# Guest Editors' Introduction: RTL to GDSII—From Foilware to Standard Practice

**Dwight Hill**
Synopsys

**Andrew B. Kahng**
University of California, San Diego

**CHIP IMPLEMENTATION,** from a RTL description in a language such as Verilog or VHDL to tapeout in the form of mask tooling data, is the process by which product concepts can become high-value realities. Designers often view chip implementation as comprising logic synthesis, placement, and routing (SP&R), which are all classic point tool arenas. Commercial logic synthesis tools have been in wide use for more than a decade, and practical place-and-route systems date back more than 30 years. Complemented by analysis point tools (for parasitic extraction, crosstalk and delay calculation, and static timing-noise-power verification), SP&R implementation tools have allowed design teams to devise flows that spiral in from RTL timing estimation and global interconnect planning to a final detailed layout. This spiraling entails successive refinement: Logic moves from functional descriptions to optimized circuits, spatial embedding of devices and interconnects become codified, and estimations of parasitics prove ever-greater accuracy.

## Not a tool, but a flow

Most of the work in RTL to GDSII has burgeoned in industry, not academia. This is in part a result of the complicated and somewhat messy nature of the problem. To demonstrate results, a flow needs competitive technology in several areas, which involves more software than a typical graduate student group can manage and maintain. Much of the complication stems from the flow's multiple representations. The unified implementation flow is sometimes called *physical synthesis*, or more often, *RTL to GDSII*. But both of these names oversimplify the actual process. It is not just RTL logic that enters the flow and GDSII that comes out. It is true that in a typical flow, the system accepts a collection of RTL files for entry. But to begin synthesis, the tool typically needs many libraries that describe available cells and underlying technology constraints. For example, timing libraries describe the delay properties of the cells. These libraries usually begin in textual DCL (delay calculation language) or .lib formats, but are often compiled into binary representations that are more convenient for the tools. Physical libraries describe the geometry of individual logic cells, as well as I/O buffers and locations. The physical models of the logic cells can originate in GDSII, but GDSII itself is insufficient to process them. Because GDSII does not define the properties of its layers, other files are necessary to interpret the GDSII geometry. Other technology files describe the available layers, the routing and placement rules, and the site array types (even for standard-cell chips). And most RTL-to-GDSII flows actually begin after some other tool has created a valid floorplan. The floorplan provides the context for placing the logic, including the I/O locations, RAMs, and other blocks, both movable and fixed.

Adding to this set of inputs are the timing constraints that define the circuit's specific timing properties. Although flow diagrams often overlook them, the timing constraints can represent almost as large a data volume as the original RTL. Today, most tools accept timing constraints in the Synopsys Design Constraint (SDC) language, a dialect of Tcl. Typically, timing constraints start out small with just a few clock definitions. But the presence of derived and gated clocks, false paths, and multicycle paths tends to bulk them up quickly. If the logic design starts out hierarchical (as most do) and is then flattened, the constraints tend to multiply when a hierarchical pin is elided, because constraints bound to hierarchical pins might need to be mapped onto many lower-level objects.

The complexity of timing constraints reflects the overall objective of the design implementation process, *timing closure*. There's no point in making a design log-

0740-7475/04/$20.00 © 2004 IEEE Copublished by the IEEE CS and the IEEE CASS

ically correct or 100% layout-rule correct if it doesn't meet timing constraints. Traditionally, static timing analysis was run at the beginning of the process at a milestone called RTL handoff, and at the end of the flow at a mask sign-off milestone. But this has proven inadequate because the physical realization process must modify not only the logic, but also the way that timing is abstracted. All RTL-to-GDSII systems have embedded timing analysis, and much of the power, sophistication, and complexity (not to mention much of the CPU time and memory cost) of modern design implementation flows comes from this timing analysis operation. But even the best timing analysis, working on complete constraints, is useless without accurate timing abstractions; this has been the rub.

## Spiraling out of control

Today's RTL-to-GDSII implementation suites owe their genesis to the nonconvergence of traditional flows. Out-of-control spirals arose when abstractions and optimization objectives were no longer valid. Before interconnect delays dominated timing, logic synthesis could focus on stage counts as an accurate proxy for cycle time. Before higher-order interconnect moments became significant, placement could use the lumped capacitance of a heuristic Steiner tree as a quick estimate of gate load delay. Before the insertion of global repeaters substantially changed the gate-level netlist, this could be treated as an engineering change order (ECO) step in the flow. But with the size and complexity of today's chips, these simplifying assumptions are no longer reliable.

Today, many more loops need closing: Crosstalk, substrate coupling, and power and thermal phenomena all affect the accuracy of circuit timing and signal integrity. Reliability and manufacturability—including electromigration, antenna checks, dummy fill for planarization, and process loading—are also crucial, even at the beginning of implementation. The need for increasingly close abstraction of nanometer-scale physical phenomena has motivated unifications across the synthesis-analysis-specification loop. Estimations must go ever deeper into the design process (to the level of a gate-level netlist, placement, and detailed routing, for example) even while performing early chip planning.

## Three categories of prediction

We observe that *predictability* is the cornerstone of any convergence-improving approach: It is the heart of scalable, top-down design. There are really only a few practical ways to achieve predictability. The first, and generally the quickest, early form of prediction is *statistical* in nature. That is, it avoids calculations that depend on the specific topology of the design under development. However, statistical methods such as wire load models, choke point analysis, occupation functions, and Rent's rule cannot deliver the requisite accuracy. This is because cycle times and noise coupling failures are maximum or extreme (as opposed to average or ensemble) statistics, and hence difficult to estimate.

A second form of prediction is *constructive* and *iterative* in nature: It estimates by doing, as in quick placement or trial routing. In common practice, these predictors tend to use history as their guide. As the tool runs, it tracks previous estimates of logic, placement, and routing congestion, and uses the estimates to formulate its next solution. In a theoretical sense, constructive predictions are undesirable because they "predict by doing"; that is, they actually run the tool to predict what it will do. This approach does not fundamentally improve the flow's scalability. However, it is perhaps today's most commonly used method, and operates both internally within monolithic tools (which are large tools, each from a single vendor) and more visibly, by running point tools in an iterative loop. Systems that expedite this approach play a valuable practical role because they make the engineering job simpler. By automatically invoking, managing, and calibrating multiple point tools, these systems significantly reduce the required engineer hours and overall elapsed time.

Finally, a third way to achieve predictability is to *assume and enforce*, which involves making a specific assertion about a physical property, and then constructing the circuit so as to guarantee that this property will remain valid throughout the design process. Examples of this method include wire planning, noise-free fabrics, and constant delay. This method generally pertains only to monolithic tools, because otherwise, it would be difficult for one point tool to make an assumption if a subsequent point tool was not set up to enforce it.

## From unity, strength

The past several years have seen successful efforts to combine implementation and analysis facilities into interlocking amalgams that deliver better results than any sequence of independent point-tool runs. An increasingly common paradigm is based on a persistent *performance analysis backplane*. From RTL floorplanning down to detailed routing, analyses can continually and incrementally obtain estimated parasitics from

synthesis results, assess performance of the current design implementation with respect to requirements, and then drive synthesis with updated constraints. Phrases such as "common timing engine" and "unified back-end database" reflect this idea. In such a taxonomy, the constructive approach results from the weaknesses of the statistical approach: Perfect optimization objectives can't drive isolated flow stages, and there must be many iterations between analysis and synthesis. Given this context, trial SP&R with an incremental analysis backplane helps the best remaining alternative reduce the pain of iteration. Added flexibility stems from overlapping degrees of freedom that can help solve a given problem. For example, a combination of placement, net ordering, shielding, spacing, driver sizing, and repeater insertion might be able to fix a crosstalk violation, although any one of these steps working in isolation, could not.

## Two perspectives on the flow

The two articles in this special section reflect two approaches to transforming RTL (and other) inputs into high-quality placement and routing. Both approaches are embodied in industrial systems used for several years to tape out many chips. The first article, "An Integrated Environment for Technology Closure of Deep-Submicron IC Designs" by L. Trevillyan et al., provides a relatively broad context for a set of tools integrated around a common data model and timing analysis engine. In this case, all the tool development took place internally at IBM. The article emphasizes the use of a common timer, and discusses the tradeoffs necessary for synthesis and placement to work together. The focus is on the synthesis-placement connection.

The second article, "Crosstalk-Aware Placement" by J. Lou and W. Chen, offers a peek at some of the detailed analysis necessary for a physical synthesis placer to work effectively amidst routing congestion. Lou and Chen emphasize the prediction of routing congestion using a constructive and iterative approach. Together, these articles—and indeed, the current body of work toward RTL-to-GDSII design flows—illustrate the growing trend toward integration and interdependency in EDA.

Several other technology descriptions could have potentially joined the two that we have selected. However, it appears that at this point, vendors of commercial RTL-to-GDSII EDA tools are reluctant to disclose differentiating technical details that would adequately appeal to the *IEEE Design & Test* audience. Of particular interest, in our opinion, are systems that aren't classical point tools per se, but instead provide utilities, metrics collection, data mining, and expert user knowledge that serve to assemble, control, evaluate, and coordinate tools from other vendors. These systems emphasize integration of the multiple data models involved and consistent handling of design components, such as libraries and performance constraints. Examples of this widely deployed, if less publicized, industrial practice include those of suppliers such as Runtime Design Automation (http://www.rtda.com), Manhattan Routing (http://www.mri-nyc.com/mri), ReShape (http://www.reshape.com), and arguably Tera Systems (http://www.terasystems.com), along with internal CAD groups and ASIC design centers ranging from Silicon Access and Cisco to Fujitsu and LSI Logic. These obviously represent only a tiny sample of the many players in the field. Google currently finds more than 7,000 links on the topic "RTL GDSII" (plus many more on "RTL-to-GDSII" and "physical synthesis"), including several links from each of the major (and most of the smaller) EDA companies.

## Still to come

Although there has been tremendous theoretical and practical progress in RTL-to-GDSII chip implementation, more challenges lie ahead. Many of these are a direct consequence of the dramatic increase in electrical and physical complexities inherent in sub-90-nanometer fabrication. New nanometer-scale physical issues are piling on at an alarming rate. These include printability, manufacturing variability, thermal reliability, resistive effects, nonuniformity of permittivity in ultrathin dielectrics with conformal barrier layers, and grain size and scattering impacts on conductor resistivity.

Another set of issues comes from the designs themselves: They are orders of magnitude larger and faster than they were only a few years ago. Design teams are not growing as fast as their designs, and they are increasingly spread across different geographical regions, and even different countries. Thus, the RTL-to-GDSII flow that works for today's chips is already nearing the breaking point for 90-nm designs. Yet with each generation of chips, design teams have come to depend increasingly on this flow.

**How will designers** meet design implementation convergence and quality goals in the long term? The key to progress will probably depend on continued advances in the predictor technologies we discussed

earlier: statistical, constructive, and assume and enforce. Of course, radically new techniques could also allow the normally disparate areas of SP&R to coordinate. As with the other aspects of the *International Technology Roadmap for Semiconductors*, there are some brick walls ahead (so to speak) that will necessitate continued fundamental innovations. It is an exciting time indeed. ■

**Dwight Hill** is a principal engineer in the physical synthesis department at Synopsys. His research interests include timing constraint analysis and timing closure, VLSI layout synthesis, interactive CAD systems, and field-programmable gate arrays. Hill has a BS in computer engineering from the University of Illinois, and an MS and a PhD in electrical engineering from Stanford University. He is a senior member of the IEEE.

**Andrew B. Kahng** is a professor in the Departments of Computer Science and Engineering, and Electrical and Computer Engineering at the University of California, San Diego. He also leads the Calibrating Achievable Design activity, which includes the Bookshelf project, in the Marco Gigascale Silicon Research Center. His research interests include VLSI physical layout design and performance analysis, combinatorial and graph algorithms, and large-scale heuristic global optimization. Kahng has an AB in applied mathematics (physics) from Harvard College and an MS and a PhD, both in computer science, from the University of California, San Diego. He is a member of the IEEE and the ACM.

■ Direct questions and comments about this special issue to Andrew B. Kahng, Departments of CSE and ECE, Univ. of California at San Diego, La Jolla, CA 92093-0114; abk@ucsd.edu.