VLSI Hardware Design by Computer Science Students: How early can they start? How far can they go?

Ney Laert Vilar Calazans, Fernando Gehm Moraes
Faculdade de Informática - Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre - RS – BRAZIL - ZIP Code: 90619-900
E-mail: {calazans, moraes}@kriti.inf.pucrs.br

***Abstract** - This work describes a novel approach to teach computer organization concepts with extensive hands-on design experience very early in Computer Science curricula. While describing the proposed teaching method, it addresses relevant questions about teaching VLSI design to students in computer science and related fields. The approach involves the analysis, simulation, design and effective construction of processors. It is enabled by the use of both, VLSI hardware prototyping platforms constructed with reconfigurable hardware and powerful computer aided design tools for design entry, validation and implementation. The approach comprises a 4-hour a week lecture course on computer organization and a 2-hour a week laboratory, both taught in the 3rd semester. In the first two editions of the course, most students have obtained successful processor implementations. In some cases, considerably complex applications such as bubble sort and quick sort procedures were programmed and run in the designed processors.*

Introduction

Computer architecture and organization are well-established disciplines in Computer Science and Computer Engineering curricula. Quite often, these contents are covered early in undergraduate courses, and the approach is absolutely sound. This happens because of the crucial need to provide the student soon with a strong understanding of the relationship between the physical reality of hardware and the software abstractions it implements.

On the other hand, VLSI digital systems design is a subject usually covered, if at all on elective courses, at the end of the undergraduate curriculum. Also, these VLSI courses frequently deal mostly with low level aspects of the design process. Examples of such low level aspects are tackling only physical and logical abstraction levels.

However, computers *are* VLSI digital systems, formed either by a composition of several VLSI subsystems or even by a single VLSI chip. Thus, an excellent way to teach and to understand their power and limitations is by effectively designing them. Two recent technological advances allow this to become a reality in the classroom. First, the availability of cheap, powerful VLSI hardware prototyping platforms based on reconfigurable hardware such as FPGAs and CPLDs. A good example of the profusion of available hardware aids is the list maintained by Guccione [1]. Second the existence of easy to use, powerful, free and/or commercial computer aided design tools for high-level design entry, validation and implementation. Examples of these tools are the current simulators and synthesizers based on Hardware Description Languages (HDLs).

This work describes a novel approach to teach computer organization through the analysis, simulation, design and effective construction of processors using the aforementioned advanced facilities. The next Section describes the context and structure of the implemented courses. Section 3 considers the merging of VLSI design techniques into computer organization, and suggests ways to do this into Computer Science and Engineering curricula. In Section 4, the teaching methods employed are introduced and discussed. Section 5 assesses issues arising from the choices of tools and problem solving strategies. Section 6 presents the current state and the future of the reported work, while Section 7 provides a few conclusions.

Courses Context and Structure

The proposal for computer organization teaching is part of a revised Computer Science undergraduate 9-semester curriculum, which went into effect in the 1st semester of 1997 (starting in March) at the authors' institution. It was implemented as two required courses, a 4-hour a week course on computer organization and a 2-hour a week laboratory course both taught in the 3rd semester. The basic prerequisites for both courses are an introduction to Digital Circuits, a course on Algebraic Structures and another on Physics for Computer Science. These courses provide the student with traditional combinational and sequential logic design techniques, lattice and Boolean algebra theory, and a brief account of circuit and electronics theory and instrumentation, respectively. Three other required courses, two on computer architecture and one on microprocessors follows those on computer organization. There is also the possibility of taking some elective courses on selected advanced topics on digital systems. In this way, the student is exposed during the whole curriculum to hardware issues regarding his future profession. This was one of the objectives of the 1997 revision.

The computer organization (INF46183) course contents are distributed into four units:

- Unit 1: The digital systems design process – where models for the design process like the Y-diagram [2] and others [3] are introduced, together with a discussion of digital systems' classifications and the basic digital circuit design flow with the use of CAD tools.

- Unit 2: The CPU classical model – where the datapath and control unit partitioning is explored through the presentation of both von Neumann and Harvard organization models and the development of a case study design at the RTL abstraction level.
- Unit 3: Hardware description languages (HDL) – in which is presented another, more abstract way of designing digital systems and, in particular, processors. This includes the redesign of the CPU case study of Unit 2.
- Unit 4: Advanced topics on computer organization – where the previously studied concepts are complemented by introducing advanced hardware structures for datapath and control unit performance enhancement, such as pipelining and floating point arithmetic hardware.

On the other hand, the computer organization laboratory course (INF46184) is divided into three parts:

- Unit 1: Classical digital circuit design - where the students familiarize with schematic capture and simulation tools. In this unit they implement basic combinational and sequential blocks such as adders, ALUs, counters and finite state machines.
- Unit 2: Classical CPU design – employing the same tools of the previous unit, the case study of the lecture course is implemented step by step, using RTL modules.
- Unit 3: HDL hardware design – where an HDL language is introduced and used to describe hardware at the behavioral and structural levels, contrasting the complexity of the circuits they can handle with that of the previous approach.

Computer Organization and VLSI Design Teaching

There are two quite different ways of approaching the design of digital systems at the undergraduate level, the one employed in Electrical and/or Electronic Engineering, and that of Computer Science and/or Computer Engineering.

Electrical engineering students start with a strong emphasis in linear circuits, which are used later as the foundation to investigate electronic phenomena and devices. The study of digital circuits and systems principles and techniques comes very often late in the curriculum as a special case behavior. In this way, electrical engineers are usually strongly aware of physical consequences of designing digital circuits, like clock distribution problems and timing constraints. However, they often fail to grasp architectural and software issues of a complex VLSI design.

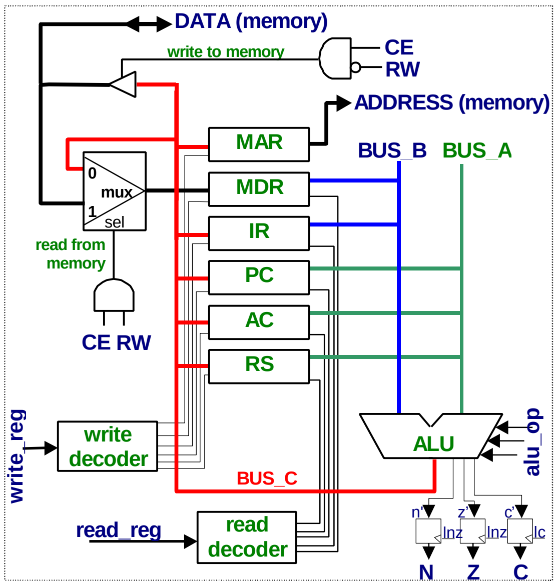
On the other hand, Computer Science students get acquainted with digital systems much earlier than Electrical Engineering students get. They employ abstract models from Boolean Algebra as basis, instead of circuit theory models. Thus, it is hard for them to deal with timing and power constraints affecting the performance or even the functionality of the whole system. However, they are usually well equipped to deal with higher levels of abstraction such as architectural and software issues of VLSI systems.

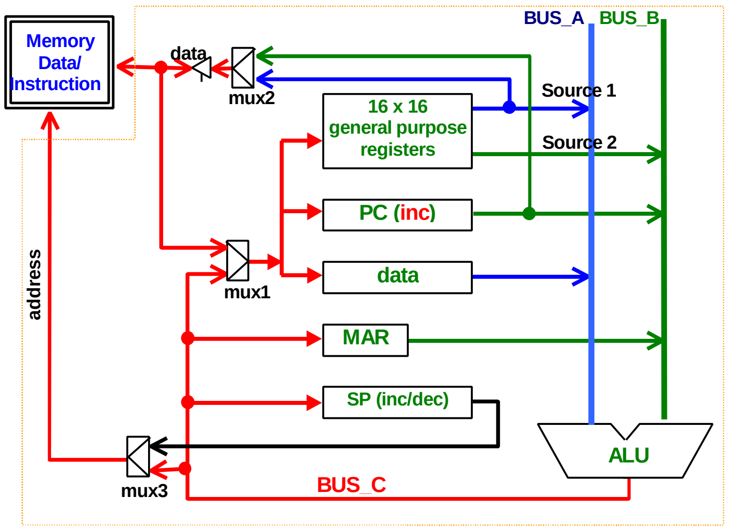
As mentioned in the beginning of this paper, two technological advances, CAD tools and fast prototyping facilities enable to teach computer organization through the effective implementation of processors. They do so by allowing designers to overlook most physical synthesis issues, shifting the design effort to more abstract levels. This approach is particularly well suited to Computer Science students, since they may then use physical synthesis as a *push-button* activity, concentrating on higher level organization and architecture problems.

Teaching computer organization and architecture in this way helps Computer Science students demystifying hardware design activities. Also, the students are able to understand more thoroughly problems appearing in future courses like compiler performance optimization, memory management and input-output bottlenecks at the system and operating system level.

However, the same set of tools can most often be used with lower level preoccupations in mind, since they include aids for performing tasks such as IC floorplanning, performance-driven placement, routing and clock distribution. This would of course be a more appropriate approach to Electrical Engineering students.

From the above discussion, it is reasonable to conclude that the ideal team to design VLSI systems, and particularly computers, must be a composition of both engineers and computer scientists. One problem with multidisciplinary teams is that they often fail to communicate properly, due to the different views each professional has of the same basic concepts. Another beneficial side effect of using the above teaching approaches is to bridge this communication gap, by providing common ground knowledge in these concepts to both classes of professionals. This issue is better understood by presenting a few examples of practical subjects where communication problems arise. Let these examples be external/internal memories and pipelining. We consider in the analysis only what computer science students acquire to bridge the mentioned communication gap. Several other example subjects and the engineering student side of bridging the gap could be devised as well. A computer science student often learns to view memories simply, as tables coupled with means to access their data. We propose a simultaneous VLSI view of memory technology, which is helpful in understanding the need to address the memory bandwidth problem not only in hardware design, but also in software and, above all, operating system design.





(a) Cleopatra datapath higher level schematics (b) Ramses datapath higher level schematics

Figure 1 – Case study and final work datapaths for 1998/2 edition of INF46183 course.

In this way, computer scientists are able to better interact with hardware designers in issues like the SRAM/DRAM trade-off. Pipelining is another point where communication can be a problem between hardware and software personnel. Software people usually deal with an abstract view of pipelines, as provided by the assembly language level, which resembles very little the multistage organization of modern hardware pipelines. The exercise of implementing VLSI pipelined hardware enables the computer science student to better understand the hardware issues as well as the implementation of the assembly language abstraction, and thus to enhance his/her capacity to communicate with hardware specialists.

Finally, there is the recent trend to provide curricula that are midway between Electrical Engineering and Computer Science such as Computer Engineering. The students of these latter courses should benefit of the approaches proposed above, applied in early and late moments of their undergraduate academic experience, respectively. The basic idea here, as well as on computer science curricula is to modify the traditional early courses on computer organization and architecture to include training with the VLSI implementation of the hardware structures needed to support the abstract view of computers in these disciplines. Second, to employ elective courses at the end of the curricula to deepen the training of the interested student in topics of VLSI and computer abstract design.

Employed Methods

Both the lecture and lab courses are based mostly on the stepwise analysis of a single processor case study named Cleopatra [4]. Its datapath is depicted in Figure 1(a). Since the emphasis is on computer organization models, the case study is neither complex nor modern. It consists in a simple 8-bit, accumulator-based von Neumann core, with 13 operation codes and 4 addressing modes, amounting to 37 distinct instructions.

The Cleopatra core assists in introducing and discussing several new concepts in the lecture course. The specification allows exploring instruction sets, addressing modes and principles of assembly language programming. Once these few architectural concepts are apprehended, a processor datapath organization of Figure 1(a) is proposed and studied. Among the concepts introduced, it is worth highlighting:

- Control and data registers and their role.
- Memory-processor interaction, differentiating von Neumann and Harvard organizations.
- ALU implementation and busing strategies.
- Control unit structure, comparing hardwired and microprogrammed implementations.

The lab course examines the VLSI design view of computer organization in detail, using CAD tools to construct increasingly more complex modules of Cleopatra.

Both courses start with a traditional, schematic-based approach of the core design. Later, the whole design work is redone with modern, HDL-based tools. There are three main reasons for doing so:

- Showing that it is possible to design at very high abstraction levels and still obtain competitive implementations, due to the current quality of current CAD tools.
- Providing students with a comprehensive insight into the panoply of hardware design methods and tools, by comparing these two significantly distinct approaches.

- Making it clear to students that HDLs are not programming languages, which is achieved by the mapping of schematic symbols and structures to the syntactic and semantic structures of the chosen HDL.

The theoretical course comprises approximately 30 2-hour lectures distributed roughly as follows:

- 25% to revisit digital circuits basics and to present design process models and taxonomy.
- 30% to present the Cleopatra case study , developing and discussing its schematic-based design.
- 25% to introduce a HDL and to redesign the case study with it.
- 20% to analyze advanced concepts in computer organization.

In the lecture course the students are required to design a load-store 16-bit processor as final work using the target HDL. The processor specification includes the instruction set, addressing modes and the datapath organization. One example datapath, provided in the last edition of the course appears in Figure 1(b). The most important design constraint imposed by the specification is a fixed number of clock cycles per instruction (CPI=2). The specification of this final work is delivered to the students during the first HDL introductory lectures. In this way, they have 45-50 days to complete the design and present a running functional HDL simulation.

The lab course comprises around 15 2-hour sessions with the following distribution:

- 60% to get acquainted with the CAD tools and explore the schematic based approach to processor implementation.
- 30% to investigate the HDL-based approach.
- 10% to introduce physical synthesis concepts and training using reconfigurable devices.

At the end of the semester, the students have obtained a deep understanding of VLSI processors design at high levels of abstraction. They write assembly language programs that run on the machine they implemented by themselves. Thus, they are able to develop software and follow the data and control information flow internal to the hardware.

Tools

It is common in teaching environments the adoption of the so called educational or academic tools to support several disciplines, particularly in those related to digital systems design and validation [5,6,7]. The justification for doing so is to avoid the alleged disadvantages associated with commercial tools. Among these disadvantages are their steep learning curves, the necessarily expensive hardware platforms they require, and their limiting licensing requirements. These last requirements reduce access in the educational institutions to a few sites and avoid that students have copies of the tools in their home computers. Also, commercial tools come professedly with very complex manuals [5]. Another frequent justification is that in some cases no commercial tool is available to explore the subject of the courses, as in the case of hardware formal verification and other recent advances.

The current academic experience of the authors leads them to disagree with all but the last of the above pleas, at least in the case of the work reported here. Steep learning curves and complex manuals are a fact of life, but it is important to avoid underestimating the capacity of the students to cope with it. On the other hand, much time is spent in the courses convincing the students that designing VLSI systems is indeed a formidable task. The complexity of the tools is an invaluable help in this sense. Educational tools may indeed ease learning, but they provide a poor picture of the real world tools used to solve real world problems. The adopted option here is to face them early with the use of real world tools, even if the problems they solve are obviously away from the state of the art by some years. In practice, students have reacted quite favorably to the use of commercial tools with regard to the educational tools employed in previous courses. This occurs in response to either the power of the tools, or to the fact they feel part of the real world of solving complex design problems. With regard to hardware platforms, the present state of the art in personal computers and CAD systems technology allows the use of professional tools on inexpensive platforms. This combination has sufficient power to solve the biggest problems found in undergraduate courses easily. As for licensing requirements, many software and hardware enterprises have formal or informal university cooperation programs that allow charging very little or nothing for an unlimited number of software licenses.

For example, the lab course mentioned above is provided with two professional systems (around 30 licenses of each), 20 hardware prototyping kits and 20 Pentium-based NT-workstations. The only investment needed to keep the lab up-to-date is around US$600 a year for maintenance of one of the CAD systems.

The authors have chosen to use VHDL as the Hardware Description Language in both courses. Again, this is not the easiest way to approach higher level abstract design entry, but they consider it worth, since VHDL is not only the most complete, but also the most used of HDLs today.

The lecture course final work must be developed using a VHDL simulator. The students have available the Aldec Active-VHDL simulator [8] for this. The lab course employs the Xilinx Foundation CAD system [9] for schematics and VHDL design entry, synthesis and implementation. Functional and timing validation for schematics is attained through the built-in logic simulator in the Foundation system. The Active-VHDL tool is used to guarantee functional validation of VHDL code. This simulator can be fully integrated in the current version of the Foundation CAD system.

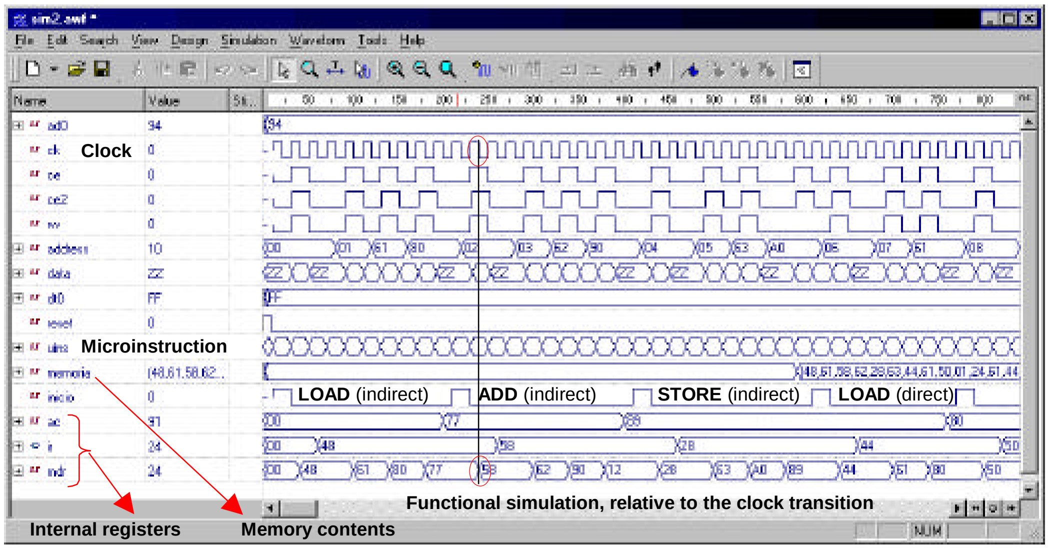


Figure 2 – Partial functional simulation of an object code program in the Cleopatra processor.

Functional VHDL simulation is the most used tool for initial validation and error correction in the first phases of design. As an illustration of the complex simulation run by students, Figure 2 depicts a partial view of an object code program execution, showing four complete instruction cycles. The Figure stresses the functional nature of the simulation, where no relative delay among signals is considered.

Also employed is the SIS public domain synthesis system from Berkeley [10], for illustrating control unit logic synthesis strategies. Assemblers and simulators have been developed by one of the students for the two case studies.

On the hardware side, the prototyping kit is the combination of two products of Xess Corporation [11]: the XS40-005XL and XST-1 boards. The first board contains a Xilinx FPGA for hardware prototyping, an Intel 8051 microcontroller, memory and a parallel port interface for communication with a PC host. The second board enhances the I/O and memory capabilities of the first, providing video and audio interfaces, as well as extended memory.

FPGA programming, software development and hardware-software downloading and execution are achieved directly from the NT-workstation through the parallel port and dedicated software tools.

Current State and Evolution

In the first two editions of the computer organization courses (98/1 and 98/2) only design entry and functional simulation tools have been employed, due to the lack of prototyping platforms, obtained only in 1999/1. The next step is adapting the course contents to the new reality, inserting lab experiments to address:

- Physical synthesis, approaching performance results of this step.
- Timing simulation with delays computed at the physical synthesis.
- Performance driven synthesis, with the use of timing or area constraints.
- Downloading and testing of designs on the prototyping platform.

This implies a major reformulation of lecture and lab contents, and a good balance between abstract design issues and effective VLSI hardware implementation must be achieved in the available time.

Another consequence of the availability of effective hardware design resources is the extension of the approach proposed here in a near future to the computer architecture and microprocessor courses in the same curriculum. The main objective is to provide students with a continued exposition to hardware design issues during their academic life. Advanced topics that are to be covered in such extensions are for example pipeline and cache hardware implementation issues.

Connecting the subjects covered here to the elective courses on digital systems at the end of the curriculum is another possibility of extending the work. The objective can be either to explore in more depth the CAD systems available to implement more realistic designs or the development of CAD algorithms. With the hardware and software support tools available, there is one recent and important subject that can be examined in elective courses, the hardware software partitioning and codesign. Last but not least, we are considering the application of this VLSI design technology to other, apparently unrelated disciplines in Computer Science, such as:

- Compilers – to develop as course work a C compiler for the case study processor and/or the prototyping platform microprocessor.

- Computer networks – to implement the lower layers of the OSI/ISO protocol.
- Computer graphics- to implement in hardware time-consuming algorithms, like rendering and texturing.

Conclusions

The main goals set when developing the approach proposed in this paper are considered attained. Integrating VLSI design and computer organization subjects were successfully achieved.

After teaching the courses twice, the authors have achieved a considerable level of satisfaction on the part of the students. Several successful processor implementations have been obtained and some of these present outstanding functional quality. In some cases, considerably complex applications such as bubble sort and quick sort procedures were programmed and run in the designed processors.

It is possible now to turn the attention to the questions in the title. *How early can Computer Science students start designing VLSI hardware design?* The authors' experience demonstrates that students may start at the 3rd semester of an undergraduate curriculum, and yet achieve a high degree of success. *How far can Computer Science students go in doing VLSI hardware design?* So far, it has not been possible to answer this question fully, since only now the hardware and software resources to do so are complete. Before these were available it was not possible to evaluate the potential of students to deal with lower abstraction levels involving timing constraints and performance driven synthesis. However, there are very optimistic expectations regarding the future accomplishments of Computer Science students in both VLSI design and computer organization, as well as on other related courses.

One very important issue we are currently addressing is the adaptation of the approach to the courses following computer organization, computer architecture I and II. This is fundamental to keep students simultaneously in contact with both VLSI and computer design during the period before they are able to follow elective courses on the subject.

A problem that only now starts to be treated by the authors is the quantitative assessment of student performance and effectiveness of the approach, and this should receive increasing attention in the next editions of the courses.

Most of the material employed in the implemented courses, like case study specifications and design, lecture presentations, course notes and free software are available at the following URLs (in Portuguese):

- http://www.inf.pucrs.br/~moraes/Dorglab.html
- http://www.inf.pucrs.br/~calazans/org_comp.html

Material that is not freely available to students can be obtained by contacting the authors directly by e-mail.

Acknowledgements

The authors would like to acknowledge the continued support of Xilinx, Inc., which through its University Program gracefully provided the authors' institution with unlimited free software licenses and prototyping boards needed to make the full lab course implementation possible. Aldec, Inc. has provided us with two years of free licenses of its VHDL simulator. The authors are also grateful to the institutions CNPq (Research grants 522939/96-1 and 520091/96-5) and FAPERGS (Research grant 94/01340-3), which through the years have been funding the research work of the authors, making possible the developments reported here. The authors are in great debt with Daniel Liedke, a former student of the computer organization courses, who implemented assembler and simulator programs for the two case studies used in both courses.

References

[1] Guccione, S. List of FPGA-based Computing Machines. Available at: http://www.io.com/~guccione/HW_list.html.

[2] Gajski, D. and Kuhn, R.. New VLSI tools. Computer, 16(12):11-14, December 1983.

[3] Calazans, N. *Automated Logic Design of Sequential Digital Systems*, Chapter 1: Introduction. Imprinta Gráfica e Editora Ltda, Rio de janeiro, RJ, 1998. pp 1-42. (in Portuguese). Also available at ftp://ftp.inf.pucrs.br/pub/ calazans/pubs/prjlog/v1.0/Cap1.ps.

[4] Moraes, F., Calazans, N., Silva, F. and Barrios,M.. Cleo-LIRMM: um experimento de implemetação de processadores dedicados em plataformas de prototopação de sistemas embarcados, *In: V Workshop IBERCHIP, 01-03/03/1999, Lima (Peru), p. 81-90.*

[5] Maurer, P. Enhancing the Hardware Design Experience for Computer Engineers, In: 1998 *Frontiers in Education Conference*, Tempe, AR. Session T1E, pp. 60-63, November 1998. Available at: http://fairway.ecn.purdue.edu/~fie/.

[6] Rodríguez Pardo, L., Moure, M., Valdés, M. & Mandado, E. VISCP: a Virtual Instrumentation and CAD tool for Electronic Engineering Learning, In: 1998 *Frontiers in Education Conference*, Tempe, AR. Session S2B, pp. 1095-1099, November 1998. Available at: http://fairway.ecn.purdue.edu/ ~fie/.

[7] Grünbacher, H. Teaching Computer Architecture/Organization using Simulators, In: 1998 *Frontiers in Education Conference*, Tempe, AR. Session S2C, pp. 1107-1112, November 1998. Available at: http://fairway.ecn.purdue.edu/~fie/.

[8] Aldec Inc. Homepage: http://www.aldec.com/.

[9] Xilinx Inc. Homepage: http://www.xilinx.com/.

[10] Sentovich,E. et al. SIS: a system for sequential circuit synthesis. Technical report UCB/ERL M92/41. Universy of California, Berkeley, CA, May 1992. Available at http://www.cad.eecs.berkeley.edu/Respep/Research /vis/usrDoc.html.

[11] Xess Corporation. FPGA products: http://www.xess.com/FPGA/.