

01 Jan 1999

# Hardware-Software Co-Verification in an Undergraduate Laboratory

Hardy J. Pottinger
*Missouri University of Science and Technology*, hjp@mst.edu

Daryl G. Beetner
*Missouri University of Science and Technology*, daryl@mst.edu



## Recommended Citation

H. J. Pottinger and D. G. Beetner, "Hardware-Software Co-Verification in an Undergraduate Laboratory," *Proceedings of the IEEE International Conference on Microelectronic Systems Education (1999, Arlington, VA)*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1999.
The definitive version is available at https://doi.org/10.1109/MSE.1999.787028

# Hardware-software Co-verification in an Undergraduate Laboratory

Hardy J. Pottinger
Daryl Beetner
Department of Electrical and Computer Engineering
University of Missouri - Rolla

## Abstract

*Skills in hardware-software co-design are quickly becoming critical to product development in high-technology computer industries. Systems-on-silicon typically include a considerable amount of software as well as custom hardware and are increasingly difficult to develop using traditional techniques. To satisfy a growing demand in industry, students in electrical engineering, computer engineering, and computer science should be introduced to concepts of hardware-software co-design at the undergraduate level. This paper examines a new laboratory at the University of Missouri - Rolla in which students in Electrical and Computer Engineering are exposed to modern system design concepts through the use of hardware-software co-simulation. Key tools used in the course including a hardware prototype consisting of an 8051 microcontroller and a field programmable gate array, and a VHDL model of the prototype are discussed.*

## 1. Introduction

The availability of powerful but inexpensive processors and synthesizable cores are changing the face of embedded computer system design. Embedded computer systems are those computers found in industrial and commercial application with non-traditional interfaces. Embedded system designers need a wide range of skills to compete in this rapidly changing field. These skills include system modeling, requirements analysis, system specification, hardware-software partitioning, and system integration. Our existing course sequence [1] includes courses in hardware design as well as software development but does little to integrate either, or to include these new skills so critical in a highly competitive industry. This paper discusses steps we are taking to correct the situation by integrating elements of hardware and software design and system modeling in a single undergraduate microcontroller laboratory.

System level modeling of a combined hardware-software system, use of C for microcontroller programming, and rapid prototyping of hardware with FPGA's are key ingredients of a new required undergraduate course at the University of Missouri - Rolla. This course follows a typical introductory logic design course in which students are exposed to industrial strength design automation tools used here as well as hardware design using FPGA's. Although most have some familiarity with C, none have used it outside of the usual introductory programming course. None of the students have had a previous course in assembly language or microprocessor hardware.

Subsequent sections will briefly discuss the hardware prototype we are using, the simulation model which enables students to 'get it right' the first time, and a sampling of the laboratory exercises the students perform using these tools.

## 2. Hardware Prototype

The companion lecture course is based on the Intel 8051 microcontroller family. C is used as the programming language and the compiler from Keil Software is used for all programming exercises. Keil's simulator, which is included with the compiler, is a good high-level language debugger but does a poor job of simulation at the hardware-software interface. In order to give the students a realistic feel for embedded systems we wanted to avoid the usual microcontroller plus dumb terminal combination. We wanted the students to be able to develop their own hardware without a lot of fabrication overhead. We also wanted the students to be able to simulate everything, not just the software, in order to get it right the first time and avoid the usual 'burn and try' approach. Even though this is often advocated as an 'advantage' by FPGA vendors we didn't believe it an appropriate philosophy to promote. Finally, we wanted the students to have the opportunity of actually seeing their design as a hardware artifact rather than simply being satisfied with a simulation model. Aside from the personal satisfaction gained from the exercise, we recognized that some phenomena

such as power requirements, EMI, ground bounce and signal integrity are perhaps best illustrated in the lab at a more concrete level.

These constraints led us to consider a combination of a Xilinx FPGA and an 8051 with either EPROM or RAM for program storage and some minimum amount of additional I/O. We are using Xess's XS40 board (www.xess.com). which consists of a ROMless 8051 variant, 32k SRAM, and a Xilinx 4005 FPGA. It interfaces to a PC's parallel port for downloading the FPGA's bit file and for loading the 8051 code into the SRAM. The parallel port can be used as a simple stimulus generator and the XS40 makes all of the 8051 pins as well as many of the FPGA's pins available on two rows of header pins. This makes it a simple matter to either plug the XS40 into a breadboard or to connect it to a logic analyzer.

At this point in time the students can write their software and debug it using the Keil compiler and debugger. They can expand the 8051 and develop special purpose peripherals using powerful design automation software tools and the onboard FPGA. Unfortunately if that were the end of the story they would be faced with a long trial and error process of painful hardware and software integration. In fact the documentation supplied with the XS40 suggests such an approach. Fortunately hardware-software co-verification can come to the rescue. The only missing ingredient was a complete, instruction set accurate simulation model of the hardware-software prototype.

## 3. Simulation Model

Real processor simulation models are not easy to come by. So-called bus functional models are in wide use and are fairly easy to write but require special scripts and do not execute actual user code. We wanted a model of the 8051 that was instruction set and clock cycle accurate that could execute object code files produced by the Keil software. We started with an incomplete model graciously supplied by Dr. Frank Vahid and his students and, after about a semester's worth of intense work by two of our students, Mr. Mike Mayer and Mr. Kyle Mitchel, we have a, still incomplete, but usuable model of the 8051. This model was incorporated into Mentor Graphic's QuickHDL Pro simulator that allows us to mix VHDL and schematic models for simulation. A model of the XS40's 32k SRAM which can read Intel Hex format object files and the student’s own FPGA design completes the typical simulation model.

## 4. Laboratory Exercises

Students are given a schematic with symbols for the 8051, 32k SRAM, and a black box for their portion of the circuit. Underneath the 8051 and SRAM are our VHDL models. A typical exercise will involve writing a small C program for the 8051 and debugging it with the software debugger. Along with the software exercise, the students are given a small hardware design to complete. One of the first exercises is to simply add an address latch and put the 8051 into expanded mode to utilize the offchip SRAM as code space for the microcontroller. Skills learned in the previous logic design course are reinforced and the students use traditional schematic capture based design and simulation for this portion of the exercise. After they are relatively confident that their code and hardware are working properly, they can use the system model to integrate and test the two. Finally, after they are confident that their software is working with their hardware, they are ready to download both the FPGA bit file and their object code to the XS40.

## 5. Conclusion

After extensive simulation, working hardware the first time is almost anticlimatic. Initial feedback from students in the first semester the lab is being taught has been favorable. One student commented that 'once a design is checked using Mentor tools and then verified with actual hardware and test equipment in the lab, I am confident the design is a good one'. Our use of simulation coupled with actual hardware implementation paid off early in the semester. The students designed a simple eight bit parallel port for the 8051, located at memory location 0xFFFF. There was enough ground bounce when the 8051's address lines switched from all 1's to all 0's to cause a glitch on the address latch enable line and a very subtle program error. Without the confidence gained through simulation that their logical design was actually correct and the error most likely was elsewhere, this problem might not ever have been found by students at this level.

Hardware-software co-verification is a powerful technique that is only starting to be used in industry. Use by undergraduates in Electrical and Computer Engineering can only hasten its adoption and thus improve the process of embedded system design as well as provide students with skills needed by industry. Our initial experience leads us to believe that this will be a common feature in future undergraduate computer laboratories.

## References

[1] H. J. Pottinger, W. Eatherton, "Using a multi-FPGA based rapid prototyping board for system design at the undergraduate level", 37th Midwest Symposium on Circuits and Systems, Lafayette, Lousiana, August 1994.