

Design Automation  
Principles  
An Essential Part of an  
Engineer's Education  
*Peter M. Maurer*

Technical Report ED-1, 1997  
VCAPP Laboratory  
Dept. of Computer Sci. & Eng.  
University of South Florida  
Tampa, FL 33620

# Design Automation Principles: An Essential Part of an Engineer's Education

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

## Abstract

The past two decades have seen an explosion in engineering technology, due in large part to design automation tools. It is obvious that engineers need to learn to use these tools, and most engineering schools have included some introduction to design automation in their curricula. We would insist that it is equally important for engineers to learn *the principles* upon which these tools are based. Such knowledge allows engineers to play an active role in the development of new tools, and to formulate realistic expectations about future developments in design automation. Because different fields of engineering require radically different types of tools, we have concentrated our efforts in the area of electronic design automation. We identify three major areas of electrical design automation, physical design automation, simulation, and high-level synthesis. These choices were made based on the importance of the areas, and the availability of educational materials. We have developed course requirements for each area, and provide a list of important topics within each area. For each of the areas, we identify textbooks and other educational materials. We also provide recommendations for laboratory exercises, and provide pointers to course materials on the world wide web. At our institution, we teach a single course in design automation, with the major topic changing from year-to-year. While this style of course offering meets our needs, we recognize that other institutions may wish to offer a single comprehensive course in the area. To this end, we have formulated requirements for a single survey course in design automation, with topics taken from the three major areas. At the present time, we provide sufficient materials for a simulation course in design automation. This includes several handouts which can be combined into a textbook, laboratory exercises, some overheads, and an extensive software package that can be used as a basis for design projects. These materials are under continual development, and will eventually be expanded into other areas of electrical design automation. This paper and the references to which it points, can give educators a place to start in developing their own courses in design automation.

## 1. Introduction.

Every year the explosion in the computer industry continues as one technological barrier after another is surpassed. Every year “the experts” predict that the limits of current technology will be reached “soon.” Every year the experts are proven wrong. Indeed, even the rate of change is accelerating every year, with one revolutionary new technology being introduced on the heels of another. A new generation of processors becomes available before the preceding generation can be fully exploited. A hard disk

that was considered top-of-the-line just two years ago is now obsolete and unavailable. The technological revolution is not confined to the computer industry. The computer industry is only the most visible tip of a universal explosion in technology that is occurring in virtually every field of engineering. Complex electronic devices of all kinds are becoming available at lower and lower prices. Today's video camera will certainly be obsolete within a few months, and be replaced by a device costing far less than today's model. Similar stories could be told about advances in materials sciences, improvements in aircraft design, new developments in flexible manufacturing systems, and in virtually every other type of engineering design.

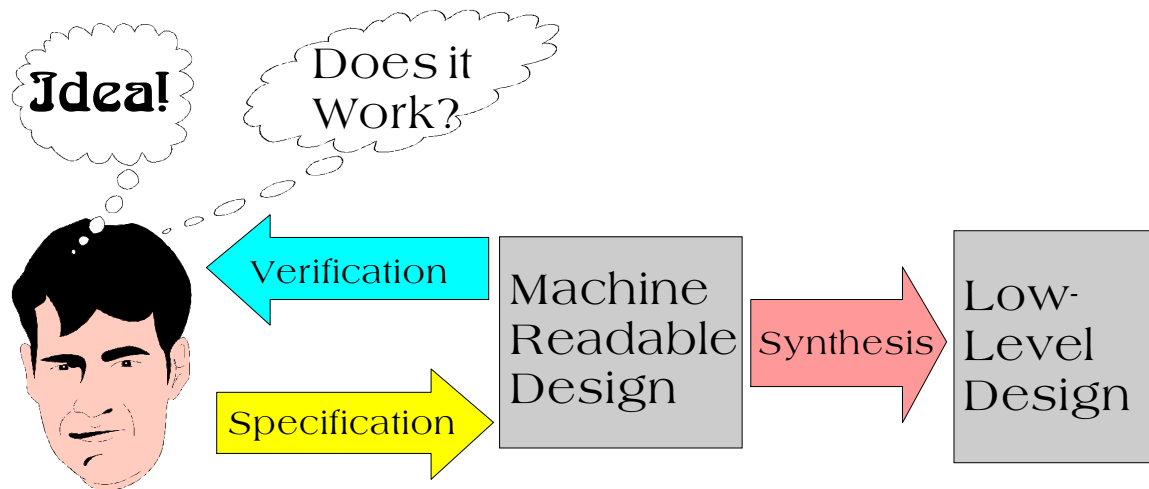
Although one cannot discount the creativity and genius of our engineers, the essential ingredient in this revolution is Design Automation. The scope and quality of today's design tools has enabled new designs to be created and tested in increasingly shorter periods of time. The time required to create new designs has plummeted, and at the same time the complexity of today's designs surpasses what was possible only a few years ago.

Design Automation is such an essential part of design at all levels, that no engineer's education can be considered complete without some exposure to modern design tools. The engineering-education community has responded well to this challenge, however simple exposure to new tools is not enough. It is absolutely necessary that students learn how these tools are constructed. It is necessary because students must not only know what can be done today, they also must appreciate what can be accomplished in the future. Without such knowledge, there is no way they can participate in the development of the very tools they will need to do their job in the future. Without such knowledge, the new engineer will tend to focus on improving the minute details of the current design process instead of focusing on the over-all design process. To quote a supervisor of a Bell Laboratories design automation group, "Creating new design tools is like finding a man pounding nails with a rock and asking him how you can help. Invariably the answer will be, 'Find me a bigger rock!'" The engineer who is not familiar with the power and potential of design automation is liable to ask for bigger rocks, while the properly educated engineer will be wise enough to ask for a hammer.

## **2. Rocks, Hammers, and Sky Hooks**

Different types of engineering require vastly different types of design tools. So much so, that there is virtually no common ground between design automation for electrical engineering (say) and design automation for mechanical engineering. Before dealing with specific issues, it is necessary to restrict the focus to a single area of engineering. Because of the focus of our department and research, we will concentrate on the area of computer engineering, and to a lesser extent, electrical engineering.

Design automation of electrical devices, or ECAD as it is usually called, can be broken down into several distinct areas, the most important of which are Physical Design Automation, Simulation and Testing, and High-Level Synthesis. These areas fit into the general framework illustrated in Figure 1.



**Figure 1. Tools and the Design Process.**

The three major tasks in designing electronic circuits are specifying the design in machine-readable form, verifying the correctness of design specifications, and automatically synthesizing new designs from old designs. Specification is the least automated of these tasks. It is generally facilitated through the use of schematic editors, and other tools, but it remains a largely manual process. Testing and design verification are the designer's most time consuming tasks. This is understandable, since errors in the machine-readable specifications lead to costly errors in the final product. Although it is technically an optional part of the design process, automatic synthesis has become an essential tool in creating complex electronic circuits. Synthesis is the process of automatically translating high-level design specifications into low-level structures. Some synthesis algorithms translate gate-level designs into silicon-based schematics. Others translate high-level language into registers and microcode. Synthesis is one of the most interesting and challenging areas of design automation.

Each of the three areas mentioned above is rich enough to provide the content for a one-semester course at the undergraduate level, although it is a good idea to include some topics from all three areas. Physical design automation focuses on algorithms for synthesizing layout from logic-level designs, simulation and testing focuses on the verification of designs at all levels, while high-level synthesis focuses on algorithms for creating hardware structures from high-level algorithmic specifications. There are also a number of minor topics, such as formal verification, that can add variety to a course that is focused on one of the primary areas.

The astute reader will note that we have ignored the problem of design specification and capture. This is intentional, because the main problems that must be solved in these tools are data management, user-interface programming, and graphical representation of data. While these topics are important, they are the proper subjects of courses in computer graphics, databases and human-interface design. Those portions of the tools that are legitimately part of design automation will also fall into the category of synthesis or simulation. Although it is not possible to completely ignore the human-interface problem, programming exercises and homework can be organized to minimize most of the human-interface design problems.

### 3. Physical Design Automation

A course in physical design automation should begin with basic logic design and basic VLSI design. If possible, the student should already have taken introductory courses in these two areas, but must have taken introductory logic design as a prerequisite. The first week to two weeks should cover the basics of VLSI design, especially the design of transistors and basic gates. At the University of South Florida, we begin this portion of the course with a discussion of semiconductor technology, because our students are drawn from both the computer science and the computer engineering programs.

There is some debate as to which topic ought to come next. Most of the existing textbooks begin with partitioning, which is the problem of breaking a circuit into two or more parts, and minimizing the connections between the parts. Partitioning is indeed the first task that one would perform when creating the layout of a circuit, but we find the topic to be difficult to motivate without some prior discussion of other issues. For this reason, we prefer to begin our course with channel routing, which is the problem of routing an area with fixed pins at the top and bottom, and floating pins at the right and left, as illustrated in Figure 2. After a discussion of several channel routing algorithms, which should include the left-edge algorithm, dogleg routing, and the YACR2 algorithm, we motivate the discussion of channel routing by discussing the placement problem.

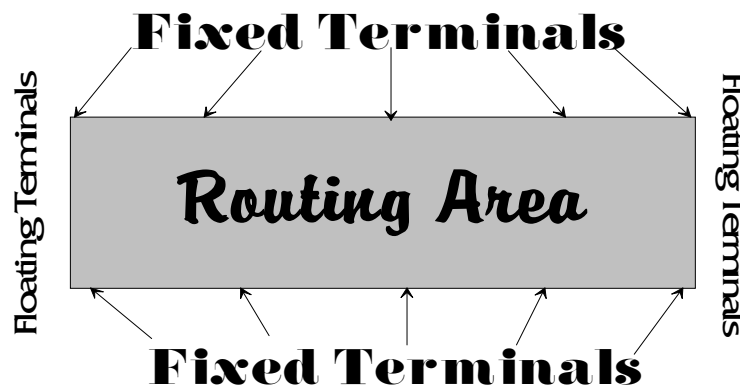
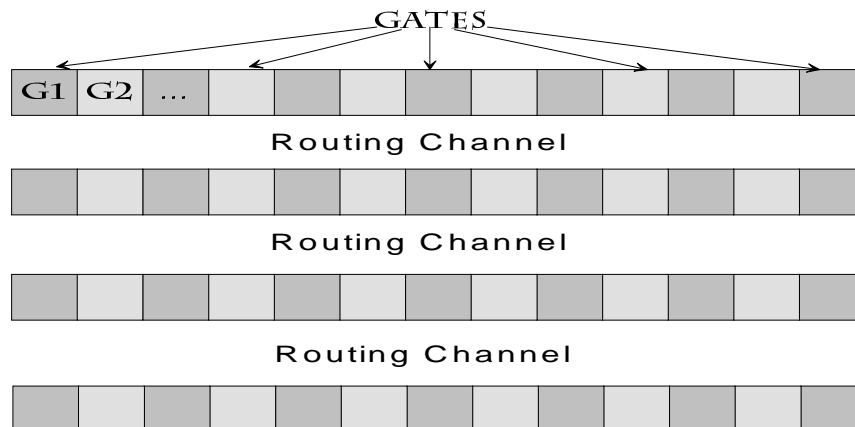


Figure 2. A channel Routing Area.

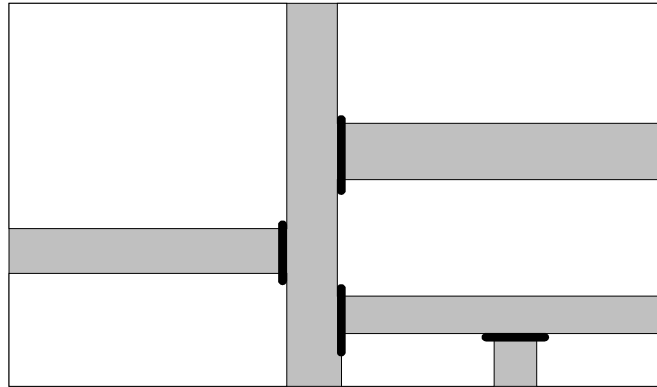
The simplest form of the placement problem assumes that a collection of pre-designed cells into has been arranged into rows, as illustrated in Figure 3. The collection of cells is assumed to be a gate-level circuit, and the cells are assumed to represent simple gates. These cells are assumed to have power and ground terminals routed through the top and bottom, which simplifies the problem of power and ground routing. Gates are connected through the routing channels between the rows, so there is immediate motivation for the channel routing problem. The placement problem is to rearrange the rows so as to minimize the amount of area taken by the channel routing algorithms. At this point, the student will understand that a well-constructed channel routing problem will consume less area than a poorly constructed problem, so there is also strong motivation for the placement problem. At a minimum, the course should cover min-cut placement, force-directed placement, and simulated annealing.



**Figure 3. The Placement Problem.**

At this point, partitioning can be introduced as a method for constructing the initial rows for the placement problem. Another motivation is the problem of breaking the circuit into pieces small enough to be handled by a placement algorithm. The basic partitioning algorithm is the Kernighan-Lin Algorithm, which is also the basis for min-cut placement. This algorithm is used to create the basic partition, which is then enhanced through iterative improvement techniques such as simulated annealing. There are other basic algorithms that can be used to create initial partition. The course should cover both basic partitioning and iterative improvement techniques.

At this point the student is conversant with the techniques used to create large blocks of layout. The next step in the course is to cover floorplanning and global routing, which can be used to combine blocks into a larger circuit. Floorplanning is the art of arranging large layout blocks so as to minimize the area consumed by the entire circuit. Despite much research, floorplanning is still a largely manual process. One can mention the automatic techniques available, but should emphasize that human intervention is usually required to obtain an acceptable floorplan. Once a floorplan is complete, global routing is used to wire the connections between the blocks of the circuit. Figure 4 illustrates a sample floorplan. The white areas represent blocks, while the gray areas represent the spaces between them. These gray areas can be divided up into channels, as indicated by the heavy lines, and routed using a channel router. It is important to route the channels in the proper order, since the floating connections at the ends of the channel must become fixed before the connecting channel can be routed. In many cases, it is necessary to route a connection from one block to another non-adjacent block. In this case, a net must be routed through more than one channel. In such situations it is usually the case that there are several paths that a net could follow. Global routing is the problem of assigning nets to channels in such a way that all nets end up being routed to their proper locations, without causing congestion in any of the channels. Congestion occurs when one channel is very heavily used, and another equally useful channel is under-used. Global routing also attempts to minimize total path length. This portion of the course should cover channel ordering, maze-running algorithms, and the Steiner tree problem. More recent approaches may also be presented.



**Figure 4. Global Routing.**

In addition to channel routing and global routing, it is also necessary to present some specialized routing techniques. The most important of these are the Lee router, which may also be covered as part of global routing, and various switch-box routing algorithms. Other specialized routing algorithms are the river-routing algorithm, clock-distribution algorithms, and power and ground routing techniques.

Finally it is necessary to cover optimization of layouts. The primary optimization technique is compaction, which is often provided as a layout-editor command. At least one commonly used algorithm, such as that used by the Magic editor, should be presented. There are a number of other topics that can enrich a physical design automation course, among these are pin assignment, equivalent terminal handling, and automatic layout verification. The course should wind up by giving an overview of simulation and high-level synthesis.

#### **4. Simulation and Design Verification**

Simulation and design verification are the most important and time-consuming parts of the design process. This is understandable since bugs found during simulation are easy to fix, while those found in the final product are expensive and time consuming to repair. In the case of high-profile circuits such as microprocessors, bugs in the final product can have far-reaching consequences that go beyond the engineering laboratory to a company's bottom line.

Oddly enough, simulation is something that everyone believes they already understand. Like many commonly held beliefs, this is not true. However, at least a portion of the course must be devoted to reassuring students that this "really is how simulators work." and dispelling false notions that they may have gotten elsewhere.

Simulation techniques can be broken into several categories that are organized more-or-less around the level of the circuit model. In other words, a design consisting of logic gates must be simulated differently from one consisting of transistors. At the highest level are models that are written in high-level languages such as C, PASCAL, or VHDL. Because of its immense popularity, it may be desirable to introduce *some* VHDL at the beginning of the course. However, VHDL modeling tends to be much like ordinary high-level-language programming, and will do little to enhance the student's knowledge of simulation algorithms. Our preference is to skip high-level modeling and move directly

to logic simulation. We also begin the course with a two-lecture review of logic design, to make sure that students have not forgotten this material.

The presentation of logic-simulation algorithms must begin with a discussion of logic models. The students will already be familiar with the two-valued logic model, and should have no trouble constructing gate simulations for this model. It is important to point out the strengths and weaknesses of the two-valued model, and introduce more complex models. At the very least, it is necessary to introduce three-valued logic using one, zero, and U (Unknown). One can also introduce the Z (Tristated) value, but there is some debate about whether Z should be considered a value or a signal-strength. In VHDL simulations, Z is considered to be a real value, but other simulators take the opposite view. It is also very important to point out that many complex VLSI designs have been verified using nothing more than two-valued logic. One should also point out that even when a highly complex logic model is used, the vast majority of all gate simulations will be done using ones and zeros.

Because gate simulations are trivial, the most important topic in logic simulation is scheduling. The ultimate goal of any scheduling algorithm is to reduce simulation time to a minimum. There are two approaches to achieving this goal: making gate simulations more efficient, and reducing the number of gate simulations performed. There are algorithms that use compile-time scheduling and straight-line code at run time. These algorithms have reduced the amount of run-time code to a minimum, but cannot adapt themselves changing conditions in the input. For this reason, these algorithms are called “oblivious” algorithms. If the user submits the same set of inputs ten times in a row, an oblivious algorithm will perform ten complete simulations, ignoring the fact that if there is no change in the input, there can be no change in the output. (Clocks and one-shot timing signals are considered to be input changes.) One must contrast oblivious algorithms with event-driven algorithms, which attempt to reduce the number of gate simulations to a minimum.

It is also necessary to go through the various timing models. In particular, it is necessary to contrast zero-delay simulation with unit-delay and multi-delay simulation. It is important to point out to the students, that except for the terms “zero-delay” and “unit-delay,” there is little agreement about the meaning of various terms. In our courses, we use the terminology in the following way. We use the term “multi-delay” to denote a timing-wheel-based simulation. Gate delays differ from one another, but it is assumed that only a rough estimate of the gate delay has been obtained, perhaps from a library or some standard formula. We also assume that the gate delays are expressed as reasonably small integers (less than 1000). We use the term “nominal-delay” to denote a more precise simulation using priority queues rather than a timing wheel. Gate delays are expressed as real numbers, and it is assumed that the numbers have been obtained from a preliminary layout of the circuit. The delay is constant, and is unaffected by changes in the circuit. We use the term “timing-simulation” to denote simulation that is based on differential equations, or some approximation thereof. Simulations of this nature are generally transistor-based, and do not fall into the category of logic simulation.

In addition to the basic timing models, important variations on the multi-delay and nominal-delay models should also be covered. These include, rise-fall delays, transport delay, inertial delay, and min/max simulation.

The logic-simulation portion of the course should also cover fault simulation. The important algorithms are fault-propagation, and concurrent fault simulation. It is also important to cover the predominant fault models, such as stuck-at faults, and bridging faults. Logic simulation is sometimes equated with fault simulation, but in reality the two are completely different. Fault simulation is, effectively, the simultaneous simulation of several circuits, with the aim of determining whether the various circuits can be distinguished from one another using a particular set of tests. Logic simulation is used to evaluate a circuit, fault simulation is used to evaluate a set of tests. Although a fault-simulation technique may be based on a commonly known scheduling algorithm, the simultaneous simulation of faulty circuits introduces significant problems. Among these are hyperactivity, and accidental creation of sequential elements.

At the transistor level, there are two basic approaches to simulation, switch-level simulation and circuit simulation. Switch-level simulation treats a transistor as a logical device that can be conducting or non-conducting. An exponentially nested model of signal strengths is used to determine which values dominate others. Sub-blocks of the circuit are represented as systems of Boolean equations that are solved using Gaussian Elimination.

Circuit simulation uses a mathematical model of the circuit and a timing schedule to calculate voltage levels at various nets at a sequence of predetermined times. Tools such as Spice and Pspice fit into this category. We prefer not to cover circuit simulation in our design automation course, because the techniques require a significant amount of background material for complete understanding. Logic simulation and switch-level simulation consume all but about two weeks of our course, and we prefer to spend the final two weeks of the course presenting material from physical design automation and high-level synthesis. In particular, the material on multi-delay and nominal-delay techniques can consume a significant portion of the course. By the time this material has been covered, the students are ready for some variety. Circuit simulation could be covered in a senior-level seminar course with four or five students, but we would not recommend it for a general course in design automation.

## 5. High-Level Synthesis

The area of high-level synthesis is one of the most diverse and interesting fields in design automation. In this category we find everything from simple macro processors to full-blown compilers that purport to create a complete circuit from a simple high-level description. Despite the many claims of success, high-level synthesis remains an area of intense research. Much of this research is very interesting, and much of it has resulted in successful commercial tools.

The area of high-level synthesis contains several well-defined sub-areas, including behavioral synthesis and logic synthesis, as well as a number of less well-researched areas. Behavioral synthesis is the problem of creating a gate-level circuit from a high-level-language description of an algorithm, while logic synthesis is the problem of constructing an efficient gate-level implementation of a set of Boolean equations. Research in high-level synthesis has also spawned a large research effort into Binary Decision Diagrams, or BDD's. This research has had impact in other areas, most notably in the area of simulation.

In our courses we have concentrated on the area of behavioral synthesis, which consists of four distinct steps, scheduling, register assignment, data-path design, and microcode generation. In the scheduling step, the algorithm is broken down into a collection of simple operations, the data-flow dependencies between the operations are determined, and the operations are scheduled using the available hardware and the data-flow dependencies. Two of the important scheduling algorithms are “in time” scheduling and force-directed scheduling. The objective of register assignment is to assign the variables of the algorithm to different registers, while minimizing the number of registers required. Register assignment has been shown to be equivalent to the maximum-clique problem, which is known to be NP-Complete. Data-path design is the process of creating the required computational circuits and the data paths between the computational units and the registers. The objective of this step is to minimize the amount of hardware required, but this is also highly dependent on the results of the scheduling step. The final step is microcode generation, which creates the control logic necessary to execute the schedule created by the scheduling step. This step may actually generate microcode, or it may create an equivalent hardwired control.

At the present time we have not devoted a major segment of a course in high-level synthesis, but we may do this in the future. We would include topics from both behavioral synthesis and logic synthesis, and also include a segment on simulating high-level designs. This type of course would be an ideal place to include some of the more interesting, but less important areas of design automation, such as graph-isomorphism based layout verification, and automatic verification of designs.

## 6. Survey Courses

Because we have a number of researchers in the area of design automation, we prefer to organize our course around one of the major areas, and offer the course repeatedly for additional credit. However, other departments may prefer to offer a single standard course that covers the basics of all areas. In physical design automation, the essential topics are Kernighan-Lin partitioning, channel routing, global routing, and min-cut based placement. In simulation the essentials are delay models, including the basics unit-delay, zero-delay, and multi-delay models. The basics of event-driven and compiled code simulation should also be covered. In high-level synthesis it is necessary to cover the basics of behavioral synthesis, especially in-time scheduling and force-directed scheduling. These topics, taken together, should give the student a general idea of how design-automation tools work.

## 7. Laboratory Exercises

One cannot expect students to complete an entire design automation tool in a single semester, however it is possible to provide students with a framework that simplifies the development of certain algorithms, thus permitting one or more development projects to be completed in a single-semester course. The quickest way to facilitate programming exercises is to provide highly-stylized input for the various algorithms. For example, if the left-edge channel routing algorithm were assigned as a project, the input could be in the form of a sequence of pin-numbers, rather than a more realistic type of layout. For simulation, a pre-written system with a parser and a human interface could be provided. A student exercise could be to replace one component of this system. For high-level

synthesis, a parser could be provided that reduces input data to a predictable set of data-structures.

As far as projects are concerned, we recommend the left-edge channel routing algorithm, and the Kernighan-Lin partitioning algorithm as half-semester or full-semester projects. In simulation, we recommend the levelization algorithm, or the gate simulation algorithm. If it is desirable to have the students finish a complete simulator, one could add leveled simulation. We would recommend running this algorithm interpretively (that is, as part of the compilation process) rather than generating compiled code. In high-level synthesis, we would recommend any of the scheduling algorithms, as long as properly parsed input could be provided.

## 8. Educational Resources

There are excellent text-books available for physical design automation. See references [1-3] for details on obtaining these texts. Reference [1] is excellent, but is somewhat dated. It consists of a series of articles written by different experts in the field. Reference [2] is comprehensive and up to date, but is not always clearly written, and does not always cover each topic in sufficient detail. Of the three we recommend reference [3], which is up to date, readable, and aimed at the undergraduate level.

There are fewer good textbooks in simulation. Reference [4] contains some information about simulation, but is essentially a book on testing, rather than a comprehensive work on simulation. For this area, we provide our students with our own notes, which are available from the web-site mentioned in reference [5].

In synthesis, there is at least one excellent book in logic synthesis [6]. For behavioral synthesis, one could start with the excellent survey paper mentioned in reference [7], and add material from the ICCAD, and DAC conferences as well as current articles from IEEE Transactions on Computer Aided Design, and ACM Transactions on Design Automation Systems.

## 9. Conclusion

As stated in the title, we believe that no engineer's education is complete without some introduction to the internal workings of design automation tools. Design automation is a rich subject that allows one to design several very different courses. At the same time, the basics can be covered in a single-semester survey course. We are currently offering this course as an elective, with the subject material changing from area to area each semester. We plan to revise this course somewhat, to cover a more broad range of topics that will stay the same from year to year. We believe that this course would be a valuable addition to any curriculum, that would add breadth and insight to an engineer's educational experience.

## 10. References

1. Preas, Bryan and Lorenzetti, Michael (eds.) *Physical Design Automation of VLSI Systems*, The Benjamin/Cummings Publishing Company, Menlo Park California, 1988. (ISBN 0-8053-0412-9).
2. Sherwani, Naveed, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, Boston, 1993. (ISBN 0-7923-9294-9).

3. Sait, Sadiq and Youssef, Habib, VLSI Physical Design Automation: Theory and Practice, IEEE Press, New York, 1995. (ISBN 0-07-707742-3)
4. M Abramovici, M. Breuer, A. Friedman, *Digital Systems Testing, and Testable Design*, IEEE Press, New York, 1995. (ISBN 0-7803-1062-4)
5. Maurer, Peter, Course Materials web site:  
<http://www.csee.usf.edu/~maurer/courses.html#UGDA>.
6. Hachtel, Gary and Somenzi, Fabio, *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers, Boston 1996. (ISBN 0-7923-9746-0)
7. M. C McFarland, A.C. Parker, and R. Camposano, Tutorial on High-Level Synthesis, Proceedings of the 25<sup>th</sup> Design Automation Conference, pp. 330-336, 1988.