

# The Case for More Digital Logic in Computer Architecture

Mark E. Hoffman

Department of Computer Science and Interactive Digital Design  
Quinnipiac University  
275 Mount Carmel Avenue  
Hamden, CT 06518

mark.hoffman@quinnipiac.edu

## Abstract

New topics, most notably the World Wide Web, have put considerable pressure on the Computer Science curriculum. *Computing Curricula 2001* represents a consensus that topics in the core must be reduced to accommodate new topics as they emerge. Unfortunately, digital logic has been reduced to 1/3 its original coverage. We argue that more core coverage should be given to digital logic, and that it should be included in Computer Architecture, not Discrete Systems. Digital logic is fundamental theory necessary for all Computer Science graduates; it provides an indispensable link between theory and practice; and it demonstrates recurring concepts, most notably “levels of abstraction.” We propose a Computer Architecture course model that includes a hands-on lab to support the core hours of digital logic we consider appropriate.

*Keywords:* digital logic, computer architecture, hardware

## 1 Introduction

After the publication of the *Computing Curricula 1991* (CC1991, Turner 1991) several unanticipated topics emerged, such as object-oriented programming, human-computer interaction, graphics and multimedia, security and cryptography, and most notably, the World Wide Web. In 1996, five years after its publication, Allen Tucker (1996) remarked on the omission of these topics, and suggested “a more responsive process that would allow integration of new principles and artifacts soon after they emerge” rather than waiting the traditional decade. This exciting, yet frustrating, constantly changing character of Computer Science places a great deal of pressure on the curriculum. The task force that developed *Computing Curricula 2001* (CC2001) made a conscious decision to “reduce the required level of coverage in most areas so as to make room for new areas.” The decision is formalized as Principle 7: *The required body of knowledge must be made as small as possible* where it further states that “the best strategic approach is to *reduce* the number of topics in the required core so that it consists only of those topics for which

there is a broad consensus that the topic is essential to undergraduate degrees.” (Italics is in the original.)

As it applies to digital logic there is danger in the CC2001 “consensus” of creating a self-fulfilling prophecy: fewer graduates will be exposed to digital logic strengthening the consensus over time that digital logic is unnecessary. A recent report on the state of the Computer Organization and Architecture course found from a survey of approximately 80 faculty that “[n]early 2/3<sup>rd</sup> of those who responded teach the course outside of their primary area of interest” (Cassel and Kumar 2002). For small colleges we may conservatively assume the fraction is greater. And, although, it may be said many of those who teach outside of their primary area of interest “work harder to improve the course” (Gousie 2003), they typically look to a standard such as CC2001 for guidance. This is true for textbook authors as well as for publishers who want to appeal to the broadest possible audience.

In this paper we make the case that more digital logic than is specified by CC2001 should be in the core topics, and that it should be taught as part of a Computer Architecture course rather than Discrete Systems. Our case is based on the fact that digital logic is a fundamental theory that all Computer Science graduates

	CC1991	CC2001	Change
Total Core Hours	271 (283 <sup>1</sup> )	280	+3% (-1% <sup>1</sup> )
Computer Architecture Hours	59	36	-39%
Operating Systems Hours	31	18	-42%
Systems-Level Hours	90	54	-40%

**Table 1: CC1991 and CC2001 Total and Systems-Level Hours**

should know, as it provides a means to link the theory to a practical understanding of how computers operate, and it tangibly illustrates recurring concepts, such as levels of abstraction, that may be unclear from other areas, such as object-oriented programming. We make a case for more digital logic in the spirit of Ivanov (2002) who eloquently argues for the interdependence of hardware topics in the larger curriculum, and Krishnaprasad (2002) who suggests additional hours for digital logic and offers

---

Copyright ©2004, Australian Computer Society, Inc. This paper appeared at *Sixth Australasian Computing Education Conference (ACE2004)*, Dunedin, New Zealand. Conferences in Research and Practice in Information Technology, Vol. 30. Raymond Lister and Alison Young, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

---

<sup>1</sup> Includes optional *Introduction to Programming Languages* (Turner 1991).

<b>CC1991</b>	<p><b>AR4: Assembly Level Machine Organization (15 hours)</b></p> <ol style="list-style-type: none"> <li>1. Basic organization; von Neumann block diagram, data paths, control path, functional units (e.g., ALU, memory, registers), instruction cycle</li> <li>2. Instruction sets and types</li> <li>3. Assembly/machine language</li> <li>4. Addressing modes (e.g., direct, indirect, register, displacement, indexing)</li> <li>5. Control unit; instruction fetch and execution, operand fetch</li> <li>6. I/O and interrupts</li> <li>7. Hardware realization</li> </ol>
<b>CC2001</b>	<p><b>AR3: Assembly level machine organization (9 hours)</b></p> <ol style="list-style-type: none"> <li>1. Basic organization of the von Neumann machine</li> <li>2. Control unit; instruction fetch, decode, and execution</li> <li>3. Instruction sets and types (data manipulation, control, I/O)</li> <li>4. Assembly/machine language programming</li> <li>5. Instruction formats</li> <li>6. Addressing modes</li> <li>7. Subroutine call and return mechanisms</li> <li>8. I/O and interrupts</li> </ol> <p><b>AR5: Functional organization (7 hours)</b></p> <ol style="list-style-type: none"> <li>1. Implementation of simple datapaths</li> <li>2. Control unit: hardware realization vs. microprogrammed realization</li> <li>3. Instruction pipelining</li> <li>4. Introduction to instruction-level parallelism (ILP)</li> </ol>

**Table 2: CC1991 and CC2001 Assembly Language-Related Hours**

course models in support. In the next section we discuss the state of Computer Architecture in the Computer Science curriculum and digital logic within Computer Architecture. We consider the perceived threat to assembly language programming compared to digital logic; various proposed models for Computer Architecture courses; and the distribution of digital logic topics between Computer Architecture and Discrete Systems. In Section 3 we make the case for more digital logic in Computer Architecture. We propose a digital logic lab to accommodate the necessary coverage in a Computer Architecture course in Section 4. And we offer conclusions in Section 5.

## 2 Digital Logic in the Computer Science Curriculum

CC2001 contains a 280 hour core that may be interpreted as seven three-credit courses—assuming a typical three-credit course over a 15-week semester—although the exact packaging of the core topics as courses is left to individual institutions. This is approximately the same number of hours contained in CC1991: 271 hours, or 283 hours if the optional *Introduction to Programming Languages* is included (Turner 1991). New topics have been added to the core, such as 15 hours of Net-Centric Computing and 3 hours of Graphics and Visual Computing; others have been eliminated, such as Numerical and Symbolic Computations; and some have been adjusted, such as the reduction of Operating Systems (OS) hours to 18 from 31. Computer Architecture (CA) is also hard hit with a reduction from 59 hours in CC1991 to 36 hours in CC2001 leaving only 61% of the CC1991 core allotment. Taken together, CA and OS incur a 40% reduction in hours devoted to system-level courses. (See Table 1.)

Reaction to the reduction of Computer Architecture core hours has taken the form of conference papers and panels.

Ivanov hosted a panel at *CCSC-Northeast 2002* to discuss the role of hardware courses in the small college curriculum (Ivanov, Mallozzi, Nunna, and Stephan 2002); he further defended the relevance of hardware courses in the small college curriculum based on the importance of interdependencies between hardware and other courses (Ivanov 2002). Krishnaprasad (2002) also defended the relevance of hardware courses suggesting increases in particular core topics, such as AR1: Digital Logic and Digital Systems (from 6 hours to 12 hours). He offered two course models to implement his suggestions. At *CCSC-Northeast 2003* a panel discussed issues implementing the architecture, assembly language and operating systems components of CC2001 (Ford, Gousie, Kumar, Sattar, and Wilkens 2003). The greater part of discussion defended assembly language programming instruction, there was some discussion of digital logic, and almost no discussion of operating systems.

### 2.1 Assembly Language and Digital Logic in the Computer Architecture

It is instructive to look at exactly what is changed in CC2001 with respect to assembly language and digital logic; both are perceived as threatened by the new curriculum. Assembly language is included in the knowledge unit AR4: Assembly Level Machine Organization in CC1991 with 15 hours devoted to its coverage. In CC2001, Assembly Level Machine Organization is now core topic AR3 with 9 hours devoted to its coverage. However, on closer examination we find that AR5: Functional Organization with 7 hours allocated to its coverage includes topics such as implementation of simple datapaths and control unit that were included in AR4 in CC1991. Therefore, CC2001 actually devotes 16 hours to cover topics originally categorized as Assembly Level Machine Organization in CC1991. That is an overall increase of one hour—granted some new topics

<b>CC1991</b>	<p><b>AR1: Digital Logic (12 hours)</b></p> <ol style="list-style-type: none"> <li>1. Basic logic elements and switching theory; minimization of functions</li> <li>2. Propagation delays and hazards</li> <li>3. Technologies: types of flip-flops</li> <li>4. Devices (e.g., demultiplexers, multiplexers, decoders, encoders, adders, subtractors, comparators, shift registers, counters, PLD-type devices)</li> <li>5. Memories (e.g., ROM, PROM, EPROM, EAROM, RAM)</li> <li>6. Analysis and synthesis of synchronous circuits, asynchronous vs. synchronous circuits</li> </ol> <p><b>AR2: Digital Systems (6 hours)</b></p> <ol style="list-style-type: none"> <li>1. Register transfer notation, conditional and unconditional</li> <li>2. Algorithmic state machines, steering networks, load transfer signals</li> <li>3. Tristate and bus structures</li> <li>4. Iteration, top down/bottom up, divide and conquer</li> <li>5. Decomposition, trade-offs, economics</li> <li>6. Block diagrams, timing diagrams, transfer language</li> </ol>
<b>CC2001</b>	<p><b>AR1: Digital Logic and Digital Systems (6 hours)</b></p> <ol style="list-style-type: none"> <li>1. Overview and history of computer architecture</li> <li>2. Fundamental building blocks (logic gates, flip-flops, counters, registers, PLA)</li> <li>3. Logic expressions, minimization, sum of product forms</li> <li>4. Register transfer notation</li> <li>5. Physical considerations (gate delays, fan-in, fan-out)</li> </ol>

**Table 3: CC1991 and CC2001 Digital Logic-Related Hours**

such as instruction pipelining have been added. If we look closer at Table 2 we see that AR3 topics such as instruction sets and types, instruction formats, addressing modes, etc. (topics 3, 5-8) may reasonably be taught from an assembly/machine language perspective. Therefore, of the 12 topics listed, half directly benefit from the study of assembly language programming. That immediately allows for 8 hours devoted to its coverage. However, 8 hours is not adequate coverage to become a proficient assembly language programmer. Assembly language programming may be perceived as threatened because CC1991 includes 59 hours for Computer Architecture—approximately 20 hours short of the two full courses typically offered. In many cases the extra time was devoted to a course in assembly language programming. CC2001 does not afford the luxury.

In CC1991, AR1: Digital Logic is allocated 12 hours for coverage, and AR2: Digital Systems is allocated 6 hours making the total for both 18 hours. CC2001 combines both topics as AR1: Digital Logic and Digital Systems allocating only 6 hours for coverage. (See Table 3.) The new curriculum provides only one-third the time for coverage of both topics. And, as will be discussed further below, only 3 of the 6 remaining hours are included in the model Computer Architecture courses; 3 hours have been shifted to Discrete Systems. It is noteworthy that an overview and history of computer architecture has been added as an AR1 topic. Of the 23 hours removed from Computer Architecture, the disproportionate number of 12 hours, over half, are removed from Digital Logic and Digital Systems. Assembly Level Machine Organization with its related topic Functional Organization actually increase by one hour. It is clear that digital logic is at greater risk than assembly language programming. This leads one to ask the question: Why is there greater concern for salvaging assembly language programming than digital logic when it fairs better in CC2001?

An ITiCSE 2000 Working Group Report (Cassel *et al* 2000) found that for the CC1991 knowledge unit AR1:

Digital Logic almost everyone taught basic logic, whereas the remaining topics were only taught by a little over half, even though most felt moderately confident with the material. For AR2: Digital Systems fewer than half taught any topic, and confidence was lower than for digital logic. In contrast, for AR4: Assembly Level Machine Organization, all but two topics (hardware realizations, and microprogramming formats and coding) are taught by almost everyone with higher confidence than digital logic.

From these results we can see that digital logic and, in particular, digital systems, have weaker support than assembly level machine organization including assembly language programming. As the first rung of the ladder of software abstraction (Ford, Gousie, Kumar, Sattar, and Wilkens, 2003), assembly language appears to enjoy more support than the corresponding first rung of the hardware ladder of abstraction, digital logic. Ivanov (2002) sums up the general situation by stating that “many faculty consider hardware courses essentially useless.” However, he goes on to debunk the myth that “computer scientist = programmer;” a myth that favors assembly language programming regardless of its low level. In small colleges, where teaching loads are heavy, there are limited funds for hardware labs, and there is little room in the curriculum for hardware courses, whether hardware courses are considered “useless” is not really the problem—digital logic may be surgically removed without perceived damage and concern. CC2001 supports the notion by reducing by 2/3<sup>rd</sup> the number of core hours devoted to Digital Logic and Digital Systems, and suggesting that half, presumably the Digital Logic topics, be moved to a Discrete Systems course. It might be argued a result of the evolutionary nature of the course to drop topics as they lose currency: the physics of vacuum tubes in the 1950s giving way to transistor circuits in the 1960s and so on (Wolffe *et al* 2002). We do not believe that digital logic should

become the exclusive domain of computer and electrical engineers.

## 2.2 Computer Architecture Course Models

CC2001 offers two course models covering Computer Architecture topics: a single course CS220{c,s,t} *Computer Architecture* proposed for the compressed, systems-based, and topic-based approaches; and a two-course sequence CS221w *Architecture and Operating Systems* and CS222w *Architectures for Networking and Communications* proposed for the Web-based approach. Both CS220{c,s,t} and CS221w include 3 of 6 hours of AR1: Digital Logic and Digital Systems, where the other 3 hours are included in Discrete Systems (CS105 or CS115). Effectively, this model allots one week of lecture to Digital Logic and Digital Systems, with the prerequisite that Digital Logic has already been covered leaving time to cover Digital Systems topics only.

CC1991 (Turner 1991) does not offer course models, however, the number of hours (59) allocated to Computer Architecture requires more than one course, but somewhat less than two. Two-course organizations include various combinations of computer architecture, computer organization, assembly language programming and digital logic. In schools with engineering programs, a digital logic course might be offered as a computer or electrical engineering course taught by engineering faculty. Without engineering department support, the situation in small colleges is more problematic. CC1991 encourages the use of closed labs<sup>2</sup> to supplement lectures in Digital Logic and Digital Systems.

Two model curriculums have been proposed specifically designed for liberal arts colleges: one before CC1991 in 1986 (Gibbs and Tucker 1986) and the other following CC1991 in 1996 (Walker and Schneider 1996). The 1986 model includes CO1: *Principles of Computer Organization* as one of four core-level courses. The course is described as stressing “the hierarchical structure of computer architecture” and includes both digital logic and assembly language programming. A lab is considered a “must” for students to gain experience with computer organization. The 1996 model includes CS3: *Computer Organization and Architecture* as one of two intermediate-level courses. The course description includes the statement that “there is...the need to expose students at an early stage to lower-level abstractions such as digital logic, machine language,” and an introduction to digital logic is specifically mentioned as a course topic. A lab is not specifically mentioned, however, since CC1991, on which this model is fashioned, considers labs desirable, its importance may be inferred.

Krishnaprasad (2002) proposes two models to cover his suggested expansion of the CC2001 core hours in Computer Architecture: two required hardware courses, *Digital Logic and Computer Organization*, and *Computer Architecture* where each course includes a lab; and one

required hardware course for small colleges with a hardware lab. For the one-course model, Krishnaprasad proposes that virtually all expanded digital logic be covered in a Discrete Systems course “[t]o spread the coverage of hardware topics over several courses in the curriculum.”

## 2.3 The Role of Discrete Systems Courses

Prior to CC2001, a variety of distributions of digital logic topics into Discrete Systems were proposed, from no Boolean algebra or digital logic to virtually all digital logic topics. CC1991 includes Boolean algebra but no digital logic in its Discrete Mathematics requirement (Turner 1991). The 1986 liberal arts model includes a similar distribution (Gibbs and Tucker 1986). The 1996 liberal arts model (Walker and Schneider 1996) includes a rather radical departure from these models; instead of including Boolean algebra in MA1: *Discrete Mathematics*, it is included in the core-level (third-level) course CO2: *Foundations of Computing* that also includes complexity and recurrence among other theoretical topics. (It is interesting to note that this reverses the traditional order where Boolean algebra precedes, and serves as a foundation for, digital logic. CS3: *Computer Organization and Architecture* that includes digital logic is an intermediate-level (second-level) course that is presumably taken before CO2.) As noted above, Krishnaprasad (2002) recommends placing virtually all digital logic into Discrete Structures for the one-course model proposed for small college curricula.

CC2001 requires 43 core hours in Discrete Systems. Two model courses are proposed: a single 3-credit course CS115: *Discrete Structures for Computer Science* that includes 3 of 6 core hours of AR1: Digital Logic and Digital Systems (logic gates, flip-flops, etc.) omitting all 4 core hours allocated to Graphs and Trees; and a two-course model CS105: *Discrete Structures I* and CS106: *Discrete Structures II* where AR1: Digital Logic and Digital Systems is covered in the first course. Either CS115 or CS106 (the second course of the two-course model) is recommended as the prerequisite for Computer Architecture courses. If we look more closely at CS115, we find that 13 of 40 hours, almost 1/3 of the course, is devoted to DS2: Basic Logic (10 core hours) and AR1: Digital Logic and Digital Systems (3 core hours). As noted in the report concerning CS115, “[a]lthough such a strategy is workable, many institutions will prefer to use two courses to cover this material in greater depth.” We might reasonably conclude that Discrete Systems, although an attractive home for digital logic, is already oversubscribed, exacerbating the dilemma for small college programs. Another consideration that emerges is the perception that digital logic is theory only—a perception supported by its placement in Discrete Systems.

## 3 The Importance Digital Logic in Computer Architecture

From the preceding we see that digital logic is something of a “hot potato”—in a course taught largely by faculty outside their primary area of interest, only the most basic topics are consistently taught, in spite of moderate

<sup>2</sup> “A closed laboratory is a scheduled, structured, and supervised assignment that involves the use of computing hardware, software, and instrumentation for its completion. Students complete a closed lab by attending a scheduled session, usually 2-3 hours long, at a specific facility.” (Turner 1991)

confidence with the material. Does digital logic belong in Computer Architecture or Discrete Systems? Is it strictly theory, or is it of some practical value? Should it be abstracted away like vacuum tubes and transistors in favor of more modern topics? The “consensus” answer from CC2001 is that digital logic is theory relegated to Discrete Systems with only glancing reference in Computer Architecture. Of course, some prerequisite knowledge of basic logic is necessary from Discrete Systems to establish a theoretical foundation. However, we argue that digital logic has enormous practical value and belongs squarely in Computer Architecture.

We make this claim based on the following observations: computer hardware is fundamental to all computing, and digital logic is the fundamental theory on which hardware is built; digital logic creates a clear bridge from theory to practice that is easily demonstrated and may form the basis for greater understanding of other computing concepts; and digital logic conveniently illustrates many recurring concepts. A poignant place to begin is Pilgrim’s (1993) observation: “It is easy to forget that even small-scale integrated digital technology is new to the student.” Where as we who teach computer science may dismiss digital logic, or take it for granted, our students have no such luxury. Do we consider it so obvious to be unnecessary to teach? From the student’s perspective it is not so obvious.

Regardless of the topic, venerable like algorithms or newly-emergent like net-centric computing, somewhere there must be computer hardware to implement it. Computer hardware is fundamental to the discipline. But why, regardless of its privileged position, do students of computing need to understand hardware, and, in particular, digital logic? At a minimum it clarifies the mystery of the machine: a magical device that executes programs (Chua and Winton 1983). Gousie (Ford, Gousie, Kumar, Sattar, and Wilkens 2003) observes that “[k]nowing the inner workings of the computer gives students much better insight into their own programming, making them careful about memory usage, for example.” Kumar (Ford, Gousie, Kumar, Sattar, and Wilkens 2003) similarly comments that “[s]tudents need to know the low-level system details to understand the restrictions and limitations of programming.” Even CC2001 states that a *system-level perspective* that “encompass[es] an appreciation for the structure of computer systems and the processes involved in their construction and analysis” is a desirable characteristic of Computer Science graduates. So digital logic supports understanding of high-level topics like programming, but why not abstract it away as has been the case with vacuum tubes and transistors? Vacuum tubes and transistors are technologies that implement digital logic theory. It is appropriate that as technologies change, their inclusion is altered to represent the current standard. However, digital logic remains fundamental theory on which hardware is built independent of the implementing technology. Therefore, hardware, and digital logic, in particular, remain relevant topics for all students of computing.

As noted above, digital logic—a fundamental computing theory—may be implemented by effectively any reasonable technology. As such, it provides a convenient

bridge between theory and practice. CC2001 states that “appreciation of the interplay between theory and practice” is another desirable characteristic of Computer Science graduates. Tucker (1996) notes “[t]opics in the theory of computing need to be integrated with practical topics in the curriculum at all levels.” Digital logic provides integration of fundamental theory that may be tangibly demonstrated in a variety of practical ways. One widely supported way to integrate the interplay between theory and practice is a lab (Tucker 1996, CC1991, Turner 1991, Hartmanis 1992, Tomek 1981). Labs, particularly labs that include hands-on experience with breadboards or FPGA chips rather than simulators only, give students the opportunity to go beyond mathematical theory and their powers of visualization to see it in practice. They see the theory works, and they demonstrate it by building a device that performs a useful function, such as a 4-bit adder. Therefore, digital logic remains relevant as a means of exploring computing theory and practice.

CC1991 incorporates the notion of *recurring concepts* that “are significant ideas, concerns, principles and processes that help to unify an academic discipline” (Turner 1991). Knowledge units include a list of recurring concepts; for example, AR1: Digital Logic includes complexity of large problems, conceptual and formal models, consistency and completeness, levels of abstraction, ordering in space, ordering in time, reuse, and trade-offs and consequences. Ivanov (2002) makes a similar, if not more general, point in emphasizing the interdependence of hardware topics to the rest of the curriculum. So, on the strength of recurring concepts illustrated by digital logic, it is an important and necessary topic of study. But to make the point more clearly, we illustrate how digital logic effectively demonstrates one recurring concept: levels of abstraction. Logic gates capture the behavior of an individual logic function, such as AND, OR, or NOT. Moving down one level, these logic gates may be implemented by transistors, or some other suitable technology. Moving up, logic gates may be combined to implement a logic function or device such as a decoder or flip-flop. At this point it is simple to demonstrate how a single function may be realized by more than one configuration of logic gates, and how minimization may be used to find the configuration with the fewest gates. Moving up one more level, devices such as decoders, multiplexers and flip-flops may be use to implement modules such as registers and memory. This sequence of abstraction, aside from teaching the principles of digital logic and how complex hardware is built and tested modularly, also reinforces an object-oriented perspective popular in programming (Jipping, Marlowe, and Sherstov 2002, Neff and Sampath 2003). Where the concept of abstraction may be “abstract” when first encountered in a programming course, digital logic, especially when encountered in a lab, tangibly demonstrates it in an easily understood fashion.

Digital logic is fundamental theory necessary for anyone who studies computing. It provides a convenient and necessary bridge between theory and practice. And it illustrates many recurring concepts in an easily

demonstrated and understandable way. Regardless of how simple or intuitive we may consider the topic, it is new to most of our students. We should not discount the power of these simple ideas. For these reasons, we claim that digital logic is highly relevant and deserves more time in the curriculum beyond the 6 core hours devoted to it and Digital Systems, and that it belongs in Computer Architecture. Additionally, we believe that a digital logic lab is a necessity for making theory practical, and recurring concepts like “abstraction” obvious.

Beyond the “theoretical” argument, a future Computer Science graduate might ask: How will digital logic help me on the job? We could appeal to the breadth argument that it provides a broad-based understanding to frame practical problems, or that it provides a basis for understanding higher-level issues, such as program performance. And this has been the case, particularly for small colleges, where few graduates were headed to hardware careers. However, the proliferation of embedded systems and reconfigurable computing systems may change this. Embedded systems offer a wide range of complexity as they make their way into virtually every device, or “behind every wall” (Hadimioglu 2003). FPGA chip-based reconfigurable computing systems may be used to implement simple functionality that does not require general-purpose processors, and by-passes hardware designers. Programming language extensions are being developed that allow programmers to specify code segments to be implemented in hardware (Najjar *et al* 2003). Najjar *et al* point out that “because they typically lack circuit-design skills and knowledge of hardware description language, application programmers tend to find hardware design paradigms inaccessible.” Knowledge of digital logic plays a very practical role in providing future graduates with knowledge to produce a better design—what code segment should be implemented in hardware and an understanding of the performance tradeoffs. Therefore, from both the theoretical and practical perspectives, digital logic remains a necessary and relevant topic.

#### **4 Integrating More Digital Logic into Computer Architecture**

The concern that lead the CC2001 Task Force to find an appropriate balance of old with new topics is necessary and valid. And we agree that overall CC2001 is a good curriculum standard, however, we have argued that it does not include enough coverage of digital logic. As a matter of fact, we have argued that reducing the coverage to 1/3 of its original amount and moving half of the remaining coverage to Discrete Systems is misguided. What remedy can we offer? How much digital logic should be taught in a Computer Architecture course given the constraints of small colleges? How can additional hours be included in a course some argue is already “too big?”

We make the following recommendation based on our study of the literature and experience teaching digital logic in Computer Architecture at a small college. The CS220{c,s,t} *Computer Architecture* course model included in CC2001 is a worthy starting point. It includes a consensus of contemporary topics valuable to Computer

Science graduates. The syllabus provided includes topics 2-5 of AR1: Digital Logic and Digital Systems that if proportionally covered already implies that nearly 5 of 6 core hours specified be necessarily included in the course. We instead propose that at least 3 hours, or one week of a 15-week course, be devoted to in-class discussion of digital logic. Covering all 6 hours would be better where some of the hours may be integrated with other topics, or may be “stolen” from elective or contemporary architecture topics. We further suggest that overall 12 hours be devoted to digital logic where the remaining 9 hours are in a closed, hands-on lab. The lab should be either breadboard-based or FPGA chip-based. Recent papers describe a breadboard-based lab (Ivanov 2003) and an FPGA chip-based lab (Hoffman 2004) where the cost is small enough for budget-conscious departments at small colleges. We think the benefits are certainly worth the relatively small expense. These papers include examples of digital logic labs that are closely integrated with the course. We do not recommend the use of simulators only because it does not give students the practical experience of seeing a design work on a standalone device. Hands-on labs also benefit students with concrete learning styles (Stiller and LeBlanc 2003).

The course we propose provides adequate coverage of digital logic topics for students to understand the fundamental theory, relate it to what they have learned as mathematical theory in Discrete Systems, and make practical connections to how it is used to build a working computer. In that process, students are exposed to recurring concepts such as levels of abstraction that may be practically demonstrated as successively more complex circuits are built by abstracting building blocks at the next level. Hands-on labs provide the practical experience and extra time for students to learn. And, as an added benefit, students genuinely like the labs, and the balance it gives the course (Ivanov 2003, Hoffman 2004).

#### **5 Conclusion**

We have presented a case for including more digital logic in Computer Architecture courses. The consensus of CC2001 threatens the place of digital logic in the Computer Science curriculum, particularly its place in Computer Architecture. We believe that covering digital logic exclusively in Discrete System, as is implied by CC2001, gives the false impression that it is somewhat expendable theory, and may soon be removed completely from the curriculum. We have argued that digital logic is the fundamental theory of computer hardware and that all computing ultimately appeals it; that digital logic provides a valuable link to teach and bridge theory and practice; and that digital logic reinforces many recurring concepts that unify the discipline. In particular, the recurring concept “levels of abstraction” directly supports the general concept of computer layers, as well as, layers in a network protocol stack or operating system layers. Finally, we have proposed a way to integrate adequate digital logic coverage as an extension of CC2001 through the use of a hands-on lab that gives students concrete opportunities to learn.

## 6 Acknowledgements

The author thanks Michael Gousie, Haldun Hadimioglu, and Lubomir Ivanov for comments offered on drafts of this paper. The author also thanks the anonymous reviewers for their helpful comments.

## 7 References

- Cassel, L., Holliday, M., Kumar, D., Impagliazzo, J., Bolding, K., Pearson, M., Davies, J., Wolffe, G.S., and Yurcik, W. (2000): Distributed Expertise for Teaching Computer Organization and Architecture. *ITiCSE 2000 Working Group Reports*: 111-126.
- Cassel, L. and Kumar, D. (2002): A State of the Course Report: Computer Organization and Architecture. *ITiCSE '02*:175-177.
- CC1991 (1991): Association for Computing Machinery and Institute of Electrical and Electronics Engineers Computer Science Joint Task Force: *Computing Curricula 1991*.
- CC2001 (2001): Association for Computing Machinery and Institute of Electrical and Electronics Engineers Computer Science Joint Task Force: *Computing Curricula 2001*.
- Chua, Y.S. and Winton, C.N. (1983): Hardware Component of an Upper Level Computer Science Curriculum. *SIGCSE '83*: 36-40.
- Ford, F., Gousie, M., Kumar, A., Sattar, A. and Wilkens, L. (2003): Implementing the Architecture, Assembly Language, and Operating Systems Components of Curriculum 2001. *JCSC 18*(5): 118-122.
- Gibbs, N.E. and Tucker, A.B. (1986): A Model Curriculum for a Liberal Arts Degree in Computer Science. *Communications of the ACM 29*(3): 202-210.
- Gousie, M. (2003): Personal E-mail. Received July 8, 2003.
- Hadimioglu, H. (2003): Personal communication. Received July 18, 2003.
- Hartmanis, J. (1992): Computing the Future. *Communications of the ACM 35*(11): 30-40.
- Hoffman, M. (2004): An FPGA-Based Lab for Digital Logic in a Computer Architecture Course. (Submitted to *Ninth Annual Consortium For Computing Sciences in Colleges Northeastern Conference, Schenectady, New York, April 23-24, 2004*).
- Ivanov, L. (2002): Hardware Courses and the Undergraduate Computer Science Curriculum at Small Colleges. *JCSC 18*(3): 177-184.
- Ivanov, L., Mallozzi, J., Nunna, R., and Stephan, J. (2002): The Role of Hardware Courses in the Computer Science Curriculum in Small Colleges. *JCSC 17*(6): 164-166.
- Ivanov, L. (2003): A Hardware Lab for the Computer Organization Course at Small Colleges. *JCSC* (To be published).
- Jipping, M.J., Marlowe, S. and Sherstov, A. (2002): Using Java to Design and Test Hardware Circuits Over a Classroom Network. *SIGCSE '02*: 162-165.
- Krishnaprasad, S. (2002): Relevance of Computer Hardware Topics in Computer Science Curriculum. *JCSC 18*(2): 328-336.
- Najjar, W.A., Böhm, W., Draper, A., Hammes, J., Rinker, R., Beveridge, J.R., Chawathe, M. and Ross, C. (2003): High-Level Language Abstraction for Reconfigurable Computing. *Computer 36*(8): 63-69.
- Neff, N. and Sampath, G. (2003): An Object Framework for Teaching ALU Component Design in Architecture Courses. *JCSC 18*(5): 23-30.
- Pilgrim, R.A. (1993): Design and Construction of a Very Simple Computer (VSC): A Laboratory Project for an Undergraduate Computer Architecture Course. *ACM SIGCSE Bulletin 25*: 151-154.
- Stiller, E. and LeBlanc, C. (2003): Creating New Computer Science Curricula for the New Millennium. *JCSC 18*(5): 198-209.
- Tomek, I. (1981): HARD-Hardware Simulation in Education. *SIGCSE '81*: 268-270.
- Tucker, A.B. (1996): Strategic Directions in Computer Science. *ACM Computing Surveys 28*(4): 836-845.
- Turner, A.J. (1991): Computing Curricula 1991. *Communications of the ACM 34*(6): 69-84.
- Walker, H.M. and Schneider, G.M. (1996): A Revised Model Curriculum for a Liberal Arts Degree in Computer Science. *Communications of the ACM 39*(12): 85-95.
- Wolffe, G.S., Yurcik, W., Osborne, H. and Holliday, M.A. (2002): Teaching Computer Organization/Architecture with Limited Resources Using Simulators," *SIGCSE '02*: 176-180.