Laboratories Teaching Concepts in Microcontrollers and Hardware-Software Co-Design

Daryl Beetner, Hardy Pottinger, and Kyle Mitchell
Department of Electrical and Computer Engineering
University of Missouri-Rolla
Rolla, MO 65409

Abstract *– Hardware-software co-design is becoming increasingly important to the embedded systems industry. It will soon be fundamental to digital systems design. As such, students in Electrical and Computer Engineering and in Computer Science should be introduced to hardware-software co-design early in their undergraduate education.*

We are designing laboratory modules which introduce concepts of hardware-software co-design in an undergraduate's first course on microcontrollers and digital systems design. Students use design automation tools to develop FPGA-based hardware for use with an 8051-microcontroller and use common software development tools to develop microcontroller-software in C or assembly language. Co-simulation of hardware and software is enabled using a simulation model of the 8051 that we developed.

Preliminary results are encouraging. Students who take the lab perform better in the associated lecture class than those who do not take the lab and appear to develop a greater appreciation for digital systems design. We believe such laboratories will be a common component of computer engineering classes in the future.

Introduction

The rapid reduction in size of integrated circuits (ICs) has allowed more and more functions to be implemented in a very small area, to the point that entire systems may now be implemented on a single chip. These systems are referred to by such acronyms as Systems on Silicon (SOS), Systems on a Chip (SOC), and Systems in a Package (SIAP). As the number of chips shrinks and the complexity grows, the product designer's ability to resolve problems becomes increasingly more difficult and expensive. Signals once available for analysis are now hidden within the chip and development of multiple hardware prototypes during the debugging process can be ineffective as well as cost-prohibitive. What's more, designers face incredible pressures to bring their product to market very quickly. A product's success or failure often depends on hitting a vanishingly short market window. Consequently, products must be developed fast and virtually error-free.

Adding to the design problem is the increasing ability to implement functions in either hardware or software or both. Where once the boundary between hardware and software was quite distinct, now the designer must spend considerable effort deciding how best to partition his problem between the two areas. His decision affects not only the performance of his product, but cost, power, risk, and more. Design of hardware and software must be closely coupled for a successful result.

To bring products to market faster and to identify problems earlier in the design cycle, industry has turned to design processes which rely heavily on hardware-software co-design and co-verification and on reuse of complex building blocks (reusable cores and/or intellectual property (IP)) [1,2,3,4]. Co-design refers to a process where design of hardware and software is very closely linked throughout the design process. In the traditional approach, tasks for hardware or software are identified immediately and are then developed largely independent of one another. They are only brought together at the end of the design process when the product may be only weeks from a delivery date and when errors can be especially difficult to fix. The progressively intimate relationship between hardware and software has made this development process risky at best.

The functionality of a product can be improved and the risks in development reduced using hardware-software co-design. Co-design typically stresses a top-down approach. By immediately capturing high-level functionality of a design, several possible architectures can be tested quickly and critical flaws can be eliminated early in the design process. As the design progresses, greater levels of detail are incorporated and virtual prototypes of hardware and software working together are tested at regular intervals. Key to successful implementation of the hardware-software co-design process is co-simulation on an accurate hardware-software model. The goal of this process is to make final implementation in hardware an anti-climactic event, free from unwanted surprises.

The recent interest of industry in hardware-software co-design is highlighted by the explosive growth of articles published in this area. A search of Compendex shows no articles on this subject prior to 1993, but shows more than 300 since. Integrated Systems Design Magazine, a publication on industry concerns and applications, has printed more than 43 articles on co-design in the past 2 years.

While use of hardware-software co-design is growing in industry, few institutions offer courses that introduce it at the undergraduate level. Those that do offer it at the senior or graduate-level, often as an elective [4,5,6]. As hardware-

software co-design will soon be fundamental to digital systems design, we believe it should be introduced to students early in the electrical engineering, computer engineering, and computer science curriculum. Early introduction not only ensures that all students are exposed to these important concepts, but also allows simplified integration of these concepts in upper level courses like VHDL modeling, VLSI design, microprocessor and microcomputer design, and others.

The following paper documents laboratories we are developing to introduce hardware-software co-design in a junior-level course in microcontrollers and digital systems design.

Course Design

Laboratories are being developed around a new course at University of Missouri-Rolla which teaches digital systems design from the perspective of microcontrollers and embedded systems. This junior-level class is a "gateway" to upper-level courses in computer engineering, as it is a requirement for many senior and graduate level classes. It is required for students in computer engineering, but also attracts a healthy mix of students from electrical engineering and computer science, as well as some students from other disciplines. Students come into the course with a basic knowledge of digital logic, use of industrial strength design automation tools, and use of FPGAs for rapid prototyping of digital designs. Most, but not all, have had a course in C++ programming. Few have any real knowledge of assembly programming or of computer architecture. Laboratories are being developed as an adjunct to the lecture course, but are only required for computer engineering students (though they are not required to take the lab and lecture simultaneously). We chose this course because it is taken early in the curriculum, it attracts students from a broad range of disciplines, and because an embedded systems course offers the opportunity to experiment with unique hardware interfaces and to showcase exciting, down-to-earth, applications that actively engage the student and increase their level of interest.

The laboratories are built around the 8051 microcontroller. The 8051 was chosen because of its wide-spread popularity, its simple architecture, and the likelihood that it will remain an important processor for years to come. There are more than 20 independent vendors of 8051-based components and more than 12 current textbooks on this controller. The fundamentals learned on this controller should transfer easily to other processors or to more sophisticated architectures.

Laboratories are being designed to teach fundamentals of microcontrollers, digital systems, and hardware-software co-design through hands-on development of simple embedded systems [7]. Students follow each project from conception to implementation. They design hardware and software from a given set of specifications, they co-simulate their hardware and software design to test for proper operation, and they implement their design in hardware to verify their final result. Key to this process is extensive system-level simulation before implementation. We believe hardware implementation is important not only because of the satisfaction a student gets from building something "real", but because there are many problems which arise in the lab which do not appear during simulation, such as power considerations, ground bounce, and so on.

While laboratories are still under development, concepts we hope to convey include:

Microcontrollers:

- Programming microcontrollers in assembly language
- Programming microcontrollers in a high-level language (C)
- Analog and digital I/O
- Signals and timing necessary to read and write external devices
- Memory usage and addressing modes
- Interrupts, timers, and counters
- Inter-device communication
- Real-time operating systems

Hardware-Software Co-Design:

- Hardware-software co-development
- Hardware-software co-simulation
- Partitioning tasks between hardware and software
- Re-use of complex blocks (IP)
- Implementation of digital logic in an FPGA
- Use of laboratory tools and equipment to test computer hardware

Of course, being a lab, students also learn how to deal with problems which arise when dealing with "real" hardware and learn to work within teams to reach a common objective.

Tools

While the focus of advanced design processes like hardware-software co-design is applications like systems on a chip, building systems on a chip is not feasible for most laboratory settings. A good alternative is a board incorporating a microprocessor-FPGA combination. Software is designed for the microprocessor and digital hardware is designed and downloaded to the FPGA.

For our labs, we have chosen the XS40 board from Xess corporation [8], which combines an 8051-family microcontroller with a 9000-gate Xilinx FPGA. The board also sports a 7-segment LED, a VGA monitor interface, and a parallel port connector for communication with a host computer. It comes with a generous supply of wire-wrap pin-probe points, which allow students to readily access bus signals with logic-analyzer probes or to attach additional hardware. The pins are configured such that the XS40 may be plugged directly into a breadboard if desired.

Students develop their software in C or assembly language using Keil Software's software development tools

[9], though C is used in most labs. Before implementing their software with hardware, students thoroughly simulate their software using Keil's software simulation tool. This tool is good for discovering high-level problems with the code, but not for simulating the hardware-software interface. After simulation, their code is compiled to an Intel-format hex file.

Students develop custom digital hardware to be implemented in the FPGA using Mentor Graphics' Design Architect. Mentor Graphics is a good choice for hardware design and simulation in an academic lab because their powerful industry-grade software is available quite inexpensively to institutions of higher-education [10]. Hardware is simulated (independent of software) using Mentor Graphics' QuickSim.

Once hardware and software have individually been simulated successfully, students may co-simulate their hardware-software design. Key to this step is a VHDL behavioral model of the 8051 we developed for this purpose. We felt a bus-functional model was too abstract and inconvenient for the typical junior-level student and that a synthesizable model was too complex and would simulate too slowly. Features supported by our clock-cycle-accurate model at the time of this publication include:

- Entire 255-instruction 8051 command-set
- 4-port digital I/O
- 4Kb of internal program memory
- 256 bytes of internal (byte) memory
- Up to 64Kb of external code or data memory, with accompanying (timing accurate) control signals
- Critical SFRs (e.g. ACC, PSW, DPTR, etc)
- External interrupts
- Bit-addressable memory

An essential feature of our model is a VHDL "wrapper" for the internal ROM which loads and interprets an Intel hex-format file upon simulation.

Using the model of the 8051, students co-simulate their design using Mentor Graphics' QuickHDL Pro. QuickHDL Pro allows simulation of a mixture of schematic and HDL models. Through this simulation, students verify proper operation of their hardware with their software. Complete access to signals allows students to quickly identify the source of any problems. These problems are rectified by going back to the software-development or schematic-capture tools. Once the student is comfortable with their design, they may compile their hardware to a bit-file, ready for download and verification on an FPGA.

The final step, hardware verification, is performed using the XS40 board. The hardware configuration file is downloaded to the FPGA and the compiled software is downloaded to the on-board SRAM. After so much simulation one would expect few surprises during implementation of their design. The design should almost certainly work. However, surprises do still occur. More on that point later.

Experiments

Experiments were developed to meet the goals listed in the "Course Design" section of this paper. While lab development is still underway, some of our current labs include:

- **Programming the 8051 using Keil uVision-51**. An introductory lab. Students write a simple assembly program using Keil uVision-51 to read and write ports of the 8051. They debug their code using Keil dScope-51 and create an assembled Intel hex-format file of their program. Emphasis is placed on basics of assembly language programming and on use of the Keil software development toolchain.
- **Hardware-software co-simulation with the 8051-simulation model, QuickSim, and the XS40 prototyping board.** Students use QuickHDL Pro from Mentor Graphics to simulate hardware signals generated by the program they developed in the last lab. After simulation, their software is implemented on the XS40 board. Students verify that signals from the 8051 are the same when simulated as when measured. An emphasis is placed on use of design automation tools, on co-simulation, and on use of the XS40 board
- **External memory addressing -- logic implementation with an FPGA and programming in C.** Students design hardware and software to write numeric messages to the 7-segment display on the XS40 board. A subroutine is written in C which takes a numeric digit and writes the appropriate codes to external memory to show that number on the 7-segment display. Students design an address decoder and address latch to place the 7-segment display at a particular location in the 8051's external memory space. Results are verified, in part, by downloading their design to the XS40 and measuring outputs of the 8051 while slowly stepping the system clock.
- **Hardware-software partitioning.** Students write a program in C which writes numbers to the 7-segment display. Unlike the last lab, hardware is developed such that a number written to an external address is properly encoded and displayed, replacing the code-conversion subroutine of the last exercise. For example, if the number "9" is written to location 0x7FFF (the display address) then the 7-segment display will display a "9". Students continue to simulate with Keil and QuickHDL Pro before implementing and testing their design on the XS40 board.
- **VGA character display.** Students design hardware and software to display characters on a VGA monitor using the XS40 board. Students are given previously designed hardware "IP" for a VGA timing generator, a frame buffer, a character ROM, a character

synchronizer, and a row-column generator. Students use these modules to create a VGA character display unit implemented in the FPGA. Students write a program in C which interacts with their hardware to scroll a message across the VGA display. Interaction between hardware and software is verified in Mentor-Graphics before implementation.

Labs under development include:

- Serial Communication and Message Encoding with a Pseudorandom Noise Sequence;
- Digital Thermometer Implemented with a Dallas-Semiconductor 1-wire Device;
- Communication using Quadrature-Phase Pulse-Width-Modulation;
- Real-Time Control of an Embedded System.

Each of these labs requires development and testing of both hardware and software. In many labs, students are given pre-designed hardware or software building blocks which they incorporate in their design. Hardware-blocks are often designed by students in an upper level VHDL course. These students tend to work harder and do a better job with their VHDL design because they know their design will be implemented by others. These students are given an exposure to design for re-use and an added appreciation for their work.

Toward the end of the semester, students taking our lab complete a self-directed team-based design project. They choose a device they would like to build, they specify its characteristics, they plan their design, they build it, simulate it, and finally implement it in hardware. Their progress is tracked through regular contact with the lab instructor and through an on-going project web page. When complete, they give an oral presentation to the class and demonstrate their final product. Some projects built by the students include:

- Remote control and display for a PC-based MP3 player
- "Simon" game
- Four function calculator
- Digital LCD alarm clock
- Automated pet feeder
- Virtual pet
- Electronic piano
- Autonomous robot
- "Pong" game implemented with an 8051 and a VGA display

Many projects use the XS40 board, but many students also design their own circuit board around a discrete 8051-family microcontroller. Students especially appreciate a project they can take home with them at the end of the semester and show off to their friends.

Preliminary Results

Complete assessment of these laboratories is still underway, but preliminary results are promising. Since offering this class, there has been a significant increase in the number of students using microcontrollers in their senior design project. This increase indicates the student's enjoy the material and also that they are walking away from the course with some applicable knowledge. An exit survey of students from the lab supports this contention. The overall consensus is that they enjoyed the lab, particularly the opportunity to make something "real". The lab also appears to improve performance in the lecture course. While data are preliminary, a study of test scores over 1 semester show that students who chose to take the lab scored an average of 13 points higher (approximately 1.3 letter grades) on tests in the lecture than students who chose not to take the lab. One semester's worth of scores is not enough to make a definitive statement of performance, but it is encouraging.

An advantage of implementing designs in hardware is that students often encounter problems that would not appear in simulation. One problem encountered by our students occurred when they attempted to locate an external device at address 0x7FFF. Rapid switching of the address lines from all 1's to mostly 0's created a sufficient ground bounce to cause a fluctuation in the address-latch enable line and a very confusing error to the students. Without simulation, they would have spent significant effort fruitlessly searching their code for a cause. Because they simulated first, they were confident that their code and the basic structure of their hardware was correct and could immediately look elsewhere. We should note that Xess has corrected the ground bounce problem in the second version of the XS40 when they added separate power and ground planes.

A discouraging result is that students would often abandon the simulation-heavy design process stressed in the lab when they implement their own project. They would immediately jump from code and hardware creation to hardware implementation, the classic "burn-and-try" approach (If it compiles, it must work, right?). In the future, we would like to adjust our labs to better emphasize the time and effort which can be saved through proper simulation.

Our laboratories have also resulted in beneficial side-effects unrelated to student learning. An example is the fate of the Simon-game project mentioned earlier. This project was taken up by students in a later semester who polished and formalized the design, producing a compact printed circuit board that could be manufactured en mass. It is now used in a pre-college outreach program, where students solder the necessary components on the printed circuit board and walk home with a small game they helped build. The hope is to get these students excited about electronics and computers so they might consider a career in electrical or computer engineering.

Conclusions

Hardware-software co-design is quickly becoming a critical skill to industries which develop computer-based products. Introduction to these skills at an early level allows higher level courses to seamlessly incorporate these concepts in their curriculum, without significant overlap, and also ensures exposure to a maximum number of students. Our 8051 processor model was key to successful implementation of our labs. Students enjoyed the labs and appear to do better in the lecture as a result of taking them. The labs improve their understanding of hardware-software co-design and of embedded systems, which has had a beneficial ripple-effect on upper level courses. Future adjustments are needed, however, to truly convey the importance of simulation over the simpler "burn and try" approach. While our data is preliminary, results indicate that these labs are successfully achieving their goals and can be an important part of the electrical engineering, computer engineering, or computer science curriculum.

Acknowledgements

The authors wish to thank Dr. Frank Vahid of University of California – Riverside for supplying a rough model of the 8051 which we built upon to create the model now used in our labs. Mike Mayer, who has now graduated, deserves our thanks for his effort updating this model to its current level. We would also like to thank Dr. Jim Frenzel of University of Idaho for his continuing feedback and suggestions for lab improvement. This work was supported in part by the National Science Foundation's Course, Curriculum and Laboratory Improvement program under grant no. DUE-9952540.

References

1. Sangiovanni-Vincentelli, A., Rowson, J., "What You Need to Know About Hardware/Software Co-Design," *Computer Design*, August 1998, pp. 63-9.
2. Ernst, R., "Codesign of Embedded Systems: Status and Trends," *IEEE Design & Test of Computers*, Apr-Jun 1998, Vol. 15, pp. 45-54.
3. De Micheli, G., Gupta, R.K., "Hardware/Software Co-Design," *Proceedings of the IEEE*, 1997, Vol. 85, No. 3, pp. 349-65.
4. Hamblen, J.O., Henry, L.O., Yalamanchili, S., Dao, B., "An Undergraduate Computer Engineering Rapid Systems Prototyping Design Laboratory," *IEEE Transactions on Education*, 1999, Vol. 42, No. 1, pp. 8-14.
5. Carnegie-Mellon University, "Advanced Digital Design Project," http://www.ece.cmu.edu/~ee545.
6. University of California-Berkeley, "EE291a – System Design," http://amber.berkeley.edu:5037/cgi-bin/soc/search_gencat.pl.
7. Pottinger, H.J., Beetner, D.G., "Hardware-Software Co-Verification in an Undergraduate Laboratory," *1999 IEEE Computer Society International Conference on Microelectronic Systems Education*, 1999, pp. 41-2.
8. Xess Corporation, http://www.xess.com.
9. Keil Software, http://www.keil.com.
10. Mentor Graphics higher education program, http://www.mentor.com/partners/hep/index.html.