Shortening the Time to Volume Production of High-Performance Standard Cell ASICs

Coding guidelines for behavioral modeling and a process for generating wire load models that satisfy most timing constraints early in the design cycle are some of the techniques used in the design process for standard cell ASICs.

by Jay D. McDougal and William E. Young

As time-to-market pressures continue to increase, the need for shorter design cycle times is more urgent than ever. At the same time, the demand for high performance in standard cell ASICs is also increasing. These trends are expected to continue as customers look to get the most return on investment from the latest IC process technologies. Many of the design considerations needed to achieve these high-performance goals compete directly against achieving quick time to market.

A typical standard cell design process includes several iterations in which individual steps must be repeated to adjust for data determined at later steps. A common example of this is the need to redesign portions of the circuit when physical results such as extracted parasitics are fed back into a simulator and performance goals have not been met.

This paper presents methods that help reduce or even eliminate the need for design iterations by increasing the chance of "first time perfect" at each design step. First, we discuss the methods developed for each of the steps in our ASIC design process (see Fig. 1) to get shortened throughput time and reduced design iterations while still producing high-performance components. Next, we present the results of applying these methods to the design of a CMOS ASIC that is used in HP's X-terminal products.

Behavioral Modeling

Since high-level behavioral modeling is done very early in the design flow, the way in which the code is written can have a dramatic effect on the downstream processes. We have developed a set of hardware description language (HDL) guidelines that, if followed, will reduce throughput time for the entire design flow. These guidelines were collected from several designers within HP and have been updated and modified as areas for improvement have been identified in each of the downstream processes.

The HDL coding guidelines include sections on clocking strategies, block hierarchy and structure, flip-flops and latches, state machines, design for test, techniques for ensuring consistent behavioral and structural simulation, and Synopsys-specific issues (Synopsys is an automatic design synthesis tool from Synopsys, Inc.).

Following these guidelines allows us to avoid many time-intensive steps later in the design process such as name remapping, race analysis, and iterative structural and behavioral simulation.

Synthesis

Our goal during synthesis is to be able to produce a design that, when routed, will meet performance goals with the smallest possible area. We also want to do this with as few iterations in the behavioral model and the synthesis scripts as possible.

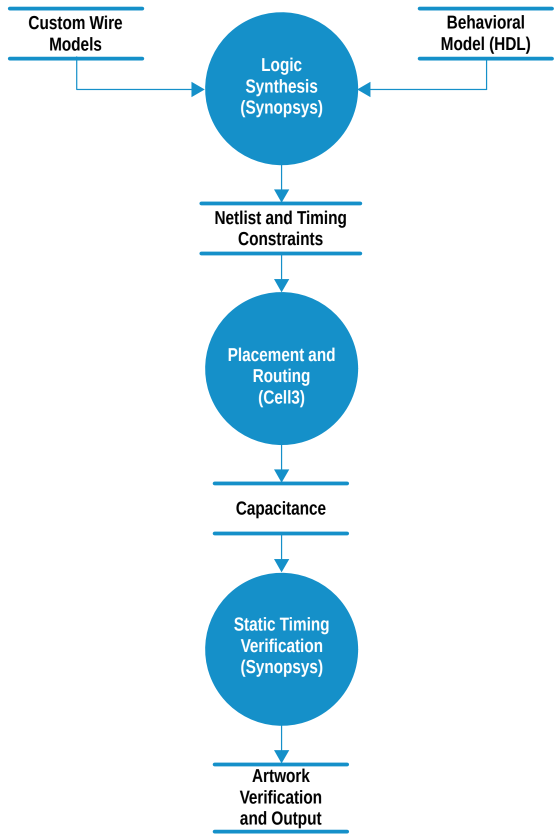
Custom Wire Models
Logic Synthesis (Synopsys)
Behavioral Model (HDL)
Netlist and Timing Constraints
Placement and Routing (Cell3)
Capacitance
Static Timing Verification (Synopsys)
Artwork Verification and Output

Fig. 1. Basic ASIC design flow.

Coding Guidelines. The HDL coding guidelines mentioned above enable the creation of behavioral models that synthesize predictably in a shorter amount of time with better performance than those created without guidelines.

Generic Synthesis Script. To streamline synthesis we create generic synthesis scripts for our design that contain all of the design-specific constraints such as flip-flop types, clock speed, behavioral model locations, and so on. These scripts also include input and output timing constraints, external loading, and drive capability. The scripts are used to synthesize all submodules in the design from the bottom up. Only those modules that do not meet timing requirements when incorporated into their parent module are resynthesized with scripts written specifically for them. This allows the majority of the blocks to be synthesized very quickly. Fig. 2 shows a portion of a generic synthesis script.

Custom Wire Load Models. One of our primary goals is to perform a single route without any iteration. To accomplish this, the wire loading (capacitance on the line) estimates that are used during synthesis have to be conservative. However, if they are too conservative then it is not possible to meet performance goals. To determine a wire loading model with the appropriate amount of conservative estimation for our library and tool methodology, we performed several wire loading experiments. These experiments consisted of synthesizing modules of various sizes and design types, routing them, and then verifying that their performance with actual wire loads satisfies the timing constraints defined for the module. Several passes were done for each module using a wire load model with varying levels of conservatism. Fig. 3 summarizes the process we used to generate the wire load model. This process was necessary because initial wire load estimates might be too optimistic. For example, the initial capacitance estimate for one connection between two inverters might be 0.04 pF, but after placement and routing the actual capacitance could be 0.1 pF, which might cause the chip's timing constraints not to be met.

```
/* These are fragments from the generic Synopsys dc_script */
/* the full script can be easily modified for a given model */
/*   ––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––          */
/*   Read in Verilog HDL:                                         */
/*   ––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––          */
read -f verilog mymodule.v
check_design

/*   ––––––––––––––––––––––––––––––––––––––      */
/*   Set the wire load model:                    */
/*   ––––––––––––––––––––––––––––––––––––––      */
set_wire_load block –library wire_loads

/*   ––––––––––––––––––––––––––––––––––––––      */
/*   Define the clocks:                          */
/*   ––––––––––––––––––––––––––––––––––––––      */
create_clock CLK –period 20 –waveform {0 10}
set_clock_skew –plus_uncertainty 0.5 –minus_uncertainty 0.5 –propagated CLK

/*   ––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––          */
/*   Set block operating environment:                                        */
/*   ––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––          */
loader_pin = hp_cmos26g_table_slow/INVFF/A
driver_pin = hp_cmos26g_table_slow/NINVFF/Q
set_load 3 * load_of(loader_pin) all_outputs()
set_load load_of(loader_pin) all_inputs()
set_drive drive_of(loader_pin) all_inputs()
set_input_delay 10 –clock CLK all_inputs()
set_output_delay 10 –clock CLK all_outputs() –max

/*   ––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––          */
/*   Compile the design
/*   ––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––          */
compile

/*   ––––––––––––––––––––––––––––––––––––––––––––       */
/* Write out results                                    */
/*   ––––––––––––––––––––––––––––––––––––––––––––       */
write –hierarchy –f verilog –output mymodule.vopt
write_constraints –format sdf –output mymodle. sdf –max_paths 1000
report_design >> mymodule.sn_rpt
report_hierarchy –full >> mymodule.sn_rpt
report_timing >> mymodule.sn_rpt
```

Fig. 2. A portion of a generic synthesis script.

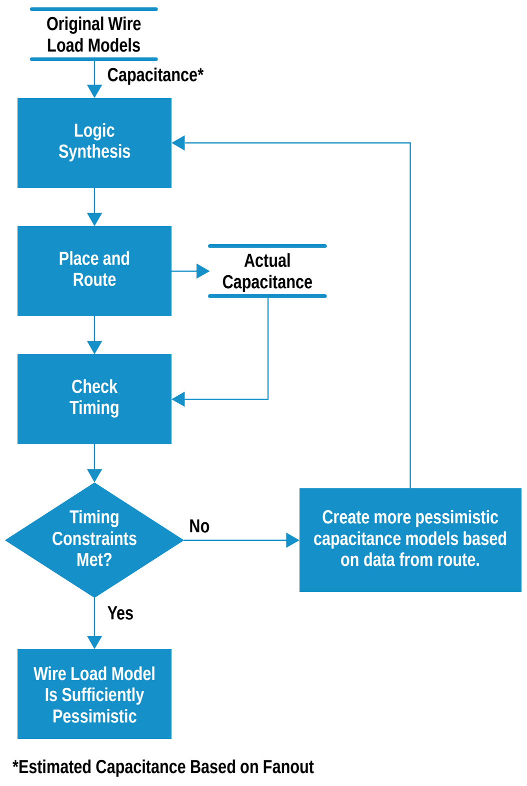
Original Wire Load Models
Capacitance*
Logic Synthesis
Place and Route
Actual Capacitance
Check Timing
Timing Constraints Met?
No
Create more pessimistic capacitance models based on data from route.
Yes
Wire Load Model Is Sufficiently Pessimistic
*Estimated Capacitance Based on Fanout

Fig. 3. The process for generating wire load models.

During these experiments, each module was first synthesized using average wire load estimates. After routing, if the modules failed timing, additional experiments were performed using a progressively more pessimistic wire load model. The wire load model that was used was generated by selecting a point in the distribution of actual capacitances for each fanout that is greater than a given percentage of nets. Fig. 4 shows a sample distribution of interconnect capacitance for nets with a fanout of two in a typical module. In this example, we wanted to use a model that would predict a capacitance such that 90% of the wires would typically have actual capacitance less than the predicted value. To do this we simply used the capacitance value from the distribution that was greater than 90% of the other wire capacitances. This was done for each fanout to create a synthesis wire load model called a "90% model."

We used this method to test models that fell within the 50-to-95-percent range. We found that unless we used at least a 90% wire loading model we had some timing violations after back-annotation* that were not present during synthesis with estimated loads.

We also discovered that the magnitude and distribution of the routed capacitance were fairly consistent across a wide

* Back-annotation in this context refers to the process of taking the actual capacitance values extracted from routing and using them during static timing analysis.

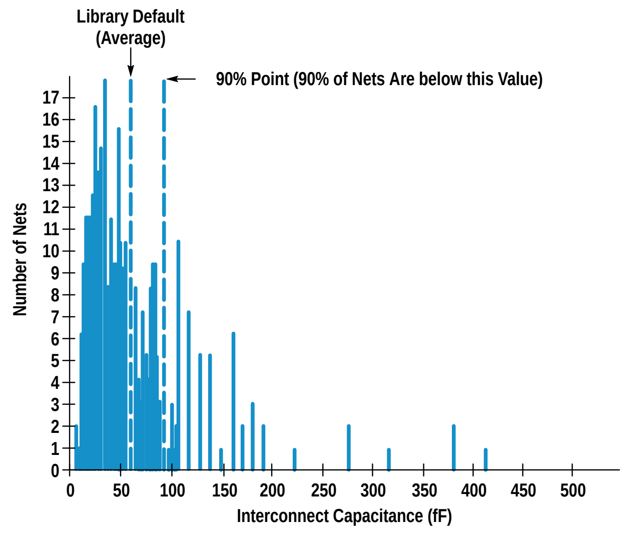

Fig. 4. A sample distribution of interconnect capacitance for a fanout of two in a typical module.

range of module sizes and design types. This was a surprise and it allows us to use the same wire load models for all designs done in the same library without having to repeat the experiment for each design.

We use the 90% wire load model derived from the above process to help guarantee single-pass routing for all modules using our library. In addition, we create another model for wires that are used for global* module interconnect. This is done using the same method described above except that the wires used to create the capacitance distributions are limited to global wires. This model more accurately predicts the higher capacitance associated with top-level interconnect. Synopsys, our logic synthesis tool, can automatically select the appropriate wire load model to use for each global wire.

Table-Driven Models. Another factor in our ability to achieve high performance and minimize cycle time is the accuracy of the Synopsys library timing models.

The biggest problem created by inaccurate synthesis timing models is the requirement for numerous iterations between synthesis and timing verification to fix timing violations. These iterations involve changing synthesis constraints, overconstraining, and replacing cells by hand in some cases. Timing inaccuracy also leads to poor optimization decisions, resulting in nonoptimal circuits in terms of performance and area.

These and other problems are essentially removed by the new nonlinear table-driven timing models in Synopsys. With these models, timing can be made to match the Spice characterization tables for each cell in the library. The transition time definition can also be made to match exactly so that it can be constrained properly.

The HP C26102SH library supports this new model and allows us to get accurate timing from our synthesis package so that there are no timing or operating condition violations in the timing verification step. Besides the obvious benefit of reducing iterations, this new library and timing model gives us performance and density improvements. The performance is improved because the synthesis tool is now working on the correct paths and is able to generate faster circuits. Density is also improved for the same reason. Since incorrect paths are no longer being optimized incorrectly and overconstraining is not necessary, the overall design size is smaller. In addition, the improvement in transition-time modeling and constraints avoids the need to apply a global transition-time constraint to the design which can lead to oversizing many cells.

* Global wires are the interconnecting wires between submodules on a chip.

Timing Constraints

As an additional method of ensuring that our performance goals are met without having to do multiple routing passes, we have added the process of driving the placement with constraints derived from synthesis.

Because Synopsys timing is very accurate with table-driven models, it can be used directly to create timing constraints imposed on the router. This is done using the Standard Delay Format timing output in Synopsys. Critical path timing is written in Standard Delay Format which can then be converted to Design Exchange Format and input to Cell3** with the netlist. These timing constraints become part of the overall constraint equation for the placer.

Several thousand constraints can be quickly and easily generated using this method. However, only those paths that are within a few percent of failing with estimated loads should be constrained. If an appropriate wire load model is used, the rest of the paths should meet their timing without being constrained. Having too many paths constrained will slow the placement process and may produce inferior results. However, we have successfully constrained up to a thousand paths.

Place and Route

Our goals during placement and routing are to implement the design in the smallest possible area, meet all specified performance goals, and minimize the number of iterations through the process.

Timing-Driven Placement. Timing-driven placement is the process of driving the placer with the timing constraints output from a timing analyzer (Synopsys, in this case). The following factors are important in successfully using this technique.

- Accurate timing models used for static timing analysis. As mentioned above, accurate timing models are critical to ensure that the synthesis program works on the right paths and that the timing constraints are accurate.
- Accurate cell delay information fed to the placer. Our placement program, Cell3, uses a two- or three-parameter delay equation. The two-parameter equation calculates delay with an intrinsic component and a load dependent component. The three-parameter equation modifies the intrinsic delay to reflect its dependency on the input slope. To drive the placer with accurate data, it is important that Cell3's delay calculation match that used by the timing analyzer as closely as possible. Table I shows the correlation obtained from each of these models compared to the data used by the timing analyzer. The data in the table is for 100 representative paths, varying in length from 4 to 71 cells.

** Cell3 is the placement and routing program we use, which comes from Cadence Design Systems.

Table I
Cell3 to Synopsys Correlation

	Timing Analyzer Data	Cell3 (two-parameter)	Cell3 (three-parameter)
Minimum Path	3.47 ns	3.56 ns	3.43 ns
Maximum Path	15.54 ns	18.51 ns	14.55 ns
Least Error versus Synopsys		+1.4%	0.0%
Greatest Error versus Synopsys		+26.2%	+7.4%
Error Range versus Synopsys		1.4% to 26.2%	−7.3% to 7.4%

As shown in Table I, Cell3's three-parameter model provides greatly improved delay modeling compared to the two-parameter model. Because of its improved accuracy, the three-parameter model is now used as the standard during timing-driven placement.

- Accurate estimates of interconnect. For synthesis, interconnect delay is specified by the 90% wire load models. For placement, it is important that the placer has a good idea of what it can expect in terms of average per-layer wiring capacitance for each signal. These numbers are determined by analyzing wiring distributions on previously routed chips.

Clock Tree Synthesis and Verification. Typical ASICS are driven by single or multiple high-speed clocks. These clocks drive thousands of flip-flops and must have low insertion delay and skew to meet performance requirements. To implement these balanced clocks, special placement and routing features must be used. Cell3's clock tree synthesis tools are used to insert a buffer tree for each clock (see Fig. 5). The clocks are then prerouted using various forms of Cell3's balanced routing techniques. This method has met with mixed success. Excellent results can be obtained by using the detailed balanced router, which is included as part of Cell3's clock tree synthesis option. However, since this tool has proven not to be as robust as we would like, we have found that an easier and much more robust method is to use Cell3's balanced global routing, which meets the needs of most high-performance standard cell ASICs.

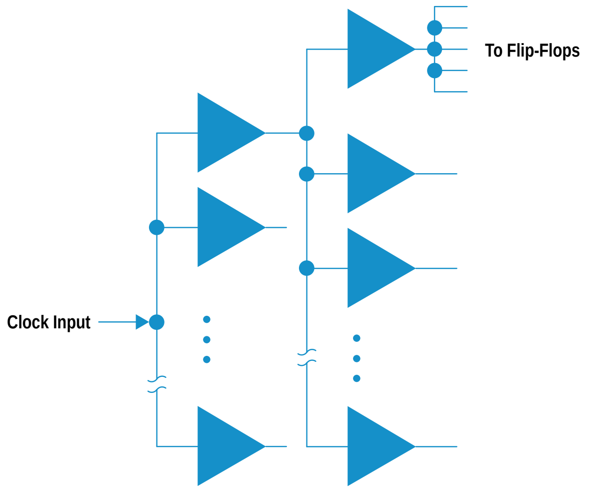

Fig. 5. A typical clock buffer tree. Insertion delay for this configuration is the average delay from the clock input to the flip-flops. Skew is the difference between the shortest and longest delays.

Actual clock delay and skew are verified using the RC extraction capability provided in CheckMate.* The general steps involved in using this approach include:

- Complete the physical implementation of the clocks.
- Extract (via CheckMate) RC information for all nets in the design.
- Create a Spice netlist containing only the cells and nets in the clock trees. (The desired nets and cells are specified to CheckMate's Spice writer.)
- Add custom circuit conditions (external loads, etc.).
- Run the Spice job.

Cell3 can also provide clock and skew information, but the steps listed allow us to obtain more accuracy and to build confidence in the Cell3 numbers.

Automatic Scan Insertion. Automatic scan insertion is done during the routing process. This makes it possible to include scan logic without having to design it in during behavioral coding. However, there are rules in the HDL coding guidelines that must be followed to make automatic insertion possible. Using automatic insertion reduces the complexity of the behavioral design and eliminates iteration because of scan clock skew, scan chain ordering, and timing performance issues. The use of automatic scan insertion is made possible by the use of an internal tool that performs insertion and optimization.[1]

Results

A chip in which the processes described above have been applied is a CMOS ASIC that is used in HP's X-terminals. Its main functions are memory and data path control. The chip contains 270,000 transistors and runs at multiple clock frequencies of 50, 100, and 33 MHz.

For this chip, the 1000 timing paths with the smallest timing margin were input as constraints fed to the Cell3 placer. Cell3 met all of the constraints on the first pass and subsequent timing analysis verified that all the paths were satisfied. More important, no other timing violations were created because of the constraints on the tightest paths. If that had occurred, we were prepared to exercise the Synopsys in-place optimization flow. This flow does in-place up and down sizing of cell drives as necessary to meet timing constraints. This step proved to be unnecessary for our X-terminal ASIC. The 90% wire load models were adequate.

The multiple clock domains on the ASIC are all derived from a 100-MHz input clock. The chip has three domains and each contains approximately 600 flip-flops. To meet performance goals, the skew across all of these domains had to be less than 0.5 ns and the insertion delays had to be matched within 1 ns. After placement and routing, clock delay and skew verification was done using the CheckMate RC extraction tool. Fig. 6 shows the Spice results obtained on one representative clock tree. All clock trees were found to have acceptable skew. This tight clock skew was a key factor in achieving the ASIC's aggressive performance goals.

* CheckMate is an artwork verification and parasitic extraction tool from Mentor Graphics Corp.

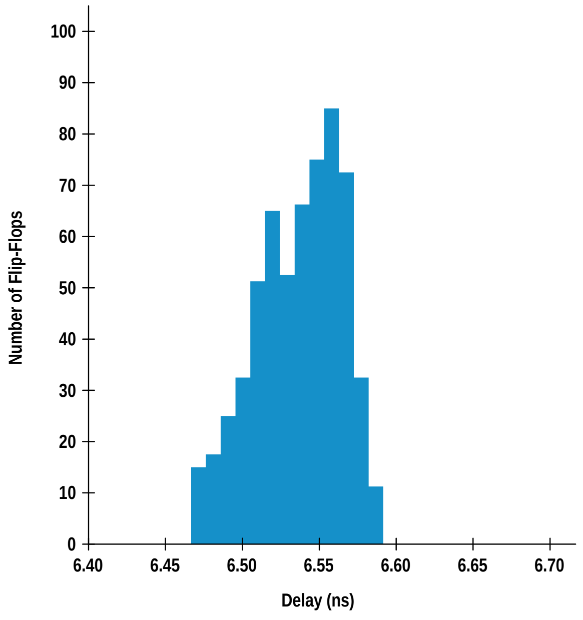

Fig. 6. Spice clock delay and skew results.

It is our goal to characterize Cell3's delay calculation to the point at which we can use its delay numbers for clock skew verification without the additional complexity of using Spice. Table II shows the correlation obtained between Cell3 and Spice for the clocks on the X-terminal ASIC.

Table II
Cell3 to Spice Correlation

	Cell3	Spice	Delta
Best Case	2.84 ns	2.56 ns	−9.9%
Worst Case	2.96 ns	2.57 ns	−13.2%

The results of this correlation are very encouraging. The offset is relatively constant, and the Cell3 numbers are always more conservative than the Spice numbers. This is because Cell3's per-layer capacitance constants are intentionally skewed toward the conservative end of the range.

Contributing Factors

The following factors played a part in producing the results mentioned above and providing a chip that met the specifications.

Library Design. One of the major contributing factors for meeting our density and performance goals is the use of high-quality libraries such as the new HP C26102SH (0.8 μm) standard cell library. This library is a scaled-up version of the HP C14104SH library developed for our CMOS14 (0.5 μm) process.[2]

One major improvement introduced with the HP C26102SH library is that it is tuned for Cell3 while maintaining compatibility with other routers. For a roughly square core, the library allows a very even distribution of horizontal and vertical routing resources. This gives the placer more freedom to find an optimal solution. The use of advanced layout techniques, along with the exploitation of new process design rules, allows smaller cells that provide the same functionality as previous library versions. Finally, advanced

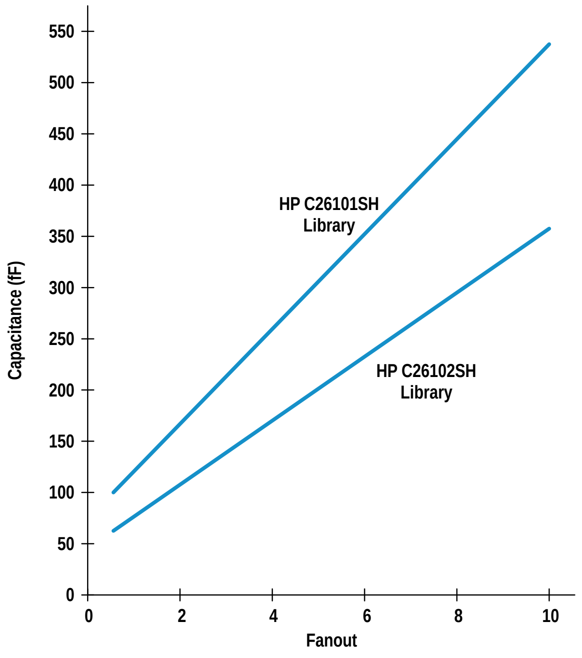

Fig. 7. Wire capacitance versus fanout in two cell libraries. The HP C26101SH library is the older CMOS26 library.

router model generation provides fully gridded routing and optimum pin locations, which improves both density and run time. These factors result in a reduction in average wire length for a given fanout. The improvement in wire capacitance as compared to the previous cell library is shown in Fig. 7.

Another area of improvement is in the drive sizes of the cells. The "stairstep design"[3] approach was used to determine appropriate drive increments. The stairstep design approach is a method of sizing gates to provide optimum area versus performance characteristics for each cell in the library. As a result of using this approach, the HP C26102SH library provides more drive increments for Synopsys to map to, avoiding the excessive area penalty imposed when a higher drive cell must be swapped into a path that is just missing timing.

Finally, this is the first library to use the more accurate table-driven Synopsys models. As discussed previously, this allows more accurate delay calculations and eliminates correlation issues with other timing verification tools.

Since the HP C26102SH library is a scaled-up version of the HP C14104SH library, all these advantages will continue into the next process generation.

Design Methodology. Our design methodology was carefully constructed to produce a chip that met aggressive timing goals in a reasonable amount of time. The 90% wire load models improved the chances of first time perfect through the routing cycle, albeit at some loss of performance and die size. This was an intentional choice designed to minimize the design cycle time, while still meeting performance targets.

The HDL coding guidelines enabled the creation of a design that was easily synthesized. They also enabled the use of automatic scan insertion and test vector generation.

Finally, by imposing a set of automated naming rules during synthesis, tool compatibility issues were minimized. These naming restrictions eliminated name remapping and cross-referencing throughout the design flow.

Point Tools. Several key point tools are necessary to support the design flow discussed in this paper. Synopsys with table-driven delay models is required to provide accurate static timing analysis and valid constraints to the placer. Cell3's timing-driven placement is required to implement the critical path timing output from Synopsys. An automatic test generation (ATG) tool is required to insert and optimize the scan chain automatically and produce appropriate test vectors.

Possible Improvements

Although we have been successful with the methods described in this paper, there are still some areas for improvement.

Timing Verification in Synopsys. Eliminate Verilog functional timing simulation by relying on Synopsys static timing analysis for verification.

More Automation of Clock Insertion. As mentioned, further characterization of Cell3's delay calculation is necessary to eliminate the need for Spice clock verification. Another area of improvement here is to work with Cadence to improve the balanced routing capability of Cell3.

Alternate Ways of Prelayout Capacitance Estimation. Other methods for accurate early capacitance estimation are available that don't require a preliminary route. Promising improvement areas here include using advanced floor planning earlier in the design cycle and netlist-based capacitance estimation that includes not just fanout but other factors such as estimated chip size and types of cells connected to each wire.

In-Place Optimization. Use less conservative wire loads for preroute estimation with expanded use of links-to-layout during the synthesis and placement loop to enable higher performance with minimal impact on cycle time.

References

1. B. Jung and J. McDougal, "An Optimal Scan Chain Auto-Connection Methodology and Scan Signal Insertion Scheme to Reduce Chip Area," *1993 HP Design Technology Conference Proceedings,* pp. 343-347.

2. S. Ratner, J. Eaton, A. Martinez, and H. Youn, "Development of a New Dense and Router Independent BiCMOS Compatible, Standard Cell Library Floorplan for (Bi)CMOS14," *1993 HP Design Technology Conference Proceedings,* pp. 477-484.

3. J. Eaton, "Stairstep Library Design: The Application of Optimization Techniques to the Design of the CMOS14 Standard Cell Library," *1993 HP Design Technology Conference Proceedings,* pp. 391-398.