Resistance is Futile!

Building Better Wireload Models

Steve Golson

Trilobyte Systems
33 Sunset Road
Carlisle MA 01741
Phone: +1.978.369.9669
Fax: +1.978.371.9964
Email: sgolson@trilobyte.com

ABSTRACT

Wireload models are like the weather. Many people talk about them, but not many people *do* anything about them! This paper will explore some of the myths and realities of wireload models:

- why wireload models are important, and why *nobody* understands them
- why your intuition is wrong
- why you shouldn't trust your silicon vendor
- why floorplanning sometimes doesn't matter
- why having an accurate wireload model is a *bad* idea

A technique for measuring the quality of wireload models will be described. Real-world results will be discussed. Cool graphics will be shown. A desperate plea for future work will be given.

1.0 Synthesis basics

The typical methodology or flow used with Design Compiler is shown in Figure 1.

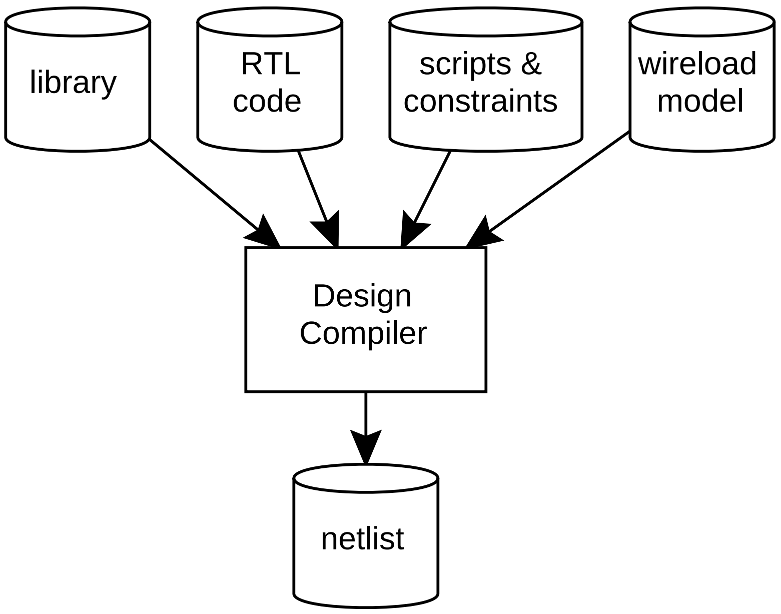

Figure 1. Typical Flow

The quality of the netlist is determined by:

- synthesis tool (vendor and version)
- technology library
- RTL code
- compile scripts and constraints
- wireload model

Much has been written on how to write better RTL code ("better" means the resulting netlist is smaller, faster, lower power, etc.). Also extensive work has been done on examining synthesis scripting styles and how to properly constrain your design. In contrast, very little has been said about wireload models.

A *wireload model* is what the synthesis tool uses to estimate wire characteristics (e.g. interconnect delay) in the absence of physical layout data. For a wire with a given fanout, the wireload model specifies the capacitance, resistance, and area of the wire. (Here *fanout* is defined to be one less than the total number of pins on the net.)

Although the synthesis tool has complete control over the netlist, the resulting timing is greatly affected by the physical layout. The wireload model is the *only* information that the synthesis tool has about the back-end place and route flow.

2.0 Wireload model basics

ASIC vendors typically develop wireload models based on statistical information taken from a variety of example designs. For all the nets with a particular fanout, the number of nets with a given capacitance is plotted as a histogram. A single capacitance value is picked to represent this fanout value in the wireload model. If a very conservative wireload model is desired, the 90% decile might be picked (i.e. 90% of the nets in the sample have a capacitance smaller than that value).

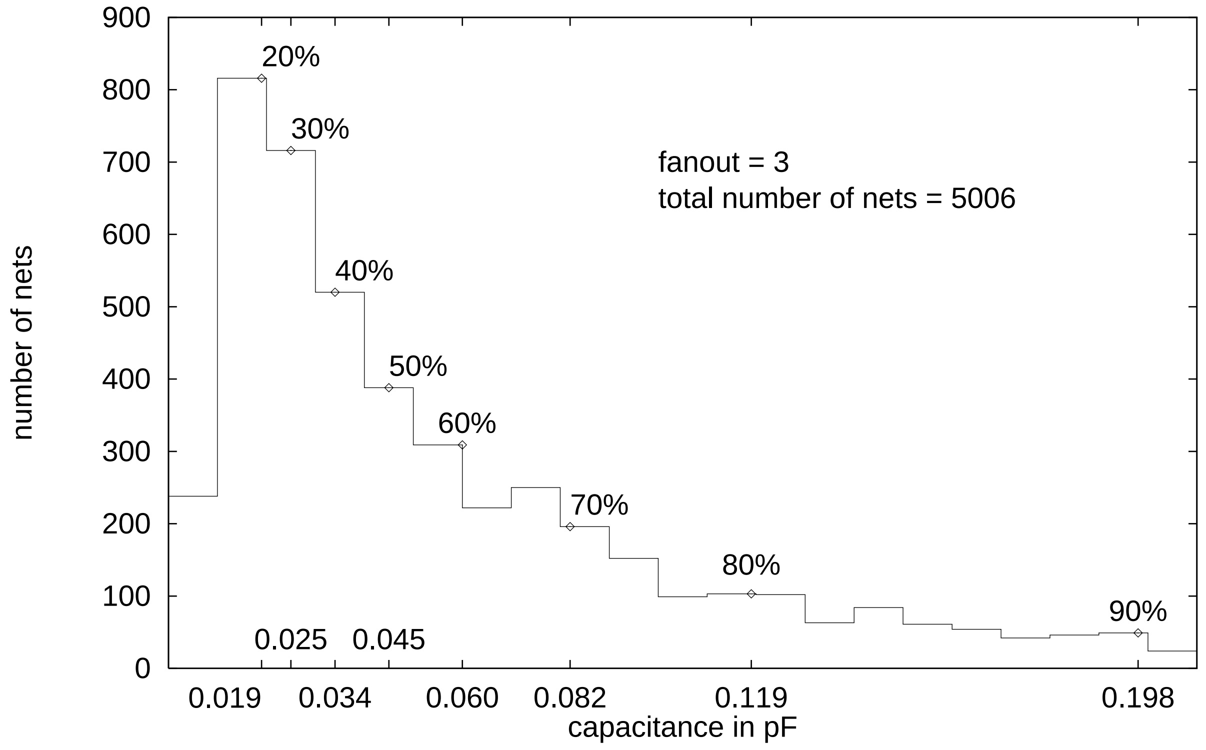

Figure 2. Example statistical distribution showing deciles

Figure 2 is an example histogram plot. The distribution typically has a very long tail; in this case the 95% net has a capacitance of 0.358 pF while the 100% net (i.e. the worst-case net) has a capacitance of 2.130 pF. Sometimes these long tails are trimmed before the deciles are calculated. A smoothing function is applied to guarantee that capacitance increases monotonically with fanout.

Similar statistics are gathered for resistance and net area.

Usually the vendor supplies a family of wireload models, each to be used for a different size design. This is called *area-based wireload selection*.

Example wireload models from three different vendors are shown in Figure 3. All three models are for a 20k-gate module size in a 0.25µ process. Two theoretical curves are also shown, one from Lee Bradshaw [1] and the other from the Berkeley Advanced Chip Performance Calculator (BACPAC) [17]. Note that vendors B and C and the BACPAC model are all linear.

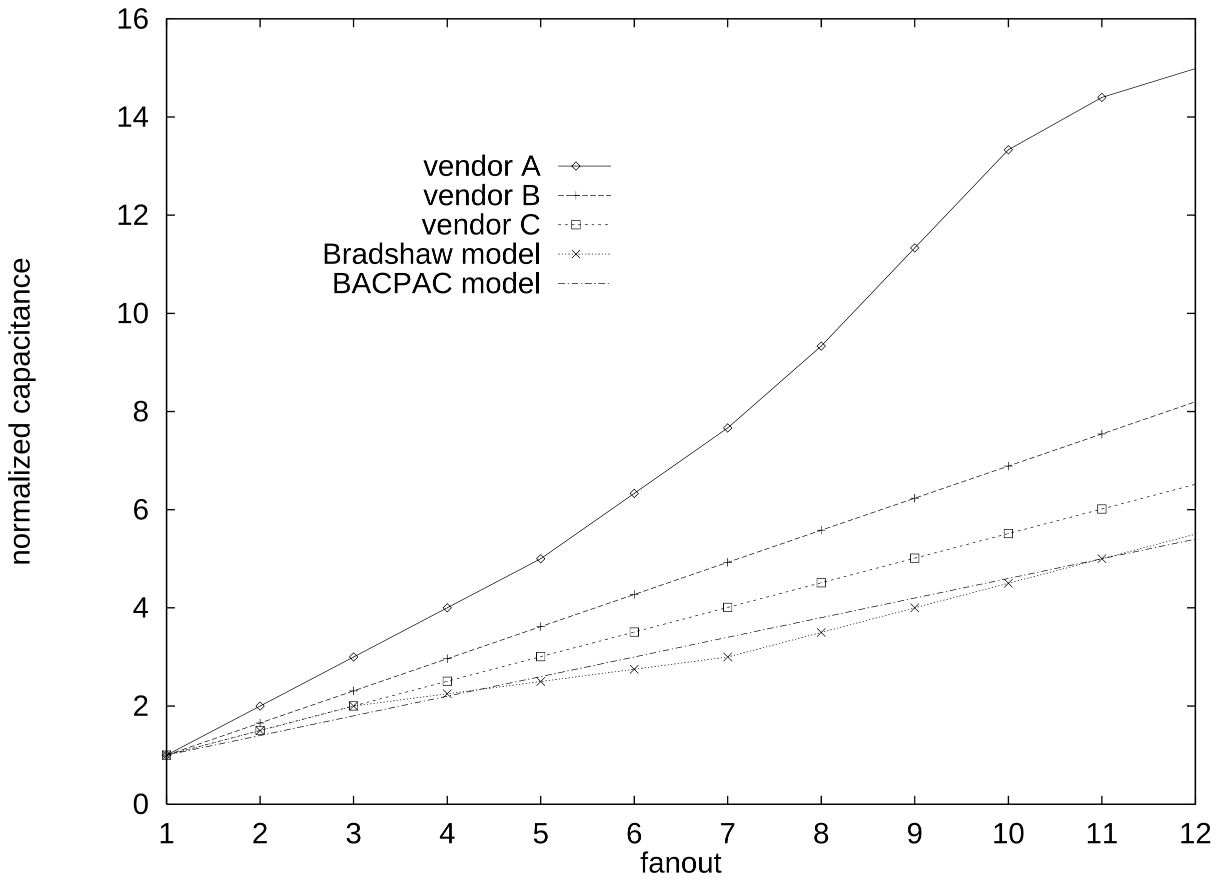

Figure 3. Example wireload models

The curves are normalized such that a fanout of one has a normalized capacitance of one, allowing easy comparison. An alternative normalization would be to divide all capacitances by the input capacitance of a typical 2-input NAND gate from each library. Interestingly, for these three vendors, the normalized value for fanout of one would still be very close to one! In other words, in these 0.25µ processes, the wire capacitance for a fanout of one is about equal to the input pin capacitance of a typical gate.

Bradshaw's wireload model was based on actual design data and some theoretical modelling done by Kurt Baty [26][27].

Smith [11] gives an excellent overview of interconnect delay and the statistical nature of wireload models.

Much more information about wireload models (including example syntax) can be found in the Synopsys Online Documentation [14][15][16].

3.0 Conventional wisdom (myths)

There is much conventional wisdom about wireload models and how to properly use them. Over the years some of this “wisdom” has been shown to be more myth than truth. Here is my collection of wireload myths, with a brief discussion of each.

Myth #1: Select the wireload model based on the area of the block that will be placed and routed as a unit

This is absolute dogma, repeated by virtually every paper, book, and users manual that mentions wireload models [19]. However it isn’t necessarily true.

Mohsin [8] introduces a technique called “hierarchical estimated wire load (HEWL)” where different wireload models are used at different levels of logical hierarchy within a single place and route block. The idea here is that the placer will keep logically-related cells in close proximity. Thus cells will be placed within an area that is reflective of the wireload model used, even without floorplanning.

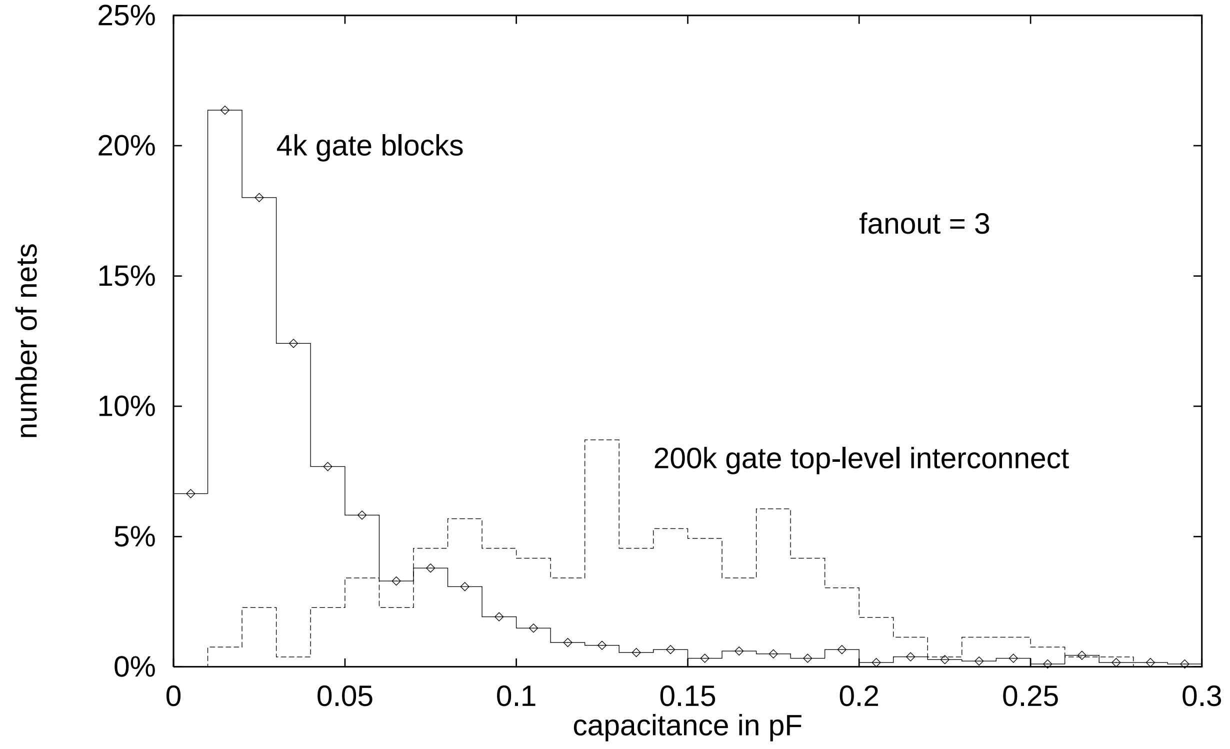

Figure 4. Statistical distribution of nets vs. capacitance for two levels of hierarchy from a single layout block

Statistical analysis seems to bear this out. Figure 4 shows two histograms taken from a single block that was placed and routed with no floorplanning. The “4k” curve includes nets enclosed by all leaf modules with a size of 4k gates or less. The “200k” curve includes only top-level interconnect nets at the 200k-gate module size, i.e. nets that are enclosed by the top-level hierarchical module and that are *not* enclosed by some lower level of hierarchy. Thus the allocation of nets to statistical population is identical to how Design Compiler allocates nets to wireload models using the `enclosed` wireload model mode [14]. Similar statistical results were reported by Smith [11].

It appears that larger blocks should have more pessimistic wireloads, however not shown in the figure is that the tail of the 4k distribution is very long, much longer than the tail of the 200k curve. For very pessimistic wireloads (>90% decile) this technique may not work.

Myth #2: One wireload model per area is enough

Not necessarily. Following the analysis of Myth #1 above, these hierarchical modules might all need different models, even though they are all the same size:

- 20k-gate random logic block
- top-level interconnect for five 4k-gate modules
- 20k-gate random logic block, that is part of a larger place and route block
- top-level interconnect for five 4k-gate modules, that is part of a larger place and route block

Alternatively, there is evidence that a *single* wireload model can give good results, regardless of area [3][24].

Myth #3: Wire resistance should be set to zero

Many vendors have resistance set to zero in their wireload models. The result is that *interconnect delay is zero*.

This means that the interconnect model (e.g. `best_case_tree`, `worst_case_tree`, `balanced_tree`) selected as part of `set_operating_conditions` actually doesn't have any effect.

Note that wire capacitance still has an effect on cell delay and transition delay [15].

The typical excuse for zero resistance is "but our extraction tool only gives capacitance!"

Even if resistance is non-zero, many vendors still use the original `wire_load` library format which forces resistance, capacitance, and area to be proportional. The newer `wire_load_table` format allows more flexibility and accuracy [15].

Myth #4: Wire area should be set to zero

Real wires *do* have area, and this can greatly impact your layout. If Design Compiler has a good idea of wire area it can make better trade-offs during gate-level optimization.

Toshiba [7] reports that wire area is critical to get a routable netlist.

Myth #5: You can always trust the vendor wireload model

Not necessarily. Even if your vendor avoids Myth #1 through Myth #4, you might still have problems because:

- your design is larger than the examples that the vendor used to derive their wireload models
- your netlist characteristics are different (more nets, more routing congestion, more IOs)
- your routing blocks have a different aspect ratio
- your design flow may be different. McDougal et al. [6] reported that an optimistic wireload model gave best results for a flow including IPO, while a pessimistic model gave best results with no IPO.

Keep in mind that your vendor may have a very different agenda from you. The vendor tends to want conservative wireload models so they can easily meet pre-route timing estimates. The customer wants more optimistic wireloads with very little margin. Some vendors are attempting to make the predictions of the wireload model as accurate as possible, leaving the inclusion of margin to the designer's judgement.

Myth #6: Custom wireload models are always better

This is almost as pervasive as Myth #1, and it is just as suspect.

What is commonly called a custom wireload model is more precisely called a *design-specific wireload model*. The idea is to first do a trial place and route of your own design, and then use the resulting statistics to generate new wireload models that are specific to your design. In theory this will give better results. However Joshi [9] found that some blocks got worse results from their "custom" wireload models, compared to the "custom" models from another block.

Several things may go wrong:

- you might get a small statistical sample from your one design, rather than the many designs used by your vendor
- you typically generate the wireloads based on an early netlist, and the netlist characteristics may change as the design is refined (e.g. area gets bigger, number of nets changes)
- the model creation tool may not use a methodology you agree with (especially if you want to use different models within a single place and route block)
- your flow may require `uniquify` so that each instance gets its own wireload model

  An alternative is to create a *design-specific wireload library* which replaces the vendor wireload library, but where the individual models are not tied directly to floorplan or layout blocks. This allows the compile flow to remain unchanged.

The generation and use of design-specific wireload models is discussed by Bradshaw [1], Rusu [4], and McDougal [6].

Myth #7: Wireload model should agree with post-route statistics

This is the most heinous of all myths. It is the underlying force behind Myth #6.

The purpose of a wireload model is *not* to agree with post-route net statistics.

Rather, the purpose of a wireload model is to accurately predict post-route timing.

The actual shape of the model doesn't really matter. How it is generated doesn't really matter. All that matters is that the model *accurately predicts post-route timing with appropriate margins*.

4.0 Measuring the accuracy of a wireload model

Several techniques have been presented to measure the accuracy of wireload models, by comparing the predicted pre-route timing with actual post-route timing.

The timing information for a design can be generated from the top of the design hierarchy with the following dc_shell commands

```
      set_false_path -from all_inputs()
      set_false_path -to   all_outputs()
      report_timing -nosplit -path end
```

The resulting report gives the path delay, required path delay, and slack for each endpoint in the design:

```
 ****************************************
 Report : timing
         -path end
         -delay max
 Design : merced
 Version: 1998.02-2
 Date   : Mon Feb  1 16:02:08 1999
 ****************************************

 Operating Conditions: WORST_TREE   Library: P858
 Wire Loading Model Mode: enclosed

 Design           Wire Loading Model      Library
 ------------------------------------------------
 merced                 10M               mylib_v1.3
 x86                    1M                mylib_v1.3
 bus_int                100k              mylib_v1.3


 Endpoint                         Path Delay     Path Required     Slack
 ------------------------------------------------------------------------
 core/cache_ctrl/regs/u_ff_2/D (FF1) 10.59 r          9.69        -0.90
 regs/ctrl/be_reg_ff/D (FF2)         11.23 f          9.67        -1.56
```

Because of the `set_false_path` commands only flop-to-flop paths are reported, thus avoiding any inaccuracies due to unrealistic input and output delays.

Using such data, Joshi et al. from Texas Instruments [9] compared the pre- and post-route slack for each endpoint in a design. Subtracting the pre-route predicted slack from the post-route actual slack gives a difference slack value or *delta slack*. Plotting the number of paths with each such value creates a *delta slack histogram plot*. Figure 5 shows an example.

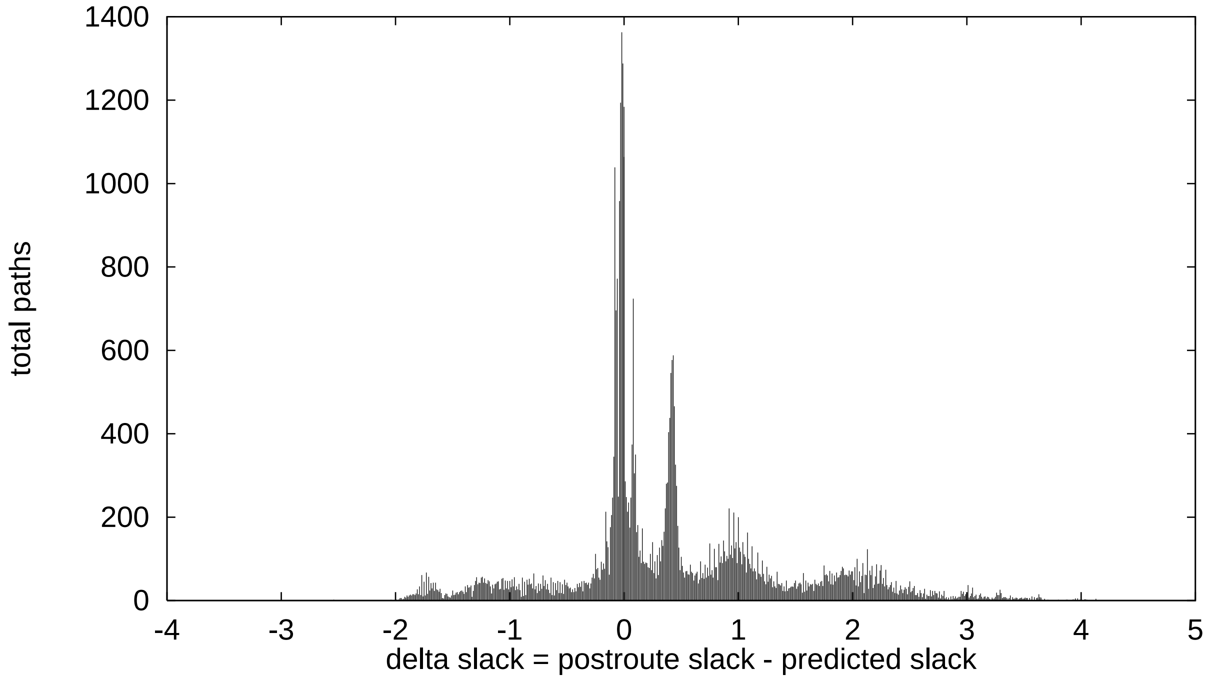

Figure 5. Delta slack histogram plot

Plots with a peak around zero show a good correlation to the predicted values. Plots with a majority of paths to the right of zero indicate that the predicted values (and thus the wireload models) are pessimistic, while paths to the left of zero indicate optimistic predictions.

Toshiba [7] added another dimension by plotting the pre-route predicted path delay versus the post-route actual path delay. Each data point represents a single path in the design. Figure 6 shows such a *path delay scatter plot*.

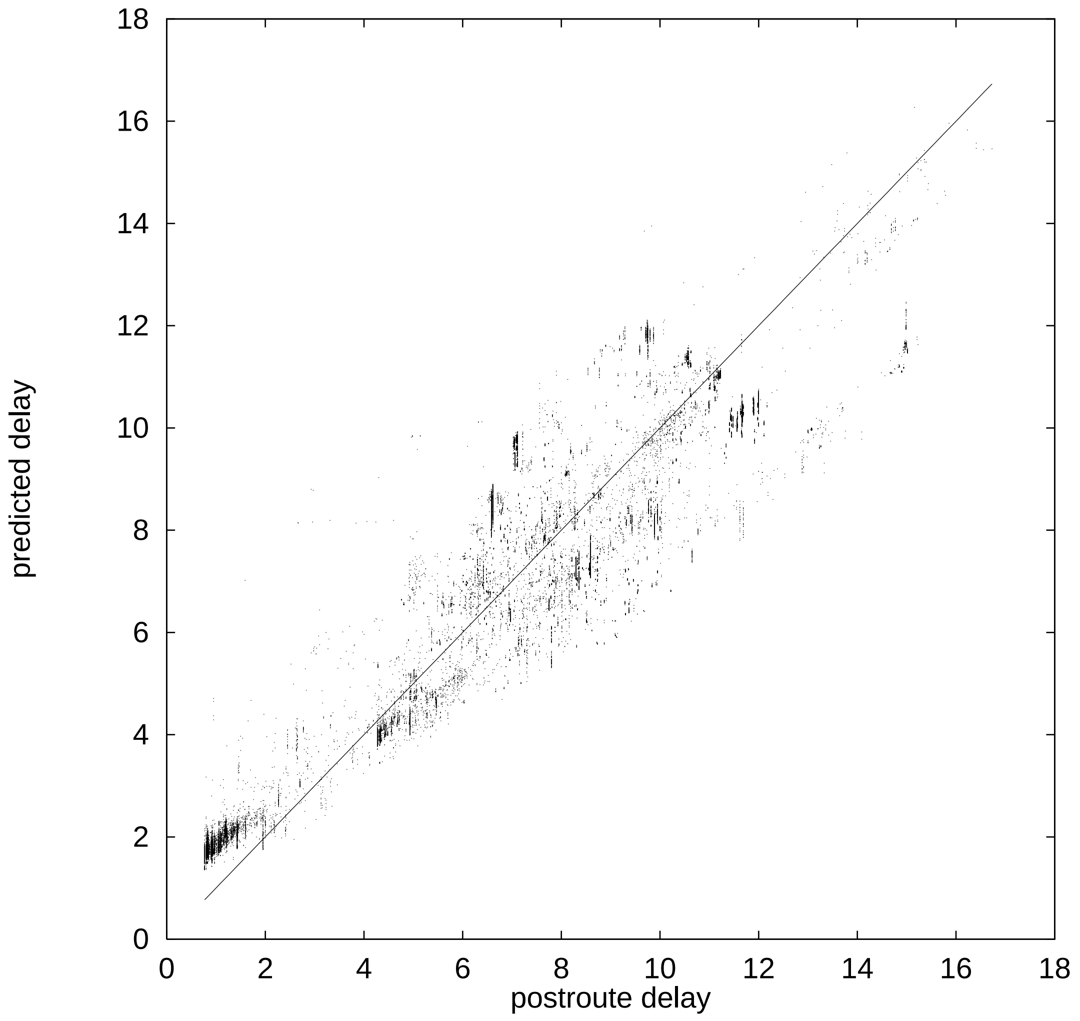

Figure 6. Path delay scatter plot

The diagonal line indicates perfect agreement between pre- and post-route delay. Points to the left of the diagonal represent paths whose predicted values were pessimistic, while points to the right represent optimistic predictions. Note that if all the points are projected onto a line perpendicular to the diagonal, the result is the delta slack histogram plot of Figure 5.

This plot is useful for showing rough correlation, but has several problems. Comparing total path delay isn't very interesting, because a mispredicted path that has lots of slack is not the critical path. Slack can be used for comparison, however slack alone is not adequate, because with multifrequency designs (due to multiple clocks or multicycle paths) there are different flop-to-flop path requirements. For example a slack of -2ns on a 10ns path may be more important than a -3ns slack on a 40ns path. A much better measure is the *slack ratio* defined for a given path as

$$\text{slack ratio} = \frac{\text{slack}}{\text{required delay}}.$$

Plotting the pre-route predicted slack ratio versus the post-route actual slack ratio gives a *slack ratio scatter plot*. Figure 7 gives an example using the same data from Figure 6. Lucent Technologies [24] introduced slack ratios and slack ratio plots and has used them for several years.

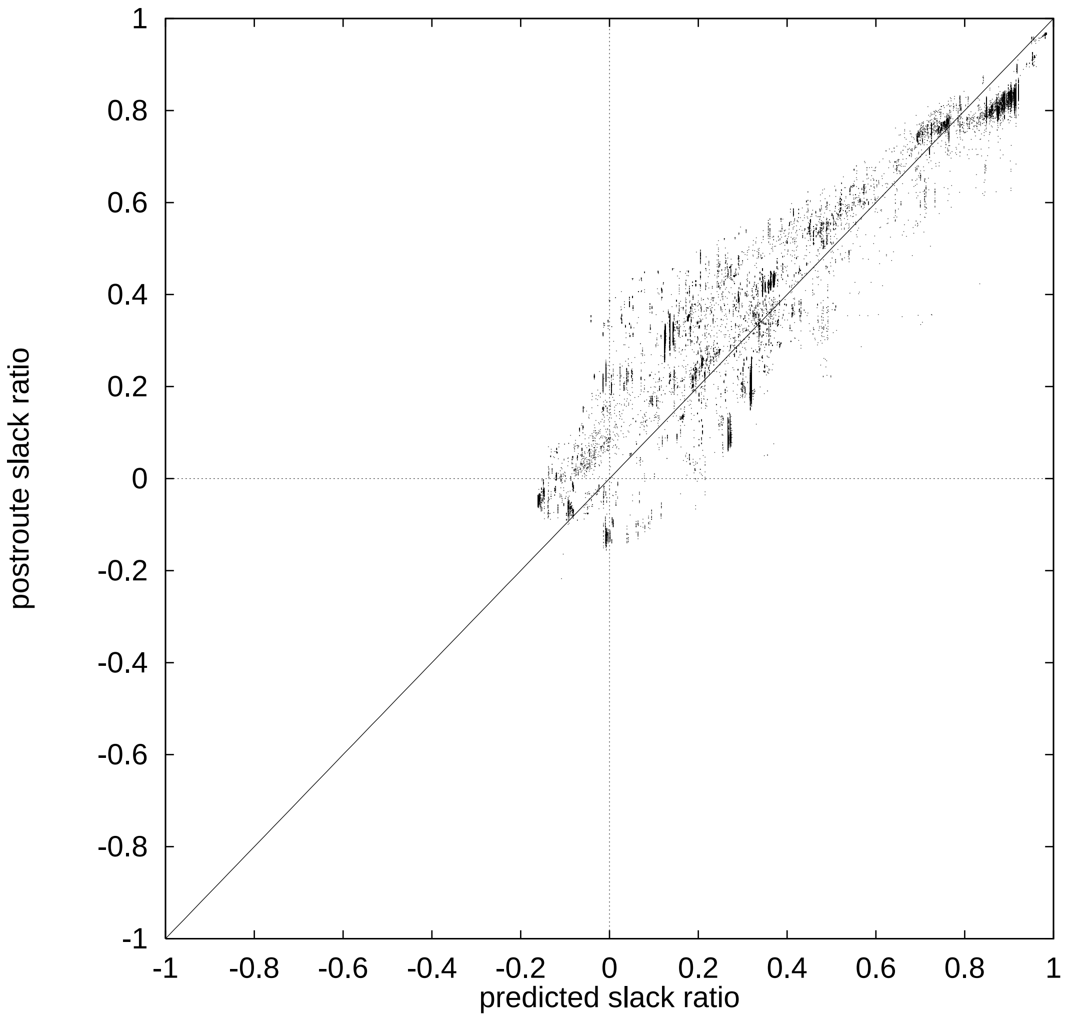

Figure 7. Slack ratio scatter plot for 42,052 paths

The diagonal line indicates perfect agreement between pre- and post-route delay. Points to the left of the diagonal represent paths whose predicted values were pessimistic, while points to the right represent optimistic predictions.

Points to the left of the *y* axis are paths that were predicted to not meet their required timing (they have negative pre-route slack). Points below the *x* axis are paths that have negative post-route slack and therefore actually do not meet their required timing.

Figure 8 shows the same design after in-place optimization (IPO). Note that the points below the *x* axis have been "swept up" to the axis, and now all paths meet their required timing.

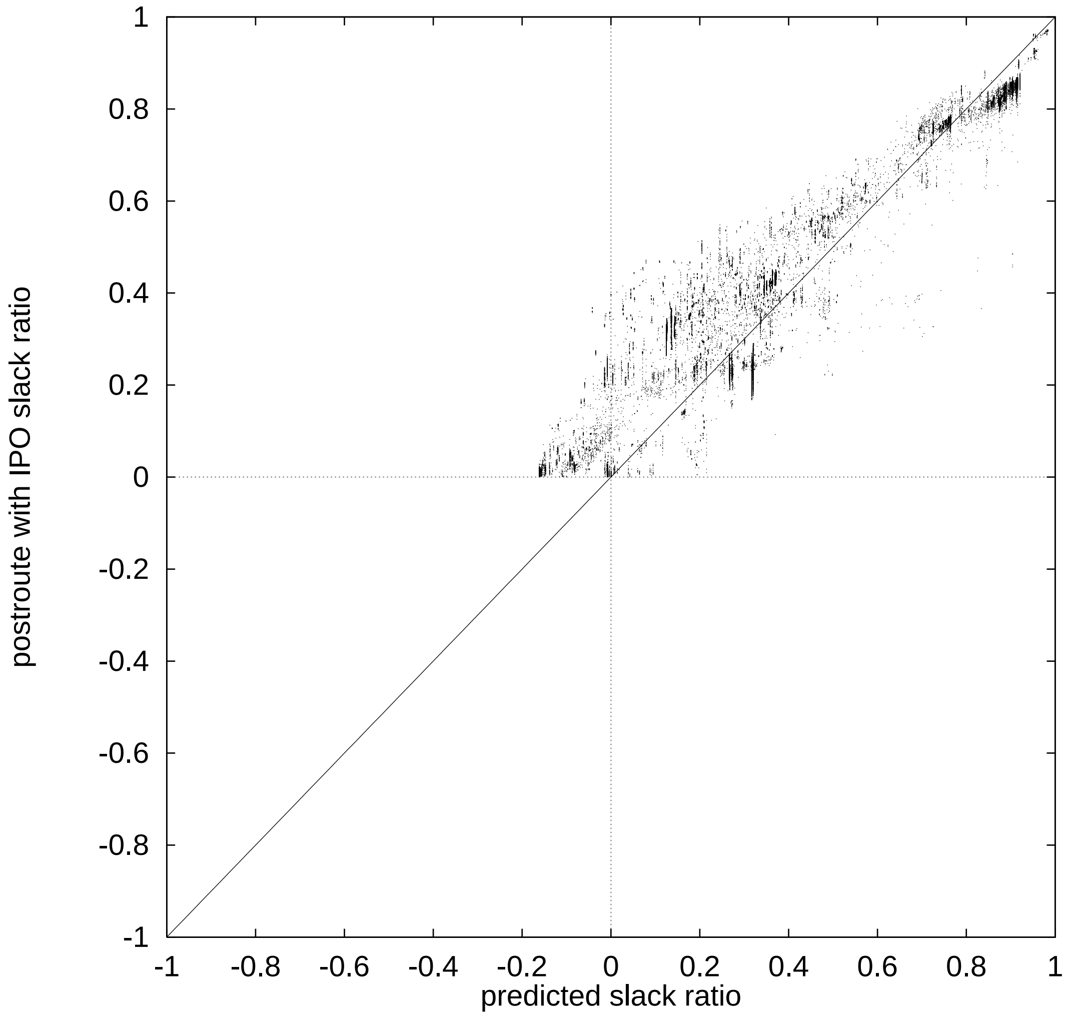

Figure 8. Slack ratio scatter plot showing IPO results

Figure 9 shows a different view of the same data. By plotting the post-route slack ratios versus the post-IPO slack ratios we can see which paths are affected by the IPO. Most paths have not been changed significantly, but the paths that were missing timing (below the *x* axis) have been "bent" up to the axis. This is called a "hockey stick" plot [24].

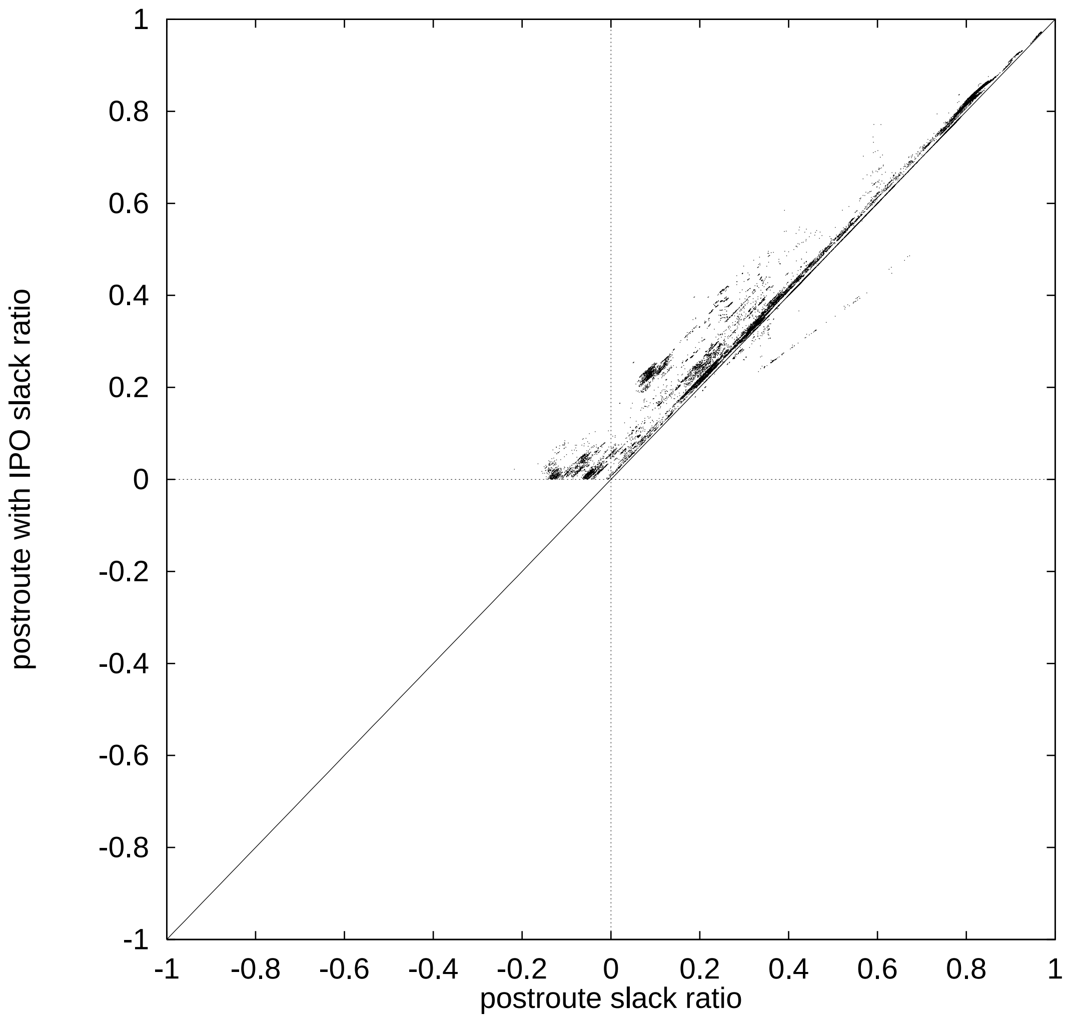

Figure 9. Slack ratio scatter plot showing IPO results

Yet another way to display slack ratios is to plot the cumulative number of paths that have less than a given slack ratio. This is a *slack ratio percentage plot* and an example is given in Figure 10. This shows the same data as Figure 9. Note that after IPO there are no paths with negative slack ratios. The entire bottom part of the curve has shifted to the right showing critical paths being fixed. Slower paths with larger slack were mostly unaffected.

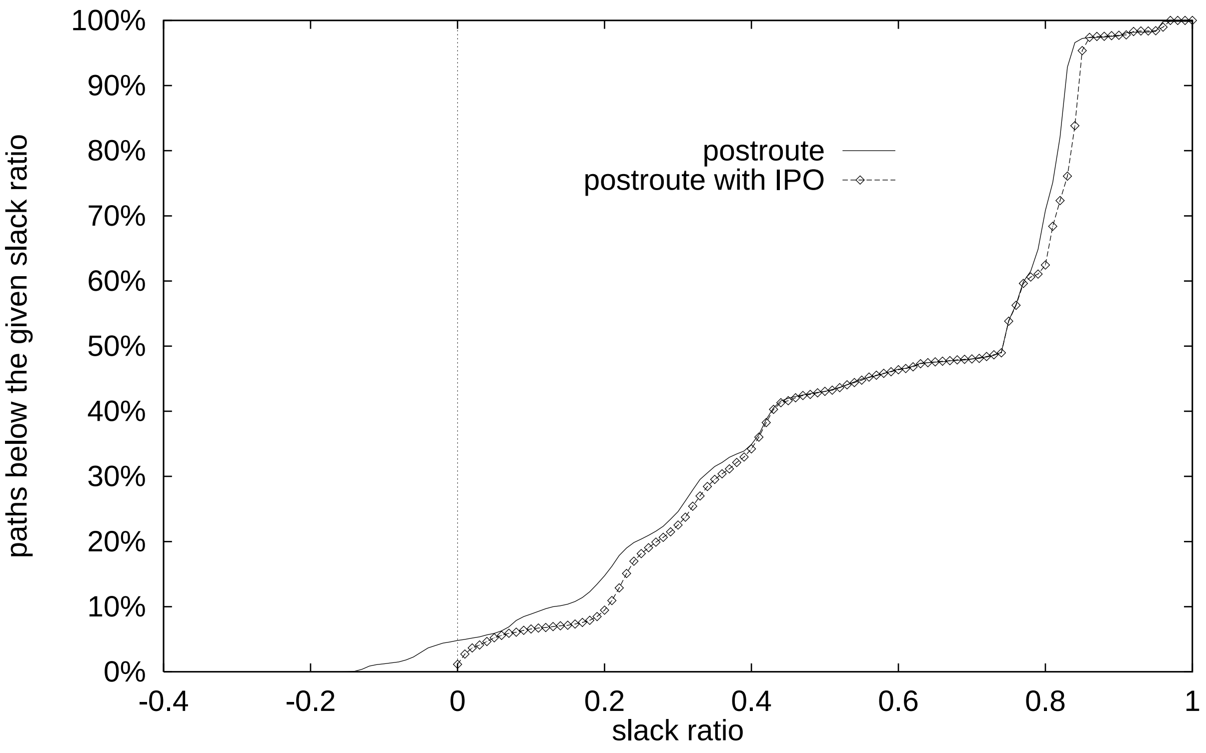

Figure 10. Slack ratio percentage plot showing IPO results

5.0 Suggestions

Ask your vendor how their wireload models are generated. What methodology do they use? What sorts of example designs? Do their models show good correlation between pre-route predicted delays, and post-route actual delays? What sort of margin do they have? Ask your vendor for slack ratio plots. Do them yourself for your own designs.

When generating your own custom wireload libraries be sure and use a unique library name, perhaps incorporating a date or revision number of some sort (e.g. `mylib_11_4_b`). The top of your timing reports will clearly show which version of your wireloads are being used. Note this may cause problems if you use `write_script` because you will get lines such as

```
  set_wire_load “100k” -library “mylib_11_4_b” …
```

throughout your script, which will cause many errors when you change the library names.

If you wish to create your own design-specific wireload models or libraries, plan on at least three layouts: the first using the vendor default wireloads, the second using your new custom wireloads, and a third that fixes up the wireloads that broke things on the second pass. For best results each trial should use identical RTL code, scripts, and constraints. The only change should be the wireload model. Generating these layouts will take some time. Your vendor may not want to do it. Your manager may not want to give you the time.

Make sure your models increase monotonically with fanout. If you are using automatic area-based wireload selection then your models must be monotonic with area as well.

It is difficult for a wireload model to accurately model nets within a design that are connected to a hard macro (such as a RAM). It is better to keep all hard macros in a separate level of hierarchy that is only interconnect. This allows the macro port nets to be forward-annotated with `set_load`.

6.0 Future work

How sensitive is Design Compiler to variations in wireload models? If the capacitance value for a fanout is changed by 1% will the netlist be different? What about 0.1%? 0.01%?

Use Howard Landman's techniques [12] to plot the results of the entire synthesis and layout flow. As the wireload changes, is Design Compiler's behavior stable or chaotic? What about the back-end tools?

The example slack ratio plots used all paths in the entire design. If different wireload models were used at different levels of the hierarchy, how can we measure the accuracy of each model? One technique may be to report flop-to-flop paths that are contained entirely within the particular level of hierarchy.

What sort of wireload models should be used during in-place optimization (IPO)? Perhaps this is a situation where the wireload model should agree exactly with post-route statistics. Does the model still need to be monotonic?

Typically all nets are treated equally when gathering post-route statistics. What happens if nets are weighted by whether or not they are on the critical path? Nets from paths with smaller slack would be given more weight in generating the statistical histograms. How does this change the shape and accuracy of the resulting wireloads?

Following the work of Baty and Bradshaw, generate simulated wireload statistics. Assume a square block of length L on a side. To model a net with fanout of n, randomly place $n + 1$ points in the block. Approximate the length of the minimum rectilinear Steiner tree that connects all the points (e.g. by calculating the half-perimeter measure or complete-graph measure [11]). Plot the resulting histograms and calculate the deciles. What happens if you change the aspect ratio of the block? For a given decile, is there a closed-form solution for the fanout vs length curve? How do these theoretical statistical results compare to actual data from real designs? How is this simple model different from a real place and route tool?

7.0 Conclusions

Back in 1994, an LSI Logic presentation stated that “Wiring is becoming THE MAJOR FACTOR for the overall delay in Submicron Technologies.”[1][2] It is clear that very soon the statistical nature of interconnect delay will overwhelm even the most careful wireload models. Wireload models will become obsolete. Such a simple model of interconnect delay is not sufficient. Accurate modelling of actual net delay will be necessary.

The current flow in Design Compiler is

1. Logic-level optimization
2. Map to technology
3. Gate-level optimization

Only step 3. requires the use of interconnect delay modelling. Eventually this flow must combine placement and synthesis [21][23] as follows:

1. Logic-level optimization
2. Map to technology *and* initial placement
3. Gate-level optimization *and* placement optimization

Currently the gate-level optimization step takes up the majority of synthesis runtime, and adding placement will make it even longer. However much of the gate-level optimization may no longer be necessary, since the synthesis tool will not need to improve parts of the netlist that are placed optimally. Routing is not necessary; with accurate placement the interconnect delay can be estimated with sufficient accuracy.

If full placement is too difficult then perhaps only the critical paths need to be placed in detail, and other cells can be given a rougher placement. Perhaps a relative placement can be done for critical path cells, rather than an absolute placement. Some FPGA synthesis tools do this today [25].

Note this new step 3. is very similar to the existing location-based optimization (LBO) supported by Floorplan Manager with the `reoptimize_design -in_place` command [13][16].

1. Emphasis in the original. Note the buzzword “deep submicron” had not been invented!

8.0 Acknowledgements

All data in this paper was taken from a 450k-gate design implemented in a 0.35µ triple-metal process. The plots were generated using `gnuplot` and some simple `perl` scripts.

Thanks to the following people for many stimulating discussions: Jerry Frenkil (Senté), Kurt Baty (WSFDB Consulting), Ken McElvain (Synplicity), Brent Dichter (Artisan), Francis Cheung (Mitsubishi), Mohammad Mohsin (Toshiba), Jay McDougal (Hewlett-Packard), Lee Bradshaw (Alantro Communications) and especially Michel Remillard (Lucent).

I am eternally grateful for the support and understanding of my wife Terry.

9.0 References

[1] Lee Bradshaw, "The Why & How of Creating Your Own Wire Loading Tables," *ESNUG*, Post 141 Item 3, August 1993.

[2] LSI Logic, "Synthesis Flow using Synopsys," in *SNUG 1994 Proceedings*.

[3] Jay D. McDougal and William E. Young, "Shortening the Time to Volume Production of High-Performance Standard Cell ASICs," *Hewlett-Packard Journal*, pp. 91-96, February 1995.

[4] Stefan Rusu, "Advanced Timing and Wire Models in Synopsys," in *SNUG 1995 Proceedings*.

[5] Jeff Buckles, "New Synthesis and Design Methods Reduce Gate-Array Layout Iterations," in *1996 On-Chip System Design Conference Proceedings.*

[6] Jay McDougal, Tim Brown, Jeff Hintzman, "Creating and Using Custom Wire Load Models," in *SNUG 1996 Proceedings*.

[7] Toshiba, "Toshiba/Synopsys Links to Layout Methodology," in *SNUG 1996 Proceedings*.

[8] Mohammad Mohsin, "Hierarchical Wire Load Estimation," in *SNUG 1997 Proceedings*.

[9] Hemant Joshi, Nirmala Karumanchi, and Vijayanand Angarai, "Optimizing Wire Load Models (WLMs) to Reduce Synthesis-Placement and Routing Iterations," in *SNUG 1997 Proceedings*.

[10] Doug Matzke, "Will Physical Scalability Sabotage Performance Gains?" *Computer*, vol. 30, no. 9, September 1997, pp. 37-39.

[11] Michael John Sebastian Smith, *Application-Specific Integrated Circuits*, Addison-Wesley, 1997.

[12] Howard A. Landman, “Visualizing the Behavior of Logic Synthesis Algorithms,” in *SNUG 1998 Proceedings*.

[13] Sandra Ma, Tzong-Maw Tsai, and Jean-Paul Meunier, “Logic Synthesis,” in *SNUG 1998 Tutorials*.

[14] “Design Compiler Reference Manual: Constraints and Timing,” Synopsys Online Documentation v1998.02.

[15] “Library Compiler User Guide,” Synopsys Online Documentation v1998.02.

[16] “Floorplan Manager User Guide,” Synopsys Online Documentation v1998.02.

[17] Dennis Sylvester, William Jiang, and Kurt Keutzer, “BACPAC - Berkeley Advanced Chip Performance Calculator,” http://www-inst.eecs.berkeley.edu/~dennis/bacpac/index.html.

[18] Dennis Sylvester and Kurt Keutzer, “Getting to the Bottom of Deep Submicron,” in *Proc. of International Conference on CAD*, pp. 203-211, 1998.

[19] Michael Keating and Pierre Bricaud, *Reuse Methodology Manual*, Kluwer Academic Publishers, 1998.

[20] Scott Hamilton, “Taking Moore’s Law Into the Next Century,” *Computer*, vol. 32, no. 1, January 1999, pp. 43-48.

[21] “SRC Synthesis Task Force Report,” December 1998, http://www.src.org/areas/design.dgw.

[22] Jay McDougal and Bill Young, “Achieving Timing Convergence between Synthesis and Place & Route,” in *1999 On-Chip System Design Conference Proceedings*.

[23] Barbara Tuck, “Linking logical and physical design,” *Electronic Systems*, vol. 38, no. 1, January 1999, pp. 58-64.

[24] Michel Remillard, Lucent Technologies, personal communication.

[25] Ken McElvain, Synplicity Inc., personal communication.

[26] Kurt Baty, personal communication.

[27] Lee Bradshaw, personal communication.