Digital Nano-CMOS VLSI Design Courses in Electrical and Computer Engineering Through Open-Source/Free Tools

Elias Kougianos[1], Saraju P. Mohanty[2], and Priyadarsan Patra[3]
NanoSystem Design Laboratory (NSDL), University of North Texas, Denton, TX, USA.[1,2]
Intel Architecture Group, Intel Corporation, USA.[3]
E-mail: eliask@unt.edu[1], saraju.mohanty@unt.edu[2], priyadarsan.patra@intel.com[3].

***Abstract*— Digital VLSI design courses are a standard component in most electrical and computer engineering curricula. Electronic Design Automation (EDA) or Computer Aided Design (CAD) tools and frameworks are an integral and indispensable part of such courses. In this paper we present our findings during the preparation and setup of such a course, centered around nanoscale CMOS, standard cell based design. Practical issues, such as the choice of licensing model, hardware and software platform selection, point tool identification and deployment as well as the availability of readily useable standard cell libraries are discussed. In addition, a design flow that incorporates the tools in a comprehensive framework is presented. A sample syllabus and a suggested teaching methodology are also given.**

I. Introduction

Digital VLSI design courses are an integral and well-established part of undergraduate electrical and computer engineering curricula. They are no longer considered specialized courses but are part of a well balanced program, on equal footing with other, older subjects such as Digital Signal Processing, Control Systems, Communications, Microprocessors etc. Even though all such courses rely on specialized software, digital VLSI design differs in that the required software is immensely more complex, difficult to master and expensive. Furthermore, it does not extend to other topics in the same way other software packages (for example, MATLAB) do.

To benefit a student, a digital VLSI course must provide, in addition to coverage of fundamental topics, early and continuous exposure to the various Electronic Design Automation (EDA) or Computer Aided Design (CAD) tools and frameworks that must be used to complete successfully even a small size project. Peculiar to this subject is the related topic of a *design flow* and how such a flow is achieved by judicious combination of point tools. Very important for a student's education, with implications to future employment, is the solid understanding of the entire design process and demonstrated (via a project) familiarity with EDA tools. For these reasons, a successful VLSI design course will include substantial exposure to such methodologies and tools.

Integrated circuit EDA tools are prohibitively expensive. A complete seat of tools (acquired through one license for front and back end IC design costs hundreds of thousands of dollars. The major EDA vendors have understood the need for undergraduate student exposure to such tools and provide reasonable, at first glance, packages at "nominal" cost. The intentions of these offers are noble but the actual implementation of these special academic programs leaves a lot to be desired. The "nominal" cost is of the order of $5K - $10K *per year*, but they do not come with any customer support. In addition, the computing and licensing servers (or platforms) in which they are parked are expensive. The skilled man power as a CAD support engineer (with system adminstration skills) needed to maintain such facility is quite expensive. For smaller schools even this cost is prohibitive and creates a serious discrepancy between the education of students from schools that can afford the software and those that cannot. In the opinion of the authors, a one-time perpetual license fee would be more reasonable. Worse still, usage of these tools is governed by very complicated legal agreements, which must be approved by each institution's legal department. This process can literally take months. Other problems with commercial EDA is the inability of students to run the software on their machines, the very selective distribution policies, lack of support, annual renewal hassles etc. Paradoxically, smaller EDA companies are even worse in their attitude towards educational institutions: either they outright demand full price for educational use of their products or their "academic" prices are ridiculously high compared to the large companies. The authors understand that EDA companies are private business entities with profits as their objective but catering *effectively* to students (who will be their future customers) is good business. The universally strong dislike of EDA companies in the IC design world, stems partially from this lack of endearing students, and their educators, in our opinion.

To resolve these problems in our courses and provide the ability to our students to experiment on their own, on their personal computers, while providing them with a meaningful educational experience, we investigated, with the help of an NSF grant, the possibility of using open source and/or free EDA tools in our courses. Open source is preferred, whenever available, to free. Free software commonly lucks the availability of source code and this presents dangers regarding the long-term viability of these projects. This paper presents our preliminary findings during the planning stage.

The rest of the paper is organized as follows: in Section II, we discuss hardware and software choices and give recommendations for setting up a VLSI laboratory. In Section III, we present a sample generic syllabus for a VLSI course. We examine open source/free tools and internet resources in

[0]This is supported in part by NSF grants CCLI-0942629 and CNS-0854182.

Section IV and show how design flows can be implemented using these resources in Section V. Our proposed teaching methodology is given in Section VI while conclusions and extensions are discussed in Section VII.

II. Hardware / Software Platform Alternatives

The selection of appropriate hardware and software platforms is, to a large extent, dictated by the choice of EDA and CAD tolls that will be used throughout the course. In the past these tools were primarily commercial (which are very expensive) and only available for a limited number of proprietary Unix workstations. Fortunately, with the advent of inexpensive x86-based personal computers and the open source movement, the situation has altered drastically and true, complex IC design can be performed on low-priced hardware with the software being mostly free.

On **the hardware side**, the selection of platform is simple: x86-based personal computers are used. Their ubiquitous presence, ready availability and low-price make them the obvious choice. The only question that needs to be addressed is whether the computational model will be client-server or client-only. Both approaches have their merits but, after consultation with departmental IT (Information Technology) support (i.e. Computer Technical Support), it was decided that the client-server model is most suitable in a class project environment. The main advantage is that only one machine (the server) needs to be custom configured in terms of operating system and software installations. In addition, centralized accounting management is possible on the server via a network login service. The Lightweight Directory Access Protocol (LDAP) was chosen because of its simplicity and the availability of a robust open source implementation (OpenLDAP, [1]). All student accounting information as well as their working storage is kept on the main server. This places, of course, the burden of maintaining backups on the server side. A very efficient approach to solving this issue is through the use of RAID arrays which address the problem of one or more hard disk failures while maintaining the integrity of data. In the lab dedicated to the course, RAID 1 (i.e. Redundant Array of Inexpensive Disks 1 or mirroring) was chosen as it offers very effective real-time backup with hardware failure tolerance and zero downtime. Other choices are RAID 5 and 6 (i.e. disk spanning with parity redundance) when the number of available hard disks is limited. RAID 6 should be preferred because it offers protection in the event of simultaneous failure of two disks, as opposed to RAID 5 which would experience complete data loss in such an event.

The selection of the client-server model has one downside: the hardware requirements on the server are substantial and depend on the number of students that are enrolled in the class. In our initial deployment, 35 students are enrolled which means that the server should be able to accommodate all these students simultaneously. The server used in this project is a four-CPU (Intel i7) computer with four cores each, 16 GB of RAM and 4 TB of RAID 1 storage. If a substantial server configuration is outside the financial resources of the department offering such a course, a workstation-only model can be used as described below. Also, the heavy communications between the clients and the server dictate that there should be several (ideally four or more) network ports on the server and the network connecting the clients to the server should be of gigabit speed.

The clients are x86-based personal computers configured as workstations. Since the performance requirements are much less severe than the server, a less powerful CPU configuration (even older Pentium IV class machines are acceptable) is required. 2-4 GB or RAM and approximately 250-500 GB of hard disk storage are sufficient. In addition, a gigabit ethernet connection should be available.

On **the software side**, several factors affect the choice of operating system including the following:

- Whether the environment will be client-server or workstation only.
- Whether there is a dedicated lab available or general access labs are used.
- Whether the software running on the server and the workstations will be 32 bit or 64 bit.

A. Client-Server Model

The client-server approach is preferred and is followed in this project due to the advantages inherent in this model. Since the vast majority of open source/free EDA/CAD tools have their roots in older, Unix-based projects, the operating system (OS) on the server side should be a variant of Unix. Due to the popularity of Linux, it was chosen as the server OS. A problem with Linux is the bewildering array of available distributions. Since enterprise class performance and stability are needed, the field of available choices is more narrow but some solutions are commercial and require annual maintenance contracts. The Community Enterprise Operating System (CentOS, [2]) version 5.5 was chosen due to its robustness, extremely wide support, frequent updating, and open source, no-cost availability. The 64-bit variant is used to take full advantage of the large amount of available RAM (16 GB).

The client side OS choice is more critical. There are two realistic alternatives: Microsoft Windows or Linux. Each alternative has its own advantages, disadvantages and challenges.

If a General Access Lab (GAL), shared among departments and colleges, is used by the students to complete the course project, then it is unlikely that Linux or dual-boot workstations are available and most probably the workstations are running a 32 or 64 bit variant of Microsoft Windows. In this case, the role of the workstation is reduced to that of a "dumb terminal" used to communicate with the server. All software, and the students' working storage resides on the server. A free X server, such as Xming [3] or free virtual network software, such as TightVNC [4] are used for this purpose. This scenario is viable but places resource strain on the server and the network. In addition, students can use their own laptops to connect to the server. This setup is shown in Fig. 1.

B. Workstation-Only Model

If a GAL is not available or if a lab is dedicated to the course projects but the cost and maintenance associated with a server

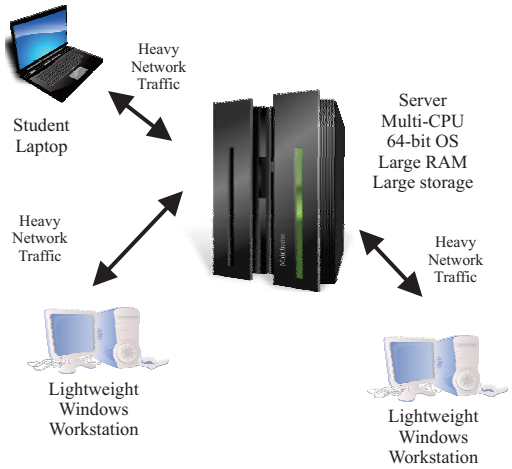

Fig. 1. Client-Server model of lab setup.

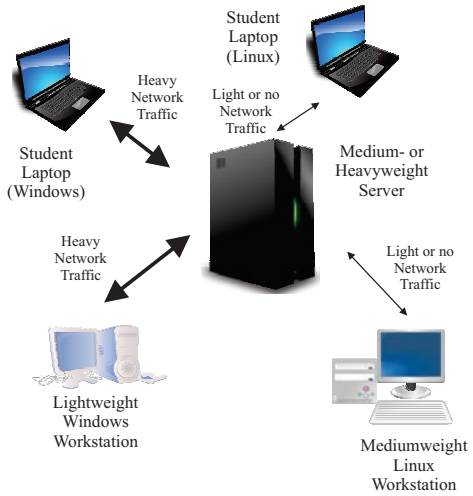

Fig. 2. Mixed-Mode model of lab setup.

cannot be justified, a workstation-only approach can be used. With the central server removed, each individual workstation must be responsible for user authentication, local working storage and tool execution. Since the tools will be running locally, a 32 or 64 bit (depending on available memory) version of Linux is installed as the OS. The workstation can be running CentOS or an EDA-specific version of Linux, the Fedora Electronic Lab [5]. To facilitate maintenance, a student is assigned a specific workstation for use throughout the course. Students are also encouraged to install the OS and tools on their personal laptops and use them instead of a workstation. Most students find the OS installation and configuration of their laptops in a dual-boot mode a valuable experience.

C. Mixed-Mode Model

The most flexible configuration arises when a powerful server is available and the course instructors have the ability to configure individual workstations (in a GAL or dedicated lab or both) to fit the course project requirements. In this usage model the server handles authentication and storage but the execution of the tools can take place locally, on the workstation, or remotely, on the server. The home directories of the students are also mounted, upon login, to the local workstation via NFS, or, if the workstation functions in "dumb terminal" mode, they can be on the server. Similarly to the workstation-only model, the local machines run a 32 or 64 bit version of Linux as the OS but Windows PCs with Xming or TightVNC can also be used. Student laptops can be part of the system after installation of the proper software. This model reduces the hardware requirements on the server at the expense of additional configuration and maintenance of the workstations. This setup is shown in figure 2. Because of its flexibility, this approach has been followed in this project.

III. A Sample Syllabus

We provide in this section a template for a syllabus targeted towards a standard-cell based digital VLSI design course. This syllabus is tentative and will be dynamically adjusted as we gain more experience with issues and requirements during actual delivery of the course. The course is scheduled to be offered in the Spring 2011 to an anticipated audience of 35-40 junior and senior electrical and computer engineering students.

A. Course Title and Credit Hours

Digital Nano-CMOS VLSI Design. 4 credit hours course that include 3 hours lecture and 3 hours laboratory.

B. Course Description

Introduction to digital VLSI standard cell-based design. Nano-CMOS device physics and technology. Simple gates (inverters, NAND, NOR). Flip-flops and memories. Introduction to layout. Simulation and characterization of gates. Standard cell libraries. Layout of standard cells. Standard cell characterization. Introduction to VHDL and synthesis. Synthesis of larger designs. Place, route and chip assembly. Students develop a standard cell library and complete a medium-size chip design.

C. Prerequisites

Digital Logic, Circuit Theory, Basic Electronics, Structured Programming.

D. Required and Optional Textbooks

Weste and Harris [6] or Rabaey, Chandrakasan and Nikolic [7] (required). Mohanty, Ranganathan, Kougianos and Patra [8] (optional). Sicard and Bendhia [9] (optional).

E. Course Objectives

Upon completion of the course, the student will be able to:

1) Understand nano-CMOS device principles and technology.
2) Model the nano-CMOS transistor for digital and analog applications.
3) Assemble nano-CMOS transistors into basic digital gates, such as inverters, NAND and NOR.

4) Perform the layout of simple gates for a given technology.
5) Simulate and characterize simple gates using an analog simulator.
6) Understand the concepts and methodologies involved in standard-cell design.
7) Design and characterize a library of standard cells.
8) Use a hardware description language, such as VHDL, to synthesize larger designs from standard cells.
9) Place, route and assemble into a chip a synthesized design.

F. Student Learning Outcomes

The students will:

1) Demonstrate proficiency in designing and analyzing digital CMOS circuits.
2) Incorporate digital gates into standard cells.
3) Demonstrate proficiency in layout of standard cells.
4) Demonstrate an adequate level in writing and synthesizing VHDL code.
5) Design a medium-size digital chip.
6) Perform functional verification of a medium-size digital chip.

IV. The Individual CAD Tools and Web Resources

The selection of the various open source and/or free point tools is dictated by the individual tasks that the students should be able to perform as part of the overall design flow as outlined in the following subsections.

A. Schematic Entry and Netlisting

The starting point in our design flow is a transistor level description of various cells, followed by analog simulation for functionality verification and characterization. An effective and programmable schematic entry tools is XCircuit [10]. It allows schematic entry and automatic hierarchical SPICE netlist generation. Additionally, the schematics can be exported in publication-quality postscript code which is useful when the students compose their project report. XCircuit is actively being developed with frequent releases and is part of most Linux distributions. Precompiled packages are also available but compilation from source code is straightforward.

B. Analog Simulation

SPICE is the standard analog simulator used in this course for gate and cell characterization. From the numerous variants available, ngspice [11] was chosen due to its active development, improved stability and support of the BSIM 4.6.5 model which is absolutely necessary for effective nano-CMOS transistor level simulation (including gate leakage). Equally important is the availability of SPICE models that reflect current nano-CMOS technology and process capabilities. In our course we use the Predictive Technology Model (PTM, [12], [13], [14], [15], [16]) which can be tailored to various technology nodes and processes (including carbon nanotubes). Postprocessing of the SPICE simulation results is done through the graphical waveform viewer GTKWave [17].

C. Layout

There are four major open source projects addressing the needs of IC layout: the venerable Magic [18], Toped [19], LayoutEditor [20] and graal, which is part of the Alliance VLSI toolset [21]. The choice is left on the individual instructor as long as the tool can export GDSII streams. In our course we use Magic and graal since they support LVS (Layout vs. Schematic) and DRC (Design Rule Checking).

D. Back-End

The tools described above will allow the student to design, simulate and layout a number of standard cells and thus create a standard cell library. To create a working chip from a library requires a set of tools that will take synthesizable VHDL and synthesize modules of a larger chip using the library. Place, route and chip assembly then follows to create a working chip. The back-end is the weakest link in open source EDA. Essentially there are two solutions but only in the form of integrated frameworks. They are the Alliance VLSI system [21] and the Electric VLSI system [22]. Electric is fully integrated which means that establishing a flow with individual tools is difficult or impossible. Parenthetically, in the integrated framework category we should mention Microwind [23] which is supported by two excellent textbooks [9], [24] but does not cover the back-end, is not open source and is Windows only.

On the other hand, the Alliance VLSI system consists of a large number (over 30) individual tools that cover complete front-to-back standard cell VLSI design. Specifically, these tools are capable of:

- Hierarchical layout with DRC and LVS.
- Graphical Finite State Machine (FSM) entry, minimization and synthesis.
- VHDL simulation (including customized C modules) with waveform viewer.
- VHDL to Register Transfer Level (RTL) synthesis and optimization.
- Standard cell placement and routing.
- Routing for pads.

These tools are tightly integrated into a complete back-end flow which is fully documented but also allows the instructor to customize it. Having the ability to manually execute and explore the various steps in this flow is of high educational value, as well. In addition, the Alliance VLSI system is part of many Linux distributions and comes in precompiled binary packages, as well as complete source code. Most importantly, Alliance provides a set of λ-based CMOS libraries including standard cells, memories and pads. Alliance has been field-proven with industrial designs such as a 400K transistor gigabit router [25] and an 875K superscalar microprocessor [26]. Additional scalable libraries are also available by other researchers [27]. The license is very liberal (GNU General Public License, GPL) and it allows even for fee-free commercial designs, an aspect which may be attractive to entrepreneurial students (or instructors!).

In our course we use the Alliance VLSI system for the back-end due to its completeness, easy manipulation of design flow and the availability of scalable standard cell libraries.

V. Design and Simulation Flow

The proposed design and simulation flow for the course projects is logically divided into two portions: front-end and back-end. In the following discussion, front-end means the completion of a standard cell, from specification to schematic to simulation and final **layout**. Therefore, in our flow, standard cell layout is part of the front-end. Back-end means the conversion of synthesizable VHDL to RTL and the place and route of the pre-characterized standard cells to implement the RTL. The design is completed with the addition of pad cells.

A. Front-End Flow

The front-end of the flow is presented in Fig. 3.

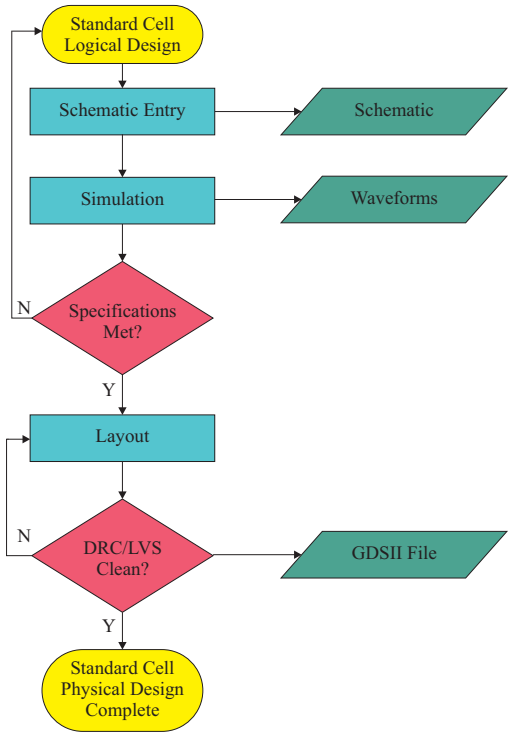

Fig. 3. Front-end of the design flow.

The entire flow starts with the logical (transistor-level) design of a standard cell. The cell can be very simple, with a small number of transistors, such as an inverter, or fairly complex, such as a 16-bit multiplier with transistor count in the hundreds. In all cases, the transistor count of the cell is not excessive and therefore analog simulation is feasible within very reasonable times. The schematic of the cell is entered through XCircuit [10] which also generates the netlist. Subsequently, the netlist is hand-edited to include the required SPICE analyses and any additional modifications needed. This step is of extreme educational value, giving the students an opportunity to see and edit the actual netlist (as opposed to working with a graphical editor that hides all the details) and provides them with a solid understanding of how SPICE works and what it expects as input. ngspice [11] is then invoked on the command line (another educational bonus) and any errors must be corrected by editing the *netlist* itself, unless the error is due to the logical design. The results of the simulation are then viewed with GTKWave [17] and if the design passes the functional verification tests and meets specifications, physical design (layout) follows with Magic [18] or graal [21]. At the end of this process the students have a functional, characterized library of standard cells and the back-end flow follows.

B. Back-End Flow

The back-end of the flow is presented in Fig. 4.

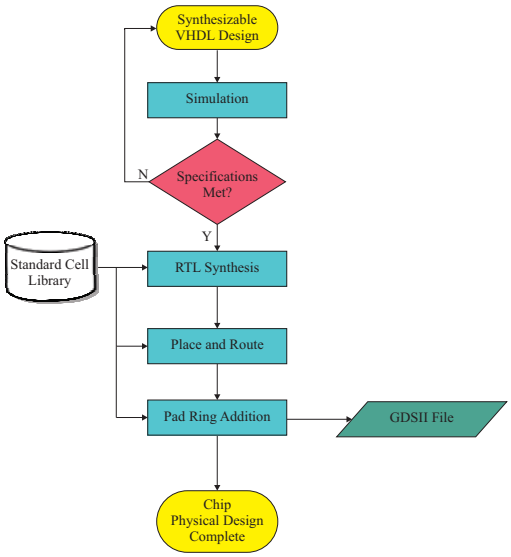

Fig. 4. Back-end of the design flow.

The entire back end is done using Alliance tools. First, a synthesizable VHDL description of the chip is derived. This step is typically broken into several substeps as the chip is gradually built from previously derived VHDL subsystems. For functional simulation any VHDL simulation tool can be used but we prefer to use ASIMUT which is the simulator provided in Alliance. The VHDL subset supported by Alliance is restrictive so if a design simulates correctly with ASIMUT, then the probability of its successful synthesis (within the Alliance framework) is high. Once the design has been verified, RTL synthesis follows. This step uses the previously generated standard cell library (derived in the front-end part of the flow) or it can use another library provided with Alliance or another researcher ([27], for example). RTL synthesis is followed by place and route of the standard cells and the chip is completed with the place and route of a pad ring. The final three steps (RTL synthesis, standard cell place and route, and pad ring place and route) are, as expected, of significant complexity and involve a large number of tools and intermediate formats. This process is extremely well documented in the Alliance help system [21] and allows a significant level of customization from the instructor or the students. The Alliance suggested flow also provides for intermediate step verification and validation so the confidence level for a working final design is high.

VI. Suggested Teaching Methodology

The conceptual framework of the course is straightforward: starting from first principles of operation of CMOS transistors,

students progress to gates, to layout, and then generate a standard cell library; using VHDL, they design a chip and implement it physically using the generated library.

Even though the description is simple, the scope is tremendous. The course can be taught in one semester in its entirety or it can be broken into a two-semester course sequence for deeper coverage of all topics. The deployment (one or two semesters) is up to the individual instructor, within the constraints of the relevant curriculum. We propose to follow both approaches with the one-semester version being taught in the Spring 2011 semester and report our findings in regards to the two approaches, in a future publication.

At an early stage of the course, the instructor will have to make a choice: whether to have the students create their own standard cell library or use one of the provided libraries. In a one-semester course, the latter option might be the only one viable, but a significant portion of educational experience will be lost in the process. In our first course offering we will attempt to cover both library and chip creation. In a two-semester course, library creation is a must. We have also incorporated a lab component to the course which is, in our opinion, indispensable. It is in the lab sessions that the students will learn the "mechanics" of translating an abstract course lecture into something that can be actually manufactured. It is also important that the lab be taught not just by a teaching assistant but that the instructor should be also present, at least for portion of the allocated lab period.

One of the advantages of using open source tools only is that the students can work on assignments and projects on their own personal computers or laptops. In fact, **this should be strongly encouraged**. The assignments should have a strong practical flavor with concrete deliverables (e.g. design an 8-bit register) that can be used to build the library and the chip.

The most important facet of this course is the project on which the students work throughout the semester. For a one-semester course, the concrete deliverable could be a final physical design of a chip while for a two-semester course the deliverables would include the library, in addition to the chip. The specific nature of the chip is left to the instructor and/or the students. It would be an excellent educational experience for the students to form a 2-3 person team and research possible chip candidates which would then be proposed for instructor approval. Common examples are CPUs, DSP cores (filters, FFT, DCT/IDCT), arithmetic cores (CRC generator, ALU, FPU), communications controllers (ethernet, UART, CAN, I2C, USB, IrDA), cryptographic hardware (AES, DES, RSA, SHA) etc. Many such examples, complete with VHDL code, can be found at the OpenCores web site [28].

VII. Conclusions and Extensions

We presented in this paper a conceptual organization of a digital, standard-cell based VLSI design flow using exclusively open source or free tools. The flow can be adapted to be part or the core of a one to two semester VLSI design course and produces a working chip design (and associated standard cell library) as concrete outcomes. The entire process is broken into a front-end and a back-end. The front-end uses traditional open source simulation and layout EDA tools while the back-end is part of the well-established open source Alliance VLSI system and allows for full customization.

Our overall objective is to demonstrate that high quality undergraduate education in VLSI is possible with open source or free tools and should be encouraged as part of the ethics training that is expected of modern engineering curricula.

It is our future plan to attempt this type of open source flow to semi-custom or even full-custom digital design and eventually to analog and mixed-signal IC design. There are many aspects of such flows that are missing from the open source tool repertoire but we hope that our work will stimulate further developments in this direction.

As a final note, the authors would like to invite commercial EDA companies to less restrictive (and cheaper) academic licensing which would make industrial-strength EDA training and education really accessible to **all** the students.

References

[1] The OpenLDAP project, http://www.openldap.org/.
[2] The Community Enterprise Operating System, http://www.centos.org/.
[3] Xming X server, http://www.straightrunning.com/XmingNotes/.
[4] TightVNC software, http://www.tightvnc.com/.
[5] The Fedora Electronic Lab, http://fedoraproject.org/wiki/Electronic_Lab.
[6] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design*, 4th ed. Addison-Wesley, 2011.
[7] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*, 2nd ed. Addison-Wesley, 2003.
[8] S. P. Mohanty, N. Ranganathan, E. Kougianos, and P. Patra, *Low-Power High-Level Synthesis for Nanoscale CMOS Circuits*. Springer, 2008.
[9] E. Sicard and S. D. Bendhia, *Basics of CMOS Cell Design*. McGraw-Hill, 2007.
[10] Xcircuit, http://opencircuitdesign.com/xcircuit/.
[11] Mixed - Mode Mixed - Level circuit simulator, http://ngspice.sourceforge.net/.
[12] Predictive Technology Model, http://ptm.asu.edu/.
[13] Y. Cao, et al., "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," in *Proceedings of the IEEE Custom Integrated Circuits Conference CICC*, 2000, pp. 201–204.
[14] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm early design exploration," *IEEE Transactions on Electron Devices*, vol. 53, no. 11, pp. 2816–2823, November 2006.
[15] A. Balijepalli, S. Sinha, and Y. Cao, "Compact modeling of carbon nanotube transistor for early stage process-design exploration," in *Proc. International Sympo. Low Power Electronics and Design*, 2007, pp. 2–7.
[16] Y. Cao, et al., "The predictive technology model in the late silicon era and beyond," *Foundations and Trends in Electronic Design Automation*, vol. 3, no. 4, pp. 305–401, 2009.
[17] GTKWave waveform viewer, http://gtkwave.sourceforge.net/.
[18] Magic VLSI layout tool, http://opencircuitdesign.com/magic/.
[19] Toped IC layout editor, http://www.toped.org.uk/.
[20] The LayoutEditor, http://www.layouteditor.net/.
[21] Alliance VLSI CAD, http://www-asim.lip6.fr/recherche/alliance/.
[22] Electric VLSI system, http://www.staticfreesoft.com/.
[23] Microwind, http://intranet-gei.insa-toulouse.fr/~sicard/microwind/.
[24] E. Sicard and S. D. Bendhia, *Advanced CMOS Cell Design*. McGraw-Hill, 2007.
[25] Z. Belkacem, R. Vincent, P. Frédéric, G. Alain, and D. Anne, "RCube: A gigabit serial link, low-latency adaptive router," in *Proceedings of the Hot Interconnects Symposium IV*, 1994.
[26] G. Alain, L. Luis, F. Wajsbürt, and W. Laurent, "Design of a high complexity superscalar microprocessor with the portable IDPS ASIC library," in *Proceedings of the European Design and Test Conference (EDAC-ETC-EUROASIC)*, 1994.
[27] VLSI and ASIC technology standard cell library, http://www.vlsitechnology.org/index.html.
[28] OpenCores, http://opencores.org/projects.