good test generators, WISE works with Java, a language that has seen only limited deployment in the embedded community.

The Save-IDE [20] is an integrated development environment for component-based embedded systems. It supports component-based design, provides generated C language skeletons, and works directly with an integrated model checker. The Save-IDE presumes the presence of an operating system layer that supports the component-based design paradigm; in contrast, our work targets a lower level of the system, allowing for direct testing of operating system layers.

The importance of automated testing tools has been highlighted in [2], which admonishes practitioners not to introduce coverage analysis and visualization tools in projects without a reasonably usable automated test suite.

Finally, pedagogical research continues to reinforce the importance of test-driven design and test-first strategies in and out of the classroom:

> "We found that test-first students on average wrote more tests and, in turn, students who wrote more tests tended to be more productive. We also observed that the minimum quality increased linearly with the number of programmer tests, independent of the development strategy employed."[9]

2. THE LABORATORY ENVIRONMENT

We have based Xest on the Project Nexos [25] suite of tools, a cooperative, multi-school effort to bring hands-on embedded systems laboratory exercises into core undergraduate computer science and engineering courses. Extensive details on the software, hardware, and curricula are available in the existing literature [3, 4, 7], and half a dozen more schools have constructed or begun to construct Nexos-enabled laboratories in the past two years.

The embedded operating systems course described in this paper focuses on the Linksys wireless router as the embedded target platform. The Embedded Xinu kernel [5], a core component of Project Nexos curricula, runs on six distinct models of 32-bit MIPS-based router platforms produced by Atheros, Broadcom and their competitors, in addition to the Qemu virtual machine. While the examples in this paper are tied primarily to the MIPS router platform, the flexibility of this infrastructure has been amply demonstrated with other hardware ranging from 8-bit ATmega1280 Arduino boards, to 16-bit Motorola 68HC11 and 68HC12 development boards [11], to 64-bit dual-core Pentium workstations with PXE boot capabilities.

These systems all have two key features that we rely on:

- At least one serial port that can act as a console device for the O/S and any firmware bootloader, and
- Firmware that can upload a fresh O/S kernel over an I/O device at boot time.

These two features are available for many embedded systems, reflecting the more mature design practices visible in many segments of the embedded market. Debugging serial consoles are the norm in the consumer embedded market, and also many popular research platforms, like Berkeley Motes running TinyOS [22]. While some primitive 4- and 8-bit embedded microcontrollers are still in use, many device manufacturers have moved to inexpensive 8-, 16- and 32-bit cores with high-level firmware controlling the boot process. One-Time-Programmables and dedicated ROM program stores are no longer very common, replaced instead by flash ROM. With the prevalence of mutable forms of permanent program storage, it is now common to see embedded systems with provisions for uploading/updating system images. In this realm, rapid prototyping of embedded operating systems software is the new norm.

We organize our target platforms into a remotely-accessible pool, as detailed in [7]. Users on a "front-end" workstation compile their code with a MIPS platform cross-compiler. Specialized tools upload the compiled system image to the central server, select an available target hardware "back-end", and power it on. Control sequences sent to the embedded firmware over the serial console during power up interrupt the boot sequence, and redirect the system to upload a fresh system image over the network from the central server. The new operating system is loaded over the network into RAM on the target device, and begins execution. The serial console connection is referred back to the console of the front-end user, who can then interact with their kernel as though connected directly to the embedded device.

The advantages of this infrastructure are many in both the classroom setting and in the production environment. Specialized hardware is required both for aggregating the embedded console devices and remotely power-cycling the hardware. Commercial solutions exist for both and we provide greater detail in previous papers and at [5].

2.1 Xest Prerequisites

The Xinu External Suite Tester (XEST) framework was developed in the context of the experimental embedded systems infrastructure described above. The key prerequisites for our testing tool are thus:

- Embedded platforms that can be remotely power-cycled into a "known good" state,
- Provisions for unattended upload of software image to embedded device, and
- At least one stable I/O communication channel with a "fail safe" device driver.

If parallelized testing is to be supported, there is a further requirement of a test pool of:

- Multiple, concurrently accessible target platforms.

Testing infrastructures of this scope are not unusual in the context of desktop and server systems. Our work establishes that these prerequisites can be both beneficial and inexpensively achievable in the embedded system context.

3. XEST

Next we present the details of the Xinu External Suite Tester (XEST) system.

As established in previous sections, testing is important for checking that software meets the requirements set forth by its specifications. Test automation aims to reduce testing effort by allowing software to repeat testing tasks quickly and precisely in place of a manual tester. Automation can be applied to many types of software testing and results in cost effective and time saving benefits. Specifically, regression testing is useful for uncovering software regressions – a problem that occurs where previously working functionality stops working as intended as a result of the addition of new changes.

Regression testing of embedded systems remains difficult. The system needs to be robust, easily maintainable, and easily able to interact internally and externally with our embedded target hardware.

One significant challenge is to interact with the embedded operating system kernel with minimal code overhead. An age-old problem with systems testing, the insertion of test code changes system properties ranging from code size to deadline constraints and cache misses. Our solution is to make use of the existing "test hook" already present in the Xinu shell interface, `xsh_test.c`. By using a dedicated test hook already present in the embedded kernel, we disturb as little as possible in the rest of the system.

3.1 Implementation

While the Embedded Xinu system itself is primarily implemented in ANSI-standard C and MIPS assembly, we chose to implement Xest in two standard scripting languages, *bash* and *expect*, due to their portability, wide availability, and powerful flexibility.

An overview of the full version of Xest is given below; this instance of the tool is designed to run unattended, parallelized nightly regression tests of an embedded kernel production system. In Section 5, we present a simpler, preconfigured instance of the tool used for teaching purposes.

System Initialization

Before running Xest, various attributes are set in the system configuration file related to two main areas: version control and automatic notifications. By default, the system expects to retrieve a current copy of the target source code from a version control system, such as Subversion. Alternatively, a local directory tree can be specified. Notification options allow custom e-mail subject headers, specific targets for testing results, and modes that control output verbosity.

Test Creation

Given our serial link with the embedded hardware, testcases are comprised of three components: a test hook that runs on the embedded platform, an input to the serial console, and an expected output. The data file format is plain human-readable text, which not only allows easy debugging of the test harness, but can combine well with existing tools for automatically generating testcases for coverage or other criteria. Test hooks consist of `xsh_test.c` files to be dropped into place during kernel compilation. The test hooks follow a straightforward sentinel marker system:

```
printf("===TEST BEGIN===");
[ code to test ]
printf("===TEST END===");
```

Once the controlling `expect` script sees the `TEST BEGIN` line, it starts capturing data as the output to the testcase. Once it sees the `TEST END` line, it stops and compares the collected output with the expect output. Shorter sentinel values could be used for more time-sensitive contexts, (at 115K baud, the start sentinel will take more than 2 milliseconds of asynchronous serial processing,) but this has not proven to be a problem with the prototype system.

Solution Initialization

In an educational setting, it is often the case that simple typos in the expected output will cause a testcase to fail, even when the system is correct. One natural solution for this is to add capability for correct outputs to be generated by a "known good" reference implementation of the assignment. When such a reference implementation is available, the Xest `initialize.sh` script correctly generates expected outputs for each test input.

Within this expect script, a connection to a Xinu backend is made and a command to 'test' is sent. The embedded kernel runs the specified test hook, and output is collected. As mentioned above, the expect script waits until it sees the `TEST BEGIN` sentinel before saving results into the output buffer. This discards unnecessary output generated by target platforms booting remote kernel images, which can be substantial. Once the `TEST END` sentinel is seen, the connection to the backend is closed and the log is returned.

Optional parameters provide additional functionality. When run with no parameters, all testcases are initialized. By using a *-tc* parameter, the user can specify certain testcases to initialize. This is useful for initializing a new testcase or a testcase that failed to initialize properly the first time. Lastly, a help option exists that gives an explanation of the three ways to initialize Xest. All existing testcases can be listed for easy reference.

External Testing

Once solutions have been initialized, testing is begun by executing the `xest.sh` script. Xest determines which testcases to run – either all of them or those specified as parameters. Next, the appropriate source revision to use for testing

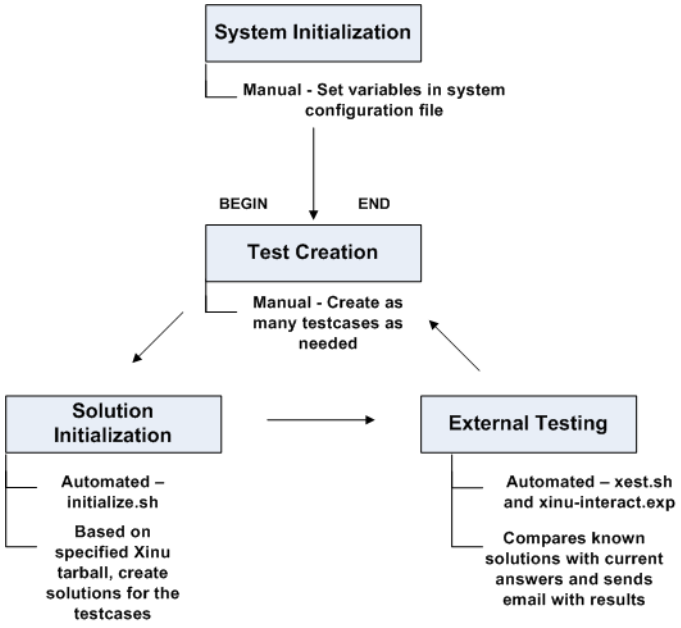


Figure 1: Xest Flow Chart

is selected based on the system configuration file. Each test case is compiled and run against a previously "known good" output.

Once all tests have completed, the `xinu-interact.exp` script returns control to the main Xest script. The log file contains the results of each testcase and is compared to each previously known solution; if the test does not pass, the result is placed into a problem output file (`badoutfile`).

Similar to solution initialization, Xest can be run with parameter specifications. If the user runs

```
./xest.sh
```

with no parameters, all tests are run. Similarly, using the *-tc* parameter, the user can specify individual tests to run. This is useful for checking a new testcase or rerunning previously failed cases.

```
./xest.sh -tc 01-testOne 02-testTwo
```

Lastly, a help option exists:

```
./xest -help
```

that behaves similarly to Xest initialization help and gives an explanation of the three ways to run Xest.

4. XEST AS GRADING ASSISTANT

While Xest was initially developed as a regression tool for the Embedded Xinu development effort, its usefulness as an assessment tool for the Nexos curriculum quickly became apparent. In this section, an overview of the components in Xest that contribute to automated evaluation of student kernels is given in Figure 3. The next section will discuss Xest components used as a tool for students to test their own work prior to submission.

Support for Multiple Courses

The Embedded Xinu operating system is used for teaching hardware systems (assembler programming), operating systems, embedded / real-time systems, embedded networking, and even compilers courses. The testing requirements of these various courses are disparate enough to warrant additional abstractions in the Xest design to support this. Components of Xest have even been employed to evaluate Java programs in an introductory programming course, demonstrating its flexibility and breadth.

Xest supports template directories which contain skeleton configurations and testing scripts for each course to be supported. These templates include tests, grading scripts and solution files, as well as assisting utilities specific to individual courses. The presence of this template system allows for faster specializations based upon project testing requirements.

Error Checking To account for errors of both the student and system varieties, various new methods of checking for errors were necessary:

1. Checking for existence of files, directories, etc.,
2. Logging errors into a separate file for later review, and
3. Attempting to recover from student homework submission errors.

4.1 Implementation

The standard cycle for student code assessment with Xest is given in Figure 3, and explained in the following sections.

System Initialization

Before using the Grading Component of Xest, the user sets various attributes in the system configuration file. Items such as the course and semester are set so that the correct course reference materials are used. Next, the user executes the `courseInit.sh` bash script. This script copies over course-specific files and directories from the `courseTemplates`.

Project Creation

Before a new directory for a homework assignment can be created, the user must specify various variables in the project configuration file (much like the system configuration file). These variables include the path containing students' submitted assignment, the particular scripts needed for grading the assignment, the number of points for the assignment, and a list of the assigned code files that the students worked on. Upon executing the `createNewProject.sh` script, a new directory for the project is created and template scripts and test cases are copied into it.

Grading

Upon executing the `rungrades.sh` script, the automated evaluation process begins.

If an optional parameter is given (the number of backends to grade on), the system can grade in parallel. To do so, the program looks up available backends using the pool management utilities and calls the `batchgrade.sh` script with names of the students to be graded. The total number of student tarballs is distributed across the pool of available backend targets and evaluated by the `grade.sh` script. If no parameter has been given (non-parallel mode), the `grade.sh` script is called on each student submission in serial fashion. The `grade.sh` script does most of the work; first, a temporary directory is created for the student's tarball. Next, the student's tarball is located from the `input` directory and untarred into the new temp directory. Any files that are necessary for grading, including special grading logic (`initialize.c`) are copied in as well. After compilation, each testcase is copied into the temp directory.

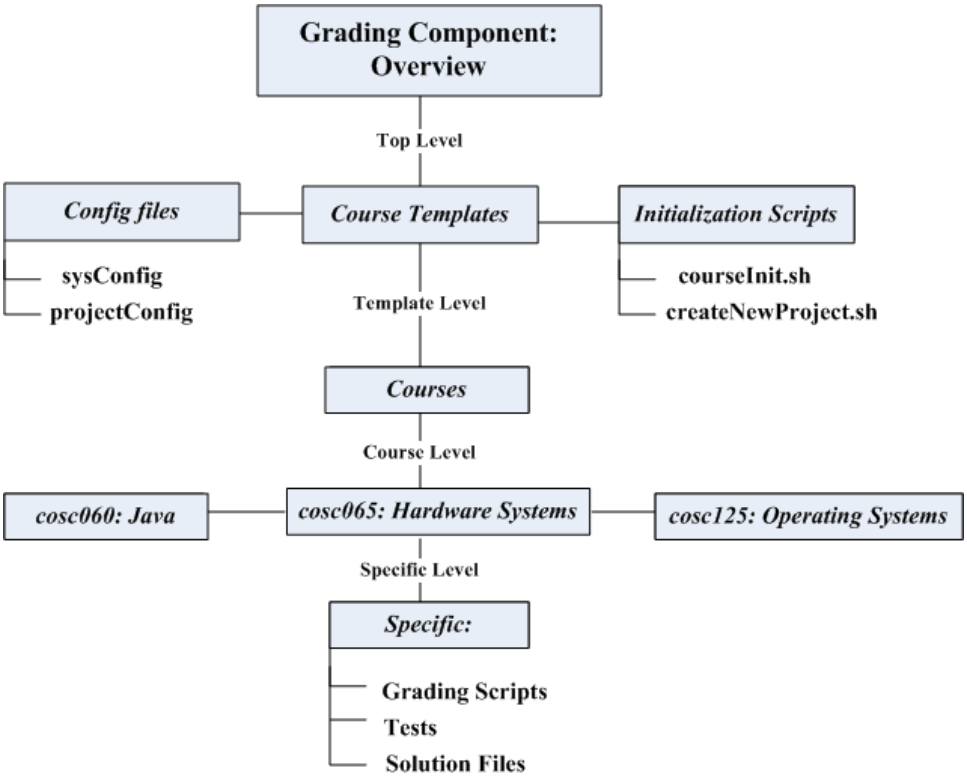


Figure 2: Xest Grading Component Overview

Each of the testcases is used by an expect script to connect to a Xinu backend, run the test, and return with resultant output. If there are no discrepancies with the expected output, the testcase passes. Otherwise, the non-matching output is stored into a file for later review. Through the entire grading process, any errors that are encountered are logged into the output file. Special errors, such as those that involve compilation or files/directories are logged in `problems.txt` for later review.

Reporting

Reports are automatically generated for each student submission, containing header information such as names, assignment number, points, and other details about the submission. Files of interest (identified by the per-project configuration file) are included in the report, and then the results of each test case are given.

5. NIGHTLY REGRESSION TESTING

We have evaluated Xest in the context of two completely different settings. The first of these is as the nightly regression test manager for our local development branch of the Embedded Xinu operating system. The second, described in the next section, is as a teaching tool to help habituate a test-first implementation strategy for students developing their own kernel components in an embedded operating systems course for undergraduates.

Automated regression testing has long been a best practice for loosely organized development teams collaborating on complex software implementation. What is novel about our use of Xest for nightly regression tests is that it involves taking control of a pool of dedicated target hardware to run the complete testsuite on the embedded platforms.

An entirely unattended cron job checks out the latest revision of the code base each night. Each test item in the suite is recompiled into a separate test kernel; this is expensive in overall cross-compilation time, but assures minimal overhead in the resource-sensitive embedded memory footprint. Results are compiled automatically, and are waiting in the e-mail inboxes of the developers when they awake in the morning.

By compiling each test segment directly into the kernel, Xest is able to provide in-depth unit and integration testing of the system. This includes stress testing of subunits not accessible from the user-level API, independent validation of kernel data structures, independent validation of timing properties, and loopback testing of device drivers and their hardware. Injecting test network packets destined for the remote target is also supported, but is rarely a good idea when parallel testing on a private network or when the serial console is monitored using a RS232-over-Ethernet serial aggregator on the same subnet. Instead, internal compilation of the test packet data increases memory overhead on the target machine, but eliminates a major source of nondeterminism that can cause false positive test failures.

The results of a typical Xest run of the kernel internal tests is shown in Figure 4. The tests include validation of the C language execution environment on the embedded target, preemptive multitasking, interprocess communication primitives, hardware device drivers, the micro C libraries, the memory and buffer pool allocators, and the various layers of

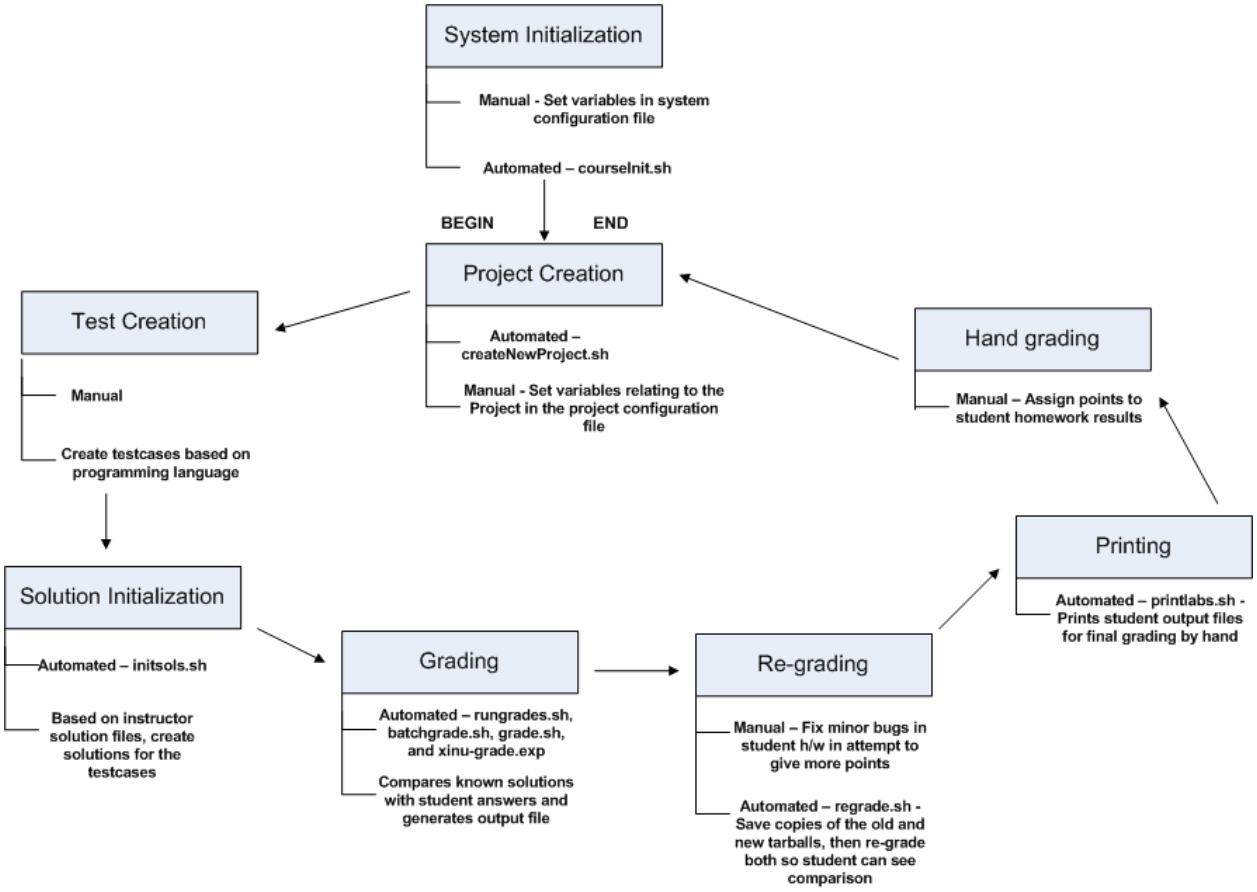


Figure 3: Xest Grading Component Flow Chart

the TCP/IP networking stack.

Xest has been deployed as a nightly regression tester for more than eight months, and has thus far detected six distinct regressions that had slipped past developers. The system has been particularly valuable in that it can be configured to perform the testing on each of the embedded platforms supported by the unified Embedded Xinu kernel, ranging from little-endian MIPS processors with a Broadcom "Silicon Backplane" architecture to big-endian MIPS cores with Atheros peripherals and a cascaded programmable interrupt controller. This allows us to catch errors caused by changes that work on one platform, but break on another; relying on developers to manually run regressions on each of the supported platforms each time they commit a code change was simply not practical.

6. XEST AS TEACHING TOOL

It comes as no surprise to experienced software engineers that test-driven development tends to produce higher quality code than a "write first, test later" approach. Faculty have published much work on their struggles and successes with entry-level testing frameworks like JUnit [14] for Java programmers [17, 24]. Unfortunately, equivalent tools have not been freely available for embedded systems.

Proprietary testing tools exist that provide unit, integration and system testing for typical embedded languages like C and Ada [21], or focus on safety-critical areas like medical device validation [16]. Such tools are generally not freely available to educators or students, and cater to professional developers with many years of experience. Thus, we do not consider these solutions to be good choices for college-age students new to embedded development. In this respect, we believe that Xest provides a one-of-a-kind, easy-to-use, entry into the world of embedded regression testing.

In Spring of 2010, we deployed a simplified version of Xest into our undergraduate embedded operating systems course; the structure of the course, syllabus, and assignments are detailed in [4]. Earlier versions of the tool had been used by the instructors for several years to assist in grading student work. Spring 2010 marked our first effort to put this tool into the hands of our students.

The simplified tool comes in two forms. The first is preconfigured to run simple input/output comparison tests between a reference implementation and a student implementation. A simple directory of testcase inputs and outputs is created, and the tool is run with the paths to the two versions to be compared. A test report presents the differences between the outputs for tests that do not match.

The second form of the tool was added in the third assignment, and includes the "test-hook" introduced in Section 3.1.

6.1 Inculcating Test-Driven Development

The fundamentals of testcase creation and the Xest inter-

```
Test Suite  1: Argument Passing     [PASS]
Test Suite  2: Priority Scheduling  [PASS]
Test Suite  3: Thread Preemption    [PASS]
Test Suite  4: Recursion            [PASS]
Test Suite  5: Single Semaphore     [PASS]
Test Suite  6: Multiple Semaphores  [PASS]
Test Suite  7: Counting Semaphores  [PASS]
Test Suite  8: Killing Semaphores   [PASS]
Test Suite  9: Process Queues       [PASS]
Test Suite 10: Delta Queues         [PASS]
Test Suite 11: Standard I/O         [PASS]
Test Suite 12: TTY Driver           [PASS]
Test Suite 13: Character Types      [PASS]
Test Suite 14: String Library       [PASS]
Test Suite 15: Standard Library     [PASS]
Test Suite 16: Type Limits          [PASS]
Test Suite 17: Memory               [PASS]
Test Suite 18: Buffer Pool          [PASS]
Test Suite 19: NVRAM                [PASS]
Test Suite 20: System               [PASS]
Test Suite 21: Message Passing      [PASS]
Test Suite 22: Mailbox              [PASS]
Test Suite 23: Ethernet Driver      [PASS]
Test Suite 24: Eth Loopback Driver  [PASS]
Test Suite 25: Network Addresses    [PASS]
Test Suite 26: Network Interface    [PASS]
Test Suite 27: ARP                  [PASS]
Test Suite 28: Snoop                [PASS]
Test Suite 29: UDP Sockets          [PASS]
Test Suite 30: Raw Sockets          [PASS]
Test Suite 31: IP                   [PASS]
Test Suite 32: User Memory          [PASS]
```

Figure 4: Xest Testsuite for stock Xinu kernel

face were presented as part of the first laboratory demonstration at the start of the term. The advantages of test-driven development were discussed, but students were given a choice how to proceed. We asked students to submit their testcase suites if they developed any, but made it clear that points would still be assigned, as always, based upon the correctness of their submission. Critically, no additional points were offered for students to develop or submit testcases; we presented the testing tool, and allowed students to draw their own conclusions on how useful the tool would be.

When the first laboratory assignment was submitted a week later, 35% of the students chose to submit testsuites in addition to their implementation source code. Anecodotally, students who chose to build an extensive testsuite demonstrated a deeper understanding of the assignment structure. Quantitatively, students who employed this test-first methodology scored an average of 150% more points than their non-testing peers in a class of 38 students. Again, to emphasize, no additional points were assigned because of student testing – but the students who choose to build a testsuite significantly outperformed their peers on the faculty grading test set. In laboratory observations, we noted only a couple of cases where students first completed the assignment, then went back to build a test suite. Students generally either committed to testcase creation from the outset, or reverted to their usual ad-hoc testing methodologies.

For the second assignment, voluntary testsuite submission increased to 40% of the group. Word had spread once graded work from the first project was returned – testing really pays off. It was highly instructive for us to see students moving to a test-first approach because they perceive it to be a best practice. They perceive this not because we've *told* them it is a best practice – even though our students, like those in many other programs, are told this from early in their careers – but because they can see their colleagues developing markedly better software as a result.

6.2 Results

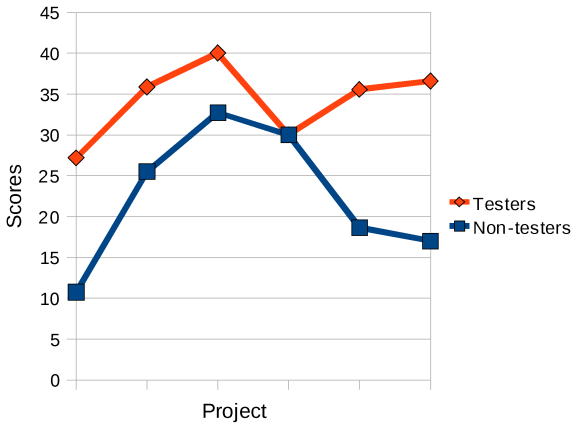


Figure 5: Testing Students vs. Non-Testing Students

Figure 5 shows relative student performance across six project assignments. The first group elected to build a testsuite and use the Xest tool for regression testing during development; the second group used only ad-hoc testing methods, and was generally observed to use a "test last" strategy for assignments. The groups are not fixed – members of the testing group were detected in each project by the presence of a Xest testsuite in their submitted directory structure. Students who did not have such a directory present were assumed to not be using Xest for test-driven development.

Project 4 is salient in the results because that was the only assignment in which students were *required* to submit a Xest testsuite with their work.

As the data in Figure 5 shows, students who elected to build a Xest testsuite outperformed their counterparts by between 22 to 153%, for an average of 84% overall. By the time we reached Project 7 in the course, more than 80% of the students were electing to build their own testsuites as part of their development cycle. Data for Projects 7 through 10 are not presented because the non-testing cadre of students had become too small of a remnant for fair comparison.

In summary, the intervention was quite successful. Over the course of the term, nearly all of the students chose to at least partially embrace test-driven development tools, and for each assignment, those who did outperformed those who did not. Moreover, even though only a third of the students initially were interested in this approach, the vast majority came to adopt early testing when they perceived that their peers were having greater success using this methodology.

7. CONCLUSION

Regression testing continues to be an important factor in the creation of robust software. To incorporate this type of testing into the Embedded Xinu project, we created a flexible framework of components for both internal (code in the kernel) and external (code on the test hosting computer) testing of an embedded operating system. Our system runs on real embedded hardware and has been configured for regression testing of an active kernel development project, for assessing student assignments, and for students themselves to use as a test-driven development tool in their coursework.

Students who adopted the Xest testing tool in our course consistently produced better results than those who relied on ad-hoc testing methods. By the end of one semester, the vast majority of students involved in the course were choosing to use our embedded regression testing tool on their assignments as a natural part of their development cycle.

While static analysis techniques, model checking, simulators, and domain-specific programming languages all have an important role to play in making embedded systems safer and more robust, automated testing remains an important piece of the puzzle. Our work demonstrates that automatic testing of embedded O/S kernels can be both beneficial and inexpensively achievable. Our software is used by both students and developers working with the Embedded Xinu experimental system infrastructure, and is available from the Embedded Xinu software repository [25].

7.1 Future Work

Our next round of improvements for the Xest system will include expanded support for time-sensitive testing, using high-granularity timers on the target to annotate output stages during testing for comparison against timers on the host. We are also interested to see if continued tracking shows that students who turned toward test-driven development in this course will continue in subsequent courses.

8. REFERENCES

[1] C. Berg, J. Engblom, and R. Wilhelm. Requirements for and design of a processor with predictable timing. In *Perspectives Workshop: Design of Systems with Predictable Behaviour*, Dagstuhl, Germany, 2004.

[2] S. Berner, R. Weber, and R. K. Keller. Enhancing software testing by judicious use of code coverage information. In *ICSE '07*, pages 612–620, 2007.

[3] D. Brylow. An experimental laboratory environment for teaching embedded hardware systems. In *WCAE 2007: Workshop on Computer Architecture Education*, pages 44–51. ACM Press, June 2007.

[4] D. Brylow. An experimental laboratory environment for teaching embedded operating systems. In *SIGCSE '08*, volume 40, pages 192–196, March 2008.

[5] D. Brylow. Embedded XINU project, 2010. `http://www.mscs.mu.edu/~brylow/xinu/`.

[6] D. Brylow and J. Palsberg. Deadline analysis of interrupt driven software. *IEEE Transactions on Software Engineering*, 30(10):634–655, October 2004.

[7] D. Brylow and B. Ramamurthy. Nexos: A next generation embedded systems laboratory. *SIGBED Review*, 6(1), January 2009. URL `http://sigbed.seas.upenn.edu/`.

[8] J. Burnim, S. Juvekar, and K. Sen. Wise: Automated test generation for worst-case complexity. In *Proceedings of ICSE '09*, pages 463–473, 2009.

[9] H. Erdogmus, M. Morisio, and M. Torchiano. On the effectiveness of the test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(3):226–237, 2005.

[10] C. Ferdinand, R. Heckmann, M. Langenbach, et al. Reliable and precise WCET determination for a real-life processor. In *EMSOFT 01: First International Workshop on Embedded Software*, LNCS volume 2211, pages 469–485, October 2001.

[11] Freescale Semiconductor, Inc. *CPU12 Reference Manual: M68HC12 and HCS12 Microcontrollers*, 2006. URL `http://freescale.com`.

[12] D. Gay, P. Levis, J. R. von Behren, et al. The NesC language: A holistic approach to networked embedded systems. In *Proceedings of PLDI 03: ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1–11. ACM Press, 2003.

[13] J. Gustafsson, A. Ermedahl, C. Sandberg, et al. Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution. In *Proceedings of RTSS '06*, pages 57–66, 2006.

[14] Junit.org. URL `http://www.junit.org/`, 2010.

[15] P. Levis, N. Lee, M. Welsh, et al. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In *SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 126–137, 2003.

[16] Indesign LLC. URL `http://www.indesign-llc.com/medical_device_validation.asp`, 2010.

[17] M. Olan. Unit testing: test early, test often. *Journal of Computing Sciences in Colleges*, 19(2):319–328, 2003.

[18] J. Polley, D. Blazakis, J. Mcgee, et al. ATEMU: A fine-grained sensor network simulator. In *Proceedings of SECON '04: IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[19] J. Regehr, A. Reid, and K. Webb. Eliminating stack overflow by abstract interpretation. *ACM Transactions on Embedded Computing Systems*, 4(4):751–778, November 2005.

[20] S. Sentilles, A. Pettersson, D. Nystrom, et al. Save-IDE - a tool for design, analysis and implementation of component-based embedded systems. In *ICSE '09*, pages 607–610, 2009.

[21] V. Software. Url `http://www.vectorcast.com/`, 2010.

[22] TinyOS.net. The TinyOS printf library, 2010. URL `http://docs.tinyos.net/index.php/The_TinyOS_printf_Library`.

[23] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: Scalable sensor network simulation with exact timing. In *Proceedings of IPSN'05: Fourth International Conference on Information Processing in Sensor Networks*, pages 477–482, April 2005.

[24] M. Wick, D. Stevenson, and P. Wagner. Using testing and JUnit across the curriculum. In *SIGCSE '05*, pages 236–240, March 2005.

[25] Project Nexos / Embedded Xinu wiki, 2007. `http://xinu.mscs.mu.edu/`.

A Modular, Robust and Open Source Microcontroller Platform for Broad Educational Usage

André Stollenwerk Andreas Derks Stefan Kowalewski

Embedded Software Laboratory
RWTH Aachen University
{stollenwerk,derks,kowalewski}@embedded.rwth-aachen.de

Falk Salewaski

Lacroix Electronics
f.salewski@lacroix-electronics.com

ABSTRACT

In current curricula, more and more courses endeavor to give practical examples on the usage of embedded hardware. Either by demonstrations in e. g. lectures or as hands-on practice in lab courses. In order to motivate the students a reference to current technological developments is desired. All these requirements lead to a point where a reliable and fast modifiable hardware platform is needed.

This paper describes a microcontroller-based platform, which was developed in 2004 and refined over the years at RWTH Aachen University. In addition to this platform, we also developed several extension modules in order to embed current technologies like RFID or a digital photo frame. Besides all the technological issues, we also had to pay attention to non-functional requirements like expense for the whole platform or the robustness needed for educational usage of the components.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education; B.3 [**Hardware**]: Memory Structures; B.4.3 [**Input/Output and Data Communications**]: Interconnections (subsystems); B.m [**Hardware**]: MISCELLANEOUSt'; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Microprocessor*

General Terms

Design, Experimentation, Documentation

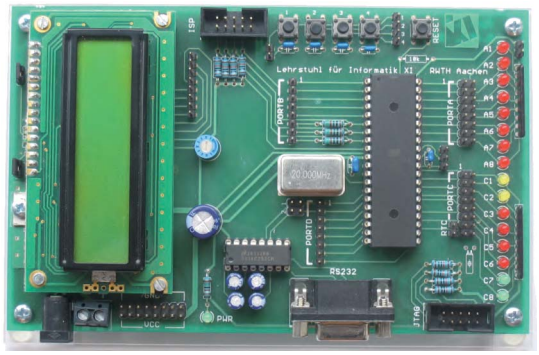


Figure 1: The basic microcontroller platform [8]

Keywords

Computer Science Education, Lab Course, Microcontroller, Evaluationboard

1. INTRODUCTION

In recent embedded systems related curricula one can note a change. Formerly the students were often taught about technologies or methodologies, on how to program embedded systems. In these cases they did not necessarily understand the interaction between the embedded system and its environment (so called cyber physical systems) [10]. One attempt solving this problem is teaching embedded systems with a direct link to an application unsing these cyber physical systems. This results in the need of more applicable examples within the courses. Either in lectures to show the students not only the theory but also how these systems act, not least with the attempt to motivate them to continue working on these examples at home, or in lab courses where they can directly work hands-on with the hardware and interact with their environment.

Both scenarios arise certain requirements to the used hardware. On the one hand, the used hardware must be reliable in terms of usability and robustness, it should be modular, to be able to augment the applications and last but not least

both, students as universities have an interest in inexpensive hardware.

We faced this challanges in 2004 when we started to create a hardware platform for our courses. After some discussions we decided to primary work with microcontrollers, due to the fact that these devices should be more intuitive to computer science students than programmable logic devices (e. g. FPGA or CPLD); especially towards freshmans. The resulting platform is shown in Figure 1.

In this article, we describe the experience we gained over the years while developing and working with our microcontroller platform and designing extension modules for it. Since we publish most of our designs including part lists on our websites [8] we invite interested others to participate in these experiences. Therefore, we give explanations on why we designed certain circuits and what the pitfalls are.

Concerning the microcontroller platform, we had several requirements. First of all the platform should be inexpensive in order to buy a sufficient number for our courses. This includes the platform itself, but also the required cables and not to forget: programming and debugging devices as well as the required tool chain. Moreover, we had in mind that it could be interested for some students to rebuild this platform for own experiments at home. Therefore we built up this platform in "old fashioned" through-hole technology (THT) and did not use any SMD components. In addition, we used sockets for most of our ICs. This enables us to exchange a broken IC without any soldering. A second educational requirement was to come up with a very simple platform to demonstrate the low number of components you need to run a complete embedded computer.

In order to allow also more sophisticated functions, we came up with a modular approach consisting of a rudimentary main unit in combination with a LCD and a collection of extension modules as described in the following chapters. Finally, an important requirement is the robustness of the platform as in our case up to 350 students work with it each semester. This requirement for robustness is including the electrical but as well the mechanical parts of the platform.

1.1 Curricular usage

We use this microcontroller platform in several courses targeting mostly CS students. Starting with some demonstrations in a first semester undergraduate lecture on *computer engineering* followed up by a lab course in the second and third semester each. During the first lab the students renew their *electrophysical basics* and in the second lab (*hardware programming*) the students get in touch with embedded software by programming an operating system on the microcontroller [16].

In addition to these mandatory courses where all computer science students of RWTH Aachen University face our microcontroller platform there are elective courses in undergraduate and graduate studies where they get in contact with the platform again. We offer some embedded system specialized lectures where the microcontroller is used for examples and demonstrations. In addition, there are optional lab courses where the students continue working with the microcontroller platform [15]. Here they e.g. implement an elevator control to gain practical knowledge about safety and reliability of embedded systems.

2. HARDWARE

As introduced in Chapter 1, many practical courses need an easy to handle, robust and inexpensive hardware platform, if they want to give the students practical experiences, which exceed just working with a software-based simulator. Simulators often have problems to show real time behaviour in an intuitive way and hide several problems of the real hardware. Moreover whatching simulations is typically less interesting. Therefore, we started to design a printed circuit board (PCB) whose elements are based on low cost and widly available components and nevertheless still is modular, so that each special application has the possibility to access the whole microcontroller. Since many of our students get very enthusiastic during working with this platform we put most of the PCB layouts and the according part lists on our website [8] to give them a possibility to rebuild the platform for their private use. The overall expense for all components is about 35 Euro and the PCB is another 30 Euro (this value scales with quantitiy).

During the first courses, we recognized the patch cords to be a regular source of errors. In addition there were many cases in which a wrong wiring caused a short-circuit. In order to make our hardware more robust towards these errors we designed many on-board configuration options as jumpers and integrate most of the single pins to either pin headers or connector plugs. With these measures, we mostly avoided loose contacts and short-circuits. In addition, wrong wirings are prevented due to the orientation and fixed pin positions within the plugs. Figure 2 shows the different components of the microcontroller platform and the possible wirings between them.

2.1 The Microcontroller

The main component of our evaluation platform is the microcontroller. There are many requirements and restrictions to this device. On the one hand, it should support many interfaces and debugging technologies like USART, TWI, SPI, ISP or JTAG as a debugging interface on the other hand it should also be cheap in acquisition. In addition, there should be existing knowledge about the microcontroller and it has to have a development environment, which is easy to use, free and has a good support for debugging purposes. It is important that the students can access the development environment without any problems; either as free software or due to any available academic licenses.

All these points lead our decision to the Atmel AVR ATmega family. We picked the ATmega32 [4] and ATmega644 [5], which are pin compatible and thus can be freely exchanged on our platform. In comparison the ATmega644 has double the capacity in all memories (64 kB Flash, 2 kB EEPROM and 4 kB SRAM).From the functional point-of-view these two microcontrollers differ just in marginal points.

There are different programming interfaces for the Atmel AVRs. First to mention is the AVRISP mkII [2]. This device can program the microcontroller but is not able to do on-chip debugging; it is available for about 35 Euro. The next step is the AVR JTAGICE 1, which is no longer produced by Atmel. Nevertheless there are different reproductions available as starting of 20 Euro [12]. Unfortunately this device is just able to program up to the ATmegs32. In order to program the ATmega644 a more comprehensive programmer is needed. At this point the AVR JTAG ICE mk II [1] (approx. 300 Euro) is needed. Thus, we try to stick for most courses

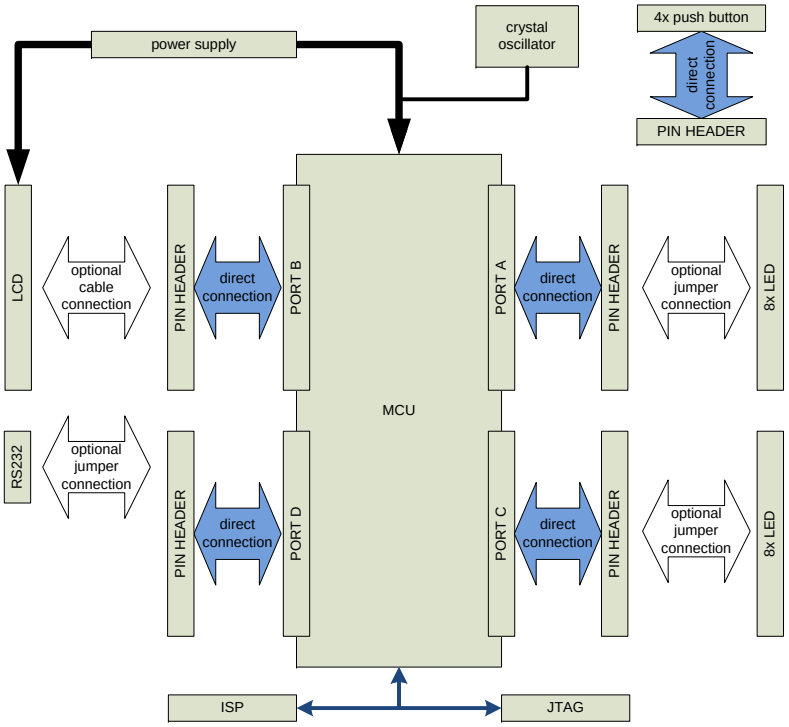


Figure 2: Block diagram of the basic microcontroller platform

to the ATmegs32, which makes the acquirement cheaper for students, who want to use this platform in private context. For some tasks, we came to the experience that a microcontroller with 2 kB of SRAM is very tight calculated. This is why we have both microcontrollers supported as freely exchangeable on our platform.

In general, there are many different ways to download the designated program onto the microcontroller. Usually in our courses the JTAG interface is used. Using this interface the students do not only have the possibility to transfer the machine code but also to debug the running device like setting breakpoints, dumping the memory or systematically manipulate the memory or registers in order to test the microcontrollers' behavior. Therefore, the microcontroller needs a connection to the development environment – the personal computer. This means some (in this case: four) of the I/O pins are permanently used by the JTAG interface. Since there are some applications that need these pins and the user does not need to be able to do in-system debugging we also placed an ISP interface on our platform. This interface enables the user to download machine code to the device but also to use all the microcontrollers' pins during runtime.

There are different development environments with enclosed tool chains to transfer assembler or high-level-language code (like e.g. C or C++) to the microcontroller (like AVR Studio or Eclipse with an according plugin). Right now Windows and Linux as operating systems are widely supported and there are even some implementations able to run directly on MacOS. In our approach, we use AVR Studio [6], which comes directly from Atmel. This software has the big advantage that the students do not need to know at which address the according registers are stored at or which configuration results out of the pattern stored in an according register. All these information is placed within AVR Studio. AVR Studio is only available for Windows.

In addition to AVR Studio, we use WinAVR [18] to be able to program in C instead of assembler. In one of our courses the students have to implement a whole operating system from scratch, which usually end up in $15-40$ kB of machine code. Implementing this in assembler would end up in a very complex program where it would be nearly impossible to support the students if they run in any problems.

Due to the fact that the ATmega microcontroller family is developed since 1996, there are many different active communities all over the world working with these microcontrollers. There are many hobbyists working with the ATmega organized in communities like *EmbDev.net*, *AVR-freaks.net* or *Arduino.cc*, whose freely available knowledge one can benefit from. However, there are also many projects run in academia based on ATmega microcontrollers, like [17, 14] as an educational example but also in research projects.

2.2 Interfaces

In the previous section, we presented and motivated the decision to the type of microcontroller we have chosen for our platform. This makes up the heart of our platform but at this point, the user is not able to interact directly with the hardware. To do so we also placed some additional devices directly on the PCB. The focus with these devices is on the one hand to keep it inexpensive but on the other hand to offer many possibilities.

In general, the microcontroller offers several interfaces. For most of them, it is necessary to supply a certain wiring between the pin of the microcontroller and the according plug on the PCB. Since all the ports of the microcontroller are available to the user, there is still the chance to connect

some more complex or extensive special purpose extensions to our platform.

2.2.1 User Interfaces

The first class of user interface (UI) embedded to our platform are buttons. Four of them were placed on the PCB. This either offers the user the chance to interact via interrupts or polling directly with the hardware. These Buttons are debounced by a 100 nF capacitor each. For one of the buttons, the students can disable this debouncing by removing a jumper in order to understand the problem of bouncing buttons better and how to solve this problem in software.

As a second class of UIs there are light emitting diodes (LEDs) placed within the PCB design. There are 16 LEDs on the platform, which can be connected freely. They enable the students to get a very simple feedback from the microcontroller.

The LEDs are connected to ground, which results in a logical `0` illuminating the LED. This configuration goes back to the time of microcontroller with only open collector or open drain outputs. Since the students shall work also with existing hardware in their later life we decided to show them also the small pitfalls that may occur in real life.

The big disadvantage of LEDs is that you need one output pin per LED, which ends up in half of the microcontrollers' pins being blocked if you want to use all 16 LEDs.

Because of the mentioned ratio between entropy and pin usage which is poor for LEDs, we added a liquid crystal display (LCD), mounted on top of the PCB. We have chosen a 2×16 characters LCD, which is already the most expensive part out of the partlist (approx. 15 Euro). This display is connected through four data and three control pins. We supply the necessary drivers for this display to the students. Because the display has other features like background illumination and adjustable contrast there are some additional circuits for wiring of the LCD on the PCB.

These devices already enable the user to implement simple interaction between the microcontroller and the user without any additional hardware.

2.2.2 Ports

In addition to the above-mentioned primitive interfaces, we also added some standardized interfaces respectively the possibility to connect with these interfaces to the microcontroller. On the one hand there are interfaces like SPI or I^2C (two wire interface) direct accessible through the microcontrollers' pins. On the other hand, there is a RS-232 interface (female 9-pin D-sub connector DE9) accessible.

The RS232 interface needs in addition to the physical port a level shifter and a crystal oscillator. The level shifter is needed because this interface needs negative voltages between -3 Volt and -15 Volt and positive voltages of up to 15 Volt. All other parts on the PCB just need up to $+5$ Volt, so this is the supply voltage. There are special purpose level shifters exactly for this application (e.g. MAX232).

The microcontroller in general does not need an external oscillator. It is able to generate the needed clock by an internal semiconductor based oscillation circuit. Unfortunately, these internal oscillating circuits are very inexact. This ends up in the fact that for proper communication through RS232 an external clock is needed. Therefore, we placed a crystal oscillator in the PCB layout.

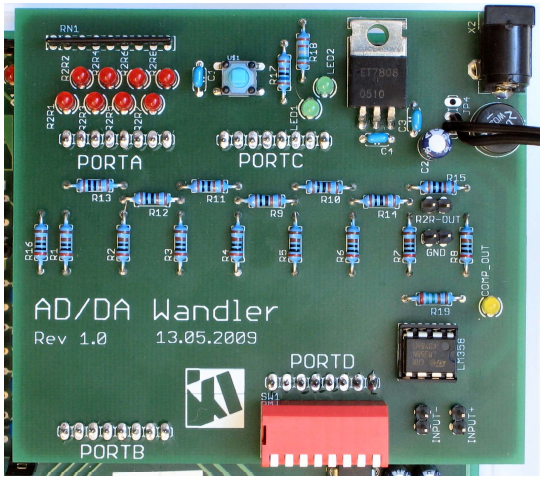


Figure 3: ADC extension module

2.2.3 Supplying

Besides all described parts on the PCB, the microcontroller of course needs a supply voltage, as also some of the other devices do. In order to be more robust towards maloperation we placed a fixed voltage regulator and a rectifier on the PCB. This leads to an acceptable input voltage of 7 Volt to 25 Volt. Thus, the students cannot destroy the microcontroller or other parts of the platform by accident. In addition, for better diagnosis we placed a LED that indicates the correct connection of the supply voltage to the platform.

The microcontroller platform has a chockstone in addition to the regular connector jack. This enables the user to connect e. g. a 9 Volt battery-block via a connector clip. During some of the courses it is helpful to have a kind of master platform that is portable and independed of any external voltage supply.

2.3 Extension Modules

Over the time, a growing need to implement functionality exceeding the above-mentioned ones arose. There were other fields of applications and in addition, we wanted to offer the students tasks, which are related to current technical developments. Last but not least, this adaption, to use modern techniques, like RFID or a chip-card reader, has a very positive impact on the students' motivation.

Most of our extension modules can be directly stacked onto the external pon headers of the microcontroller platform. This enables fast changes in configuration and prevents voltage reversals or erroneous wirings. All the PCBs used for the presented extension modules were designed at out chair suiting to the presented microcontroller platform.

Table 1 shows a listing of all existing extension modules. In the following some of the modules are presented in detail.

2.3.1 ADC

One of our first extension modules is a 8 bit analog-to-digital converter (ADC). Though the microcontroller has an internal ADC we decided, for didactical reasons, to design an external module to offer the students the chance to understand a conversion step-by-step. The students implement a up-/down counter and a successive approximation register

Table 1: Existing extension modules

name of module	application	used in course
ADC	ADC with integrated DAC to do voltage conversions	lab course *electrophysical basics* 2nd semester CS Bachelor
DCF	receiver for german radio clock signal DCF77	lab course *electrophysical basics* 2nd semester CS Bachelor
DPF	digital photo frame consisting of a color LCD and a MMC/SD card reader	lab course *hardware programming* 3rd semester CS Bachelor
LED	primitive optical coupler to realize serial data transmission	lab course *hardware programming* 3rd semester CS Bachelor
memory	8 kB SRAM memory extension in combination with 8 bit latch	lab course *hardware programming* 3rd semester CS Bachelor
motor	step motor driver and power supply	lab course *hardware programming* CS Master
PS/2	PS/2 keyboard adapter	lab course *hardware programming* 3rd semester CS Bachelor
RFID	RFID antenna and RF signal decoder chip	lab course *hardware programming* 3rd semester CS Bachelor

(SAR) on this module.

The module internally needs a digital-to-analog converter (DAC) which is realized by a resistor ladder. In addition there is a DIP-switch which enables the students to use this input UI to manually perform an analog-to-digital conversion and hence better understand the action of converting analog to digital values and vice versa. As additional device there is an operational amplifier on this module in order to do the comparisons between any input voltage and the voltage created by the internal DAC. This step is needed by the AD conversion. Finally, there are eight LEDs in order to visualize the status of the ADC. Our surveillance showed these hands on experience to be very comprehensible to the students.

Because the ADC should be able to convert in the range from 0 Volt to 5 Volt this module needs a higher supply voltage than 5 Volt. Our platform is only able to provide 5 Volt. Therefrom an own (higher supply voltage) is needed for this extension module. Nevertheless, the main platform can be cascaded behind the ADC module, why in sum one voltage source is sufficient. The ADC extension module is shown in Figure 3.

2.3.2 RFID

With the radio-frequency identification (RFID) extension module (shown in Figure 4) the user has the chance to communicate with 125 kHz RFID-Tags. The module comprises the antenna needed for wireless communication as printed coil on the PCB. In addition, the module has an EM 4095 chip, which takes care of modulating and demodulating the radio-frequency signal. Thus, the user can directly communicate in digital signals with the RFID module. The signals read from the according RFID tag is manchester coded.

Because the resulting communication protocol is simple, we do not offer our students any templates or drivers for this module. As resulting application the students implement a access control with based on RFID tags.

2.3.3 Digital Photo Frame

The digital photo frame is based on a LCD originally used in a mobile phone and a MMC/SD card reader. In addition,

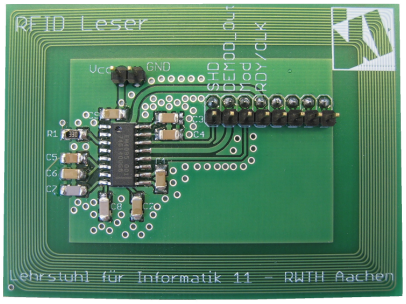


Figure 4: RFID extension module

there are some circuits to supply these devices. Like the ADC module, some of the devices (e.g. the background illumination of the LCD) need a supply voltage higher than 5 Volt. Thus this module needs to have a supply voltage of 12 Volt which can be cascaded with the main platform. The display we use is a Siemens S65 compatible (LS020), 132 × 176 pixel and 16 bit color-depth LCD. Both devices (the LCD and the MMC/SD card reader) communicate through SPI with the microcontroller.

Figure 5 shows the digital photo frame extension module. On top the LCD is fixed with a pane of acrylic glass. To the right of this LCD the power connector is placed and a cable for cascading this module with the basic platform . On the left side of the module parts of the SD/MMC card can be seen. The reader is mounten on the bottom side of the modules PCB.

As file system we use file allocation table (FAT)[11] with already existing drivers from Elm Chan [7]. This leads to the point that the students can directly access the content of the chip card and do not need to take care of any low-level file system problems. There are also some libraries available for the LCD [9] which already allow several image manipulating operations. The pictures can be stored as either bitmap or any compressed pixel-based file format. If the picture-data is saved in uncompressed bitmap format it can be directly sent to the display. In case of any compressed file format, the

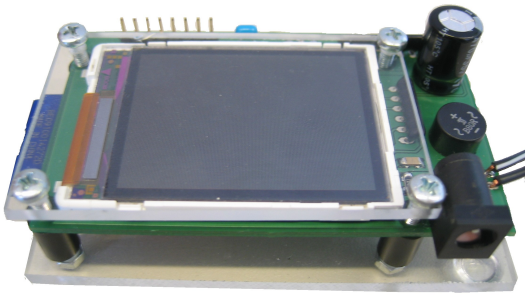


Figure 5: digital photo frame extension module

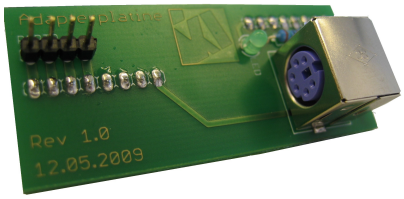


Figure 6: keyboard extension module

student will have to implement image-processing functions in order to get data the LCD will be able to work with.

During our work with the memory cards, we made the experience that there are huge differences in quality between the different available brands. One will have to test them before using them in a lab course where a robust and reliable behavior is needed. On the one hand, some of the memory cards were very slow in transfer rates which ended up in picture changes which took several minutes. In addition to this, some of the cards got flawed after short usage. Therefore, we had to replace them. Unfortunately, there are no general criteria to determine which card is well suiting for this purpose.

Because the MMC/SD card has to be read block wise this operation needs 512 bytes of memory (which equates to the size of a block). Therefore, it is challenging to suit all the data in the memory of an ATmega32. This is why we generally use an ATmega644 for this application.

2.3.4 PS/2 Keyboard Extension

Another extension is the PS/2 keyboard module (shown in Figure 6). This enables the microcontroller to be connected to a PS/2 keyboard. PS/2 in general is a synchronous serial protocol, which the ATmega's USART can handle. The electrical characteristics are designed for 5 Volt operation, that is why no additional voltage converter or the like is needed.

A keyboard encodes the communication in so-called scan codes. Each possible keystroke (or release) is encoded in a string consisting of a single or several bytes. Since we do not want, the students to copy over all these data into an array

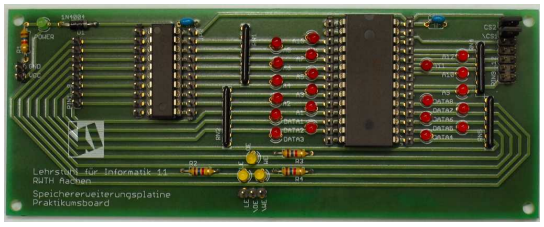


Figure 7: memory extension module

instead of spend their time in working with the keyboard we offer them prepared arrays with the according scan codes. In general, for this application one has to be careful with cheap keyboards, since they usually do not support all the scan code sets or even sometimes do not work completely according to the specification.

2.3.5 Memory Extension Module

The last presented extension is the external memory module. This is a 8 kbyte SRAM chip (HY6264A) in connection to a 8 bit data-latch (74HC573). The latch was needed because not sufficient connection pins to the microcontroller would be available otherwise. The SRAM has a data and a separate address bus. Each line on this module is equipped with a LED to visualize the processes of reading and writing to the SRAM. Most of our students stated that this is a descriptive exercise in order to understand the dialogue of embedded hardware. The memory extension module is shown in Figure 7.

3. EDUCATIONAL USAGE

The above-mentioned platform with all its extension modules is used widely in education and also in some research projects at our chair. There are several lab courses, which use this platform [15, 16]. These courses vary from implementing a motor control to the realization of a whole embedded operating system including scheduler, memory management and boot loader. Due to the fact that this platform is robust and inexpensive we were able not only to use it in graduate courses but also in undergraduate courses where we have to handle up to 350 students. In these cases, it is very important to have reliable hardware. It is impossible to run a course with small groups and resulting many appointments a week if the used hardware regularly fails.

In addition to the lab courses, we use the platform in different lectures for specific presentations. The experience shows that this more vivid way of presenting embedded hardware behavior (in comparison to presenting simulations) result in a better educational result with our students.

Beside the usage in our courses, the platform was also used in several students' thesis. Either for the evaluation of e.g. a single sensors' behavior but also to develop the whole project.

4. COMPARISON TO OTHER PLATFORMS

When we started designing this platform, in 2004 just a very small range of starter-kits or evaluation platforms for the ATmega was available and none of them was meeting our requirements. Thus, we came up with the idea to design a new PCB for our needs. In this section, we nevertheless

want to show the similarities and differences to other existing designs.

First to mention would be the Atmel STK 500 starter kit [3]. This development platform comes directly from the manufacturer of the microcontroller (Atmel) and is designed to program a variety of microcontrollers not only limited to the ATmega. The user has the choice between many different programming interfaces e.g. JTAG, ISP or high-voltage-programming. The variety of possible microcontrollers as converse argument causes a high complexity in configuration of this platform. There are 8 LEDs but no LCD on this platform. This causes a smaller possibility to debug since all output must be either encoded to the LEDs or sent out through RS232 interface. This platform costs about 80 Euro.

There are other platforms similar to the STK500 mostly sold by electronic distributors (e.g. [13]). These platforms in general have the problem that most of the ports have a fixed wiring to several devices and the user cannot change this wiring.

Not only the industry built evaluation platforms so far, also academia did. Some of these platforms like [17] gave us inspirations for our platform. The microcontroller hardware in [17] already showed the principle of modular extensions and pointed out that it is important to give students the chance to link the different devices by themselves in order to better understand these links.

5. CONCLUSIONS

In this paper, we presented a microcontroller platform for educational use developed over the years since 2004 at our chair. We combined advantages of different existing platforms to a robust and modular evaluation platform for education.

The presented platform allows us to run undergraduate lab courses with up to 350 students since 2007 but nevertheless gives us the freedom to upgrade these courses with ongoing technical developments. Due to the conservative designed electric supply circuits, it is resistant against operating errors. The positive aspects for a usage for educational purposes are completed by an inexpensive acquisition of these microcontroller platforms and extensions modules.

Besides the pure attributes, we give many backgrounds for the design of our platform in this contribution. This should, together with the freely available PCB designs and part lists on our website [8], enable others to create a platform best suiting their needs or just reusing ours without spending unnecessary time in debugging hardware.

6. REFERENCES

[1] Atmel Corporation. AVR JTAGICE mkII. http://www.atmel.com/dyn/products/product_card.asp?part_id=3353.

[2] Atmel Corporation. AVRISP mkII In-System Programmer. http://www.atmel.com/dyn/products/product_card.asp?part_id=3808.

[3] Atmel Corporation. STK 500. http://www.atmel.com/dyn/products/product_card.asp?part_id=2735.

[4] Atmel Corporation. Atmel AVR ATmega32, July 2009. http://www.atmel.com/dyn/products/product_card.asp?part_id=2014.

[5] Atmel Corporation. Atmel AVR ATmega644, Jan 2010. http://www.atmel.com/dyn/products/product_card.asp?part_id=3694.

[6] Atmel Corporation. AVR Studio 4.18 SP2, February 2010. http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725.

[7] E. Chan. FatFS Module, May 2010. http://elm-chan.org/fsw/ff/00index_e.html.

[8] Embedded Softeare Laboratory, RWTH Aachen University. An atmega evaluation board – design and components list, 2008. http://evaboard.embedded.rwth-aachen.de/.

[9] C. Kranz. Using the siemens s65-display, Oct 2005. http://www.superkranz.de/christian/S65_Display/DisplayIndex.html.

[10] E. A. Lee. Introducing embedded systems: a cyber-physical approach: extended abstract. In *WESE '09*, pages 1–2, Grenoble, France, 2009. ACM.

[11] Microsoft Corporation. Fat32 file system specification. Hardware white paper, Redmond, WA, USA, December 6 2000.

[12] OLIMEX Ltd. AVR-JTAG-L JTAG dongle for programming and emulation. http://www.olimex.com/dev/avr-jtag.html.

[13] Pollin Electronic GmbH. Atmel evaluations-board v2.0.1, Nov 2007. http://www.pollin.de/shop/dt/NTI5OTgxOTk-/Bausaetze/Diverse/ATMEL_Evaluations_Board_V2_0_1_Fertigmodul.html.

[14] J. J. Richardson. C programming and the atmega16 microcontroller, Aug 2005. http://www2.tech.purdue.edu/ecet/courses/referencematerial/atmel/.

[15] F. Salewski, D. Wilking, and S. Kowalewski. Diverse hardware platforms in embedded systems lab courses: a way to teach the differences. *SIGBED Rev.*, 2(4):70–74, 2005. WESE05.

[16] A. Stollenwerk, C. Jongdee, and S. Kowalewski. An undergraduate embedded software laboratory for the masses. In *WESE '09*, pages 34–41, Grenoble, France, 2009. ACM.

[17] B. Weiss, G. Gridling, and M. Proske. A case study in efficient microcontroller education. *WESE 05*, 2(4):40–47, 2005.

[18] J. Wunsch. Winavr, Jan 2010. http://winavr.sourceforge.net/.

Discussion: What have we learned today?

Abstract:

Workshops like WESE are designed to stimulate the exchange of ideas within the discipline. For WESE 2010, there have been presentations concentrating on textbooks, individual courses, course sequences, hardware platforms, industrial perspectives, optimization techniques, and various software concepts. During this discussion, WESE 2010 participants are encouraged to reflect on the ideas and topics presented throughout the workshop and to consider how these ideas can be disseminated and advanced within the community.