Teaching Processor Architecture with a VLSI Perspective

Mircea R. Stan
ECE Department
University of Virginia
Charlottesville, VA 22904
mircea@virginia.edu

Kevin Skadron
CS Department
University of Virginia
Charlottesville, VA 22904
skadron@cs.virginia.edu

Abstract—**This paper proposes a new approach to teaching computer architecture by placing an explicit emphasis on circuit and VLSI aspects. This approach has the potential to enhance the teaching of both architecture and VLSI classes, to improve collaboration between CS and ECE departments and to lead to a better understanding of the current difficulties faced by microprocessor designers in industry.**

Keywords: computer architecture, microprocessor design, VLSI design

I. INTRODUCTION

The teaching of computer architecture typically focuses on the interaction of instruction set architecture (ISA), instructions per clock cycle (IPC), and processor clock rate. Yet the circuit-design exigencies that profoundly impact the implementation of architecture-level concepts often receive little consideration. For example, the popular Hennessy and Patterson textbooks [1], [2] and others, despite their many strengths, have very limited information about logic and circuit issues. On the other hand, the VLSI and digital integrated circuit textbooks [3], [4] rarely consider the implications of their methods for microprocessor design at the architecture level. This division is often perpetuated by traditional academic boundaries. **In this paper we make the case that a new course is needed that crosses these boundaries and teaches computer architecture with an explicit VLSI perspective and vice-versa.**

A. *Why teaching computer architecture with a VLSI perspective*

Teaching computer architecture, as any other discipline, is different from school to school, but there have been attempts to unify it, either in an informal, grassroots way, *e.g.*, by the increased popularity of some textbooks that are widely adopted and dominate the field; or in a formal way by the different accreditation mechanisms, *e.g.*, ABET,

CSAB, and the creation and publication by IEEE/ACM of generic curricula for Computer Science and Engineering degrees. [1] In such a proposed curriculum, the main computer architecture concepts are covered in a "core" class, CS 220 - Computer Architecture, with more detailed microarchitecture and circuit issues being left to the non-core, "advanced" classes, *CS 320 - Advanced Computer Architecture* and *CS 323 - VLSI development.* We agree that not all students can, or should, learn all the details normally presented in these three classes, but we also think that it is important to teach the microarchitecture and VLSI aspects *together* for those students that elect to learn the advanced concepts and prepare for careers as microprocessor architects or circuit designers. In brief, we propose the creation of a combined class, *CS 320/323 - Advanced Computer Architecture: a VLSI Perspective*, see figure 1.

Such a class would be useful from many points of view. First, it breaks the artificial boundary between microarchitects and circuit designers. Both in industry and in academia, such differences clearly exist but are mostly detrimental. When architects do not have a good understanding of VLSI/circuit issues, they may take unwise decisions that penalize overall cost and performance; when circuit designers don't understand the overall architecture, they cannot fully take advantage of the degrees of freedom in design or exploit synergistic design choices across multiple levels of abstraction. A course like *CS 320/323 - Advanced Computer Architecture: a VLSI Perspective* would prepare students with a complex view of both architecture and circuit aspects.

Second, the class would also help as a bridge for academic programs in Computer Science, Computer Engineering and Electrical Engineering. A quick search of different existing classes and programs at different universities reveals that Computer Architecture classes are many times taught in both CS and ECE departments, with more of them on the CS side, while VLSI classes are mostly taught in ECE and EE departments, with few CS departments offering them. This is exactly the case at the Univer-

[1] http://www.computer.org/education/cc2001

sity of Virginia, where there are two classes in Computer Architecture, one in the CS, the other in the ECE department, but only one VLSI class, in the ECE department. A course like *CS 320/323 - Advanced Computer Architecture: a VLSI Perspective* would be equally attractive to both CS and ECE students and departments.

The third and final point is that such a class would bring new ideas and excitement into teaching both Computer Architecture and VLSI. While in industry the emphasis on circuit design aspects is clearly required for the high-performance microprocessors of today and tomorrow (as supported by the many publications at ISSCC and in JSSC), this trend is not yet fully reflected in the computer architecture classes being offered in academia. The situation with the VLSI classes is even more serious as very little progress has been made in the teaching VLSI since the seminal textbook by Mead and Connway. Even the newest VLSI textbooks still use the same bottom-up approach of first presenting device physics, followed by simple logic circuit design, combinational and sequential, followed by layout and finally a few case studies [3], [4]. Such an approach, quite successful in the past, has become slightly dated as it clearly targets "hard-core" Electrical and Computer Engineering students and is not interesting to most Computer Science students. Even the VLSI textbooks focusing on ASIC design are not appropriate for microprocessor designers, who need a balanced approach that combines both custom and semicustom design methods. A course like *CS 320/323 - Advanced Computer Architecture: a VLSI Perspective* would make both Computer Architecture, and especially VLSI design, more attractive to a wider spectrum of students and give them greater breadth of training.

II. COMPUTER ARCHITECTURE WITH A VLSI PERSPECTIVE: A BIRD'S EYE VIEW

The goal of the class is to give equal weight to both computer microarchitecture and circuit design aspects. In order to do this effectively the topics will be presented in parallel, with architecture concepts being used to provide a "natural" way to introduce VLSI and circuit design concepts. Accommodating both architecture and VLSI will necessarily entail sacrificing some material from traditional advanced-architecture and VLSI syllabi. Our philosophy is that with a sound training in fundamentals, the details are easily learned independently. For example, once the fundamentals of branch prediction and caching are understood, students can as needed teach themselves the various advanced branch-prediction and caching schemes, as well as variations like value prediction and prefetching. As another example, once the fundamentals of [MIRCEA:

VLSI].

To minimize the need for pre-requisites, the class will assume only a sophomore-level assembly-language and introductory computer-organization course as pre-requisite. CS 320/323 will start with a quick overview of Architecture ("Computer Architecture 101") and VLSI ("VLSI Design 101") to introduce the main ideas.

A. Overview: Processor Architecture

The overview of processor architecture topics will start with a classic, single-issue (scalar) processor. We plan to use a modern embedded processor example, like the Digital StrongArm, or its successor, the Intel XScale.This will include the basic operations of instruction fetch, instruction decode, register file access, integer and floating-point execution, and result writeback. In a generic fashion, we will also introduce the notions of pipelining and pipeline control, result forwarding, instruction and data caches, control and data hazards, exceptions, etc. The quantitative evaluation of performance through benchmarking and simulation will also be introduced here.

B. Overview: VLSI design

The overview of basic VLSI design concepts will start with a brief introduction of active device behavior and circuits, first at the switch level and only later with more detailed analysis and circuit-level modeling and simulation. Next we will touch on combinational vs. sequential logic and circuits, static vs. dynamic circuit concepts, and basic ideas of possible design flows, including custom, semicustom, and fully automated. We also briefly touch on the idea of a layout, and the corresponding CAD steps of floorplanning, placement and routing.

Following this quick introduction the course will get into more detailed discussion of each processor architecture topic and its "associated" VLSI circuit concept. There will be a clear attempt to present both architecture and circuit issues in a logical manner, in general by following the typical order of a processor pipeline for the architecture concepts, and associating the most important and natural circuit issue to the architecture, such that there are few or no repetitions and all the important aspects are covered.

III. INSTRUCTION FETCH AND DECODE: COMBINATIONAL LOGIC DESIGN

The classical processor pipeline starts with instruction fetch and decode, so it is natural to start our detailed treatment here as well. Since caches are used both for instruction and data, and since memory structures are not naturally best suited as a first introduction to circuit concepts,
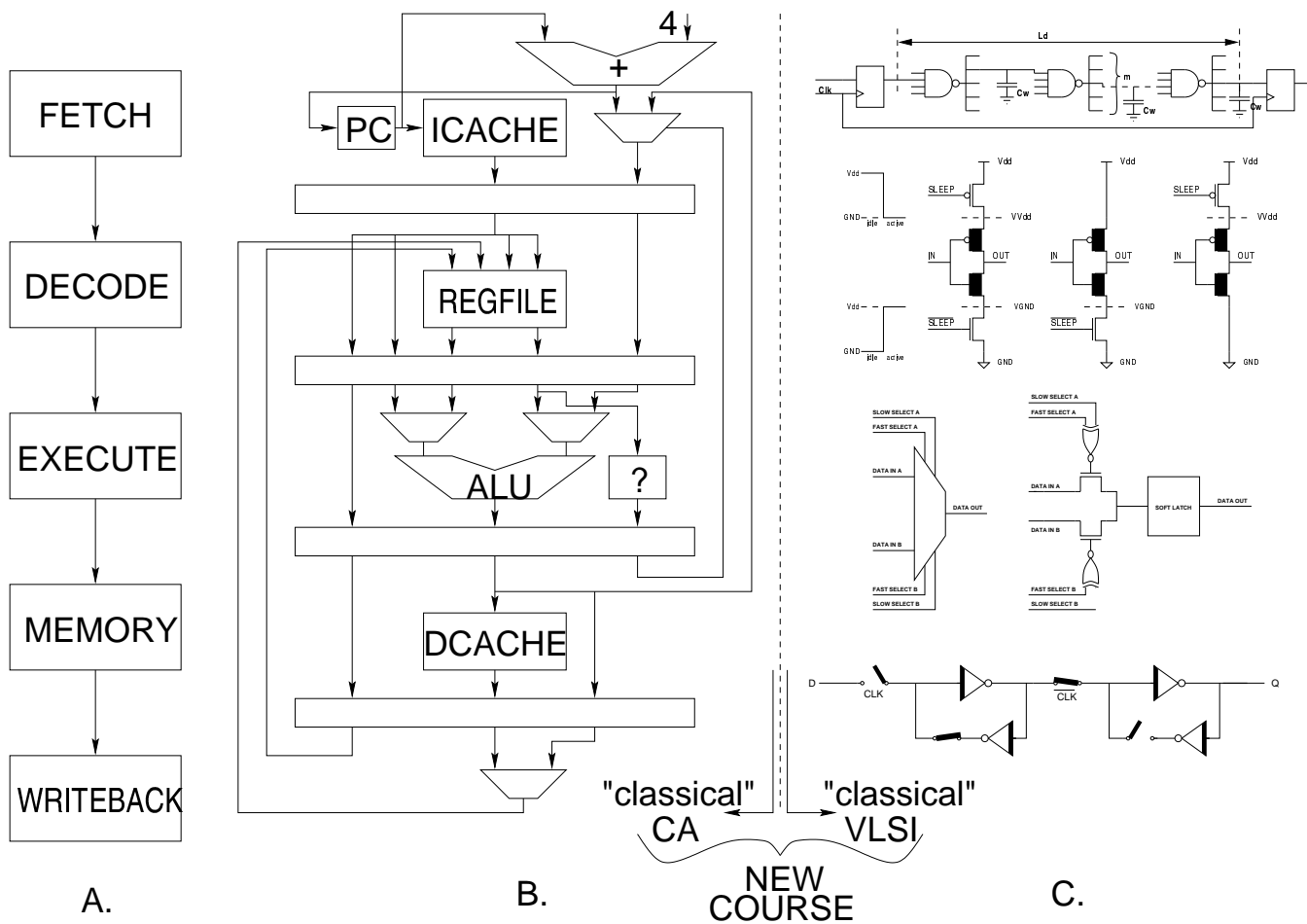
Fig. 1. New class on Computer Architecture with a VLSI perspective will combine elements from "classical" Computer Architecture classes and from "classical" VLSI design classes: A. typical 5-stage processor pipeline, B. typical simplified microarchitecture, C. typical VLSI concepts at the logic and circuit levels.

we postpone the actual discussion of VLSI concepts for memories to a later section.

A. Architecture: Instruction Fetch and Decode

Here we start with a quick discussion of instruction formats, the CISC vs. RISC debate, and how decoding a RISC instruction set is "easy" compared to a CISC. We will use MIPS as an example RISC architecture (we would have used Alpha, but now it is a "defunct" processor line) and x86 as an example of CISC, thus covering both extremes.

B. VLSI: Basic Combinational Logic

We can use decoding as a typical example of combinational logic circuits, and use it to illustrate the most important circuit design concepts. We start with simple static circuit techniques, including complementary static CMOS, pass-transistor and pass-gate logic, and show how they apply to simple logic gates with muxes and decoders

as a typical example. We then explain the advantages of complementary static CMOS (general applicability, robustness, regeneration of logic levels) as well as its disadvantages (size, suboptimal performance). We then show that other particular logic styles can outperform complementary CMOS in special cases, and exemplify with pass-transistor logic and pseudo-NMOS for muxes and decoders. We postpone the issue of dynamic combinational circuit design to a future section.

C. VLSI: Layout

Here we introduce layout techniques and the main figures of merit for VLSI circuits: performance (propagation delay), area (cost), power dissipation, reliability, robustness to noise, etc. We also introduce the notion of digital design as a trade-off among the possible figures of merit. We show simple bottom-up "polygon pushing" design steps.

IV. PIPELINING: SEQUENTIAL LOGIC DESIGN

One of the most effective ways to increase processor performance is to use pipelining of the various operations. This provides the perfect motivation for looking at the circuit design of sequential circuits.

A. Architecture: Pipelining

We first present the "classical" 4-stage and 5-stage pipelines, demonstrate the increased throughput that pipelining achieves, and explore the tradeoffs between latency and throughput for a pipelined processor. We follow up with more advanced concepts like superpipelining, and show the trade-offs due to an increase in the work per pipeline stage vs. overhead due to latch overhead.

B. VLSI: Floorplanning

The simple existence of a pipeline gives a level of regularity to the design that can be used for top-down floorplanning. Here we explain the importance of block adjacencies for reduced area (less routing) and increased performance (shorter wires).

C. VLSI: Synchronous Sequential Circuits

A pipeline is based on the overlap (in time) of the different functions; this overlap can be achieved with either synchronous or with asynchronous methods. Virtually all current processors are synchronous, so we start by explaining simple synchronous design concepts such as set-up and hold times and propagation delay, edge-triggered flip-flop vs. transparent latch vs. pulsed register, etc. We present simple static CMOS implementations of such flip-flops, registers and latches, then introduce dynamic logic, followed by dynamic versions of these state elements for higher performance but also higher power and less noise immunity. We exemplify with a few of the most important types of flip-flops used in several microprocessors, including TSPC, the Earle latch, etc.

D. VLSI: Clocking

The issues of clock generation, clock distribution and their influence on clock skew are explained. We explain the trade-offs for clock-spines, clock-planes, H-tree and X-tree clock distribution schemes, as well as the notions of centralized and distributed clocking schemes. Here we also discuss the issue of optimally driving large loads through the placement and sizing of buffers.

E. VLSI: Low-Power Design

We explain the differences between dynamic and static power, power consumption and power dissipation, etc.

More advanced concepts like time-borrowing and dynamic voltage/frequency scaling are also presented here, as well as clock-gating and other low-power methods. We also introduce the energy-delay product.

F. VLSI: Asynchronous Design

In order to provide a balanced view, we also present asynchronous design concepts such as micro-pipelines, wave pipelining, and "hybrid" methods such as globally-asynchronous, locally-synchronous approaches. We also give the (few) examples where such methods have actually made it into real commercial microprocessors (*e.g.*, wave-pipelining for address decoders).

V. EXECUTION UNITS: DATAPATH STRUCTURES

After instruction fetch and decode, the next step in a simple, scalar, processor is register read and execution. We postpone discussing register file issues to the next section and discuss execution units here.

A. Architecture: Integer Execution

Here we discuss briefly different issues related to integer datapaths, especially microarchitecture and logic level computer arithmetic algorithms, including addition, subtraction, multiplication, division and transcendental operations. Two's complement notation is introduced as part of this topic. We also briefly explain MMX and other signal-processing enhancement techniques for general-purpose processors.

B. Architecture: Floating-Point Execution

We follow the integer datapath issues with the more complex issues related to FP arithmetic, including data formats like IEEE.

C. VLSI: Datapath and Computer Arithmetic

Here we explore in more depth the differences between static and dynamic combinational logic circuits, with the higher performance of dynamic logic being widely used for datapath circuits. We then present different adder circuit styles (e.g. Kogge-Stone), multiplier circuit styles, shifter styles, etc.

D. VLSI: Placement

The VLSI structures presented in previous sections were more or less "random" logic. For datapath circuits there is an obvious one-dimensional regularity (the number of "bits") that can, and should, be exploited as bit-sliced design. Bit-slices are an example of regular placement of logic along one dimension. Here we also discuss

about custom and semicustom design methodologies and give examples of custom datapath design and semicustom standard-cell-based random logic.

VI. CACHES AND REGISTER FILES: MEMORY DESIGN

Finally we present caches and data-array structures. Caches are used for instructions and data, while data arrays are used for register files, queues, etc.

A. Architecture: Caches

We start by presenting issues related to cache associativity, first the two extremes, direct-mapped cache and fully-associative cache, followed by "in-between" cases like set-associative cache and CAM-RAM structures. We consider the issues of write-through vs. write-back, fills and write buffers. TLBs and generic buffers are other types of memory structures that are presented here. As advanced topics we present non-blocking caches and multi-level cache hierarchies.

B. Architecture: Register Files

For register files we start by presenting architectural registers and their implementation.We consider multi-porting as well as split-phase register access.

C. VLSI: Memories and Data Arrays

In order to implement memories and data arrays we present the main circuit building blocks. We start with the row and column decoders, followed by memory-cell design. Static/6T vs. dynamic/1T or 4T as well as word-lines and bitlines, precharging, for read and write are then discussed. Sense amp design and issues related to leakage and threshold wrap-up the design aspects. We follow by a brief discussion of defects, yield, and redundancy methods (spare rows and columns with reconfiguration) for increasing yield for memory structures.

D. VLSI: Routing

Physical design issues for memories are extremely important, in particular the issue of pitch-matching for the various subsections. This is an example of self-routing by abutment which shows the importance of regularity for VLSI design. General routing for "random" logic is a much more difficult problem.

VII. PIPELINE CONTROL: STATE MACHINES

A. Architecture: Pipeline Control

We first show how forwarding works and how the PC gets updated. We then introduce branch prediction and show how instructions get "squashed". As an advanced

topic we introduce multiple (in-order) issue-superscalar and the associated scoreboarding and contrast this with VLIW techniques.

B. VLSI: State Machines

Here we discuss difficulties of longer pipelines in terms of forwarding complexity and misprediction penalty. We introduce PLAs as an alternative for combinational logic implementation.

VIII. VLSI: INTERCONNECT, BUSSES AND I/O

We present major difficulties related to long interconnect, RC and RLC delay issues, and revisit buffer-insertion to reduce quadratic delay. We also show I/O design and system-interconnect issues, including the need for multi-voltage design.

IX. WHEN THINGS GO WRONG: EXCEPTIONS, VERIFICATION, TESTING

A. Architecture: Exceptions

An essential part of architecture is exception handling. We discuss precise vs. imprecise exceptions, explore the challenges of exception handling from the ISA level, and then proceed to describe the requisite hardware structures. We first present interrupt/trap hardware, supervisor mode, exceptions and trap vectors. We then trace the sequence of steps for syscall trap, I/O interrupt. For dealing with exceptions while already handling an exception we explain the need for interrupt masks, processor status word, etc. As an advanced topic we present the BIOS and describe the process of bootstrapping the computer.

B. VLSI: Verification, Testing, and Packaging

We explain the issues related to verification and validation (making sure that the design is correct) as well as to testing and built-in self test (BIST—making sure that a correct design is correctly fabricated). The notions of defects, faults and errors is explored in more detail. A brief overview of manufacturing, packaging, binning is also presented here.

C. VLSI: Power distribution

With reduced voltages and increasing power, the currents that need to be distributed on chip are increasing at an alarming rate. Here we discuss issues related to IR-drop, electromigration, and their influence on performance and reliability. We briefly mention aluminum and copper interconnect and SOI.

X. Out-of-Order Execution: VLSI Methodology

A. Architecture: Out-of-Order Execution, Register Renaming

Here we explain the benefits of out-of-order execution (OOE) and the need for renaming. We briefly describe basic OOE structures (register update unit vs. issue queues, etc.) as well as wakeup and select logic and renaming logic.

B. VLSI: Queues and VLSI Methodology

The issue queue has become one of the most complex structures in a modern out-of-order superscalar microprocessor. We choose the issue queue to do an in-depth analysis and exemplify with multiple case studies of real designs. We use this as a motivation for a look at different design methodology alternatives with their advantages and disadvantages.

XI. Conclusion

We have made the case for a class that teaches processor architecture with a VLSI perspective. We believe that such a class would have a strong impact in academia and will also better prepare students for jobs as either architects or circuit designers. We expect the class to be quite popular with a wide spectrum of students in CS and ECE departments. Since no current textbook uses this approach we also believe that there are significant opportunities for filling this void with a "new and improved" textbook that could be used, either for teaching Computer Architecture with a VLSI perspective, or, alternatively, for teaching VLSI for Computer Science students.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach, Second Edition*, Morgan Kaufmann Publishers, 1995, ISBN 1-55860-329-8.

[2] D. A. Patterson and J. L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface*, Morgan Kaufmann, San Mateo, 1993.

[3] Jan M. Rabaey and Massoud Pedram, Eds., *Low Power Design Methodologies*, Kluwer Academic Publishers, Boston, MA, 1996.

[4] Neil Weste and Kamran Eshraghian, Eds., *Principles of CMOS VLSI Design*, Addison Wesley, Reading, MA, 1993.