Requirements for a Research-Oriented IC Design System

by

Jonathan Allen

Research Laboratory of Electronics
and
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

Computer-aided design techniques for integrated circuits have grown in an incremental way, responding to various perceived needs, so that today there are a number of useful programs for logic generation, simulation at various levels, test preparation, artwork generation and analysis (including design rule checking), and interactive graphical editing. While the design of many circuits has benefitted from these programs, when industry wants to produce a high-volume part, the design and layout are done manually, followed by digitizing and perhaps some graphic editing before it is converted to pattern generation format, leading to the often heard statement that computer-aided design of integrated circuits doesn't work. If progress is to be made, it seems clear that the entire design process has to be thought through in basic terms, and much more attention must be paid to the way in which computational techniques can complement the designer's abilities. Currently, it is appropriate to try to characterize the design process in abstract terms, so that implementation and technological biases don't cloud the view of a desired system. In this paper, we briefly describe the conversion of algorithms to masks at a very general level, and then describe several projects at MIT which aim to provide contributions to an integrated design system. It is emphasized that no complete system design exists now at MIT, and that we believe that general design considerations must constantly be tested by building (and rebuilding) the various subcomponents, the structure of which is guided by our view of the overall design process.

We see these overall design processes as a set of conversions through a succession of representations, starting at the level of an algorithmic description and proceeding to a mask specification via a set of transformations. Each of these representations is keyed to a particular design focus. Thus, for example, there is an initial level at which the algorithm must be represented, and following levels where the architectural structure, detailed logic design, electrical circuit, and geometrical layout must be described. There is no unique partitioning into levels, so that some systems may want to dispense with an architectural description, while others may want to add a topological layout level. Clearly, these levels pose vastly different entities that must be characterized, so that each of the levels requires a design language appropriate to the semantic primitives (e.g. logic gate or circuit node) that are relevant at that level. Despite this diversity of representation, however, the transformations that are used to convert between them must preserve the original algorithmic intent. That is, the constraints that hold in the original algorithmic description (which must, of course, be represented in some appropriate linguistic form) must also hold at all succeeding levels, although they will certainly have to be interpreted in terms of the entities native to that level. Thus, the final mask specification must be a realization of a scheme to execute the original algorithm, and while the algorithm is expressed by some linguistic means with its own set of relevant structures to be manipulated, the mask specification will have to represent the base for task execution in terms of static geometrical structures. It is clear that a major part of the task of building a design system is the determination of the levels at which representations are desired, followed by a clear semantic definition of what the semantic entities are at that level and the (limited) ways in which they can be manipulated. Transformations across levels must then express how these semantic entities link up. We feel that it is important to conceive of the design process in terms of such linked programs with clear semantics, and that representational issues (such as common data bases) must follow from the more basic linguistic specifications.

From the start, we have felt that the biggest weakness in the present arsenal of IC design tools has been the lack of appropriate linguistic means to specify algorithms. Certainly algorithms are frequently described in a common programming language, such as ALGOL, but for purposes of IC design, such representations are inappropriate. Programming languages have generally been devised to specify algorithms that will be executed on a single-sequence (e.g. von Neumann) computer, and the usual kinds of statements referring to the time or space requirements of an algorithm assume such an architecture. In hardware systems, however, there is much opportunity for parallelism, and so the usual procedural representation of an algorithm is inappropriate. What is needed, then, is some means to represent the underlying "competence" of the algorithm, devoid of any performance bias, so that only those constraints that are common to <u>all</u> algorithms that execute the given task contribute to this "deep structure". This permits us to then explore the ways in which the task can be executed as a separate problem, and hence provide the designer with a flexible means to pick the implementation that gives the best space/time tradeoff for the particular application at hand. We have mounted two attacks on this problem. In one, the job has been to find the underlying structure common to all equivalent algorithms, while in the other, we have looked for interactive means to explore the various algorithmic alternatives. These two projects are thus complementary, and we discuss them next.

Building from considerable experience with the construction of computer-based debugging tools for electrical circuit analysis, it has become clear that the construction of constraint diagrams is very useful for encapsulating what is known about a circuit so that inferences can be made about related parameters in a circuit. Typical constraints include Kirchoff's voltage and current laws and Ohm's law, the latter expressing a constraint between the voltage across and the current through, a resistor. Constraint diagrams for electrical circuits can be readily drawn, so that inferences can be made about circuit variables which control an observed value. One way to think of the constraint diagram is to regard it as an expression of the enduring, time-independent truth about the circuit. That is, the constraints characterize what must hold in the circuit under any parametric conditions.

Next, we observe that this notion of constraint should be useful for _any_ system that performs a task, and since the constraint representation characterizes what must hold during any execution of the given task, it provides a common "deep structure" for all possible ways to perform the task. The neutral constraint representation thus provides an ideal starting point to begin the design of an integrated circuit. The designer is free to pick out any implementation consistent with the algorithmic constraints that will provide the required performance.

Attractive as this view may be, there are at present several missing pieces limiting the realization of its potential. We need to establish that constraint networks can be created for a broad and significant class of algorithms, and that an appropriate formalism emerges for this purpose. We also need to develop a variety of techniques for building performance strategies on the constraint representations, and hence provide a means to project several space/time performance means from a single underlying constraint network.

Constraints are of course a natural way to represent design rules, and we are working toward the structure of a broader class of rules than just those specified in terms of the layout geometry. Constraints such as current density, power, speed, capacitance, and other performance parameters should be integrated with those expressed at the layout level. This is an example of a general formalism being used to control the design at several levels of representation, a capability which is likely to be increasingly useful.

Constraint network representations emphasize the multiplicity of ways in which a task can be performed, so it is natural to ask how the designer can select a particular performance strategy. One way to do this is to start with <u>some</u> algorithm for performing the task, and then provide means for changing its performance via a set of transformations on the original algorithm expression. We have constructed linguistic techniques for originally specifying an algorithm, and then transforming it into a number of forms which vary throughout the space/time design space. For example, some algorithm blocks consist of statements that can be executed in any order, and hence can be done in parallel (p-blocks), while still other blocks contain statements, all of whose right-hand-sides should first be evaluated, and then all assignments should be done in parallel (c-blocks). Transformations have been defined to convert between any pair of these blocks, thus allowing the designer to explore various algorithms which are equivalent in terms of input/output behavior. Standard do-loop and for-all control structures will also be provided. Once the desired architecture is obtained, then it should be compiled into logic, represented by a low-level hardware design language.

Space and time are the fundamental architectural factors which can be traded. We expect that the requirements of IC design systems will cause the nature and control of these tradeoffs to be heavily studied. It is interesting that the design freedom presented to the IC designer makes these considerations much more prominent for large circuits than they have been in the past. At the current stage of research, we feel that it is appropriate to give the designer full control of these tradeoffs. Possibly in the future the nature of design may be sufficiently understood to encapsulate the design exploration process in a program, but we are not there yet.

From the architectural level, representation at the logic and circuit level must be obtained, leading to the layout and artwork level. We have not focused our attention on the logic and circuit levels, but we have felt that the geometrical level provides problems in the management of complexity that need immediate attention. Large integrated circuits must take advantage of modular subcomponents, or macrocells, and these have to be integrated into the semantic representations at the various levels. In recent years, the initial standard cell approach, using small pre-compiled functions arrayed in rows, has yielded to more general and larger cells, representing memories, ALU's, and other structures, which enjoy complete freedom of placement on the chip surface. This escape from layout rigidity has led to the design problem of specifying placement for these cells, and providing the required interconnect. A particular problem of interest is the extent to which hierarchical grouping of these cells should be automatically enforced as opposed to allowing normal grouping. We feel that it is important to benefit from previous designs, and to permit fast design of large systems based on these modules, even if substantial inefficiencies remain. Perhaps, by delaying the binding of the geometrical layout of these cells, topological representation can be stored for them in the designer's library, providing flexibility of connection and shape which can be made responsive to the overall layout constraints of the circuit. Of course, the ways in which these topological specifications are bound into geometry affects the corresponding circuit performance, so that as the geometry is instantiated, it is necessary to enforce the kind of extended design rules discussed above which not only restrict placement on the surface, but also constrain the circuit performance. Rapid artwork analyzers are needed to keep the geometrical and circuit representation linked up and related to the design constraints. The circuit representations must permit ready analysis and simulation for a detailed view of performance. Thus we feel it is extremely important to view the layout problem in terms of all its consequences, and to provide high-speed interactive means to explore these relationships. So just as we have placed heavy emphasis on the initial pole of the design trajectory, i.e. the algorithmic specification, we are also focusing heavily on the set of issues tied to the binding of layout geometry at the other final pole of the design process. Both areas require flexible, semantically clean representations, with options for binding of decisions available interactively to the designer.

The need for highly interactive computing facilities, particularly for graphic editing of the various representations, has led us to build an extended version of the MIT AI Laboratory LISP computer. This machine is designed as a personal computer, capable of providing very fast response. It has a large (24-bit) address space, and gives performance competitive with many large time-sharing systems. Two graphic bit map displays are provided, one for black and white, and one for color. These displays are refreshed from main memory while the computer is running, so that the displays are updated immediately as new data is computed. since the two displays run simultaneously, large figures can be displayed on the black and white monitor while detailed views are available on the color display. The computer provides 128K 32-bit words of main memory plus 16K words of writable control store and a 300 megabyte disk. Both keyboard and "mouse" are supplied for manual interaction. This machine is connected by a local network to several other machines at MIT, so that large data bases and control services such as printing can be accessed. LISP is seen as an excellent language for the construction of data structures as well as interpreters at a variety of levels, and we feel that the flexibility provided by the combined hardware and software resources is important during research on the structure of an IC design system. A DECSYSTEM-20 computer is also available, and several simulation, artwork analysis, and plotting programs are used on this facility, which is also tied to the local network. In time, we expect that several LISP machines may be devoted to the IC design effort, all linked together, and to other large machines for data base management and less interactive tasks using established programs available within the research community. Our view is that computing facilities of this type are essential to the design of a comprehensive design system, and will form the base of practical systems of the future.

The emphasis which we have placed on linguistic specification and manipulation of representations coupled with layout editing and highly interactive graphical computing must lead, of course, to a complete design system which permits the designer to convert an algorithm to a set of mask specifications. In order to focus these efforts, we are designing several circuits which should provide insight into the design system requirements. These include a speech synthesizer, a key encryption system, and a local network interface. Each of these projects is based on an original MSI design of between 120 and 150 dips, and each algorithm designer is also working on aspects of the IC design problem. We are even designing a microprocessor to serve in a next-generation LISP machine, so that our IC design efforts are reinforcing the tools available for such work. The study of various aspects of a design system, such as circuit simulation, may also lead to the design of special computing facilities for these tasks, particularly when these processes are time-consuming and expensive on conventional machines.

Acknowledgement. The IC design research described in this brief overview is part of a project devoted to means to convert algorithms to integrated circuits sponsored by AFOSR, of which the author is the principal investigator. Gerald Sussman and Guy Steele have been studying constraint systems, and the work on algorithm transformations is contained in Glen Miranker's doctoral thesis. Ronald Rivest and Andrea LaPaugh are working on generalized placement and routing, and Paul Penfield is devising artwork analysis algorithms. Clifford Fonstad and Dimitri Antoniadis have been interested in new device technology and process modeling, while the LISP machine has been designed and built by Tom Knight, Jack Holloway, and Richard Greenblatt. The cross-disciplinary interaction of these people has led to a number of refreshing insights, and the start of many new research interests motivated by the needs of IC design.