The Silicon Compiler: Panacea, Wishful Thinking, or Old Hat?

Jerry Werner, Editor-in-Chief

"As recently as two or three years ago, most IC designers were disappointed at the apparent lack of progress in the computer-aided design field. The expectations of potential users were exceeded only by the visions of the field's prophets who predicted killer *systems capable of automatically designing the most dense and complex circuits. The power of the computer would allow the elimination of many arduous and time-consuming tasks . . . "*

–from an abstract describing the informal session entitled "Computer-Aided Design Tools for VLSI" at the 1980 International Solid State Circuits Conference

"In hardware design, machine specifications are in a constant state of flux, but radical changes to the design are unaffordable if large portions of the design must be redone, as is usually the case if the design consists of geometric primitives. [Silicon] compilers make changes affordable, giving the designer freedom to explore many design strategies."

–from a 1981 Caltech Ph.D. thesis by David Johannsen, entitled "Silicon Compilation"

"The term silicon compiler is used to denote a program with the translational attributes of a compiler, yet whose object is a hardware device rather than software. It is, in a sense, the ultimate CAD tool."

–from a presentation by A.M. Peskin of Brookhaven National Laboratory, entitled "Toward a Silicon Compiler," at the 1982 Custom Integrated Circuits Conference

* * *

In a very real sense, the so-called "silicon compiler" *is* by definition a *killer* system for VLSI design. Conceptually, the silicon compiler is similar to commonly used software compilers in that a high-level input (a behavioral or functional definition of a chip) is "compiled" down to a low-level description (mask artwork).

"There is an enormous similarity—an enormous relevance—of the techniques that are applied in classic compiler writing to the problems of silicon compilation," says Steve Johnson, a Bell Labs (Murray Hill, NJ) researcher who heads a 3-person silicon-compiler project. (Johnson was part of the group that developed UNIX and the C compiler at Bell Labs.) "When you compile a FORTRAN program, the compiler chooses the registers for you. I consider sitting and staring at geometry to be right on a par with sitting and staring at assembly language," he says.

Clearly a Research Issue

Unlike other software (circuit and logic simulators, automatic layout algorithms) that started in the research labs of universities or industry but migrated into product-development groups in industry, work toward an "ideal" silicon compiler is still in the research domain. Universities such as MIT, Caltech, Brown, Stanford, the University of Utah, and the University of Illinois are actively researching the topic, as are federal labs (Lincoln and Brookhaven National Laboratories) and industry (IBM, Bell Labs, Honeywell, DEC, GTE, and others).

Although researchers (especially those with a computer-science background) laud the potential and prospects for silicon compilers, designers and managers responsible for shipping new LSI and VLSI ICs are not convinced. "The problem of designing circuits has nothing to do with the problem of writing software," flatly states Bernard Murphy, head of the CMOS IC design department at Bell Labs (Murray Hill, NJ). "Software is basically a one-dimensional problem. Designing a chip is qualitatively different—it's a two-dimensional problem," he adds. Murphy headed the team responsible for the BELLMAC 32A Processor, a large, complex (more than 100,000 transistors) CMOS device that was designed using a variety of CAD tools, but that was *not* automatically laid out. "The use of the term 'silicon compiler' trivializes the problem in the sense that it seems to imply that because we can write good compilers for software we ought to be able to write good 'compilers,' or automated layout tools, for integrated circuits," Murphy explains, adding "nothing is further from the truth—I think it's a total misuse of the word."

Murphy says that the design tools should support a human being who "directs" the design and who has what he calls "renowned pattern recognition ability." "Total automation is just inappropriate—either now or in the forseeable future—in anything where you have a competitive need for performance," he concludes.

Interestingly, Howie Shrobe of MIT, although primarily in a university rather than an industrial environment (he is, however, on the staff of Symbolics, Inc.), has a very similar perspective on silicon compilers. "The main reason [silicon compilers] are not all that useful yet is that there's a lot of complicated reasoning that goes on in deciding how to make a real chip work. If you're trying to make a sophisticated chip, then the silicon compiler will have to be an AI (artificial intelligence) program." Shrobe says that MIT to date has concen-

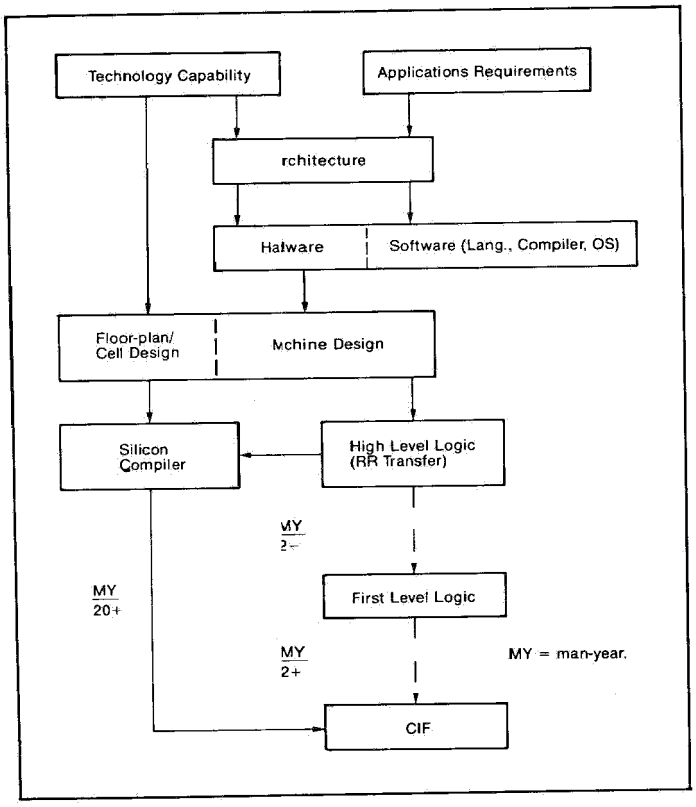

FIGURE 1. Role of a silicn compiler. This flowchart, by Art Rideout of IBM (a particpant in Caltech's Silicon Structures Project) predicts that a silcon compiler will cut IC design time by a factor of 20.

trated on developing "high level module generators," rather than full-chip "compilers."

An Appealing Concept

Just the thought that a computer could perform IC design tasks in minutes or hours that here to fore would have been counted in tens of *man-years* inevitably makes the concept very attractive. Art Rideout, a senior engineer from IBM's Burlington, VT facility who is winding up a one-year stint in Caltech's Silicon Structures Project (SSP), believes that a silicon compiler should reduce the IC development effort by a factor of 20. This is accomplished, he says, by enabling a chip design to go from a register-transfer-type high-level description directly to a layout, bypassing the manual first-level logic-design stage (see Figure 1).

One indication that the silicon-compiler concept is gaining notoriety is the fact that quite a few recent IC or computer conferences—including the 1982 COMPCON, Custom Integrated Circuits Conference (CICC), Design Automation Conference (DAC), and International Circuits and Computers Conference (ICCC)— have included at least one session or paper dedicated to silicon compilers.

Is the silicon compiler an idea whose time has come? Or is it simply the computer science community's "wish syndrome" that will eventually, unavoidably run into the hard facts of circuit design and behavior, and the realities of producing manufacturable ICs?

The Basic Problem: One of Definition

Those questions are hard to answer for one simple reason—there is a whole lot of confusion regarding just what *is* a silicon

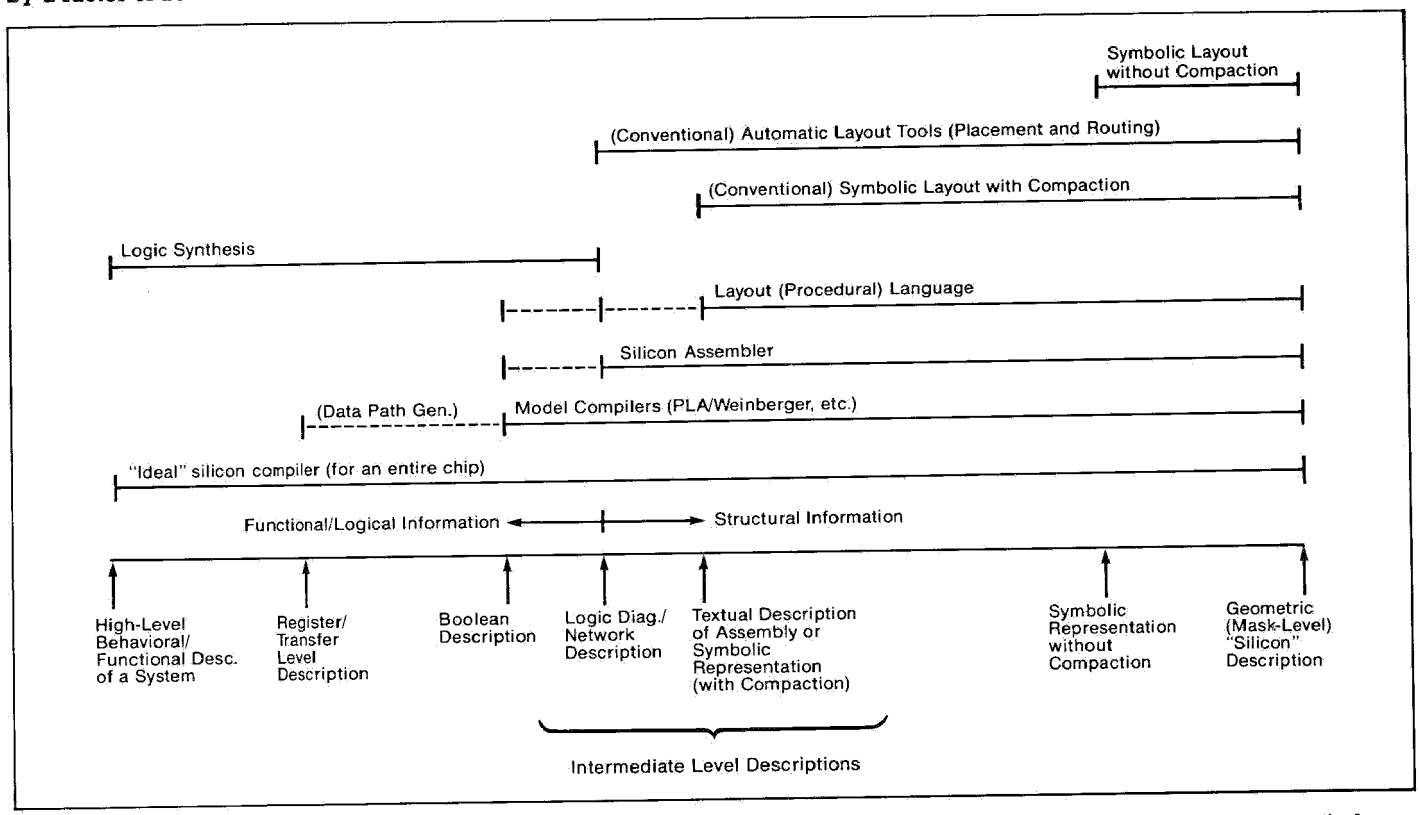

FIGURE 2. An "ideal" silicon compiler would span the IC design spectrum. Unfortunately, there is no universally accepted definition of a silicon compiler, and various other design tools (data-path generators, module compilers, silicon assemblers/layout modeling languages) often share the same name.

compiler. Theoretically, the "ideal" silicon compiler would generate IC mask artwork automatically, using only a high-level functional description (see Figure 2). Unfortunately, the term silicon compiler means different things to different people. The definition ranges from layout modeling languages, "silicon assemblers," and hierarchical PLA generators (Ayres 1979), which operate at the *structural* end of the IC design spectrum, to logic-synthesis systems which operate entirely on the functional (technology-independent) level.

Even industrial people who are working to understand what the silicon compiler is are hard-pressed to give a black-and-white description of it. "The main goal of our work is to understand them [silicon compilers] well enough so that once somebody finally figures one out, we can recognize it and also be able to analyze and compare it," says Charles Rupp, manager of the Exploratory Research Group at Digital Equipment Corporation in Maynard, MA. Rupp says that one of his projects in 1983 will be to do a chip design using three silicon compilers that are "sort of" available, and to "compare them in terms of the philosophical approach." However, he declines to identify the systems.

Is the "Silicon Compiler" Concept Really New?

Some IC design systems which proponents are calling "compilers" are definitely similar (both in terms of the overall system as well as individual computerized tools) to more "traditional" design systems. "There is a misunderstanding that the silicon compiler is brand new, when IC layout aids have been around for some time," says Chuck Gwyn, IC design department manager at Sandia National Laboratories.

The SCULPTOR system under development at Caltech's Silicon Structures Project (SSP) is a case in point. SCULPTOR includes individual module (datapath, PLA) generators and "general [module] interconnect" programs. Of course, PLA generators are widely used in industry (and have been for many years), and automatic routing software has been used extensively in industry since the mid-70s. But Gary Clow, Motorola's representative on the SSP, says that SCULPTOR's contribution is not so much in the individual module generators, but rather in the "modular and expandable" nature of the system.

One common approach to combinational logic used in silicon compilers is the so-called "Weinberger array," which is similar to a PLA but allows NOR-gate logic configurations of arbitrary depth (PLAs allow only two levels of NOR gates). MIT Lincoln Laboratory's "MacPitts" language/compiler uses this approach for combinatorial logic (Siskind *et al.* 1982), as does the University of Illinois, which modified it to let I/O nodes appear on all sides of a cell (Luhukay *et al.* 1982). However, the Weinberger-array concept has been around since the late sixties (Weinberger 1967).

The "Ideal" Compiler

Few systems purport to span the entire range of the design spectrum. One of the first, Caltech's Bristle Blocks, was used to design datapath chips which have a relatively fixed floor-plan (Johannsen 1979 and 1981). Bristle Blocks remains an approach whose universality is unknown and whose implementation is unproven. Only "five or six" Bristle Blocks designs have ever been fabricated, says Johannsen; of those, only one—a relatively small frequency-scaler chip— was ever tested. The frequency-scaler chip did not function properly, although the

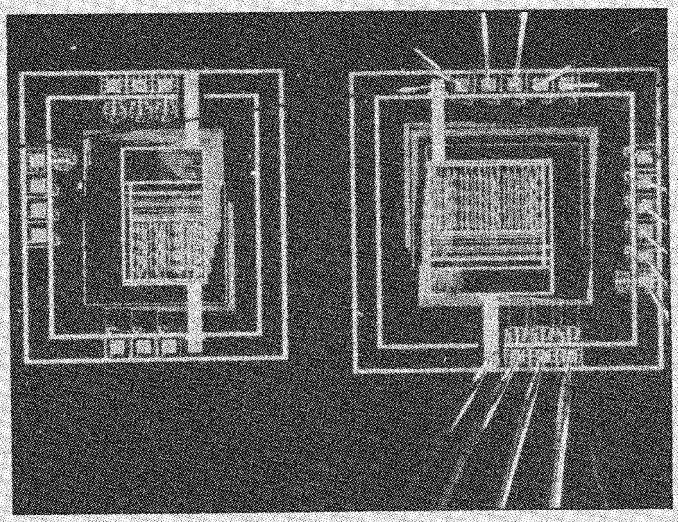

FIGURE 3. These small chips (4-bit parallel in/out left/right shift register on the left, 4-bit resettable modulo-n counter on the right) were designed by an early version of the MacPitts system. Larger devices have yet to be fabricated.

device failure was partially attributed to a specification error. Why has only one device been tested? "Basically, lack of time," Johannsen acknowledges, adding that "some of the chips didn't get back before I graduated [with a Ph.D. from Caltech]." He is one of the founders of a small start-up firm called Silicon Compilers, Inc. in San Francisco, CA. At present, the company is not giving out specific details of its future products, although a spokesman says it is not pursuing the strict Bristle Blocks approach.

Bristle Blocks typifies one similarity between most "ideal" silicon compilers: they work on a "target" architecture, not on a general class of designs. Although Rupp of DEC postulated "an almost completely general hardware compiler for the generation of integrated-circuit geometry" last year (Rupp 1981), he now concedes—primarily due to what he terms the "side assignment problem"—that such a system may be beyond the reach of today's knowledge. "It does not appear that the general layout problem is soluble—nor does it need to be," he concludes.

Another system that uses a "target architecture" is MIT Lincoln Lab's MacPitts system (Siskind *et al.* 1981). (The name MacPitts is a combination of the names of two early researchers [McCulloch and Pitts] who studied neurological systems from a mathematical and logic standpoint.) It compiles a chip using data-path generators and Weinberger array generators (which are used for the control section). The MacPitts specification language is more behavior-oriented, as opposed to Bristle Blocks language, which is more structurally oriented. The MacPitts compiler consists of two levels of routines. A higher-level routine extracts a technology-independent intermediate-level description in terms of data-path specifications, control equations, and state assignments; and a lower-level routine translates the intermediate-level description into mask data (expressed in CIF).

Although MacPitts has been used to "compile" LSI-complexity ICs (a 16-bit microprocessor with a very simplified instruction set, containing approximately 6500 transistors), only small test circuits (4-bit counters and 4-bit shift registers) have been fabricated to date (see Figure 3). According to Jeff

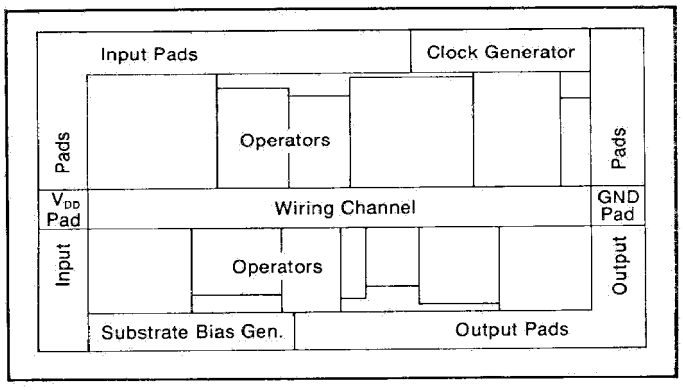

FIGURE 4. A typical "FIRST" floorplan. To minimize chip area, tall blocks are placed in the top row; short blocks in the bottom row. No placement optimization algorithm is used.

Siskind, an MIT grad student and primary researcher of the three-man MacPitts effort, MacPitts designs typically have a circuit density of 80 to 100 transistors/mm² (using a 5-micron nMOS process). Siskind acknowledges that little work has been done to date toward optimizing the *performance* of chips designed by the MacPitts system. However, he says, "that is not foreign to the framework of the compiler—it conceivably could be put in."

A Compiler for Signal Processor Chips

The FIRST (Fast Implementation of Real-time Signal Transforms) system, developed at Edinburgh University, is yet another example of a silicon compiler with a target architecture in mind. FIRST designs relatively fixed-floorplan signal-processing chips which use bit-serial architectures. The FIRST approach enables the chips to be specifed in a high-level system description language, which is a net list of bit-serial operators that represents a flow graph of the system to be implemented. The FIRST system contains a library of primitive operators (such as MULTIPLY, ADD, SORT, BIT DELAY, etc.) as well as a limited number of more complex procedural definitions (such as Biquad, Lattice, Butterfly, etc.) that enable a range of signal processing architectures (useful for speech synthesis/recognition and telecommunications applications, among others) to be configured.

In the FIRST approach, functional blocks called up from the library are then placed in specific locations in a typical floorplan (see Figure 4). All wiring is performed in a central channel, and functional blocks are arranged along the "waterfront" adjacent to the channel. Blocks are placed in the same relative order that they were specified in the high-level description of the system. No placement optimization algorithm is used, according to one member of the group that developed FIRST (Bergmann 1982), because the order in which the operators are specified should closely track the placement of blocks for optimal wiring. Wires from input pads enter the channel on the left side, and wires to the output pads exit the channel on the right side. (Input pads are located at the left and top sides of the chip, and output pads are located on the right and bottom edges of the chip.)

The FIRST design system also includes a high-level (functional) simulator, so that designs can be functionally verified prior to actual layout. An integrated automatic test pattern generation program has been discussed but not yet implemented. To date ten chips, which use a 5-micron nMOS process, have been "compiled" using the FIRST system.

Compiling Gate Arrays

One "ideal" silicon compiler that has a target *implementation* rather than a target *architecture* is being developed by Lattice Logic Ltd. of Edinburgh, Scotland. Lattice Logic's president, John P. Gray, headed Caltech's Silicon Structures Project (SSP) in 1979-1980, and described various silicon-compiler approaches at the 1979 Design Automation Conference (Gray 1979). Recently Gray returned to the DAC and, with two associates, described an approach to gate-array design using a silicon compiler (Gray *et al.* 1982). This approach lets a circuit be described in a hierarchical structural design language (SDL), which is then translated via a MODEL compiler into an intermediate design file, which is used by a physical design subsystem (to generate masks), test-pattern-generation software, and functional and timing simulators.

Pieces of the Compiler: "Front-End" Tools

As Lattice Logic's approach indicates, the existence of automatic layout tools and systems for gate arrays and cell-based ICs has been the driving force behind some approaches to "silicon compilation." At Brookhaven National Labs, one small project set out to add a high-level design *front end* to an existing set of layout tools (Peskin 1982). In this case, the system "compiled" a FORTRAN 77-based behavioral specification to the hardware-description language SDL (Van Cleemput 1979), and then translated this SDL description into network-description language (NDL) (a lower-level hardware description language used as the input for Sandia Lab's IC design system).

However, Brookhaven's effort is incomplete. Although NDL descriptions for several circuits (including a Roman-numeral clock) have been generated, those test circuits have not been fabricated nor, indeed, have they even been laid out. "It's not efficient and it's not bug free, so I wouldn't consider it of production quality," acknowledges Arnie Peskin, head of the Brookhaven effort. "Presumably it could be put in some shape, but that would have to wait until I had a collaborator with a really interesting problem to solve," he says.

Because work to create a *front-end* design tool is not technology specific (or even specific to integrated circuits), such a design tool would potentially have a wide range of applications. In fact, the heaviest of heavyweights are actively pursuing this type of research, under the general heading of *logic synthesis* (creating logical information from higher-level descriptions). For example, IBM recently discussed a logic-synthesis approach that is not completely automatic, but rather requires manual intervention (Darringer *et al.* 1981). Fujitsu discussed a similar approach in a late paper at the recent Design Automation Conference (Kawato *et al.* 1982). Fujitsu's approach converts designs (described in an RTL language called "DDL") first into a table, and then into successively specific circuits. Although it is presently being configured for printed-circuit-board design, the technology-dependent part of the system (which uses a library of standard TTL SSI/MSI devices) could presumably be replaced by LSI/VLSI "macro functions."

At Bell Laboratories, work is ongoing to perfect what is termed an "IC module compiler"—a program that would convert a file that describes the interconnection of parameterized

Information for Contributors

VLSI DESIGN welcomes contributed articles thatadvance the understanding, design and utilization of very large-scale integrated circuits (VLSI). Areas f interest include, but are not limitd to, the following:

1. Novel VLS architectures and design approaches.
2. Computer-aied design (CAD) and designutomation (DA) tools for IC:
3. VLSI testig, and design for testability.
4. New devicetechnologies, especially in reition to chip-design processes.
5. VLSI devie manufacturing, including vericaly integrated facilities ansilicon foundries.
6. The econoics of VLSI manufacturing.
7. The size an trends of VLSI markets.
8. Cooperative VLSI research and development programs, in the U.S. (*e.g.*, VHSIC) and abroad.
9. VLSI research programs at the universities
10. New teachig methods for VLSI designers.

If you particoate in this exciting field, and if youwould like to:

(a) raise the geeral level of understanding in te technical community, and
(b) raise your tatus among your peers,

please send a omplete manuscript (2000 to 4000 vords plus illustrations) or a 250- o 500-word abstract (typed, *triple- or double-spaced, on one side of each sheet*), to Jerry Werner, Editor-in-Chief, *VLSI DESIGN*, P.O. Jox 50518, Palo Alto, CA 94303-0518. Or call us at (415) 966-8340.

Please note hat *VLSI DESIGN* usually accepts only original, previously unpublished manuscripts.

VLSI DESIGN
P.O. Box 50518
Palo Alto, CA 94303-0518
(415) 966-8340

modules (counters, registers, multiplexers) into an input file for a logic simulator (Elias and Wetzel 1982). Although the program has not been used to create IC layouts, it should be compatible with Bell Labs' automated layout system, LTX, says researcher Norm Elias. He says that the IC module compiler "has the flavor of a silicon compiler."

The universities have also been active in logic synthesis. Researchers at Carnegie-Mellon University (CMU) recently reported (Hafer and Parker 1982) their effort to "synthesize" the logic of a Digital Equipment Corporation PDP-8/E computer from a high-level description in a language called "ISP." CMU's synthesized design required 84 SSI/MSI integrated circuits, 20 more than the number needed by an actual PDP-8/E.

That extra component count may be acceptable in some PC-board design environments, but may *not* be acceptable in a silicon compiler, where chip performance, power consumption, and circuit area are all vitally important.

Besides circuit size, logic synthesis may have other drawbacks in a silicon-compiler environment. "If you go down to gates, you've thrown away all the implied structure about what you're trying to design," says Johannsen. "If you're doing register-transfer operations, you know that could fit into a datapath, whereas if you're doing logic decoding that's a PLA or Weinberger array. If you flatten everything out to gates, all you have is gates," he concludes.

Layout-Generation Tools

The placement of the word *silicon* in front of the word *compiler* implies that an "ideal" tool would determine the topology of an IC. As Figure 2 shows, many tools can create IC-mask descriptions. The tools that show up in silicon-compiler sessions at technical conferences are often more accurately described as module compilers, silicon assemblers, or layout-modeling languages.

The first category, module compilers, includes PLA/Weinberger array generators, as well as programs such as MIT's "Data Path Generator" (Shrobe 1982), Stanford's SLIM (Hennessey 1981) and Brown University's SLAP (Reiss and Savage 1982). Caltech, too, uses module compilers in its new SCULPTOR system. These module compilers will generate specific blocks of an IC layout, but the blocks must be interconnected and output buffers/pads added to complete the design process.

Silicon assemblers and layout-modeling languages may be able to specify a larger fraction of a custom IC layout than can module compilers, because their input is at a relatively low level: it is either a textual description of "parameterized" cells (see *Designer's Corner* in this issue) or structures, or (as in the case of Stanford's LAVA language (Mathews *et al.* 1982)), a combination of electrical and topological details. These tools (along with their counterparts in the graphics realm—symbolic layout with compaction) also are often *general* in nature, unlike module compilers, which have specific design targets.

* * *

"When FORTRAN was originally implemented, it gave relatively poor results in terms of code space and speed. Once we understood it better–got it refined–most people came to realize that, for large programs, a good global-optimizing FORTRAN compiler will generate better code than hand-code. That's my desire for a silicon compiler."

—C. Rupp, Digital Equipment Corporation

"The funny thing is that any time there's a discussion about silicon compilers, I get asked to be a part of it. And the first thing I say is I don't believe in them—very much, at any rate. I think they're interesting but not all that useful yet."

—H. Shrobe, M.I.T.

"The total [silicon] compiler is not around the corner–it requires a conceptual breakthrough to make it a reality."

—B. Murphy, Bell Labs

Acknowledgements

Thanks to Gary Goates of the Boeing Aerospace Company for his helpful discussions.

References

Ayres, R. 1979. "Silicon Compilation—A Hierarchical Use of PLAs," *Proceedings of the 16th Design Automation Conference.*

Bergmann, N. June 1982. "Software Support for F.I.R.S.T." Edinburg University internal report.

Darringer, J.A., W.H. Joyner, Jr., C. L. Berman, and L. Trevillyan. July 1981. "Logic Synthesis Through Local Transformations," *IBM Journal of Research and Development.*

Elias, N., A. Wetzel. 1982. "The IC Module Compiler, a VLSI Systems Design Aid," to be presented at the *International Circuits and Computers Conference*, New York, NY.

Gray, J.P. 1979. "Introduction to Silicon Compilation," *Proceedings of the 16th Design Automation Conference.*

Gray, J.P., I. Buchanan, and P.S. Robertson. 1982. "Designing Gate Arrays Using a Silicon Compiler," *Proceedings of the 19th Design Automation Conference*, Las Vegas, NV.

Hafer, L.J. and A.C. Parker. February 1982. "Automated Synthesis of Digital Hardware." *IEEE Transactions on Computers*, Vol. C-31, No. 2.

Hennessey, J. Second Quarter 1981. "SLIM: A Simulation and Implementation Language for VLSI Microcode." *LAMBDA* (now *VLSI DESIGN*).

Johannsen, D.L. 1981. "Silicon Compilation," Technical Report 4530, Ph.D. thesis, California Institute of Technology.

Johannsen, D.L. 1979. "Bristle Blocks: A Silicon Compiler," *Proceedings of the 16th Design Automation Conference.*

Kawato, N., T. Uehara, S. Hirose, and T. Saito. 1982. "An Interactive Logic Synthesis System Based Upon AI Techniques." *Proceedings of the 19th Design Automation Conference*, Las Vegas, NV.

Luhukay, J. and W.J. Kubitz. 1982. "A Layout Synthesis System for nMOS Gate-Cells," *Proceedings of the 19th Design Automation Conference*, Las Vegas, NV.

Peskin, A.M. 1982. "Toward a Silicon Compiler," *Proceedings of the 1982 Custom Integrated Circuits Conference*, Rochester, NY.

Reiss, S. and J. Savage. 1982. "SLAP: A Silicon Layout Program." To be presented at the *International Circuits and Computers Conference* (ICCC), New York, NY.

Rupp, C.R. 1981. "Components of a Silicon Compiler System," *Proceedings of the VLSI 81 Conference*, Edinburgh, Scotland, U.K.

Shrobe, H.W. 1982. "The Data Path Generator," *MIT Conference on Advanced Research in VLSI.*

Siskind, J.M., J.R. Southard and K.W. Crouch. 1982. "Generating Custom High Performance VLSI Designs From Succinct Algorithmic Descriptions," *MIT Conference on Advanced Research in VLSI.*

Van Cleemput, W. 1979. "SDL, A Language for Describing the Structure of Digital Systems," Digital Systems Laboratory, Stanford University.

Weinberger, A. December 1967. "Large Scale Integration of MOS Complex Logic: A Layout Method," *IEEE Journal of Solid State Circuits*, Vol. SC-2.