# On the Relevance of Wire Load Models*

Kenneth D. Boese
Cadence Design Systems, Inc.
San Jose, CA, USA
boese@cadence.com

Andrew B. Kahng
UCSD CSE and ECE Depts.
La Jolla, CA, USA
abk@ucsd.edu

Stefanus Mantik
UCLA CS Dept.
Los Angeles, CA, USA
stefanus@cs.ucla.edu

## ABSTRACT

Wire load models (WLMs) are generally perceived to be inaccurate and inadequate for good optimization. The traditional wisdom is that accuracy of WLMs will worsen as die sizes expand and feature sizes shrink, and as wire loads become less predictable and more dominant over pin loads. In many industry white papers and academic works, the weaknesses of WLMs are used to motivate the unification of logic synthesis and physical layout into a single tool. We believe, however, that care must be taken in how we derive our motivations for new flows. In previous studies, evidence against WLMs was generally anecdotal or based on limited data (e.g., from a single design). Today, the maturation of Cadence Design Systems' PKS design tool affords us a unique opportunity to study WLMs in greater depth, and to quantify the timing improvements achieved by the unification of synthesis and layout. Using PKS, we have performed extensive experiments on fifteen real industry test cases. Our results confirm much of the conventional wisdom about WLMs, but also indicate that WLMs probably can still perform a useful function in the design flow.

## 1. INTRODUCTION

The typical chip implementation flow starts with an RTL description and moves through logic synthesis and optimization in the front end – to placement, routing, extraction and performance analysis in the back end. At each step of the flow, the designer must ensure that design constraints are met with respect to timing, signal integrity, power and area. If at any step the constraints are violated, the design will need to be sent back one or more steps to be re-optimized. Backward iterations – those that use current layout solutions as estimates for the next pass of logic synthesis – have been necessary because of a fundamental chicken-and-egg problem: (1) the front-end designer needs knowledge of placement to estimate net parasitics during timing-driven optimization, while (2) timing-driven placement requires knowledge of the actual netlist area, connectivity, and timing. To reduce time-to-market, minimizing the number of backward iterations is a high-priority objective for CAD tools.

The most popular way to solve the chicken-and-egg dilemma is to estimate parasitics during logic optimization with a wire load model (WLM), a lookup table that maps the fanout of a net to the corresponding estimated capacitance and resistance.[1] We formally define a WLM as a pair of functions $cap(fo)$ and $res(fo)$ where the integer parameter $fo$ denotes the net fanout. According to [6], each WLM lookup table should contain a set of capacitance pairs and resistance pairs, such as:

```
fanout_capacitance( 1, 0.007250 )
fanout_resistance( 1, 0.036048 )
...
```

e.g., fanout-1 nets have wire capacitance 0.00725 pF and wire resistance of 0.036048 ohms. In general, not all fanouts are mentioned in a given WLM lookup table. For example, a WLM lookup table might only have capacitance and resistance values for fanouts 1, 2, 3, 4, 5, 10, 20, and 99. Estimates for fanouts in the gaps (e.g., from 6 to 9) are calculated using (linear) interpolation. Estimates for fanouts outside the range of WLM table fanouts (e.g., greater than 99) are calculated using extrapolation based on the values for the two closest fanouts in the table.

WLMs are often used in pre-placement optimization to drive speedups of critical paths. Since timing-driven placement plausibly makes nets on critical paths shorter than average, some *optimism* may be incorporated into the WLM. Thus, a WLM may actually consist of more than one lookup table, with each table corresponding to a different optimism level. There are several ways to incorporate the optimism level. If we use the WLMs that come from the (ASIC vendor's) design library, usually there are several tables from which we can select. We can also increase the optimism level of a WLM by multiplying all values in the WLM by some factor less than 1.[2] For example, we can use 0.25, 0.5, or 0.75.[3]

*Research at UCLA and UCSD was supported by a grant from the MARCO Gigascale Silicon Research Center


*SLIP'01,* March 31-April 1, 2001, Sonoma, California, USA.


[1]In addition to capacitance estimates, resistance estimates are also needed to model $RC$ delay through the wires. Source-to-sink $RC$'s can be estimated based on net capacitance and resistance in a number of ways. For example, a conservative model could simply multiply $R \times C$ to get the source-to-sink $RC$'s. A more optimistic star model divides both $R$ and $C$ by fanout to give a value of $R \times C / fanout^2$.

[2]The higher the optimism level, the smaller the constant factor is.

[3]We might also want to pick the optimism according to some percentile in the distribution of loads. The easiest way to do that is to

## "WLMs Considered Harmful"

WLMs are generally perceived to be inaccurate and to be inadequate for good optimization [7, 8]. The traditional wisdom is that inaccuracy of WLMs will worsen as die sizes expand and feature sizes shrink, and wire loads become more dominant over pin loads (and less predictable). The updated solution to the chicken-and-egg problem is then to combine synthesis with place and route into a single tool, so that designers can use post-placement timing accuracy to drive logic optimization.

In popular accounts (industry white papers, academic works), unification of logic synthesis and physical layout is motivated by the "failure" of WLMs. In several studies, WLMs are claimed to worsen both final design quality and timing convergence. However, to our knowledge, it has never been demonstrated systematically that WLMs are actually "harmful", nor have the specific mechanisms by when they harm convergence or quality of results been adequately identified. Furthermore, as the EDA industry has produced such integrated tools as Cadence Design Systems' PKS, Synopsys' Physical Compiler and Magma's Blast Fusion, it seems appropriate to reassess whether WLMs were really that "harmful", and to gauge how tool unification has actually affected the solution quality achievable by design optimization.[4]

Our main goal is to exercise care in how we derive our motivations for new flows. Previous studies give evidence against WLMs that has been either anecdotal or based on limited data (e.g., from a single design). Thus, our work attempts to give a more systematic appraisal of WLMs. We are motivated by (although we are not necessarily able to answer directly) such questions as:

- How inaccurate are WLMs?
- Are WLMs necessary for good pre-placement optimizations?
- Do WLMs enable good pre-placement estimates of post-placement design quality?
- What type of WLM, if any, is appropriately used in the design optimization flow?

## Organization of the Paper

In this paper, we report extensive experiments with Cadence Design Systems' PKS, variant WLM constructions and flows, and multiple industry test cases. Our experiments attempt to isolate the positive and negative contributions of WLMs, various qualities of WLMs, as well as the impact of various optimization steps in the flow. We offer several comments regarding the appropriate criterion of accuracy for WLMs, and we present a new WLM construction that appears to offer some advantages over standard WLM constructions.

The rest of our paper is organized as follows. In Section 2, we summarize previous works that either try to show that WLMs are harmful or propose the merging of logic synthesis with place and route. Background on different design flows that have been used, and the role of WLMs in each of these flows, is sketched in Section 3. Finally, we present our experimental studies in Section 4, and conclude in Section 5.

pick a percentile, find where that percentile fits (say on 2-pin nets, which are always abundant) in terms of a factor, and then multiply all entries in the WLM by that factor.

[4]In other words, while there is no doubt that integrating the design flow into a single tool can shorten the design cycle, we need to ask how much can it be expected to improve timing estimation and optimization.

# 2. RELATED WORKS

Several previous studies have been motivated by the assumption that WLMs are harmful. The inaccuracy of WLMs is noted in [7], wherein Lu et al. ascribe negative effects to the inaccuracy. The authors of [7] propose a two-phase approach that combines technology mapping with logic resynthesis for minimizing post-placement delays. This approach alleviates the need for WLMs and uses a post-placement delay model to control the resynthesis process. However, Lu et al. do not actually show that WLMs are responsible for design flows that fail to meet timing.

In the SLIP-2000 paper [8], Scheffer and Nequist make several compelling arguments as to why *interconnect estimation* (for which WLMs are just one exemplar) can never be successful. Order statistics arguments are used to justify the EDA industry shift to constructive (= global routing based) interconnect estimation: noisy estimators that are used to predict average- or sum-based metrics (e.g., total wire length) may be reasonably useful, but noisy estimators that are used to predict maximum- or minimum-based metrics (e.g., worst-case slack over all timing paths, or worst-case overcongestion over all global routing grids) will have large errors. Essentially, the latter type of estimate fails because the value in question mainly depends on a small part of the design, rather than on the entire design. (For example, when WLM-based analysis indicates that the design can be run with $X$ clock cycle, Scheffer et al. argue that such an estimate cannot be trusted because of the *law of small numbers* (i.e., there are only small number of nets that are on the critical path, but these nets are the ones that determine the speed of the entire design, instead of a statistical value derived from all nets).)

Scheffer and Nequist also give evidence of discrepancies arising from use of WLMs as an early estimator. They show that the timing analysis based on a WLM has better slacks than the actual routed design. This implies that WLM-driven optimization would stop "prematurely" based on an incorrect timing prediction, thus requiring yet another iteration through synthesis to correct the timing. (We note in Section 4.3 the possibility of such phenomena being not solely due to WLMs.)

Finally, a relevant concept [1, 2] is the distinction between *accuracy* and *fidelity*. Accuracy determines whether a model gives an accurate estimate of actual values, while fidelity determines how likely it is for an optimal solution according to a given estimator to also be nearly optimal according to the actual values. Obviously, accuracy implies fidelity, i.e., if we have an accurate estimation of our final solutions, we usually get results that have been expected. However, the converse – that inaccuracy implies infidelity – is not always true. Boese et al. indicate that inaccuracy of an estimate does not necessarily make the estimator an inadequate tool for optimization. In the case of WLMs, even if WLMs are inaccurate, they might yet guide the tool to high-quality solutions.

# 3. DESIGN FLOW BACKGROUND

## WLM Types

For flows that run timing-based logic optimization before placement, there are three basic types of WLMs that can be used:

- **Statistical** WLMs are based on averages over many similar designs using the same or similar physical libraries.
- **Structural** WLMs use information about neighboring nets, rather than just fanout and module size information.

- **Custom** WLMs are based on the current design after placement and routing, but before the current iteration of pre-placement synthesis.

In the Appendix, we give details of a new custom WLM that we propose. Our new WLM separates the design modules into several module clusters, based on the module areas. Fanouts in each module cluster are separated into several fanout ranges, and these ranges are used to determine the fanout indices for the WLM tables. The capacitance and resistance values for the new custom WLM are obtained from the detailed routed results after one pass of the default PKS flow with structural WLMs.

## Design Flow Types

Historically, there are at least five possible design flows for timing-driven optimization.

- **No-WLM flow**: Originally, cell delays dominated interconnect delays, and so wire load could be ignored, and all timing optimizations could be safely done before placement without WLMs.

- **Flow with Custom WLMs and Iterations**: This flow iterates between the front-end synthesis and back-end place and route in an attempt to provide an accurate (or at least a faithful) WLM for synthesis. In the first iteration, synthesis is run either with or without a WLM. After place-and-route, the net capacitance and resistance values are extracted and used to generate custom WLMs, which are passed back to the front-end for another iteration of synthesis. The iterations are repeated until either the constraints are met or the tools give up.

- **Flow with Statistical WLMs**: This flow attempts to reduce the number of front-end to back-end iterations. Statistical WLMs are included with the library information, based on similar designs using the same or similar libraries. The first iteration of synthesis uses the statistical WLM, then passes the resulting netlist to the back-end. Further iterations with custom WLMs may or may not be necessary, depending on the results of the first iteration.

- **Flow with Pre-Placement and Post-Placement Optimization**: The next generation of design flows adds logic optimization to the back-end. Designs are optimized before placement using WLMs. Then, after placement, a further timing optimization is performed for which parasitic values can be extracted directly using placement locations and Steiner tree or global routing estimates of wire lengths.

- **Flow with Post-Placement Timing Optimization Only**: Finally, based on the assumption that WLMs are "harmful" to timing convergence, timing-based logic optimizations are deferred to the back-end, and only area-based optimizations are performed before placement. No WLMs are needed. This ideal flow is approached in varying degrees by the new generation automation tools such as Cadence Design Systems' PKS, Synopsys' Physical Compiler and Magma's Blast Fusion. In practice, however, the tools may perform some timing-based optimizations before placement, using internally generated WLMs (e.g., see the default PKS flow below), gain-based ("logical effort" [9]) synthesis, etc.

# 4. EXPERIMENTS

Our experiments use Cadence Design Systems' Physically Knowledgeable Synthesis (PKS) tool, which integrates the BuildGates synthesis tool with the QPlace placer and the WarpRoute router. We run our experiments on fifteen industry regression test cases whose characteristics are outlined in Table 1.[5] Test cases marked with a $^*$ have already had some pre-placement timing optimization (e.g., by a different vendor's synthesis tool) before being brought into PKS. Consequently, the results for these 8 test cases may underestimate the importance of WLMs and of pre-placement optimization in general.

| Test Name | Num Cells | Cycle Time (ns) |
|---|---|---|
| Design01 | 884 | 7.52 |
| Design02 | 7665 | 17.00 |
| Design03 $^*$ | 7858 | 7.52 |
| Design04 $^*$ | 8320 | 3.75 |
| Design05 $^*$ | 10200 | 3.75 |
| Design06 | 13207 | 15.00 |
| Design07 | 14459 | 3.30 |
| Design08 $^*$ | 20124 | 3.75 |
| Design09 $^*$ | 20268 | 15.00 |
| Design10 | 53506 | 5.00 |
| Design11 $^*$ | 69151 | 100.00 |
| Design12 $^*$ | 73608 | 15.00 |
| Design13 | 92562 | 12.30 |
| Design14 $^*$ | 160267 | 18.50 |
| Design15 | 181927 | 10.00 |

**Table 1: Test cases used for our experiments.**

In this section, we describe our four basic experiments, which respectively attempt to shed light on the four questions given in Section 1. A caveat: it is not possible to obtain absolute answers to such general and "motivating" questions, and our work sheds less light on some questions than on others.

## 4.1 How Accurate Are WLMs?

In general, WLMs estimate wire capacitances with an average wire capacitance for a given *fanout* (i.e., number of pins in the net minus one). One measure of WLM accuracy would be the average deviation from the mean of wire capacitances for a given fanout. I.e., if the distribution of wire capacitances is very dispersed for a given fanout and design, then any WLM based on only fanout will be "inaccurate". Based on this or similar measures of accuracy, previous studies of WLMs have characterized them as inaccurate. Although wire loads are highly variable and their estimates are perhaps doomed to inaccuracy, however, what really matters in optimization are the *net* capacitances (i.e., wire load plus all pin loads on each net). Thus, when we consider the inaccuracy of average wire loads relative to net capacitances as a whole, the inaccuracy will be smaller and perhaps more acceptable for optimization.[6]

Table 2 summarizes the distributions of wire capacitance and net

[5] For some experiments, results for the last test case are unavailable at submission of the camera version of this paper.

[6] Strictly speaking, it is the accuracy of the *effective capacitance* value that matters when using table-based timing models. There are many variant methodologies for deriving effective capacitance from the total (or, components of) net capacitance, e.g., [5]. Thus, here we simply report statistics of total net capacitances.

| Design Name | # Cells (# Nets) | Fanout (% Nets) | Average Net Cap/ Average Wire Cap | Median Wire Cap/ Average Wire Cap | Average Wire Cap Percentile | Wire Cap StdDev (Ave Diff) | StdDev wrt WLM (Ave Diff) |
|---|---|---|---|---|---|---|---|
| Design01 | 884 (1081) | 1(66.36) | 13.453 | 0.519 | 76.07 | 0.280(0.065) | 0.280(0.065) |
| | | 2(17.51) | 6.967 | 0.671 | 67.56 | 0.145(0.102) | 0.145(0.102) |
| | | 3(4.86) | 4.346 | 0.926 | 57.40 | 0.151(0.113) | 0.151(0.113) |
| | | 4(3.76) | 3.277 | 0.855 | 60.48 | 0.202(0.138) | 0.202(0.138) |
| Design02 | 7665 (7948) | 1(59.71) | 1.933 | 0.235 | 81.06 | 1.368(0.623) | 1.359(0.610) |
| | | 2(25.47) | 1.971 | 0.236 | 61.59 | 0.651(0.514) | 0.593(0.407) |
| | | 3(5.91) | 1.986 | 0.277 | 75.05 | 0.695(0.544) | 0.624(0.449) |
| | | 4(2.03) | 2.514 | 0.605 | 74.78 | 0.473(0.271) | 0.459(0.269) |
| Design03 | 7858 (9088) | 1(65.14) | 5.853 | 0.121 | 83.05 | 1.153(0.238) | 1.149(0.226) |
| | | 2(18.04) | 3.852 | 0.218 | 73.43 | 1.187(0.299) | 1.182(0.290) |
| | | 3(6.50) | 3.284 | 0.359 | 73.50 | 1.201(0.301) | 1.199(0.300) |
| | | 4(4.04) | 2.802 | 0.304 | 79.90 | 1.761(0.398) | 1.752(0.400) |
| Design04 | 8320 (9194) | 1(60.62) | 1.816 | 0.103 | 74.76 | 1.026(0.708) | 1.026(0.708) |
| | | 2(22.80) | 1.397 | 0.302 | 82.64 | 1.481(0.799) | 1.481(0.799) |
| | | 3(5.62) | 1.273 | 0.262 | 78.30 | 1.203(0.889) | 1.203(0.889) |
| | | 4(3.69) | 1.329 | 0.303 | 74.34 | 1.109(0.790) | 1.109(0.790) |
| Design05 | 10200 (10688) | 1(53.17) | 2.629 | 0.237 | 71.87 | 0.700(0.434) | 0.700(0.434) |
| | | 2(31.62) | 1.559 | 0.457 | 74.23 | 0.965(0.582) | 0.965(0.582) |
| | | 3(7.70) | 1.433 | 0.383 | 75.49 | 1.134(0.694) | 1.134(0.694) |
| | | 4(2.60) | 1.405 | 0.487 | 69.14 | 0.826(0.594) | 0.826(0.594) |
| Design06 | 13207 (15523) | 1(68.47) | 1.970 | 0.235 | 85.01 | 2.336(0.637) | 2.301(0.590) |
| | | 2(20.65) | 2.221 | 0.289 | 88.95 | 1.896(0.549) | 1.863(0.549) |
| | | 3(6.43) | 2.256 | 0.340 | 83.59 | 2.531(0.486) | 2.238(0.405) |
| | | 4(0.95) | 2.070 | 0.379 | 75.27 | 0.716(0.453) | 1.337(0.862) |
| Design07 | 14459 (14565) | 1(53.90) | 6.969 | 0.275 | 82.98 | 0.379(0.172) | 0.379(0.172) |
| | | 2(19.75) | 4.487 | 0.392 | 77.05 | 0.375(0.223) | 0.375(0.223) |
| | | 3(9.02) | 2.921 | 0.411 | 74.60 | 0.496(0.328) | 0.496(0.328) |
| | | 4(6.24) | 3.231 | 0.527 | 72.91 | 0.415(0.259) | 0.415(0.259) |
| Design08 | 20124 (20281) | 1(48.40) | 3.430 | 0.203 | 77.53 | 0.668(0.350) | 0.668(0.350) |
| | | 2(24.44) | 1.709 | 0.474 | 69.37 | 0.789(0.532) | 0.789(0.532) |
| | | 3(14.68) | 1.627 | 0.542 | 71.63 | 1.063(0.514) | 1.063(0.514) |
| | | 4(5.37) | 1.767 | 0.652 | 67.10 | 0.562(0.384) | 0.562(0.384) |
| Design09 | 20268 (20347) | 1(67.08) | 1.236 | 0.114 | 74.93 | 1.531(1.029) | 1.185(0.691) |
| | | 2(3.23) | 1.177 | 0.297 | 61.09 | 1.066(0.831) | 2.202(1.008) |
| | | 3(22.32) | 1.192 | 0.694 | 62.53 | 0.775(0.604) | 0.994(0.610) |
| | | 4(1.02) | 1.741 | 0.194 | 87.82 | 4.883(0.754) | 3.687(0.495) |
| Design10 | 53506 (55274) | 1(69.49) | 1.451 | 0.198 | 80.18 | 3.812(0.862) | 3.750(0.724) |
| | | 2(14.28) | 1.404 | 0.302 | 76.35 | 2.524(0.781) | 2.451(0.645) |
| | | 3(7.21) | 1.543 | 0.415 | 77.79 | 3.291(0.629) | 3.155(0.613) |
| | | 4(2.48) | 1.395 | 0.590 | 65.34 | 0.806(0.553) | 0.795(0.535) |
| Design11 | 69151 (79826) | 1(72.78) | 2.863 | 0.071 | 87.61 | 2.969(0.537) | 2.950(0.514) |
| | | 2(16.29) | 2.025 | 0.057 | 85.48 | 3.462(0.743) | 3.424(0.710) |
| | | 3(4.85) | 2.970 | 0.040 | 92.77 | 2.412(0.568) | 2.351(0.565) |
| | | 4(2.07) | 2.989 | 0.044 | 91.93 | 2.226(0.567) | 2.000(0.462) |
| Design12 | 73608 (74544) | 1(68.83) | 1.943 | 0.040 | 85.76 | 6.791(0.806) | 6.715(0.761) |
| | | 2(13.24) | 1.503 | 0.093 | 78.47 | 1.402(0.876) | 1.297(0.766) |
| | | 3(5.91) | 1.606 | 0.171 | 79.38 | 1.346(0.777) | 1.229(0.684) |
| | | 4(3.91) | 1.633 | 0.341 | 73.70 | 1.189(0.631) | 1.034(0.511) |
| Design13 | 92562 (96667) | 1(74.96) | 6.395 | 0.095 | 82.26 | 1.748(0.210) | 1.721(0.220) |
| | | 2(15.18) | 3.491 | 0.055 | 92.02 | 3.568(0.449) | 3.257(0.416) |
| | | 3(3.21) | 3.544 | 0.067 | 92.59 | 3.565(0.445) | 3.226(0.333) |
| | | 4(2.01) | 1.271 | 0.367 | 67.52 | 1.047(0.773) | 7.659(2.425) |
| Design14 | 160267 (178300) | 1(70.18) | 1.965 | 0.028 | 90.33 | 9.065(0.829) | 9.000(0.815) |
| | | 2(16.37) | 1.885 | 0.063 | 89.45 | 6.982(0.817) | 6.848(0.818) |
| | | 3(5.94) | 2.168 | 0.061 | 93.90 | 6.260(0.755) | 6.144(0.766) |
| | | 4(2.79) | 2.037 | 0.124 | 89.71 | 6.764(0.720) | 6.487(0.694) |
| Design15 | 181927 (182929) | 1(61.94) | 1.764 | 0.009 | 89.73 | 2.787(0.946) | 2.637(0.766) |
| | | 2(16.93) | 1.703 | 0.015 | 91.47 | 3.874(0.980) | 3.633(0.850) |
| | | 3(8.17) | 1.762 | 0.016 | 96.60 | 4.143(1.051) | 2.937(0.559) |
| | | 4(3.88) | 2.087 | 0.012 | 92.82 | 2.082(0.853) | 1.297(0.381) |

**Table 2: Parameters of the wire capacitance and net capacitance distributions. More details about the contents of each column can be found in the text.**

capacitance based on *detailed routing* in the default PKS flow. The first three columns give statistics of the test case netlist and fanout distributions after technology mapping. The fourth through sixth columns show the difference between wire capacitance and total net capacitance, as well as aspects of the wire capacitance distribution. Figure 1 shows the cumulative distribution of wire capacitances for Design10 (after the Structural WLM flow; see definitions below). The average wire capacitance (0.04026) is at the 80 percentile mark. and only 10 percent of the nets have wire capacitance greater than 0.1 pF. We see from the Figure (and from Table 2) that the distribution of wire capacitances is highly skewed: there are a few nets with very large capacitance while the vast majority of nets have small capacitance. This may be another factor behind the difficulty of achieving accurate WLMs: an average wire load overestimates load for the vast majority of nets, but underestimates

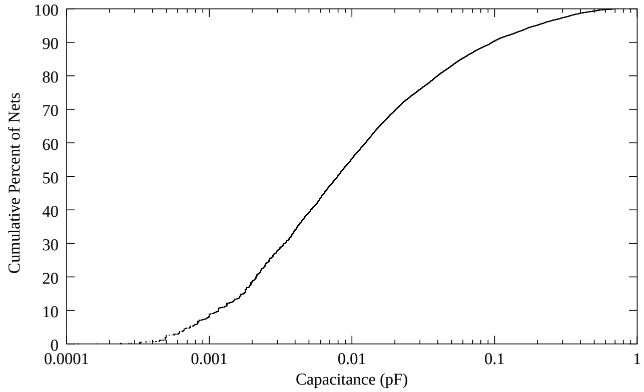


**Figure 1: Cumulative distribution of wire capacitances for all two-pin nets in the Design10 test case after execution of Structural WLM flow.**

load for the largest and, presumably, most troublesome nets.[7]

The sixth column of Table 2 gives two values: (i) (no parentheses) the standard deviation of wire capacitance, divided by average net capacitance, and (ii) (parentheses) the average absolute deviation of wire capacitance from median wire capacitance, divided by the average net capacitance. The seventh and final column of the Table also gives two values: (i) (no parentheses) a "standard error" of deviation of wire capacitance from the Custom WLM value, divided by average net capacitance, and (ii) (parentheses) the average absolute deviation of wire capacitance from the Custom WLM value, divided by the average net capacitance.[8] We see that these deviations are actually very small for small designs, but that they quickly increase with design size. Our conclusion is that WLMs can be "reasonably accurate" for small designs, but probably not for larger designs. The real conclusion is that any discussion of the efficacy of WLMs based only on the distribution of wire loads is mostly speculation. What we need is a study of how well WLMs work in real optimization flows. The following subsections present our experimental results to this end.

## 4.2 Are WLMs Necessary for Good Pre-Placement Optimizations?

To find whether WLMs are necessary for optimization, we need to run several experiments that use WLMs in several different ways (outlined in Section 3). We also use different types of WLMs to assess their effects on results. We run each test case on seven different flows, which are variations of a basic design flow template outlined in Table 3.

The seven flows we test vary in steps 3 and 5 only, as described below:

- *Structural WLM*: Step 3 = low-effort timing optimization with proprietary structural WLM.[9]

[7]However, it is possible that after some post-placement optimization, the large nets will not present a problem because (1) good timing-driven placement will allow only non-critical nets to be large; and (2) buffer insertion will cut down the size of the large nets.

[8]To be explicit: the (no parentheses) value in Column 6 (Column 7) is the square root of the average squared deviation of wire capacitance from mean wire capacitance (Custom WLM value), divided by average net capacitance.

[9]The Structural WLM we use is a proprietary WLM generated in-

| Default Flow |
|---|
| 1. Area-driven generic-cell optimization |
| 2. Area-driven technology mapping |
| 3. Pre-placement timing-driven optimization |
| 4. Timing-driven placement |
| 5. Post-placement timing-driven optimization |
| 6. Timing-driven global routing |

**Table 3: Default optimization flow.**

- *Library WLM*: Step 3 = low-effort timing optimization with most optimistic statistical WLM in the library.
- *No prePlaceOpt*: Step 3 is skipped.
- *No postPlaceRestruct*: Step 3 = same as in Library WLM; Step 5 = post-placement timing optimization with no logic restructuring.
- *No postPlaceOpt*: Step 3 = timing optimization with most conservative WLM in the library; Step 5 is skipped.
- *No WLM (WLM = 0)*: Step 3 uses a WLM in which all wire loads equal zero.
- *Custom WLM*: Step 3 uses a custom WLM generated as described in the Appendix (with "optimism" level 2).

Table 4 presents our experimental results for the seven flows on the 14 different test cases.[10] Note that, strictly speaking, our experimental results are valid only for one particular tool (PKS), and one particular version of that tool (v 4.0.0), and for the particular test cases used. Although this can be viewed as a weaknesses of our investigation, it is also closely related to one of its greatest strengths: that we are running our tests using a **real** industry tool on **real** industry designs. If we compare the average global routing slacks for the flows in Table 4, we obtain the ordering (from best to worst) of No prePlaceOpt: -1.449; Structural WLM: -1.460; No WLM(=0): -1.491; Library WLM: -1.495; No PostLogicOpt: -1.678; Custom WLM: -1.862; and No PostPlaceOpt: -6.058. Thus, flows without WLMs (no prePlaceOpt and 0-WLM) give about the same result qualities as those with WLMs. However, the WLM flows use consistently less running time, and for that reason are probably preferable to non-WLM flows. The most obvious conclusion from Table 4 is that some post-placement optimization is necessary to obtain good results: on average they improve the worst-case slacks by about 4.5 ns. We note, however, that the more complicated post-placement logic optimizations by themselves improved timing by only about 0.2 ns on average.

## 4.3 Do WLMs Enable Good Pre-Placement Estimates of Post-Placement Design Quality?

To shed light on this question, we run pre-placement optimization using our new WLM for each design (with optimism level of 1). We then compare the slacks estimated using the WLM with the actual slacks obtained after post-placement optimization and

ternally by PKS. It requires no special input from the user or from the library.

[10] All timing results are based on parasitics extracted from global routing, not detailed routing, which we justify in Section 4.5.

| Design Name | Flow | CPU (min) | GRoute-Based Slack (ns) | Steiner-Based Slack (ns) | % Row Util | % Over Cap |
|---|---|---|---|---|---|---|
| Design01 | Structural WLM | 20.96 | -0.535 | -0.488 | 81.67 | 0.16 |
| | **Library WLM** | 21.93 | **-0.489** | -0.450 | 81.61 | 0.26 |
| | Custom WLM | 8.48 | -0.539 | -0.454 | 81.57 | 0.31 |
| | No preOpt | 44.14 | -0.549 | -0.430 | 83.90 | 0.23 |
| | No WLM (=0) | 33.94 | -0.539 | -0.425 | 82.75 | 0.28 |
| | No postLogicOpt | 9.61 | -0.568 | -0.475 | 81.61 | 0.23 |
| | No postOpt | 21.83 | -0.531 | -0.518 | 80.40 | 0.09 |
| Design02 | Structural WLM | 23.07 | -4.257 | -4.262 | 64.33 | 0.29 |
| | Library WLM | 26.43 | -4.259 | -4.264 | 67.07 | 0.19 |
| | **Custom WLM** | 25.81 | **-4.253** | -4.254 | 67.04 | 0.17 |
| | No preOpt | 24.92 | -4.255 | -4.258 | 69.24 | 0.20 |
| | No WLM (=0) | 26.87 | -4.258 | -4.263 | 67.44 | 0.21 |
| | No postLogicOpt | 23.46 | -4.259 | -4.264 | 67.09 | 0.20 |
| | No postOpt | 22.43 | -4.303 | -4.337 | 68.40 | 0.13 |
| Design03 * | Structural WLM | 71.44 | -1.630 | -1.328 | 79.09 | 2.24 |
| | Library WLM | 90.16 | -1.380 | -1.347 | 80.77 | 2.51 |
| | Custom WLM | 27.90 | -1.520 | -1.390 | 80.38 | 2.28 |
| | **No preOpt** | 99.55 | **-1.370** | -1.314 | 78.98 | 2.46 |
| | No WLM (=0) | 74.71 | -1.400 | -1.332 | 80.82 | 2.31 |
| | No postLogicOpt | 49.00 | -1.460 | -1.400 | 80.80 | 2.56 |
| | No postOpt | 56.96 | -2.100 | -1.940 | 80.06 | 2.17 |
| Design04 * | **Structural WLM** | 210.00 | **-0.104** | -0.046 | 62.51 | 0.03 |
| | Library WLM | 335.77 | -0.280 | -0.188 | 61.31 | 0.03 |
| | Custom WLM | 157.29 | -0.158 | -0.093 | 61.52 | 0.03 |
| | No preOpt | 649.05 | -0.280 | -0.177 | 61.02 | 0.04 |
| | No WLM (=0) | 381.17 | -0.170 | -0.072 | 61.66 | 0.03 |
| | No postLogicOpt | 245.85 | -0.719 | -0.633 | 61.13 | 0.04 |
| | No postOpt | 53.17 | -11.599 | -11.686 | 57.08 | 0.12 |
| Design05 * | Structural WLM | 98.45 | -0.130 | -0.401 | 79.42 | 0.00 |
| | **Library WLM** | 141.70 | **-0.120** | -0.033 | 79.49 | 0.00 |
| | Custom WLM | 68.06 | -0.130 | -0.240 | 79.64 | 0.00 |
| | **No preOpt** | 114.20 | **-0.120** | -0.091 | 79.80 | 0.00 |
| | **No WLM (=0)** | 144.52 | **-0.120** | -0.033 | 79.49 | 0.00 |
| | No postLogicOpt | 109.14 | -0.150 | -0.089 | 79.55 | 0.00 |
| | No postOpt | 24.11 | -4.720 | -5.190 | 75.07 | 0.02 |
| Design06 | **Structural WLM** | 110.89 | **-0.086** | -0.126 | 75.19 | 0.00 |
| | Library WLM | 143.81 | -0.138 | -0.140 | 76.34 | 0.00 |
| | Custom WLM | 105.20 | -0.151 | -0.142 | 75.52 | 0.00 |
| | No preOpt | 373.92 | -0.298 | -0.300 | 78.14 | 0.06 |
| | No WLM (=0) | 233.52 | -0.421 | -0.491 | 78.24 | 0.01 |
| | No postLogicOpt | 156.23 | -0.500 | -0.560 | 78.33 | 0.02 |
| | No postOpt | 390.13 | -0.359 | -0.482 | 80.04 | 0.00 |
| Design07 | Structural WLM | 201.72 | -0.522 | -0.534 | 58.24 | 0.82 |
| | Library WLM | 246.51 | -0.576 | -0.549 | 57.56 | 0.88 |
| | Custom WLM | 111.94 | -0.483 | -0.483 | 57.56 | 0.80 |
| | No preOpt | 545.04 | -0.387 | -0.370 | 58.44 | 0.88 |
| | **No WLM (=0)** | 405.36 | **-0.363** | -0.352 | 58.34 | 0.89 |
| | No postLogicOpt | 127.03 | -0.665 | -0.656 | 57.07 | 0.84 |
| | No postOpt | 332.44 | -1.552 | -1.587 | 55.90 | 0.80 |
| Design08 * | Structural WLM | 930.12 | -0.430 | -0.387 | 33.46 | 0.06 |
| | Library WLM | 780.77 | -0.479 | -0.352 | 31.22 | 0.04 |
| | **Custom WLM** | 384.60 | **-0.396** | -0.320 | 31.21 | 0.03 |
| | No preOpt | 1442.58 | -0.578 | -0.498 | 31.78 | 0.04 |
| | No WLM (=0) | 815.21 | -0.529 | -0.405 | 30.98 | 0.02 |
| | No postLogicOpt | 751.37 | -0.441 | -0.352 | 31.35 | 0.04 |
| | No postOpt | 287.69 | -3.027 | -6.893 | 28.18 | 0.01 |
| Design09 * | Structural WLM | 97.90 | -3.352 | -3.356 | 72.08 | 6.84 |
| | Library WLM | 122.20 | -3.365 | -3.358 | 81.28 | 18.54 |
| | Custom WLM | 107.64 | -3.353 | -3.349 | 81.08 | 12.45 |
| | No preOpt | 153.05 | -3.503 | -3.498 | 86.16 | 21.56 |
| | **No WLM (=0)** | 108.17 | **-3.351** | -3.347 | 81.34 | 14.12 |
| | No postLogicOpt | 91.10 | -3.365 | -3.358 | 81.28 | 18.54 |
| | No postOpt | 72.95 | -6.095 | -6.780 | 80.22 | 10.16 |
| Design10 | Structural WLM | 547.53 | -1.639 | -1.072 | 75.17 | 2.74 |
| | **Library WLM** | 588.41 | **-0.850** | -0.806 | 74.58 | 1.57 |
| | Custom WLM | 376.88 | -1.092 | -1.032 | 75.14 | 1.50 |
| | No preOpt | 757.71 | -0.857 | -0.756 | 75.18 | 1.75 |
| | No WLM (=0) | 685.08 | -0.853 | -0.795 | 74.59 | 1.55 |
| | No postLogicOpt | 477.86 | -1.045 | -1.069 | 74.42 | 1.59 |
| | No postOpt | 1770.56 | -2.651 | -2.361 | 87.92 | 2.38 |
| Design11 * | Structural WLM | 392.10 | -5.500 | -5.263 | 64.70 | 0.04 |
| | Library WLM | 461.57 | -6.790 | -6.818 | 62.49 | 0.10 |
| | Custom WLM | 207.04 | -8.800 | -7.470 | 62.51 | 0.07 |
| | **No preOpt** | 407.55 | **-5.330** | -5.193 | 64.78 | 0.06 |
| | No WLM (=0) | 454.16 | -6.420 | -6.454 | 62.52 | 0.07 |
| | No postLogicOpt | 350.16 | -6.990 | -6.818 | 62.49 | 0.10 |
| | No postOpt | 290.80 | -18.100 | -16.720 | 62.57 | 0.09 |
| Design12 * | Structural WLM | 560.91 | -0.760 | 0.001 | 29.11 | 0.25 |
| | **Library WLM** | 746.05 | **-0.230** | -0.000 | 29.18 | 0.47 |
| | Custom WLM | 409.98 | -1.000 | -0.000 | 29.09 | 0.68 |
| | No preOpt | 615.70 | -0.640 | -0.004 | 29.14 | 0.45 |
| | No WLM (=0) | 745.98 | -0.230 | -0.000 | 29.18 | 0.47 |
| | No postLogicOpt | 834.05 | -0.740 | 0.004 | 29.19 | 0.46 |
| | No postOpt | 189.46 | -14.310 | -17.430 | 27.22 | 0.48 |
| Design13 | **Structural WLM** | 2364.30 | **-1.421** | -1.460 | 47.15 | 0.82 |
| | Library WLM | 3010.16 | -1.522 | -1.232 | 47.15 | 0.83 |
| | Custom WLM | 505.72 | -4.029 | -4.095 | 46.04 | 0.83 |
| | No preOpt | 3407.56 | -1.601 | -1.359 | 47.15 | 0.82 |
| | No WLM (=0) | 2602.16 | -1.714 | -1.263 | 46.84 | 0.80 |
| | No postLogicOpt | 2288.88 | -2.144 | -1.616 | 46.64 | 0.81 |
| | No postOpt | 740.74 | -10.250 | -11.035 | 45.70 | 0.87 |
| Design14 * | **Structural WLM** | 1555.44 | -0.072 | -0.277 | 30.33 | 0.05 |
| | Library WLM | 3899.80 | -0.450 | -0.083 | 30.35 | 0.05 |
| | Custom WLM | 1267.40 | -0.170 | -0.038 | 30.38 | 0.05 |
| | No preOpt | 1391.81 | -0.520 | -0.183 | 30.34 | 0.05 |
| | No WLM (=0) | 3383.97 | -0.43 | -0.083 | 30.35 | 0.05 |
| | No postLogicOpt | 3168.18 | -0.440 | -0.427 | 30.38 | 0.05 |
| | No postOpt | 3068.94 | -5.210 | -5.660 | 30.78 | 0.04 |

**Table 4: Comparison of global routing slacks for the seven main flows. Table entries with bold font show the best flow from among all other flows. Test cases with * have had some pre-placement timing optimization before being read into PKS, and so may underestimate the importance of WLMs.**

global routing. This allows us to assess whether the WLM alone is sufficient for an estimation of the final result. Table 5 shows the estimated worst slack for each design compared with the actual slacks.

| Design Name | PreOpt Slack | Placement Slack | PostOpt Slack | GRoute Slack | % Est. Error |
|---|---|---|---|---|---|
| Design01 | -0.448 | -0.546 | -0.455 | -0.518 | 0.94 |
| Design02 | -4.250 | -4.276 | -4.254 | -4.253 | 0.02 |
| Design03 | -1.417 | -2.01 | -1.34 | -1.52 | 1.37 |
| Design04 | 0.000 | -6.579 | -0.190 | -0.283 | 7.55 |
| Design05 | 0.001 | -3.63 | -0.09 | -0.15 | 4.03 |
| Design06 | -1.215 | -0.624 | -0.048 | -0.071 | -7.62 |
| Design07 | -0.388 | -1.523 | -0.443 | -0.441 | 1.61 |
| Design08 | -0.095 | -4.856 | -0.463 | -0.545 | 11.98 |
| Design09 | -3.477 | -4.555 | -3.348 | -3.350 | -0.84 |
| Design10 | -0.818 | -2.850 | -1.326 | -1.277 | 9.19 |
| Design11 | -2.878 | -25.41 | -7.63 | -9.41 | 6.53 |
| Design12 | 0.010 | -12.88 | -0.00 | -0.85 | 5.74 |
| Design13 | -0.059 | -10.822 | -1.274 | -1.401 | 10.92 |
| Design14 | -1.881 | -2.62 | -0.16 | -0.52 | -7.36 |

**Table 5: Worst slacks for each design estimated by our custom WLM (PreOpt), and compared to the Steiner-tree based slacks directly after placement (Placement) and after post-placement optimization (PostOpt), as well as to global routing slacks after post-placement optimization (GRoute). The percent error in the last column is the difference between estimated slacks (pre-placement) and the global routing slacks, relative to the target cycle time specified in the design constraints.**

Table 5 shows that the WLMs almost consistently produce slightly inaccurate estimates of the actual slack values, and that these estimates worsen as design size increases. These data seem to confirm the conclusions of Scheffer and Nequist in SLIP-2000 [8]. However, it is interesting to note that most of the larger designs have actual slack values about 12 percent worse than the predicted value. This leaves open the possibility of estimating the actual achievable slack by simply inflating WLM estimates by 12 percent.

We are currently pursuing additional experiments to determine more specifically whether estimation inaccuracies are caused by the WLM alone, or by other steps in the flow. For example, if the optimization flow results in a netlist and layout ("Layout 1") that meets timing, it is possible to put the netlist (and even back-annotated parasitics) back into the place and route flow to see whether the same timing is achieved (i.e., in "Layout 2"). Any disimprovement of solution quality will reflect a suboptimality in place and route (since we have a known achievable solution quality in Layout 1). Such experiments may eventually shed more light on interactions between WLMs, timing optimizations and layout.[11] On the other hand, it is not entirely clear what conclusions could be drawn from such experiments. If Layout 2 has worse timing than Layout 1, it might be because placement gives poor results with highly optimized netlists: it might do better with netlists that have only been partially optimized, and that have more area available to fix problems (e.g., long nets) after placement.

[11]Experiments in a similar spirit appear in Bodapati and Najm's work [3] (identifying "noise sources" in predictability of standard-cell place and route wirelengths) and in Hagen et al.'s work [4] (on quantification of suboptimality in layout heuristics).

## 4.4 What is the Appropriate Role of WLMs in the Design Optimization Flow?

The final motivating question addresses how WLMs should be used in the optimization flow, and the circumstances where they should not be used at all. If we use WLMs, what kind of WLMs should we use and what options do we need to specify to obtain the best results? What other steps need to be applied when we use WLMs? To help answer these questions, we ran the experiments on seven different flows that were presented in Table 4. We have also run the Custom WLM flow on our test cases with three different optimism levels (4, 2 and 1, where the multiplying factor for the WLMs are 0.25, 0.5 and 1.00 respectively). Table 6 describes the results of this experiment.

| Design | Optimism Level | CPU (min) | Slack (ns) | % Row Util |
|---|---|---|---|---|
| Design01 | 4 (0.25) | 8.26 | -0.539 | 81.57 |
| | 2 (0.50) | 8.48 | -0.539 | 81.57 |
| | 1 (1.00) | 8.75 | -0.518 | 81.46 |
| Design02 | 4 (0.25) | 24.82 | -4.252 | 67.03 |
| | 2 (0.50) | 25.81 | -4.253 | 67.04 |
| | 1 (1.00) | 25.49 | -4.253 | 66.90 |
| Design03 | 4 (0.25) | 36.78 | -1.410 | 80.59 |
| | 2 (0.50) | 27.90 | -1.520 | 80.38 |
| | 1 (1.00) | 41.62 | -1.520 | 80.68 |
| Design04 | 4 (0.25) | 183.34 | -0.227 | 61.77 |
| | 2 (0.50) | 157.29 | -0.158 | 61.52 |
| | 1 (1.00) | 166.02 | -0.283 | 61.64 |
| Design05 | 4 (0.25) | 49.48 | -0.130 | 79.35 |
| | 2 (0.50) | 68.06 | -0.130 | 79.64 |
| | 1 (1.00) | 59.71 | -0.150 | 79.31 |
| Design06 | 4 (0.25) | 110.27 | -0.173 | 75.84 |
| | 2 (0.50) | 105.20 | -0.151 | 75.52 |
| | 1 (1.00) | 55.48 | -0.071 | 76.44 |
| Design07 | 4 (0.25) | 137.13 | -0.489 | 57.97 |
| | 2 (0.50) | 111.94 | -0.483 | 57.56 |
| | 1 (1.00) | 117.41 | -0.441 | 57.77 |
| Design08 | 4 (0.25) | 284.96 | -0.562 | 31.03 |
| | 2 (0.50) | 384.60 | -0.396 | 31.21 |
| | 1 (1.00) | 378.11 | -0.544 | 31.17 |
| Design09 | 4 (0.25) | 107.88 | -3.348 | 81.24 |
| | 2 (0.50) | 107.64 | -3.353 | 81.08 |
| | 1 (1.00) | 116.33 | -3.350 | 81.21 |
| Design10 | 4 (0.25) | 413.24 | -0.984 | 75.32 |
| | 2 (0.50) | 376.88 | -1.092 | 75.14 |
| | 1 (1.00) | 265.06 | -1.277 | 74.87 |
| Design11 | 4 (0.25) | 198.37 | -7.020 | 62.46 |
| | 2 (0.50) | 207.04 | -8.800 | 62.51 |
| | 1 (1.00) | 211.02 | -9.410 | 62.58 |
| Design12 | 4 (0.25) | 384.77 | -0.350 | 29.03 |
| | 2 (0.50) | 409.98 | -1.000 | 29.09 |
| | 1 (1.00) | 363.11 | -0.850 | 28.90 |
| Design13 | 4 (0.25) | 888.21 | -1.407 | 47.08 |
| | 2 (0.50) | 505.72 | -4.029 | 46.04 |
| | 1 (1.00) | 862.85 | -1.401 | 46.92 |
| Design14 | 4 (0.25) | 1963.05 | -0.430 | 30.35 |
| | 2 (0.50) | 1267.40 | -0.170 | 30.38 |
| | 1 (1.00) | 1549.06 | -0.520 | 30.39 |

**Table 6: Comparison of global routing slacks for the custom WLM with three levels of optimism.**

The results in Tables 4 and 6 suggest that the type and optimism levels of WLMs used does affect the final results very much, or at least that it is very hard to predict which WLM will give the best results. Thus, it seems that our best recommendation is that the designers guess at the correct WLM type and optimism level, or if possible, try several different runs and keep the best result.

## 4.5 Slacks after GRoute vs. FRoute.

Up to this point, we have reported all timing slacks using global routing-based parasitics, due to longer running times required by detailed routing. Table 7 supports this decision by comparing the worst setup slacks based on global routing (GRoute) with the worst setup slacks based on detailed or final routing (FRoute). For each PKS flow of each design, we do the final routing to generate a routed solutions from which we get the worst final routing slacks. The last column contains the percent difference between the two quality measures relative to the cycle time specified by the timing constraints. In the table, the percent difference between GRoute and FRoute slacks is usually less than three percent. Thus, we believe it is reasonable to report GRoute-based slacks instead of FRoute-based slacks in our experiments. (Note that net parasitics generated by extraction tools such as Hyperextract or Fire & Ice are beyond the scope of this study.)

| Design Name | Num Cells | Cycle Time | GRoute Slack | FRoute Slack | % G-Cell OverCap | FRoute Violations | GR vs. FR % Error |
|---|---|---|---|---|---|---|---|
| Design01 | 884 | 7.52 | -0.535 | -0.545 | 0.16 | 0 | 0.13 |
| Design02 | 7665 | 17.00 | -4.257 | -4.264 | 0.29 | 0 | 0.04 |
| Design03 | 7858 | 7.52 | -1.630 | -1.78 | 2.24 | 3 | 1.99 |
| Design04 | 8320 | 3.75 | -0.104 | -0.043 | 0.03 | 0 | 1.63 |
| Design05 | 10200 | 3.75 | -0.130 | -0.08 | 0.00 | 1 | 1.33 |
| Design06 | 13207 | 15.00 | -0.086 | -0.044 | 0.00 | 0 | 0.28 |
| Design07 | 14459 | 3.30 | -0.522 | -0.532 | 0.82 | 0 | 0.30 |
| Design08 | 20124 | 3.75 | -0.430 | -0.397 | 0.06 | 0 | 0.88 |
| Design09 | 20268 | 15.00 | -3.352 | -3.354 | 6.84 | 32989 | 0.01 |
| Design10 | 53506 | 5.00 | -1.639 | -1.604 | 2.74 | 4669 | 0.70 |
| Design11 | 69151 | 100.00 | -5.500 | -5.42 | 0.04 | 14 | 0.08 |
| Design12 | 73608 | 15.00 | -0.760 | -0.58 | 0.25 | 7 | 1.20 |
| Design13 | 92562 | 12.30 | -1.421 | -1.358 | 0.82 | 2 | 0.51 |
| Design14 | 160267 | 18.50 | -0.72 | -0.76 | 0.05 | 60212 | 0.22 |
| Design15 | 181927 | 10.00 | -1.477 | -1.697 | 0.00 | 20 | 2.20 |

**Table 7: Comparison of FRoute slacks vs. GRoute slacks.**

# 5. CONCLUSIONS

In this paper, we have described a number of experiments that seek to isolate the positive and negative aspects of WLMs. We first analyzed the distribution of wire loads in a number of industry designs. The distributions are highly dispersed and skewed, making it hard for any model based on fanout to predict wire loads. On the other hand, if timing-driven placement is successful, the nets with the largest loads should be non-critical, giving some hope that WLMs can be useful (i.e., "faithful") even if they are not accurate. In any case, we believe that an argument based solely on the distribution of wire loads is insufficient for showing that WLMs are "harmful": the utility of WLMs needs to be tested in actual optimization flows.

Next we compared seven different flows using Cadence Design Systems' PKS optimization tool on real industry designs. Our results indicate that flows that include WLMs generally give better results than flows that eliminate WLMs. (Fortunately, PKS can generate its own structural WLMs, so users are not required to generate their own WLMs.) Thus, We believe that WLMs still fulfill an important role in optimization. We also observed that post-placement optimizations are critical for achieving good timing results–much more important than WLM optimizations before placement– particularly for larger designs. We found that applying the more complicated logic optimizations when applied post-placement consistently improves the final results, but by only a small amount. Finally, we investigated the efficacy of WLMs in predicting the timing quality after post-placement optimization. The timing estimates using WLM slack appear to become slightly less accurate as designs grow larger, but to remain within about 12 percent for the designs sizes we studied. Thus, a reasonable estimate of final path timing might be to simply pad the WLM delay estimates by a constant factor.

## Acknowledgments

We would like to thank the many members of Cadence Design Systems' Ambit development group who helped us to better understand BuildGates and PKS. We especially thank Harm Arts, Glenn Gullikson, Johnson Limqueco, and Krishna Belkhale for their insights into the internal workings and hidden options of BG/PKS.

# 6. REFERENCES

[1] K. D. Boese, A. B. Kahng, B. A. McCoy and G. Robins, "Toward Optimal Routing Trees", *Proc. ACM SIGDA Physical Design Workshop,* April 1993, pp. 44-51.

[2] K. D. Boese, A. B. Kahng, B. A. McCoy and G. Robins, "Fidelity and Near-Optimality of Elmore-Based Routing Constructions", *Proc. IEEE Intl. Conf. on Computer Design*, October 1993, pp. 81-84.

[3] S. Bodapati and F. N. Najm, "Pre-Layout Estimation of Individual Wire Lengths", *ACM Intl. Workshop on System-Level Interconnect Prediction*, April 2000, pp. 93-98.

[4] L. Hagen, J. H. Huang and A. B. Kahng, "Quantified Suboptimality of VLSI Layout Heuristics", *Proc. ACM/IEEE Design Automation Conf.*, 1995, pp. 216-221.

[5] A. B. Kahng and S. Muddu, "Improved Effective Capacitance Computations for Use in Logic and Layout Optimization", *Proc. IEEE Intl. Conf. on VLSI Design*, 1999, pp. 578-582.

[6] Synopsys, Inc., *Liberty User Guide*, Version 1999, 1999.

[7] A. Lu, H. Eisenmann, G. Stenz and F. M. Johannes, "Combining technology mapping with post-placement resynthesis for performance optimization", *Proc. Intl. Conf. on Computer Design*, October 1998, pp. 616-621.

[8] L. Scheffer and E. Nequist, "Why Interconnect Prediction Doesn't Work", *ACM Intl. Workshop on System-Level Interconnect Prediction*, April 2000, pp. 139-144.

[9] I. Sutherland, R. Sproull and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*, San Francisco, Morgan Kaufmann, 1999.

# A Appendix: A New Custom WLM

Throughout this paper, we have used a new custom WLM that uses post-routing wire capacitance and resistance, and is presumably more accurate than statistical WLMs from the library. We perform the following four steps to create the WLMs.

1. Write out net and module data.
2. Cluster modules.
3. Group net fanouts within module clusters.
4. Construct WLMs for each cluster using multi-variable linear regressions.

These steps are explained in the next four subsections.

## A.1 Writing out net and module data

We use the global routing parasitics obtained from the default PKS flow. For each module, we write out its name and its total cell area after optimization. For each net, we write out its fanout, wire capacitance, and wire resistance, along with the name of the smallest module containing all pins in the net.

## A.2 Clustering modules

We combine the modules into clusters so that each module cluster contains at least as many nets as some given target number of nets, e.g., two thousand. Each module cluster will have its own WLM calculated for it. To cluster the modules, we first sort them by cell area. We restrict the clusters to each contain a contiguous set of modules in the sorted area. We use a *dynamic programming* algorithm to cluster the modules based on the criterion: (1) The maximum difference between the number of nets in each cluster and the target cluster size. To break ties in criterion (1), we use a second criterion: (2) The average squared difference between the number of nets in a cluster and the target cluster size.

An example for target net count of 2000:

| Module Name | Number of Nets | Cell Area |
|---|---|---|
| M1 | 2,000 | 100,000 |
| M2 | 1,500 | 110,000 |
| M3 | 2,500 | 150,000 |
| M4 | 3,000 | 200,000 |

The clusters will be {M1,M2}, {M3}, {M4}. Note that M2 is grouped with M1 even though M1 contains the target number, because no cluster can contain less than the target number of nets. Note also that if criterion (2) was based on the average difference from the target, then the clustering {M1}, {M2,M3}, {M4} would have the same value as the chosen clustering, and so might be chosen instead. This explains why we use average **squared** difference, rather than just average difference.

There are some other possible design choices that can be made for this step especially on how to cluster modules (if at all) for each WLM. Some other options are the modules grouping based on cell area and the target number of nets in each module cluster. However, in our experiments, we just use the first method for clustering the modules.

## A.3 Clustering net fanouts within module clusters

We count the number of nets in the module cluster for each fanout number. We use a specified target number of nets for each fanout cluster, e.g., 50. We use the same dynamic programming algorithm to cluster the fanouts based on the number of nets for each fanout.

We can also just use the average values for each fanout with any representative net (or any number of nets over a given threshold) instead of this clustering. Also we can choose the target number of nets per fanout cluster. Similar to the module clustering, we only use the first method for clustering net fanouts within module clusters.

## A.4 Constructing WLMs with linear regressions

We use linear least-squares curve fitting to construct a piece-wise linear function for the WLM where the function is linear within each cluster. It is also desirable that the function be monotonic non-decreasing, so that nets with larger fanout are not estimated to have smaller capacitance or resistance.

Suppose for example that the fanout clusterings are {1}, {2}, {3-4}, and {5-10}. For {1} and {2}, the estimates are simply the average values for the respective net fanouts. For {3-4} and {5-10} we fit a linear function over the given range. In effect, for {3-4}, this means that the estimate for fanout {3} is its average value and that the estimate for fanout {4} is its average value. We can enforce monotonicity of the WLM as follows. If the WLM is non-monotone between consecutive fanouts, we force the WLM to have the same values for the consecutive fanouts, and re-estimate the piece-wise linear function. If the fitted line for any cluster has negative slope, then we force the slope to equal zero and re-run the least-squares regression. Other schemes could be used to fit a piece-wise linear function. For example, we could decide not to enforce monotonicity.