BY NEERAJ KAUL AND STEVE KISTER • SYNOPSYS INC

Using IC prototyping to optimize design implementation

MAKING THE RIGHT TRADE-OFFS DURING IC PROTOTYPING CAN MINIMIZE RISK AND ULTIMATELY SAVE TIME AND MONEY.

As IC-design size continues to escalate, time-to-market windows tighten, design requirements become more stringent, and device geometries shrink to nanometer proportions. Because of these constraints, employing prototyping early in the design cycle is gaining in significance. Problems lurk within these increasingly complex ICs. These problems can kill a project if you find them too late in the design cycle. Although IC prototyping solves these problems, there are as many definitions of this approach as there are techniques for its execution. This article focuses on defining prototyping for modern designs, including key requirements and useful techniques for putting prototyping into practice.

It can be costly to learn late in the design cycle that a design cannot meet your specifications. At a minimum, the schedule will slip, necessitating additional engineering time and lengthening time to market. The worst possible outcome is a canceled project, forfeiting months of work from many groups and missing the market window. IC prototyping addresses these risks. IC prototyping is similar to physical implementation in that it includes placement, routing, and optimization. However, its objective is to determine the feasibility of implementing a design within the given specifications. Early in the development cycle, a wide range of possible implementation approaches is available. Prototyping allows design teams to confirm that a chosen approach is viable. By finding potentially fatal problems early in the development process, design teams and managers can assess changing design specifications and make the architectural, functional, and die-size changes necessary to save the project from an untimely death.

Chip-design specifications fall into the primary categories of functional, timing, power, and area. The functional category defines what the design is supposed to do. The timing category describes how quickly the design should perform its functions, including such specifications as clock-cycle times. The power category defines how much power a chip can consume while in various operating states, from idle through any number of functional states. Area defines the size of the chip, which relates to cost; larger chips cost more to produce. Physical implementation directly affects area, timing, and power. If a design team cannot meet one or more of these specifications,

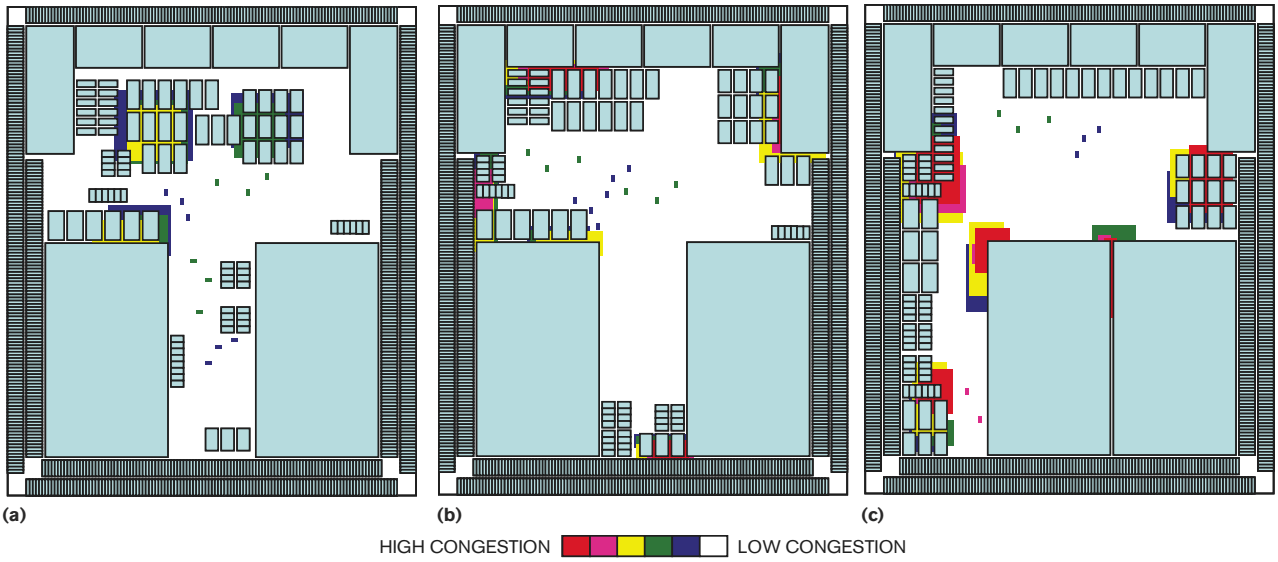


Figure 1 The congestion maps for three scripted-prototyping runs show a good implementation strategy (a), a slightly worse one (b), and a poor one (c).

the team must modify the physical or logical implementation or even the overall function of the design.

The key to IC prototyping is doing enough to find gross problems without expending the time and effort to perform the full implementation. Designers have a finite amount of time to complete a design and, therefore, limited time for prototyping. Moreover, designers can take many approaches to design implementation. Faster placement, routing, optimization, and analysis during prototyping permit designers to quickly explore many implementation strategies. This article describes the key process requirements for prototyping, as well as its objectives and benefits for netlist and constraint handling, placement, power and signal routing, and analysis.

PROTOTYPING-PROCESS REQUIREMENTS

A successful IC-prototyping strategy is one that designers can complete quickly so that they can assess the feasibility of the design as early in the development cycle as possible. This strategy requires designers to start working early with design data, which may be incomplete or inaccurate. Thus, prototyping tools must tolerate this data and help designers identify problems with the data. Proving feasibility does not mean completing an error-free implementation. Rather, prototyping shows designers the number and difficulty of the problems they will face during detailed implementation.

How much time you allocate to IC prototyping depends on your overall schedule. If the design-implementation schedule requires a few weeks, prototyping should take no more than a few days. If implementation requires several months, prototyping should take just weeks. To minimize the time to complete prototyping, designers must make trade-offs between completeness and accuracy. These trade-offs necessitate the use of analysis tools to monitor the predictability of the prototype.

Support for scripting is also important. Creating trial layouts should take minimal time to allow engineers to focus on analyzing results and determining next steps. An efficient IC-prototyping process enables designers to quickly generate multiple trial layouts. Using this analysis, designers work with one or more of the trial layouts or generate additional layouts. Once a trial layout shows promise, designers can quickly make incremental updates and perform what-if analysis.

Table 1 and **Figure 1** illustrate the results of a scripted-prototyping approach. One run of the prototyping script creates several floorplans for the design. The script also produces the HTML (hypertext-markup-language) table. Each row of the table is for one floorplan of the design. The table makes it easy for the user to see which floorplan is the best and whether any of the floorplans should become the basis for the final floorplan of the design. The **figure** also shows routing-congestion maps for three of the floorplans. The maps clearly indicate that the floorplan gets progressively worse (**Figure 1a** and **1b**) as an implementation strategy, and the map in **Figure 1c** should not become the basis for the final floorplan.

EARLY NETLIST AND CONSTRAINT HANDLING

In IC prototyping, designers start with design data that they acquire early in the process. This data likely has errors, such as missing library components; netlists referencing pins on non-

TABLE 1 SCRIPTED-PROTOTYPING RESULTS

		Setup time (nsec)				Virtual in-place-optimization setup time		Congestion*			Links
Run	Objective	Worst negative slack	No. of violating paths	Sum of negative slack of all violating timing paths	Timing	Worst negative slack	Sum of negative slack of all violating timing paths	Total	Maximum/ percentage	Wire length (microns)	Place-and-route summary report/run
No.	Hierarchical gravity/timing driven	−21.54	5048	−10,596	Report	−2.25	−1588	21,767	29/0.2	56,447,834	Place and route/log file
1	Hierarchical gravity	−19.56	4078	−5886	Report	−2.1	−1412	21,831	28/0.2	56,453,440	Place and route/log file
2	Hierarchical gravity/macros on edge	−21.27	5476	−18,410	Report	−3.07	−1980	85,565	102/ 0.3	76,652,906	Place and route/log file
3	Congestion	−19.98	4879	−6812	Report	−2.06	−1324	28,422	113/0.2	55,214,522	Place and route/log file
4	Hierarchical gravity/ congestion driven/timing driven	−27.08	4368	−12,559	Report	−2.14	−1247	87,588	144/0.3	62,468,571	Place and route/log file
5	Grouping/ macros on edge	−40.07	8742	−37,612	Report	−2.66	−2117	435,761	859/1.6	108,781,409	Place and route/log file

*Map

existent cells; and timing constraints that reference nonexistent starting points, endpoints, or even clocks. To complete detailed physical implementation, designers must resolve these errors before continuing with the design. Prototyping allows designers to proceed with the design even during the detection and reporting of errors. Designers can handle missing components as black boxes, ignore extra cell pins, and omit the application of timing constraints. Reporting is important because it allows designers to assess both the type and the quantity of the errors. The prototyping tool can also provide utilities to help the designer temporarily correct errors associated with the bad data.

Starting physical prototyping at the RTL (register-transfer level) or with RTL data that you do not map to a real gate-level netlist is problematic. The most common risk factors with these netlists are that they do not accurately reflect the size of the final netlist and that their lack of gates makes the prediction of congestion dubious. Results may appear to achieve timing closure, but, when you map the results to physical gates, the timing differs. The best approach for physical prototyping is to use the available netlist with the highest quality input—ideally, one you generate using the target physical-cell library to enable physical placement and routing. Some synthesis tools require the use of statistical or custom wire-load models to estimate interconnect loading, whereas others can read information from physical-cell libraries to more accurately predict area and interconnect loading on a design-by-design basis. This approach enables accurate assessment of an implementation strategy's feasibility.

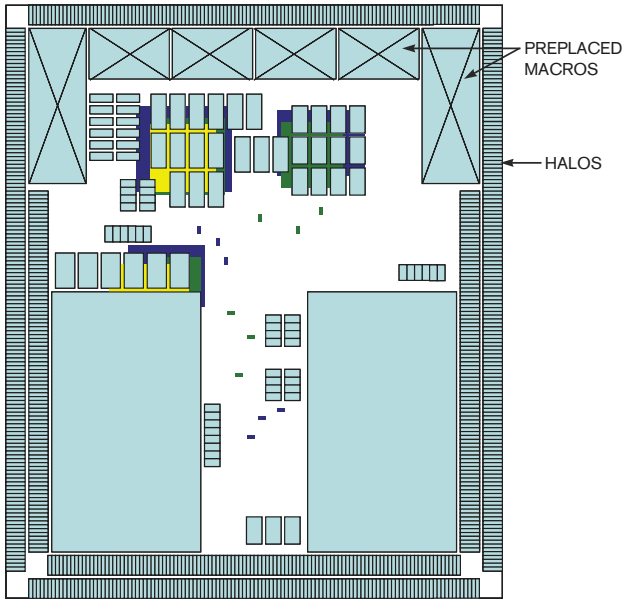


Figure 2 You can add halos to macros with congestion near the macros' edges. The two large macros (top left and top right) are good candidates for fixing in position before performing more trial runs.

PROTOTYPE PLACEMENT

The primary objectives of IC-prototype placement are to determine the best placement of the principal floorplan elements, such as I/Os, pins, and hard and soft blocks, and to quickly provide an estimate of the design's performance that you base on this floorplan. Today's designs typically have tens or hundreds of hard macros. Although they vary greatly in size, all hard macros represent routing obstructions that can create routing congestion. Given these constraints, prototype-placement algorithms must automatically and simultaneously handle hard macros and standard cells, considering routing congestion, timing, and voltage domains. Initially, designers manually place and fix the locations of critical macros and then assess the routing congestion and timing of multiple placement approaches. Some hard-macro placements work well, whereas others require refinement. The placement tools must accommodate incremental placement of the remaining

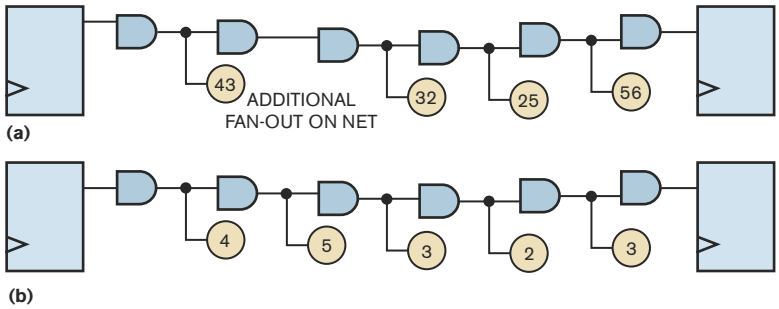


Figure 3 These paths have the same value of negative slack. Large fan-outs contribute to a significant amount of delay (a). You can easily fix it by buffering to reduce the nets. Small fan-out nets do not contribute to significant delay (b).

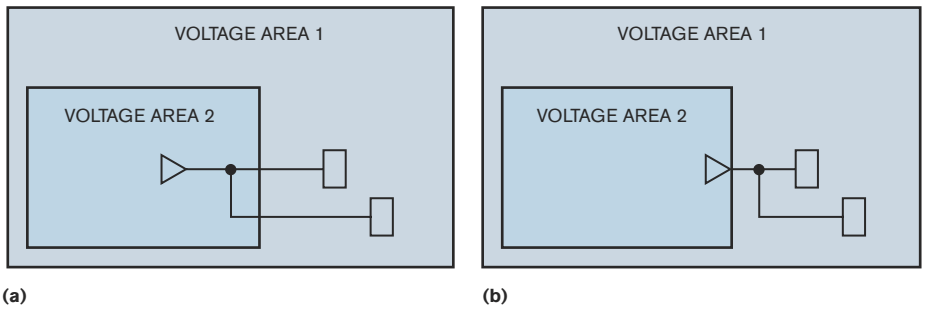


Figure 4 The output of the level shifter in Voltage Area 2 must drive long wires that cross Voltage Area 2 (a). Buffers can help only if you place them in Voltage Area 1. Placing a level shifter near the edge allows you to add buffers close to the level-shifter output and simplifies the power connections to the level shifter (b).

macros, rather than require designers to do over the placement from scratch.

In the early stages of IC prototyping, the channels between hard macros often contain congestion, due both to the macros' being too close to each other and to the presence of standard cells in the channels. You can prevent these phenomena from occurring by using IC-prototype-placement tools that support hard-macro "halos" that reserve space for routing connections surrounding the edges of the block. If the placer cannot honor halos, the designer must manually move the hard macros and add placement blockages within the channels. A few iterations may be necessary before you arrive at the appropriate channel sizes. **Figure 2** provides an example of the information designers can see in an IC prototype to help guide refinement of an implementation strategy.

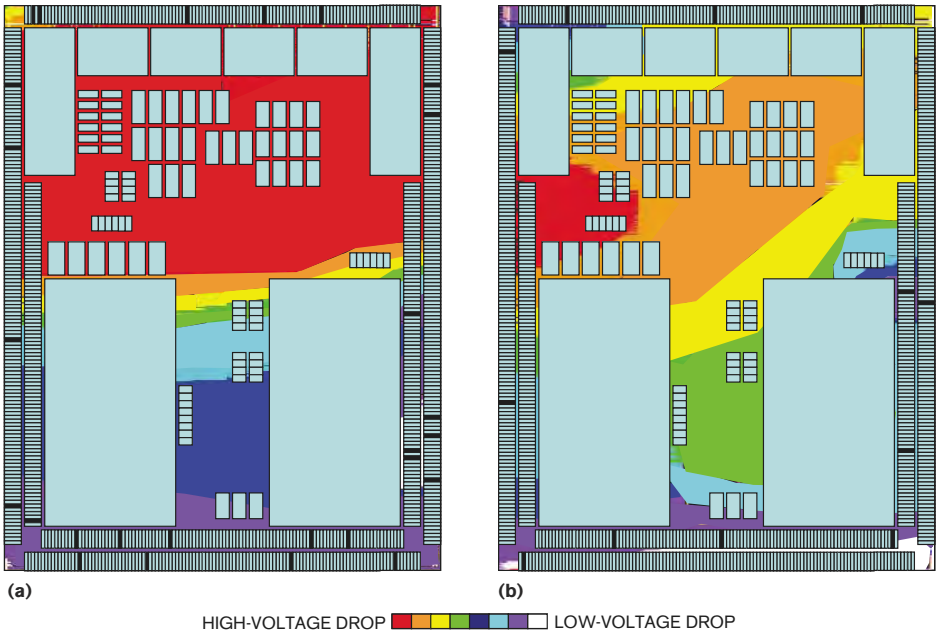


Figure 5 Placing power pads in groups or in poor locations produces poor voltage-drop characteristics (a). A design with fewer power pads (black) has better voltage-drop characteristics (b).

Netlists you develop early in the design cycle often have many high-fan-out nets; some, such as clocks, are timing-critical, and others, such as asynchronous-reset nets, are not. Physical-layout designers typically optimize these nets during detailed implementation. Prototype-placement algorithms

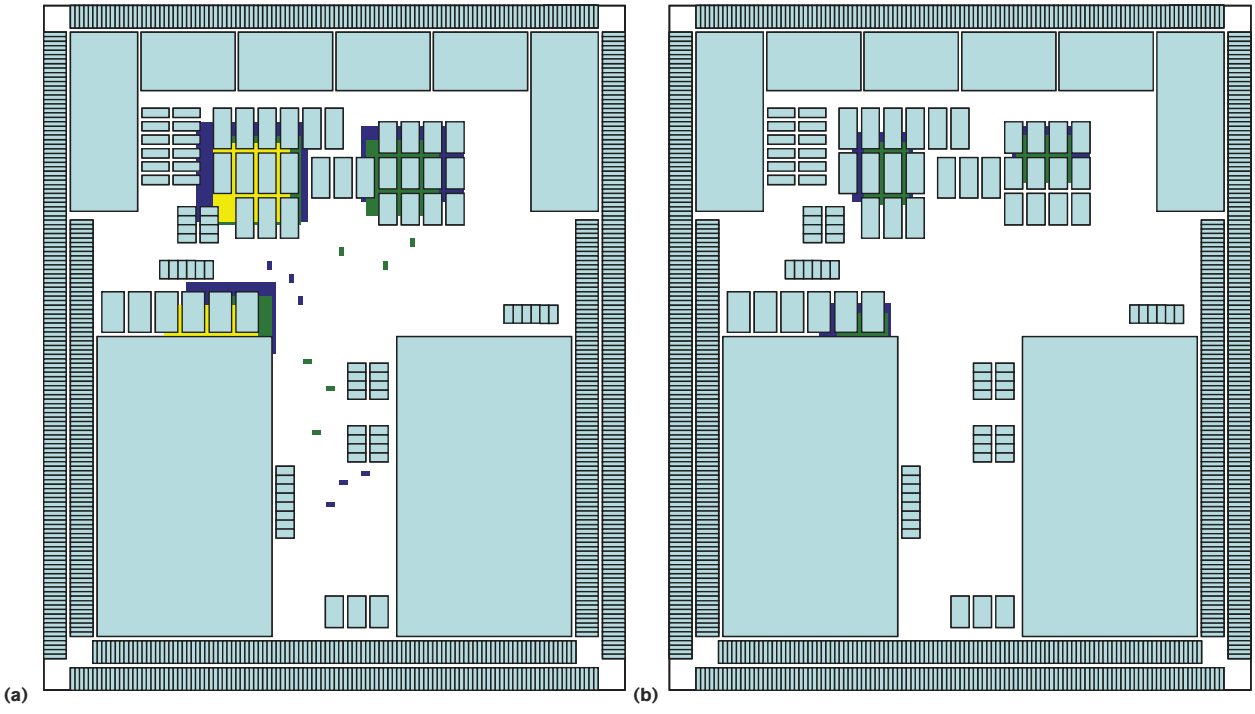


Figure 6 The congestion maps of the same design show more congestion (a) and less congestion (b). The prototype router spends less effort to remove congestion than the implementation global router does. The designer can adjust the floorplan to reduce congestion. Thus, detailed implementation routing is more likely to complete without problems.

should ignore these types of nets. If they cannot, the standard-cell logic is likely to create unnecessary congestion. Moreover, netlists you develop early in the cycle pose a problem for timing-driven placement algorithms. The netlists often contain many violating timing paths; fixing these violating paths can sometimes be difficult. Most timing-driven placement algorithms place equal weight on both those paths that are difficult to fix and those paths that are easy to fix, making it difficult to distinguish between them.

One approach to these problems, albeit time-consuming—and, hence, costly—is first to run placement focusing on alleviating congestion, next to perform in-place optimization, and then to run timing-driven placement. A more elegant approach is to use a placement engine that can execute virtual optimization on the input netlist, eliminating all but the toughest of violating timing paths. This approach produces a placement scheme that you accomplish by optimizing the hard-to-fix paths. **Figure 3** compares an easily fixed timing path with a more challenging one.

Many of today's designs have multiple voltage domains within the core area. To accommodate the logic that connects to a voltage domain, a designer must create voltage areas within the core area. But what is the best location for these voltage areas, and how large should they be? Prototype-placement algorithms must be aware of voltage domains and assemble logic that connects to one domain. The results allow designers to quickly define the size and location of voltage areas within the chip.

If the placement algorithm is unaware of voltage domains, the designer must predefine voltage areas before placement using regions and then assign logic for each voltage domain to its region. This approach forces designers to place voltage domains in the predefined regions. As a result, designers must iterate through a number of placement runs to refine the region sizes.

Multiple voltage domains also require level shifters, which are placed on nets that connect one voltage domain to another. Designers place notoriously slow level shifters at the edges of voltage areas to optimize timing (**Figure 4**). Prototype-placement algorithms must be able to recognize and appropriately place level shifters. Again, an alternative, less-than-ideal approach is to use regions. This approach requires the designer to analyze the interface nets, create regions along edges of voltage areas, and then assign the various level-shifter instances to the appropriate regions.

For large designs, logic designers commonly complete RTL coding and synthesis of some blocks before others. The nonsynthesized blocks are black boxes, because the designer knows nothing but the boxes' interface. IC prototyping supports the placement of black-box shapes, allowing the shape to vary but maintaining enough internal area for the expected "child" cells, once the designer synthesizes them.

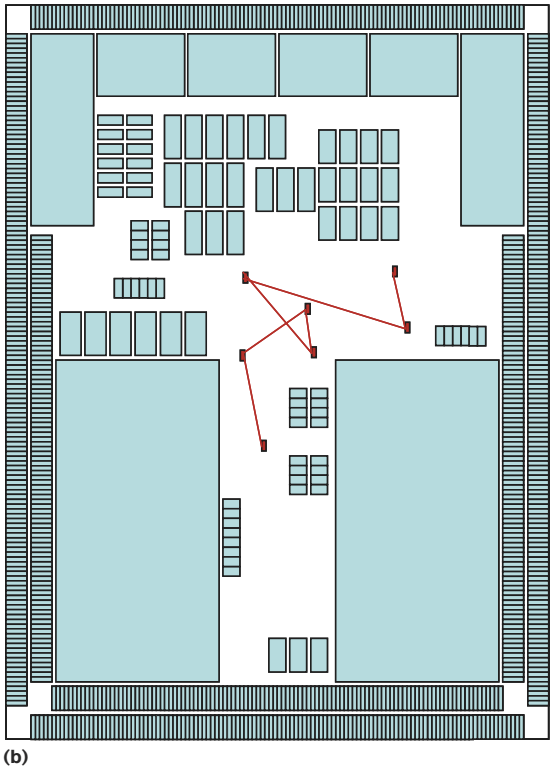
A placement tool can allow virtually flat placement of available cells during the placement and shaping of the black boxes. This approach enables block designers to proceed with the design in parallel with the development of the contents of the black boxes. An alternative strategy is to create dummy hard macros for the nonsynthesized blocks. The designer sets the size and shape of the macros and the layers and locations of pins. The trade-off in this case is that the approach forces the rest of the cell placement to work with the predefined black boxes.

PROTOTYPE-POWER ROUTING

A design's power network consumes routing resources that you could otherwise use for signal routing. IC-prototype power-network generation and analysis should rapidly generate a placeholder power mesh to accurately account for routing resources that the network consumes. This generation and analysis should also create a power network for the design that meets—but does not exceed—voltage-drop and electromigration requirements. You need to know which metal layers you should use for the power routing, how wide the wires should be, and what the distance between power-routing segments should be. Designers often rely on rule-of-thumb guidelines from power-design gurus to form the power mesh, resulting in an overdesigned mesh; that is, it easily meets requirements but at the expense of consuming excessive amounts of routing

Start-Point Name	EndPoint Name
pratichi/cnt_instrn_reg_31/CLK	isha/STACKFSM_in_reg_2_2/DO
pratichi/cnt_instrn_reg_31/CLK	isha/STACKFSM_in_reg_2_7/DO
prem/Oprnd_A_reg_2_6/CLK	prem/Zro_Flag_reg2/D0
prem/Oprnd_A_reg_2_6/CLK	prem/Zro_Flag_reg5/D0
isha/STACKFSM_ot_reg_5/CLK	isha/STACKFSM_obf_reg_1_7/DO
isha/STACKFSM_ot_reg_5/CLK	isha/STACKFSM_obf_reg_1_9/DO
isha/STACKFSM_ot_reg_5/CLK	isha/STACKFSM_obf_reg_1_3/DO

(a)



(b)

Figure 7 Cross-probing from a list of violating timing paths (a) allows designers to see the topology of the path in the prototype layout (b).

resources. Using a prototyping-power-network generator enables designers to build efficient power structures.

Some power-implementation tools require that power meshes be physically correct before designers can analyze the voltage drop. For large designs with many cells and many hard macros, this requirement means days of work to prepare the power routing for analysis. This process is not conducive to quick prototyping. A faster approach is to employ a power-network-generation and -analysis tool that can infer connections based on proximity, allowing designers to route the primary mesh structures without performing detailed connections. This approach enables designers to review voltage-drop and electromigration characteristics earlier in the design cycle and even to analyze trade-offs of varying layers, widths, and pitches within the power mesh.

In addition to analyzing various mesh structures during prototyping, it is useful to analyze the voltage-drop and electromigration characteristics of a mesh with varying numbers and locations of power-supply-pad cells. Simply adding supply pads does not always produce improvements. For example, a supply pad behind a hard macro does not easily connect to the mesh and, therefore, cannot deliver a significant amount of current. Prototype-power planning should enable fast what-if analysis of supply-cell placements rather than require the designer to perform detailed connects. The what-if analysis may prove that fewer supply-pad cells are necessary than rules of thumb would predict. Using fewer supply cells produces smaller chips in pad-limited designs (**Figure 5**).

PROTOTYPE-SIGNAL ROUTING

IC-prototype-signal routing's primary objectives are to identify highly congested areas to assess the difficulty of routing an implementation strategy and to accurately predict detailed implementation routing to provide interconnect loading for early timing analysis. Macro placement and power structures often contribute to routing congestion during implementation. Given enough time, routing algorithms of implementation tools perform hundreds of passes in an attempt to route congested areas. During prototyping, it is useful to see the congestion to determine whether changes in the implementation strategy can eliminate the congestion. Reducing or eliminating congestion reduces the effort of completing routing, which, in turn, minimizes the risk of failure to route. IC-prototype routing quickly identifies congested areas and minimizes difficult routing problems (**Figure 6**). Once you minimize congestion, prototype routing should accurately reflect implementation routing, enabling accurate extraction of interconnect loading for use in performing timing analysis on the implementation strategy.

PROTOTYPE ANALYSIS

IC-prototyping tools should provide a comprehensive set of analysis functions that allows designers to identify and as-

sess data, routing, timing, and power-consumption problems. These analysis functions also provide a key means for physical-layout designers to communicate with front-end designers. Prototyping-analysis functions should find problems in the input data; quickly identify the big problems related to routing, timing, and power requirements; and facilitate communication between physical-layout and logic designers. The adage "a picture is worth a thousand words" certainly applies to analysis of a physical prototype. A comprehensive set of reporting functions couples with a rich set of visualizations to simplify and speed analysis.

Consider timing analysis. A timing analyzer for IC prototyping highlights major timing problems without requiring the user to run detailed optimization. It accomplishes this task by incorporating virtual optimization into the timer so that the report omits easily fixed timing paths. If you then enable cross-probing from the report to a layout view, you can quickly see the topology of the difficult-to-fix timing paths. This approach also facilitates communication between physical and logical designers. Viewing the report in addition to the path layouts helps logic designers identify missed, multicycle-path, or false-path definitions (**Figure 7**).

Although IC prototyping's objectives differ appreciably from those of detailed implementation, the two approaches have similar steps. Prototyping finds potentially fatal problems early in the development cycle. If prototyping steps complete quickly, designers can assess many potential implementation strategies in the time available for prototyping. To minimize the time to complete the prototyping steps, physical designers and tools must trade off levels of completeness but maintain enough accuracy to reliably find potentially fatal problems. The payoff is a streamlined implementation process, enabling faster design cycles and speedier time to market. EDN

AUTHORS' BIOGRAPHIES

Neeraj Kaul is an R&D-group director at Synopsys Inc (Mountain View, CA), where he has worked for more than eight years. He is responsible for design planning, feasibility, and high-capacity-design methodology. He received a master's degree and a doctorate in electrical and computer engineering in 1992 from Vanderbilt University (Nashville, TN) and a bachelor's degree in technical, electrical, and computer engineering from the Indian Institute of Technology in 1986. Kaul's focus in the EDA industry is timing analysis and optimization, simulation, statistical timing, and physical design.

Steve Kister is a technical-marketing manager at Synopsys (Mountain View, CA), where he has worked for 12 years, focusing on design planning. Kister joined Synopsys in 1995 and has been in the electronics industry for 27 years. He received a bachelor's degree in electrical-engineering technology from DeVry Institute of Technology (Phoenix) in 1979. His experience includes test engineering, physical design, library development, and applications engineering.