

*Enhancing the
Design Experience
For Computer
Engineers*

Peter M. Maurer

*Dept. of Computer Science & Engineering
University of South Florida
Tampa, FL 33620*

VCAPP Laboratory Technical Report ED-2

Enhancing the Hardware Design Experience for Computer Engineers

Peter M. Maurer
Department of Computer Science & Engineering
University of South Florida
Tampa, FL 33620

Abstract

People become engineers because they love to build things. Although real engineering is comprised of designing new things, maintaining old things and redesigning things, the engineering student gets very little exposure to these activities. The main reason for this is the time and expense necessary to create a realistic engineering environment in the educational laboratory. With the recent progress in design automation tools, it is now possible to revise the core computer engineering curriculum to include realistic design experiences in virtually all courses, even those that are not usually thought of as being laboratory courses. Recently, we have established a new instructional philosophy for our computer engineering program that states that each classroom topic will be reinforced with a realistic design experience. In a traditional hardware laboratory, this would be virtually impossible due to the time and expense involved in setting up such a laboratory. However, design automation tools have elevated hardware design to the level of software design, at least in terms of the complexity of the designs that can be created in a given amount of time. Because of this we can assign a large number of simple design exercises in the low-level courses, and a few more complex projects in the later courses. This paper describes the changes that we have made in three of our core courses, Logic Design, Computer Architecture, and Digital System Design. These changes are all based around a common set of tools that will be used throughout the curriculum. Several design exercises are described, along with a suggested sequence for the various exercises. We have developed several laboratory manuals that can be used with a typical textbook. In the references we provide pointers to web sites where our laboratory manuals and tools can be downloaded. We also provide suggestions for textbooks for each course, although we expect our materials to be useful with a wide variety of texts. We also report some of our experience in using these materials in the classroom, and the student reaction to these changes.

1 Introduction

Most people don't become engineers because they like to solve differential equations. They don't become engineers because they like calculus, or because they like reading textbooks, listening to lectures, or solving homework problems. People become

engineers because they enjoy making things. They enjoy putting things together and seeing them work. They want to dream something, and then turn that dream into a reality. Oddly enough, engineering education has little or nothing in common with real engineering. Most engineering students are mature enough to realize that diligence in the classroom is the price that one must pay for the privilege of pursuing one's dreams, but this is not really the issue. Because real engineering is concerned with designing new things, keeping old things running, and renovating old things, engineering students really ought to have some exposure to these activities.

Of course, this is nothing more than "Motherhood and Apple Pie." Everyone knows that engineering students ought to do more engineering. But these things take time. A real engineering project requires a full-time commitment, usually from many talented people, not just three or four hours a week from one inexperienced student. So we compromise. We teach our students the basic principles of engineering, and merely talk about the grand things that others have done. We talk about microprocessors, but are happy if our students can debounce a switch.

Time is the underlying principle. Real engineering takes time. The less experienced a person is, the more time it takes. The biggest engineering projects take so much time that they would be a lifetime project for a single person, so many people must combine their efforts to complete a large project. Even with that, engineers still sit at workstations at 3:00 AM consuming pizza and soda, while trying to make the next deadline. It would be nice to see this kind of commitment from a student, but we can hardly demand it.

However, the engineer sitting at that workstation is vastly more powerful than the engineer of yesterday. There are people who will tell you that "today's projects are so complex that they *cannot* be completed by one man." Do not believe them. Design technology has become so powerful, that today, a single engineer can complete a project that would have required an army of engineers only a decade ago. Design automation has empowered the engineer. This empowerment has led to ever larger and more complex projects, but it has also simplified the design of more modest projects. And as educators, this is precisely what we have been waiting for. There are many times have I wished I could ask my students to design a complex arithmetic circuit, or a sophisticated microprogrammed controller, or even a small computer. Design it not just on paper, but in the laboratory, so they could see it work. So they could test it and *make* it work. If you had the money, maybe you could do it. You could buy the equipment. You could find the laboratory space. You could staff the lab with experienced people. You could buy the parts and the necessary tools. And even with all of this, it would still be difficult for a student, or even a team of students to complete a large project in a single semester.

With design automation, all of this goes away. (Well, almost. Students still have to burn their fingers on power supplies once in a while.) Those obnoxious little five-gate projects that took all week in the laboratory can now be completed in a few microseconds (or somewhat longer depending on typing speed). Those massive 200-wordline PLAs now become tiny 200-line programs. When a test fails, it's because your design is faulty, not because your lab partner accidentally sat on your breadboard. And best of all, if the software is cheap enough, you can work at home on your own computer.

To be honest, using design tools in engineering courses is "an idea whose time has come." We're not exactly the first people to think of it. What we are proposing, is not just enhancing the existing computer engineering curriculum with a few simulation

exercises. We are proposing an undergraduate curriculum that has design as its central theme, and uses a uniform set of design automation tools to facilitate student design projects. Students will not only prove theorems, but verify their correctness through simulation. Students will not only learn that $10+10=100$, they will also be asked to design a binary adder. Students will not only design an instruction set, they will also run a *sort* program using their new instructions. The circuits won't be real, they will be simulations. Nevertheless, the engineering will be real. Today's engineers spend most of their time simulating their designs. In today's world, where the physical circuits are constructed by technicians or by automatic synthesis, it is the simulation experience that is real. It is the hardware laboratory that is artificial. This is a surprising and unexpected benefit from a program that was originally motivated by economics.

2 The Starting Point

The first step in revamping our program was to select a set of software tools that could be used in the various courses of our program. We had three alternatives. We could select a set of commercial tools, such as those provided by Cadence or Mentor corporations, we could select a set of textbooks with built-in software packages, or we could design our own tools. Under normal circumstances, creating our own tools would be unacceptably complicated. However, we already had a set of tools that had been created to support our research, and it would be a relatively straightforward process to adapt these tools for use in the classroom.

Ultimately, we chose to provide our own tools, for several reasons. Commercial tools tend to have relatively steep learning curves, and often must be installed on expensive high-performance hardware. Strict licensing requirements would restrict the usage of these tools to a few specified workstations. Students could not be supplied with a copy of the tools to run on their own computers. Furthermore, if one set of commercial tools didn't provide the requirements for all classes, it would be necessary to purchase more than one commercial package.

Using the tools supplied with a textbook would eliminate many of these problems, but we also wanted to have a uniform set of tools that would be used for several courses. This is difficult to accomplish with textbook-supplied software. Furthermore, many of the best textbooks don't come with software, and even if they did, switching from one textbook to another could possibly require revision of all homework and laboratory exercises. Given the large amount of software that we already had available, creating our own tools was the natural choice. Because we distribute our software free of charge, other institutions can reap the benefits of our development efforts. This is especially valuable for smaller institutions that have limited funds for laboratory resources.

The product of our efforts is a system known as WinFHDL, which allows students to design circuits at the logic level, and simulate them. A number of powerful components are included in this software, including high-level PLA and ROM programmers, a macro processor, schematic-capture tools, a special high-level language for controlling the simulation interface, and a number of wizards that assist students in creating complex specifications. This software is designed as a collection of plug-compatible components, and can easily be enhanced with new components and features. As mentioned above the software is free for the asking, at the web site mentioned in reference [1].

3 Core Curriculum Enhancements

Our curriculum changes affected six courses, Logic Design, Computer Architecture, Digital Systems Design, CMOS VLSI Design, FPGA Design, and Design Automation. Logic Design and Computer Architecture are conventional undergraduate courses, while Digital Systems Design, is a course in the design of sequential controllers for various types of electronic devices. The three courses CMOS VLSI Design, FPGA Design and Design Automation are elective courses, while the other three are part of our core curriculum. Because our locally developed software does not include a layout editor, CMOS VLSI Design is taught using conventional commercial tools. The same is true for FPGA design, which uses the software provided by the FPGA manufacturers. Design Automation, uses our tools, but focuses on the design principles of the tools themselves, rather on the usage of the tools. In the remainder of this section we will focus our attention on the enhancements to our core curriculum.

3.1 Logic Design

For logic design, our simulation exercises start with the introductory course material. Logic design traditionally begins with such things as the commutative and distributive laws, deMorgan's laws and other equally dry subjects. Although it is necessary to go through the proofs of these laws, students don't always have the intellectual maturity to appreciate why a proof works, or to understand what the proof implies about the real world. We have discovered that we can use simulation to motivate the basic laws of Boolean logic by allowing the student to "prove" the laws through simulation. The examples given in Figure 1 illustrate typical student exercises for this portion of the course. In these exercises, Boolean equations are expressed using "&" for AND, "|" for OR, and "~" for NOT. The remainder of the circuit descriptions should be self-explanatory. The notation is that used by the WinFHDL system. The circuit "comm" illustrates the commutative law for AND, while the "dist" circuit illustrates how AND distributes over OR. "Dm1" and "Dm2" illustrate deMorgan's laws. The student is asked to design these circuits and simulate them with all input combinations. The two outputs should be identical.

comm: circuit inputs a,b outputs eq1,eq2 equation eq1=a&b equation eq2=b&a endcircuit	dist: circuit inputs a,b,c outputs eq1,eq2 equation eq1=a&(b c) equation eq2=(a&b) (a&c) endcircuit
Dm1: circuit inputs a,b outputs eq1,eq2 equation eq1=~(a b) equation eq2=(~a)&(~b) endcircuit	Dm2: circuit inputs a,b outputs eq1,eq2 equation eq1=~(a&b) equation eq2=(~a) (~b) endcircuit

Figure 1. The Laws of Boolean Logic

Experiments of this nature are used for the first 2-3 weeks of the course, gradually being replaced by experiments of a more traditional nature. The initial experiments introduce the student to the coding of basic gates as well as the direct coding of Boolean equations. Although WinFHDL permits the direct coding of functional blocks such as multiplexers and registers, we require the students to learn how to construct these blocks using basic gates. Once the student has become familiar with the internal structure of a high-level block, we introduce direct coding of these components. Our general policy in teaching this course is to back up all lectures with laboratory exercises. Figure 2 shows the table of contents for our Logic Design Laboratory Manual. In most cases, these experiments would be assigned as homework rather than as a formal laboratory exercise. The required software is available on university computers and is available as a self-installing executable that the students can use on their own computers.

Table of Contents: Logic Design Lab Manual	
Experiment 1: Elementary Boolean Functions	Experiment 13: BCD Adders
Experiment 2: The Properties of Boolean Functions	Experiment 14: Comparators
Experiment 3: Theorems and Canonical Forms	Experiment 15: Decoders and Demultiplexers
Experiment 4: Two-Level Functions	Experiment 16: Encoders and Multiplexers
Experiment 5: Nand/Nor Implementations	Experiment 17: Hamming Codes
Experiment 6: Adders and Subtractors	Experiment 18: ROMs and PLAs
Experiment 7: Code Converters	Experiment 19: Flip-Flops
Experiment 8: Seven Segment Displays	Experiment 20: Sequential Circuits
Experiment 9: Multi-Level NAND/NOR Circuits	Experiment 21: Registers
Experiment 10: Ripple Carry Adders	Experiment 22: Counters
Experiment 11: Carry Lookahead	Experiment 23: Random Access Memories
Experiment 12: Adder/Subtractors	Experiment 24: Tristate Logic

Figure 2. Logic Design Lab Manual.

3.2 Computer Architecture

We teach computer architecture as a two-course sequence. The first course is a “how-to” principles course that teaches students the inner workings of a computer, while the second is a more traditional survey of existing computer architectures. The second course is intended to be primarily lecture, with little or no laboratory work. The first course, on the other hand, is intended to be design-intensive with several reasonably large projects assigned during the semester. These projects are intended to be a continuation of the projects presented in the Logic Design course, but are more extensive and more difficult. Xxx gives the flow chart of the projects that we use during this course. Because some of these projects are extremely time-consuming, we assign only a subset of these projects to our students. We have not yet designed the last three projects: “Microprogrammed Computer,” “Hardwired Computer,” and “Modifying a Computer,” although we have plans to do so within the next year.

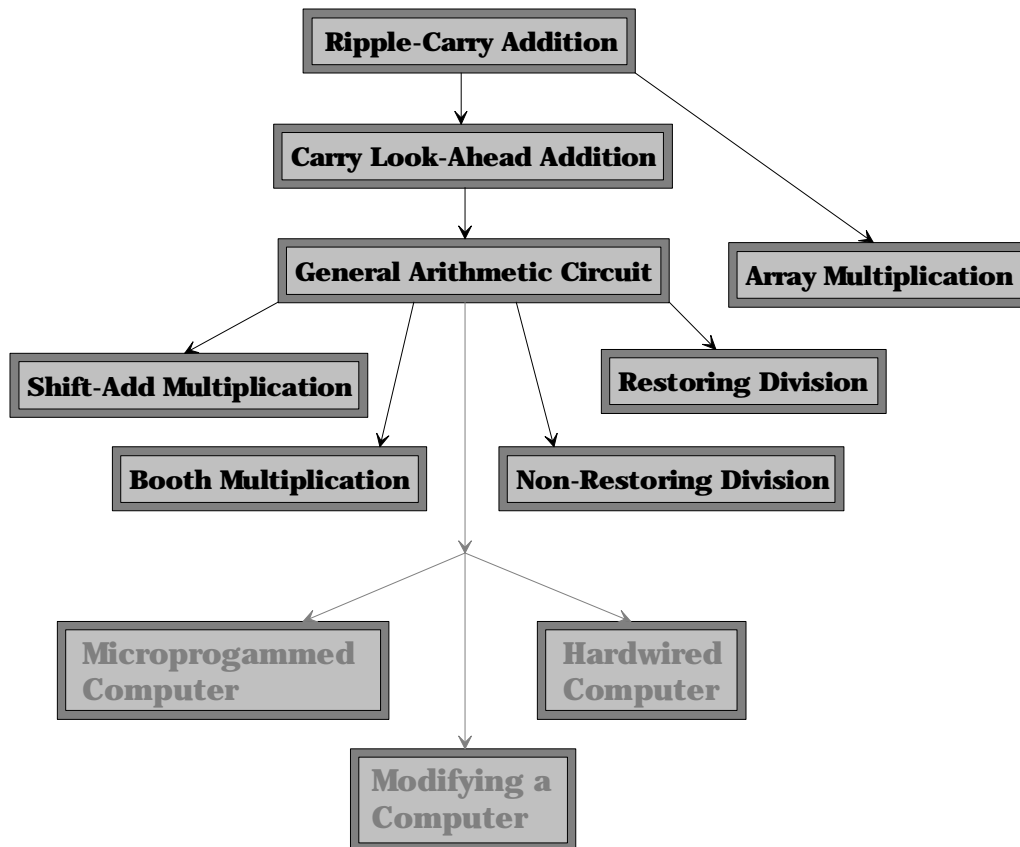


Figure 3. Projects for Computer Architecture.

The projects starting with “General Arithmetic Circuit” are significantly more difficult for the student than the “Ripple-Carry Addition,” “Carry Lookahead Addition,” and “Array Multiplication.” Starting with the “General Arithmetic Circuit” we expect students to create micro-programmed arithmetic circuits. Although such circuits are no more difficult to create than their combinational counterparts, there is something about sequential logic that represents a major shift in focus for the student. We have found it necessary to provide extensive guidance with these experiments, and in the future, will probably provide some prewritten segments for the later experiments. Originally it was our intent for the student to complete the “General Arithmetic Circuit” experiment, and then use the results for other experiments, but this turned out not to be practical.

Although Logic Design is a prerequisite for this course, it is a good idea to spend some time at the beginning of the course on a quick review of this material. To simplify this process, we break the first experiment “Ripple Carry Addition” into several segments. The first objective is to create a half-adder. The student constructs this circuit, tests it and is required to turn in the results. Then the half-adder is used to create a full adder, with similar requirements. These two experiments can be assigned the first week of the course. The full-adder is then used to create a 4-bit adder, which is then used to construct a 16-bit adder.

The later experiments are broken down into similar steps, but we do not require the student to hand in the intermediate results. At a minimum, the student should complete two combinational circuits on the scale of a 16-bit carry look ahead adder, and two sequential circuits on the scale of shift-add multiplication.

3.3 Digital Systems Design

Digital Systems Design is designed to teach the student how to build sequential control circuits for various different types of devices. For example, one of the first projects we have the student perform is to design a controller for a traffic light, with in-pavement sensors. We teach three different types of control, hardwired or state-machine control, microprogrammed control, and microprocessor-based control.

Many of the features of WinFHDH were incorporated specifically to support this course. Specifically, WinFHDH provides for direct simulation of algorithmic state machines, simulation of chip-based designs using standard off-the-shelf chips, microprogrammed ROMs using sophisticated programming and branching techniques, and for the inclusion of substantial amounts of prewritten code. These features were included specifically to support our Digital Systems Design course.

In the first portion of the course we concentrate on simpler designs using state-machine based hardwired control. The students must complete two or three projects on the scale of the traffic light controller. They must use the WinFHDH system to verify the correctness of their designs. Other types of devices that may be included in this segment include a key-pad based digital lock, and soft-drink machine.

The second portion of the course focuses on microprogrammed controls. The projects in this section are more extensive, and are on the scale of automated wiring using stepping motors, and automated railway signal light control. This segment of the course can be integrated with the hardware laboratory to provide a more realistic hardware-design experience. Exercises are based on an off-the-shelf microcode sequencer, and a standard ROM layout that uses standard opcodes and standard microinstruction formats. The microinstruction formats are flexible enough to be used in most projects. The ultimate project in this segment of the course is the design and implementation of a computer on the scale of the PDP-8.

The third portion of the course concentrates on embedded microprocessor control. This portion of the course is far less extensive than the other two, because this is essentially a mini-course in applied assembly-language programming. The designs that can be completed in this section of the course are far more powerful than the simple hardwired and microprogrammed designs, but are correspondingly harder to simulate. Simulating such designs requires simulating an entire microprocessor. This is beyond the scope of the WinFHDH project. In this segment of the course we concentrate on paper exercises, and hardware laboratory exercises using specialized development hardware.

4 Educational Materials

Our enhancements to our core curriculum are designed as laboratory exercises that will be used to enhance existing courses. For this reason, we have written, or are in the process of writing laboratory manuals to be used with existing courses. The most extensive manual available so far is that for the logic design course, which is available on the web site mentioned in reference [2]. This website also contains working versions of our other laboratory manuals.

We assume that Logic Design will be taught using a book such as [3] or [4], that Computer Architecture will be taught from a book such as [5] or [6], and Digital Systems Design will be taught from a book such as [7] or [8].

5 Conclusion

Despite the extensive work involved, introducing these changes into our curriculum was a pleasant and rewarding experience. In particular, I remember the first extensive experiment with the Computer Architecture course, namely the construction of a 16-bit two-level carry lookahead adder. This was the first experience that most of these students had at designing circuits of this magnitude and complexity. Most students were surprised that the circuits didn't work correctly the first time. Paper exercises tend to reinforce the view that hardware design is always easy, and error-free. Our laboratory exercises wasted no time in dispelling that notion. The biggest surprise that many of our students got was that their designs didn't pass when they turned them in. The common complaint was "Well, I ran lots and lots of tests and they all worked. How can you say this is broken." Such comments open the door for an interesting and diverting lecture on the science of testing.

The changes we have made in our curriculum elevate hardware design to the level of software design. No longer is hardware design limited by the expense of laboratory materials or the time required to complete the experiments. It is possible to assign large-scale design projects at the undergraduate level, and produce computer engineers who are skilled in their art and well prepared for graduate school or rewarding careers in industry.

6 References

1. Maurer, Peter, The FHDL home page, <http://www.csee.usf.edu/~maurer/fhdl.html>.
2. Maurer, Peter, The FHDL home page, <http://www.csee.usf.edu/~maurer/courses.html#UGLogic>.
3. Katz, R. *Contemporary Logic Design*, The Benjamin/Cummings Publishing Company Inc., New York, 1994. (ISBN 0-8053-2703-7)
4. Nelson, V., Nagle, H., Carroll, B. Irwin, J. *Digital Logic Circuit Analysis & Design*, Prentice Hall, Englewood Cliffs, NJ, 1995. (ISBN 0-13-463894-8)
5. Patterson, D. and Hennessy, J., *Computer Organization and Design: The Hardware/Software Interface*, Morgan Kaufmann Publishers Inc., San Francisco, 1998. (ISBN 1-55860-428-6)
6. Sima, D., Fountain, T., Kacsuk, P., *Advanced Computer Architectures: A Design Space Approach*, Addison-Wesley, New York, 1997. (ISBN 0-201-42291-3)
7. Comer, D., *Digital Logic and State Machine Design*, Saunders College Publishing, New York, 1995. (ISBN 0-03-094904-1)
8. Prosser, F, Winkel, D. *The Art of Digital Design*, Prentice Hall Inc. Englewood Cliffs, NJ, 1987. (ISBN 0-13-046780-4)