

# **Embedded System Design: A Unified Hardware/Software Approach**

**Frank Vahid and Tony Givargis**

Department of Computer Science and Engineering  
University of California  
Riverside, CA 92521  
vahid@cs.ucr.edu  
<http://www.cs.ucr.edu/~vahid>

Draft version, Fall 1999

Copyright © 1999, Frank Vahid and Tony Givargis

## Preface

This book introduces embedded system design using a modern approach. Modern design requires a designer to have a unified view of software and hardware, seeing them not as completely different domains, but rather as two implementation options along a continuum of options varying in their design metrics (cost, performance, power, flexibility, etc.).

Three important trends have made such a unified view possible. First, integrated circuit (IC) capacities have increased to the point that both software processors and custom hardware processors now commonly coexist on a single IC. Second, quality-compiler availability and average program sizes have increased to the point that C compilers (and even C++ or in some cases Java) have become commonplace in embedded systems. Third, synthesis technology has advanced to the point that synthesis tools have become commonplace in the design of digital hardware. Such tools achieve nearly the same for hardware design as compilers achieve in software design: they allow the designer to describe desired processing in a high-level programming language, and they then automatically generate an efficient (in this case custom-hardware) processor implementation. The first trend makes the past separation of software and hardware design nearly impossible. Fortunately, the second and third trends enable their unified design, by turning embedded system design, at its highest level, into the problem of selecting (for software), designing (for hardware), and integrating processors.

ESD focuses on design principles, breaking from the traditional book that focuses on the details a particular microprocessor and its assembly-language programming. While stressing a processor-independent high-level language approach to programming of embedded systems, it still covers enough assembly language programming to enable programming of device drivers. Such processor-independence is possible because of compiler availability, as well as the fact that integrated development environments (IDE's) now commonly support a variety of processor targets, making such independence even more attractive to instructors as well as designers. However, these developments don't entirely eliminate the need for some processor-specific knowledge. Thus, a course with a hands-on lab may supplement this book with a processor-specific databook and/or a compiler manual (both are typically very low cost or even free), or one of many commonly available "extended databook" processor-specific textbooks.

ESD describes not only the programming of microprocessors, but also the design of custom-hardware processors (i.e., digital design). Coverage of this topic is made possible by the above-mentioned elimination of a detailed processor architecture study. While other books often have a review of digital design techniques, ESD uses the new top-down approach to custom-hardware design, describing simple steps for converting high-level program code into digital hardware. These steps build on the trend of digital design books of introducing synthesis into an undergraduate curriculum (e.g., books by Roth, Gajski, and Katz). This book assists designers to become users of synthesis. Using a draft of ESD, we have at UCR successfully taught both programming of embedded microprocessors, design of custom-hardware processors, and integration of the two, in a one-quarter course having a lab, though a semester or even two quarters would be

preferred. However, instructors who do not wish to focus on synthesis will find that the top-down approach covered still provides the important unified view of hardware and software.

ESD includes coverage of some additional important topics. First, while the need for knowledge specific to a microprocessor's internals is decreasing, the need for knowledge of interfacing processors is increasing. Therefore, ESD not only includes a chapter on interfacing, but also includes another chapter describing interfacing protocols common in embedded systems, like CAN, I2C, ISA, PCI, and Firewire. Second, while high-level programming languages greatly improve our ability to describe complex behavior, several widely accepted computation models can improve that ability even further. Thus, ESD includes chapters on advanced computation models, including state machines and their extensions (including Statecharts), and concurrent programming models. Third, an extremely common subset of embedded systems is control systems. ESD includes a chapter that introduces control systems in a manner that enables the reader to recognize open and closed-loop control systems, to use simple PID and fuzzy controllers, and to be aware that a rich theory exists that can be drawn upon for design of such systems. Finally, ESD includes a chapter on design methodology, including discussion of hardware/software codesign, a user's introduction to synthesis (from behavioral down to logic levels), and the major trend towards Intellectual Property (IP) based design.

*Additional materials:* A web page will be established to be used in conjunction with the book. A set of slides will be available for lecture presentations. Also available for use with the book will be a simulatable and synthesizable VHDL "reference design," consisting of a simple version of a MIPS processor, memory, BIOS, DMA controller, UART, parallel port, and an input device (currently a CCD preprocessor), and optionally a cache, two-level bus architecture, a bus bridge, and an 8051 microcontroller. We have already developed a version of this reference design at UCR. This design can be used in labs that have the ability to simulate and/or synthesize VHDL descriptions. There are numerous possible uses depending on the course focus, ranging from simulation to see first-hand how various components work in a system (e.g., DMA, interrupt processing, arbitration, etc.), to synthesis of working FPGA system prototypes.

Instructors will likely want to have a prototyping environment consisting of a microprocessor development board and/or in-circuit emulator, and perhaps an FPGA development board. These environments vary tremendously among universities. However, we will make the details of our environments and lab projects available on the web page. Again, these have already been developed.