

# A senior-level course in hardware-software codesign

Patrick Schaumont  
Virginia Polytechnic Institute and State University  
schaum@vt.edu

## Abstract

*Modern electronic system design makes extensive use of programmable architectures, and requires designers to jointly consider hardware and software in their design. A senior-level course called 'Introduction to Codesign' provides a once-over-lightly approach to these complex system design issues. The course covers basic trade-offs in the design of custom architectures and software. In an associated design project, senior students analyze and accelerate a reference algorithm in C and end up with a processor and a coprocessor mapped on an FPGA board. We review the motivations for this course, the curriculum, the lab materials and tools used, and the results of the first offering of the course in the fall semester of 2006.*

## 1. Introduction

Last year, a senior-level (fourth-year) course called 'Introduction to Codesign' was organized at Virginia Tech. The course is targeted at undergraduates and explores the possibilities of hardware/software codesign as a standard design technique.

Academics and industry have praised the virtues of hardware/software co-design for many years. However, neither has shown the willingness to truly change their ways. Academic curricula tend to show a strong bias to either software design or hardware design. Industry employs the graduates in separated hardware and software teams as well. This approach is non-viable in the long term. The design space of programmable architectures is evolving dramatically. A design engineer can choose from a wide range of possible programmable target architectures, including Field Programmable Gate Arrays (FPGA), Application-specific Instruction-set Processors (ASIP), and dedicated signal processors. Hence, we need to train engineers that can jointly consider architecture design (hardware) and application mapping (software).

Typically, hardware/software codesign is considered to be a graduate-level topic. However, it is important to bring it also at the undergraduate level, as only a minority of the

**Table 1. Course Topics**

---

<i>1. Basic Concepts</i>
• Concurrency versus Parallelism
• Control Flow versus Data Flow
• Function versus Architecture
<i>2. Fundamentals of custom architecture design</i>
• Finite-state-machine with datapath
• Micro-programmed Architectures
• Coprocessors: internals and interfaces
<i>3. Methods that map applications into architectures</i>
• Digital Signal Processing/ processors
• Memory organization
<i>4. Recent developments in codesign</i>
• Codesign with Field-programmable Gate Arrays
• Application-Specific Instruction Set Processors
• Synthesis of C into custom hardware

---

student body studies at the graduate-level. Currently, many practicing engineers have to learn hardware-software codesign by 'education in the field'. While this is not necessarily bad, field-solutions are usually unstructured and dedicated to a single case.

Also, to match the challenges of modern design, our undergraduates need to become generalists; they need to be able to talk to peers in a specialization area different from their own. In this course on hardware-software co-design, students may have a hardware bias or a software bias. But they will mingle in a single design project.

We review the organization of the course, followed by a discussion of the design project. This project starts with a reference implementation in C, and ends with a processor-coprocessor implementation on an FPGA board.

## 2. Course Overview

The course 'Introduction to Codesign' has three components: lectures, hands-on assignments, and a system design project.

**Topics.** Table 1 summarizes the lecture topics. There are four parts: basic concepts; fundamentals of custom architecture design; methods to convert applications into architectures; and recent developments in hardware/software

**Table 2. Materials on Knoppix CD-ROM**

Compilers/Assemblers	
SD C Compiler	<a href="http://sdcc.sourceforge.net">http://sdcc.sourceforge.net</a>
arm-linux-gcc	<a href="ftp://ftp.arm.linux.org.uk">ftp://ftp.arm.linux.org.uk</a>
picoblaze assembler	<a href="http://www.xilinx.com">http://www.xilinx.com</a>
Simulators/Code generators	
SimIt-ARM-sfu	<a href="http://simit-arm.sourceforge.net">http://simit-arm.sourceforge.net</a>
GEZEL	<a href="http://rijndael.ece.vt.edu/gezel2">http://rijndael.ece.vt.edu/gezel2</a>
Octave	<a href="http://www.octave.org">http://www.octave.org</a>
Valgrind	<a href="http://valgrind.org">http://valgrind.org</a>

**Table 3. Sample Results by Students**

Algorithm	C lines	GEZEL lines	Area Slices	Speedup
3DES	391	558	870	1300
Blowfish	457	472	412	6.3
Misty	224	918	275	15.2
MD5	441	772	-	1.17
Salsa20	220	533	1381	2.54
Phelix	607	357	568	1.2

codesign. A key idea in the first half of the course is that a single behavior can have many equivalent implementations. So for a given C program, many hardware architectures can be built with identical functionality but a different performance [1]. In the second half of the course, design methods are added to the picture. Applications and architectures are connected by these methods and they help a designer to navigate the complex architecture space.

Throughout the lectures, the emphasis is laid on practical skills by means of examples and small in-class exercises. This pragmatic approach is motivated by the need for simple, common-sense techniques to cope with complex system design. In this context, we also found that current textbooks on hardware-software co-design are too complex and too research-oriented for undergraduate education, and we developed new lecture notes.

**Project.** Half of the grade of the course is given to practical assignments. These assignments are organized as one large system design project, in combination with five hands-on assignments serving as homework.

The design project is a coprocessor design for various block ciphers and stream ciphers (See Table 3). We found cipher algorithms to be very suitable for simple codesign experiments. The design project is split up in five phases, with a design review after each phase. The five project phases include high-level reference design (in C), embedded implementation on StrongARM, hardware-accelerator design, hardware-software integration, and FPGA-prototyping. Apart from the final phase, all performance evaluation is done using instruction-set simulation and cycle-based hardware simulation. In the final phase, the entire codesign is mapped onto an FPGA board

using the Xilinx Embedded Toolkit. The implementation step confirms the high-level simulation models, and it returns the technological cost of the coprocessor.

We used GEZEL modeling and cosimulation tools (<http://rijndael.ece.vt.edu/gezel2>). The basics of synchronous hardware design in GEZEL take no more than a single lecture to explain. GEZEL also has a path to implementation in VHDL. Neither of these qualities are simultaneously present in HDL (too much details) or SystemC (no path to implementation). We restricted the hardware-software interfaces to those offered by the Microblaze processor of the Xilinx EDK toolkit, including the loosely-coupled On-chip Peripheral Bus (OPB) and the tightly-coupled Fast Simplex Link (FSL). Both OPB and FSL interfaces are supported in cosimulation and in implementation.

**Materials.** To minimize installation and configuration issues with design software, students receive a Knoppix CDROM. As illustrated in Table 2, the CDROM comes pre-installed with various tools including cross-compilers, cosimulators, design examples, and so on. It is used to solve hands-on assignments and also supports the first four phases of the project. A live-Linux CDROM is a hassle-free way of distributing design software: virtually no installation issues were reported. For the final project phase, we made use of Xilinx XPS in combination with a Xilinx Spartan 3E starter kit. This way, students have to purchase only a personal Spartan 3E board (\$120, leveraged over multiple courses that use this board); all design software is free.

### 3. Results and Conclusions

Table 3 shows the results obtained by the students. Of particular interest is the final speedup (col 5) of the coprocessor over the initial reference implementation. Midway the project, most teams were optimistic to achieve large speedups (orders of magnitude). But many ended with a much more moderate result. This was almost always caused by excessive hardware/software communication. This result illustrates the importance of thinking across the boundaries of hardware and software. Based on the positive feedback of the first offering, we plan to continue offering the course as a design elective for our computer engineering students.

### 4. Acknowledgements

This project was partially supported through NSF CCR 0644070. We acknowledge the support of Wei Qin (SimIt-ARM's author) with tools during the course. We also acknowledge the diligence and effort of students that took this course on its' first offering.

### References

- [1] J. Madsen, J. Steensgaard-Madsen, L. Christensen, "A sophomore course in codesign," *IEEE Computer*, 35(11):108-110, November 2002.