Hardware/Software Codesign - introducing an interdisciplinary course

Micaela Serra and William B. Gardner

Dept. of Computer Science

Univ. of Victoria, Victoria, B.C. Canada

mserra@csr.uvic.ca

WCCCE Conference - Vancouver, 1998

1.0 Abstract

In 1997 we introduced a new fourth year/graduate course in the Department of Computer Science at the University of Victoria entitled “VLSI Design, CAD and Hardware/Software Codesign” under the rubric of “Special Topics”, where generic topics courses are the normal way that professors here inaugurate courses based on recent development in our research. The course was subscribed by students from both our own department and the Department of Electrical and Computer Engineering, and was received well enough to be immediately repeated the following term. In this paper, we explain our motivation, describe the course, and report on the results. The exposition is colloquial, as it reflects the script of the oral presentation at the conference itself.

2.0 What is Hardware/Software Codesign?

In this presentation, it is important that we first start by introducing the topic of Hardware/Software Codesign, as it is relatively new and may not be entirely familiar to all readers [2,7]. The following are the major definitions which capture the essence of the area:

- the cooperative design of hardware and software components;
- the unification of currently separate hardware and software paths;
- the movement of functionality between hardware and software;
- the meeting of system-level objectives by exploiting the synergism of hardware and software through their concurrent design.

Some of the definitions may seem to apply directly to any system design area. The difference, elaborated more below, is in the types of applications, and in the dynamic interaction of choices between hardware and software allocation to optimize the final product. Given a set of specified goals and an implementation technology, designers consider trade-offs in how hardware and software components work together. This leads to the need for more flexible design strategies where hard-

ware and software designs proceed in parallel with much feedback and interaction. Decisions are evaluated on performance, programmability, area, power, development and manufacturing costs, reliability, maintenance, design evolution (union of software and hardware).

Why is this topic important? First of all, we must take into consideration the advances in new architectures based on programmable hardware circuits - namely Field Programmable Gate Arrays (FPGA's) and Digital Signal Processors (DSP's). Their introduction as alternative computational units and the flexibility they offer have required different methodologies in the design process, reflecting the large number of choices. Today's computing systems deliver increasingly higher performance to end users; thus we require architectural support for operating systems, or particular hardware to expedite application-specific software (product evolution). The new architectures based on programmable hardware circuits are usually geared to accelerate the execution of specific computations or emulate new hardware designs, before they are committed to a specific (more expensive) ASIC chip. The key item here is "reconfiguration": exploiting the synergy between hardware and software.

Figure 1

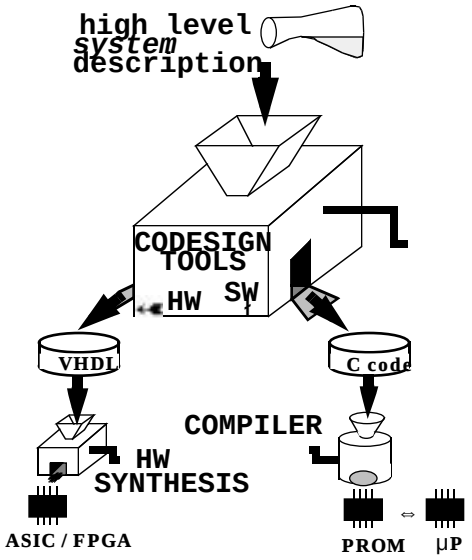


Figure 1 represents a more utopian view, where codesign and codesign tools provide an almost automatic framework for producing a balanced and optimized design from some initial high level specification. The goal of codesign tools and platform is not to push towards this kind of total automation; the designer interaction and continuous feedback is considered essential. The main goal is instead to incorporate in the "black box of codesign tools" the support for shifting functionality and implementation between hardware and software, with effective and efficient evaluation. At the end of the process, either the prototypes or the final products are output, based on currently available platforms (software compilers and commercial hardware synthesis tools). Codesign as an area of research does not aim to reinvent the wheel of system design; however, the necessary flexibility must be effectively incorporated and enhanced. For example, in the design of a real time system as a graduate project, a sub path in the figure above may indeed be followed. The difference is that the designers are given predetermined choices of hardware and software allocation and must meet the timing constraints within the specifications. Codesign introduces the research into the trade-offs of that allocation, dynamically throughout the entire process.

Mostly though we look at the largest application area of hardware/software codesign: embedded systems [3,6]. They are application specific systems which contain both hardware and software tailored for a particular task and are generally part

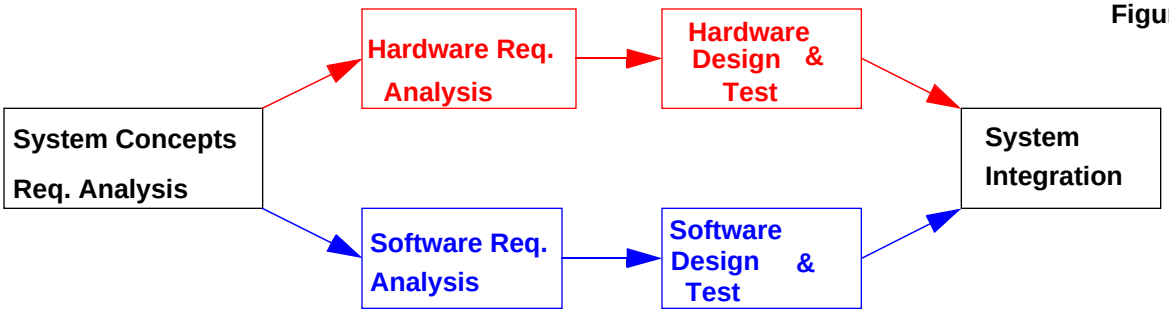
of a larger system. Examples range from the automotive fields (ABS brake controllers, almost any other monitoring sub-system), the portable communication or computing systems (the Palm Pilot, any cellular phone), to the medical area (as in portable heart monitors). Embedded systems often fall under the category of reactive systems, that is, containing sensors or similar elements which constantly interact with the external environment. The components used usually include a general purpose processor, one or more special purpose processors and some ASICs. For example, real-time systems are types of reactive systems which must meet some time constraints. A major issue in an embedded system is to provide design approaches that scale up, without a total redesign for a legacy product.

Finally the interdisciplinary aspect between computer science and computer engineering, and the intradisciplinary aspect within computer science poses a challenge to us as educators and pivots for effective technology transfer; how to provide senior and graduate students with a framework such that they can learn to work concurrently.

3.0 What is the problem with current practice and how does codesign circumvent these problems?

In the conventional design process, the hardware and software split of components is decided early, usually on ad hoc basis, creating what is commonly called a " *Model Continuity Problem*". Figure 2 shows graphically the two paths, leading to a final system integration, with no reconfiguration choices shown after the initial split.

Figure 2



Model continuity is important because:

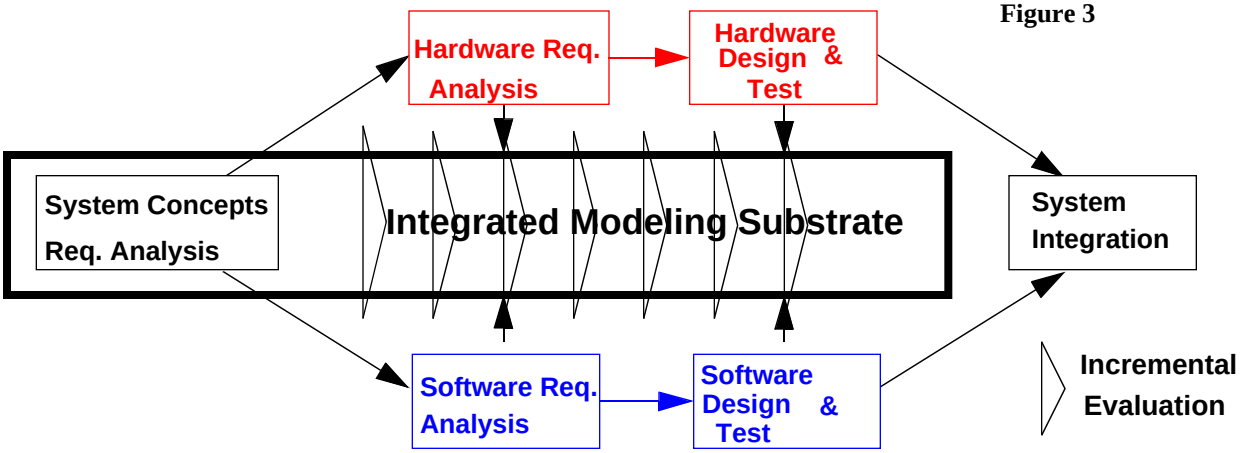
- many complex systems do not perform as expected in their operational environment;
- continuity allows the validation of system level models at all levels of hardware/software implementation;
- trade-offs are easier to evaluate at several stages.

The consequences of losing such model continuity include:

- cost increases and schedule over-runs (due to modifications late in phases);
- the ability to explore hardware/software trade-offs is restricted (e.g. movement of functionality between, modification of interfaces);
- state of the art applications require a case-by-case analysis;
- different design cultures hamper integration; here we can make a difference with an appropriate curriculum!

One of the labels given to some solution is based on the concept of a "*Unified Design Environment*", as graphically shown in Figure 3, where it is emphasized that hardware design and software design use the same integrated infrastructure, resulting in an improvement of overall system performance, reliability, and cost effectiveness. It is easy to draw such picture and assign grandiose labels. Yet here the triangles shown spanning the two paths and covering the integrated substrate do not refer to mere feedback sessions and weekly designers meetings! They represent, at a minimum, an integrated database, with powerful tools which can support the exploration, prototype implementation and rapid evaluation of the repartitioning of functionality between hardware and software, together with an essential and extremely effective cosimulation platform.

Figure 3



It is not the purpose of this presentation to teach a whole course in hardware/software codesign and present the many solution given to the issues just described. However it is useful to list the main research areas, which are:

- *Modeling:* best methodology for specifying hardware/software systems;
- *Partitioning:* how to divide specified functions between hardware and software;
- *Cosynthesis*: generating the hardware and software components;
- *Cosimulation:* evaluating the synthesized design.

Why can't CAD tools with successive simulation suffice? And why can't rapid prototyping systems suffice? A few points are as follows:

- System design space is very large if not unlimited.
- The designer should be able to manipulate during the design: Algorithms, System Structure, hardware/software partitioning and implementation technology with ease;
- A typical Rapid Prototyping system may consist of:

  - a distributed structure with a single board computers and custom boards. These boards usually run a real-time operating system.

  - the custom boards consist of processor modules and application-specific slaves, where software processes are mapped to processors and hardware processes to custom ASICs.

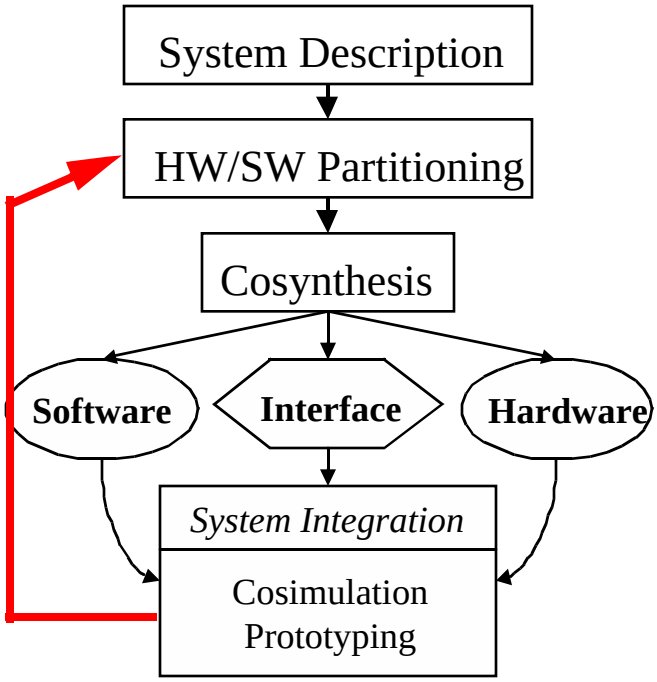
- Thus Rapid Prototyping may shorten the design path but does not answer basic questions, such as where do we divide the functionality: on a DSP or in custom hardware?

It is how the decision is made that is of research interest! Codesign tools allow the designer to avoid *local maxima* by enabling *design space exploration.*

3.1 What would be a typical context for a hardware / software codesign process?

Given the emphasis placed on interaction and the need for reconfiguration during the whole of the design process, we can summarize in figure 4 the "ideal" process flow that codesign wants to support.

Figure 4



The red "interaction and feedback" arrow is the crucial part. Another important aspect is the central "Interface" submodule, which in normal system design is often left on the sideline, causing disastrous effects at the time of integration. Given that many embedded systems which use codesign methodologies are often implemented at a very low level of programming and details (e.g. assembly code), the proper development of an effective interface becomes extremely important, even more so from the view that any reconfiguration of the design will change the critical interface modules!

4.0 How we teach Codesign

The list of subtopics organized in our course is as follows, not necessarily in the precise time order:

1. Codesign overview
2. *Overview of VLSI Design*
   - *Technology and Fabrication*
   - *Algorithms*
3. *Overview of CAD*

- *Exposure to Mentor, Synopsys*
- *VHDL*

4. Codesign methodologies
5. Embedded Systems
   - Algorithms and Unified Methods
   - Partitioning and Cosimulation
6. *FPGAs*
7. Codesign CAD tools: Ptolemy, SpecSyn/SpecCharts

The items in points 1, 4, 5 and 7 represent the main bulk of the course, discussing all the issues and solution in current codesign research. The codesign CAD tools of point 7 are distributed throughout the course with introductory labs and design assignments.

The items in points 2, 3 and 6 represent a summary of hardware and VLSI design, which is much needed by both the computer science and, often, by the computer engineering students [8]. The labels may look ambitious, but clearly only a survey of the topics is given. The difficulty is indeed in finding the necessary balance between depth and breadth. The reward was that all students find these subjects particularly interesting and extremely “empowering”, by giving them the confidence that they could hold their own in almost any design environment by being able to grasp the essentials and know where to find more information of details for both system, software and hardware issues.

The software tools we introduced prove to be a pivot for the students’ experience, as they gave them direct exposure and the power to develop significant projects (see below).

The first framework introduced is Ptolemy, distributed free from U.C. Berkeley and also available commercially. Ptolemy models systems with functional building blocks (stars, galaxies) and is ideal as a design platform at a high level, as it contains choices for many subdomains. Ptolemy can also be useful for other courses not necessarily on codesign, but focused on more general design, as it provides a sturdy platform for models based on Petri nets, CSP, etc. It contains tools for simulation and cosynthesis, but it is weak on partitioning and reconfiguration.

The second, more recent, system is SpecSyn/SpecCharts. A design is modeled using SpecCharts which supports concurrency, state transitions, hierarchy and synchronization; SpecSyn takes care of design space exploration and cosynthesis in a very effective manner. The advantage of this platform is that it is based internally on VHDL and the user programmability can be interfaced with VHDL as well. Thus it is well suited to both computer science and computer engineering students who find themselves very comfortable with a high level programming language. As an aside, VHDL is a new item for most students, and, notwithstanding any industrial or academic discussion as its advantages and disadvantages, it proved to be fascinating and powerful in its flexibility (the behavioral, structural and dataflow paradigms that VHDL allows).Figure 5 shows a schematic of Specsyn/SpecCharts and its flow of operation for a designer.

Last, but not least, programmable hardware, in the form of FPGAs is introduced in the course [8], but it is beyond the scope of this presentation. The use of FPGAs is the first step towards the research area called “*Configware*”, referring to dynamically reconfigurable subsystems. A short introduction to VHDL as a programming language is also given through one lecture and an assignment.

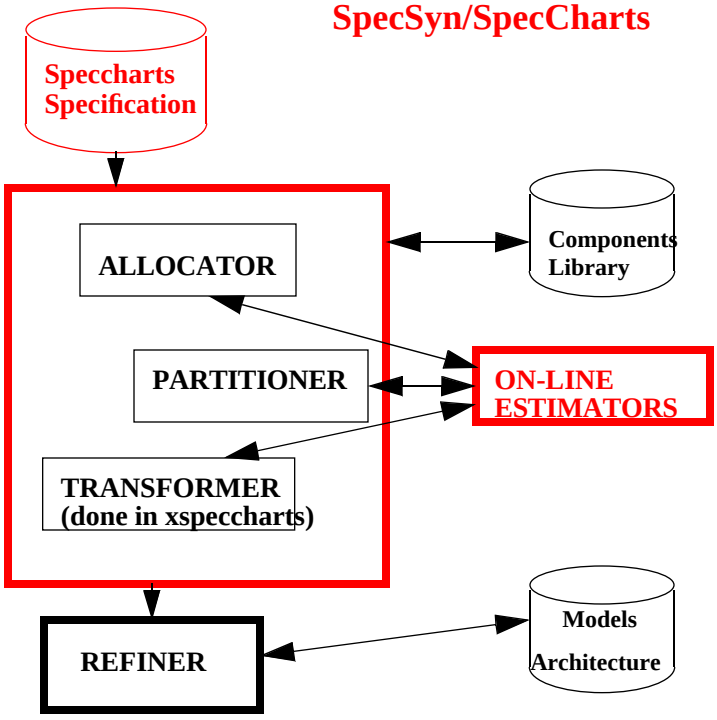


Figure 5

5.0 Projects: Students and Research

One major part of the course is presenting to students current research projects in the area developed locally and the evaluation of a student project of their choice. Students can choose a design project or a literature search on a topic beyond the scope of the lectures. The following list shows a small selection of titles of the most interesting design projects to date.

- Hand held "Golf Guy"
  - a Palm Pilot for golfer (with GSS incorporated)
- Accelerated graphic cards with FPGA's
  - an FPGA card which changes personalities
- Hardware acceleration for data compression
  - another personality inside a large system
- A rowing Coach Assistant
  - a Palm Pilot for rowing coaches
- Java Virtual Machine on the Rapid Prototyping Board [1] from the Canadian Microlectronics Corporation (CMC).

On the side of "technology transfer", three research projects were described to students: *"Intragiz"*, an object-oriented layered approach to interfaces for hardware/software codesign [5]; *"Gizgate"*, an object-oriented gateway for hardware/software codesign on the CMC Rapid-Prototyping Board [1,7]; *"Concurrent simulation of heterogeneous multiprocessor*

embedded systems", the main research in a powerful industrial cosimulator [4]. For each of these research topics, the papers and the related literature were presented.

In summary, the following points became clear in the overall evaluation of the course itself, the impact on the students and curriculum, and the students' level of satisfaction.

- The interdisciplinary aspect between Comp. Science and Comp. Engineering and the intradisciplinary aspect within Comp. Science was somewhat new and a source of great strength under all views.
- The hardware related topics were tremendously empowering to the mainly software students in Comp. Science, who found the demistification of the whole area of VLSI design and CAD software useful to their breadth, especially towards technical jobs in smaller engineering companies.
- It is fun to see a whole system designed (small, embedded) and use state-of-the-art tools. Also it is fun to see all sides of the design process.
- It is fun to hear about how things (circuits) are built and fabricated (take a few grains of sands and here is your chip!).
- The course required lots of skills - a true integration of Comp. Science streams, and a push towards breadth.

The Web page related to the course is currently still available at http://gulf.uvic.ca/~csc556; in the future a summary will remain posted.

6.0 References

1. Paul Chow and Rob Jeschke. *Rapid-Prototyping Board Users Guide.* Dept. of Electrical and Computer Engineering, University of Toronto, February 1996. CMC #ICI-068_Users_Guide R/01.
2. G. De Micheli and M. Sami, Ed. *Hardware/Software Co-Design.* Kluwer Academic Publishers, 1996.
3. D. Gajski, F. Vahid, S. Narayan and J. Gong. *Specification and Design of Embedded Systems.* Prentice-Hall, 1994.
4. W. Gardner, and M. Serra, "Concurrent Simulation of Heterogeneous Multiprocessor Embedded Systems", *Proc. of 7th Int. Symp. on IC Technology, Systems & Application,* Sept. 1997.
5. W. Gardner and M. Serra. "An Object-Oriented Layered Approach to Interfaces for Hardware/Software Codesign of Embedded Systems". *Proc. Hawaii Int. Conf. on System Sciences,* Jan. 1998.
6. S. Kumar, J. Aylor, B. Johnson and W. Wulf, *The Codesign of Embedded Systems - a Unified Hardware/Software Representation.* Kluwer Academic Publishers, 1996.
7. J. Rozenblit and K. Buchenrieder, Ed. *Codesign.* IEEE Press, 1995.
8. D. Sharp, W.B. Gardner and M. Serra, Gizgate: an object-oriented gateway for hardware/software codesign on the CMC Rapid Prototyping Board. Proc. FDP '98, Montreal, June 1998.
9. M. Smith , *Application-Specific Integrated Circuits.* Addison Wesley, 1997.