A Sensitivity Based Placer for Standard Cells

Bill Halpin
Design Technology, Intel
2200 Mission College SC12-606
Santa Clara, CA 95054
1 408 765 9867
william.halpin@intel.com

C.Y. Roger Chen
Syracuse University
Department of EE&CS
Syracuse, NY 13244
1 315 443 4179
crchen@syr.edu

Naresh Sehgal
Design Technology, Intel
2200 Mission College SC12-606
Santa Clara, CA 95054
1 408 765 4179
naresh.sehgal@intel.com

ABSTRACT

We present a new timing driven method for global placement. Our method is based on the observation that similar net length reductions in the different nets that make up a path may not impact the path delay in the same way. For each net in the design, we compute the *net sensitivity*, or the path delay reduction as a result of net length improvements. We use very accurate delay models that include the impact of waveform slope and driver loading effects. Our new timing driven algorithm uses the sensitivity information to focus on nets that have the greatest impact on improving the worst circuit paths. Our method significantly improves the worst path delay over existing published work on industry circuits.

1. INTRODUCTION

The timing driven placement problem has been the focus of much research. The objective of standard cell placement is to find non-overlapping locations for each of the standard cell modules that meet the timing constraints while minimizing design area.

Process technology advances have scaled gates more rapidly than interdo not allow the chip designer to use placement algorithms where wire length minimization is the only objective. Interconnect capacitance, or load, often exceeds the receiver pin capacitance and wire resistance can now be on par with the driver output resistance. This is important as logic synthesis does not have information about cell locations and uses a statistical wire load model to estimate the resistance and capacitance. These estimates may differ significantly from the parasitics derived from the placed design. This mismatch forces the designer to iterate between logic synthesis and placement to achieve a design that meets the timing goal. These iterations are very time consuming since they involve full routing, extraction, and timing stages. The time for each re-synthesis iteration can be an order of magnitude greater than the placement stage itself.

Recently this problem has grown as the capacitive loading of the net dramatically impacts the gate delay and slope. The delay of a gate can no longer be considered as an unchanging "intrinsic" value. When interconnect capacitance of a net increases, the output slope degrades. This poor slope increases the delay of the gates further down in the path.

Current state of the art placement algorithms [1,2] use an "intrinsic" gate delay; estimated net routing; the Elmore delay model and static path timing analysis to achieve improved timing results.

There has been extensive work on the timing driven placement problem. We classify work on the timing driven placement according to three factors: the circuit parasitics model, the path delay calculation, and how delay and parasitic information is used in placement. In [1], an iterative net weighting approach biases the placement and the Elmore model is used to calculate net delay. The half perimeter of the net pins is used to estimate the net topology. The net weight is uniformly increased for the 3% most critical nets. In [2] a powerful simulated annealing path based algorithm is presented which is able to handle large designs. A constant driver resistance is assumed and the half perimeter bounding box of the net pins is used to model interconnect. Net delay is calculated as the product of the driver resistance and net capacitance. The driver delay is constant and does not vary with capacitive load. Timing is incorporated into the simulated annealing by adding to the cost function the difference between the required time and the actual timing for all critical paths. Neither paper accounts for the effects of waveform slope nor sensitivity of the path delay to net topology changes.

2. PROPOSED MODELS FOR TIMING IMPROVEMENT

2.1 Proposed Delay Model for Placement

Referring to Figure 1, We define the following terms: A cell, or gate, g_i, implements some logic function and is the basic

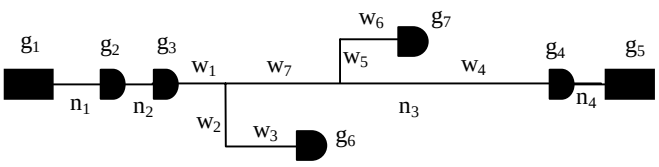

Figure 1. A path with its gates and nets

placement unit. Although our work handles gates with multiple outputs, for simplicity we denote the net driven by g_i, as n_i. We refer to the output of g_i as g_{i0} and the m inputs as $g_{i1},\ldots, g_{im}$. A k-pin net, n_i, connects the output of g_i with the k-1 input pins of n_i's receiver gates $FO_1(n_i)$, $FO_2(n_i),\ldots, FO_{k-1}(n_i)$. We define a path, p_j, as an ordered list of gate pins and interconnects, $\{g_1, n_1, g_2, n_2, \ldots, g_{nr}\}$ and the set of nets on path, p_j, as $N(p_j)$.

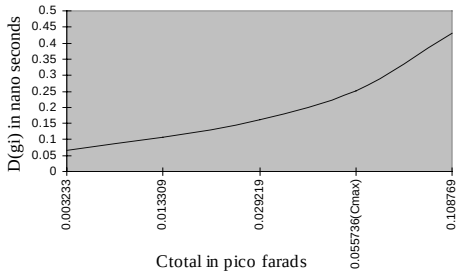

Figure 2. How the gate delay varies with capacitive load

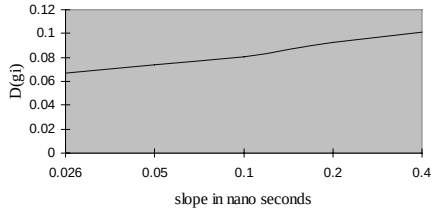

Figure 3. How the delay varies with slope

Previous placement work has assumed that the delay of the driving gate is constant, or intrinsic, for the transistor. As shown in Figures 2 and 3, the delay is not constant and varies significantly with the loading capacitance and input slope. In our work, we calculate delay based on an accurate driver and net model which includes the impact of waveform slope and the capacitive loading effects. The delay of a gate is a function of its input slope and its total load capacitance. The capacitance of input pin, g_{ij}, is $C(g_{ij})$ and the total receiver pin capacitance of n_i $C_{pin}(n_i)$ is $\sum_{j=1}^{k-1} C(FO_j(n_i))$. The load capacitance of driver g_i, $C_{total}(n_i)$, is composed of two parts: $C_{pin}(n_i)$ and the net capacitance, $C_{net}(n_i)$.We define the input slope as the time it takes the voltage at the input pin of g_i to go from its 20% value to its 80% value. We define gate delay as the time it takes from the 50% change in the input voltage to the 50% change in the output voltage. The gate delay is then

$D(g_i) = f_1(\text{input slope}, C_{total})$ and the output slope is

$\text{Slope}(g_i) = f_2(\text{input slope}, C_{total})$.

The functions f_1 and f_2 can be determined from Spice simulations. Current placement algorithms assume that $D(g_i)$ is fixed. This assumqption is invalid since placement changes affect $C_{net}(n_i)$ and therefore $f_1(\text{input slope}, C_{total})$. In Figure 2 we show how the gate delay varies with changes in C_{total} for a given slope. In Figure 3 we show how the gate delay varies with input slope. We define the delay of a path, p_j, as $D(p_j)$. We use the above gate delay model and AWE[6] to calculate the wire delay.

Referring to Figure 1, the net, n_2 is composed of the wires $\{w_1, w_2,\ldots,w_7\}$. The resistance of w_1, is $R(w_1)$ and the capacitance as $C(w_1)$. We determine the values of R(w) and C(w) values from the actual placement using a minimum spanning tree. We say that the wire parasitics are scaled by the factor, s, $0<s<1$, if their parasitic values are multiplied by s. We define $D'_{n_i,s}(p_j)$ as the path delay of p_j when the parasitics of net n_i is scaled by s. Since we are interested in the impact of a net in its path context, we define net delay, $D(n_i, p_j)$ as the difference between the path delay $D(p_j)$ with and without the parasitics of the net, n_i. The delay of the path without the parasitics of n_i is achieved by scaling the parasitics of n_i to 0, or $D'_{n_i,s}(p_j)$ where s=0. The delay of n_i, in path p_j, is then $D(n_i, p_j) = D(p_j) - D'_{n_i,0}(p_j)$.

By defining net delay in such a way, we include the increase in the gate delay that is caused by the capacitive loading of the interconnect. Our definition also includes the impact of waveform slope on gate delay. In the case of a heavily loaded gate, the rise-time of the output, as defined by CV/I, will be very slow, since the C is large relative to the current that can be provided by the gate. This slow slope will significantly increase the gate delay of the receivers.

A change in $C_{total}(n_i)$ not only affects the delay of g_i, but also the output slope of g_i. As shown in Figure 3, this slope will have a second order impact on the delay of n_i's receiver gates and their slopes. These second order effects continue until you reach a clocked element at the end of a path. These second order effects are accounted for in the computation of $D(n_i, p_j)$.

2.2 Computing Path based Net Sensitivity

The goal of the path based net sensitivity metric is to focus the net improvement on nets which have the biggest effect on timing. In previous work[1,2,4,6], all nets on a critical path were given the same weight. Focusing placement improvement within a path is paramount in timing driven placement because decreasing the length of one net usually means increasing the length of other nets. Referring to Figure 1, assume that g_1 is a very weak driver(high output resistance) and that g_2 is a strong driver(low output resistance). $D(g_1)$ will be very sensitive to changes in its wire load, $C_{net}(n_1)$, while $D(g_2)$ will not be as sensitive to the same load change in $C_{net}(n_2)$. In this case the path delay can be reduced if the length of n_1 is reduced even though n_2 will be increased. We exploit this net sensitivity to improve our placement result.

Net sensitivity quantifies the change that would occur in a path's timing due to a placement change affecting the net topology. Given two nets on critical paths, if we separately reduce each of them by the same amount, we will typically get different results due to differences in gate, loading and slope behavior.

We define the delay bound on a path, $B(p_j)$ as the required time of the path. $B(p_j)$ is defined by the desired operating frequency of the circuit.

We define, $S(p_j)$, the slack of p_j as the difference between $B(p_j)$ - $D(p_j)$. If $S(p_j)$ is negative, the path is said to have negative slack, meaning that it is not meeting the timing requirement. Let $P(n_i)$ be the set containing all the paths which traverse n_i. We define $P_w(n_i)$ as the path, in $P(n_i)$ with the worst, or lowest, slack.

We calculate sensitivity after we have globally placed and extracted design parasitics. We use sensitivity information to improve the circuit performance through iterative placement changes. Since we are interested in maximizing design performance we focus on a net's impact on its most critical path, $P_w(n_i)$. We then determine the sensitivity of that path response to changes in the net parasitics.

We compute the raw sensitivity, $RS(n_i,s)$ of n_i to a placement change as the change in delay of $P_w(n_i)$ in response to scaling all of the wires of n_i by a factor of s, where $0<s<1$. $RS(n_i,s) = D(p_k) - D'_{n_i,s}(p_k)$ where $p_k = P_w(n_i)$. We are only scaling the parasitics of the net and not actually moving the cells, so determining the sensitivity involves only recomputing the timing for the path and does not require adjusting the cell placements of net topology. We would expect that for any scaling factor $0<s<1$ that $RS(n_i,s)$ would be positive, i.e. that the slack would improve. To compute $RS(n_i,s)$, we must compute the path response and cannot simply scale $D(n_i)$ by s. This is due to the slope effects discussed in Section 2.1.

Since we are interested in improving the path delay , we form the net sensitivity by computing the delay changes in all of the other nets $N(p_j)$. The sensitivity of n_i, $Sen(n_i)$, is computed based on the raw sensitivities of all of the nets in $Pw(n_i)$. $Sen(n_i)$ is

$$\frac{RS(n_i, s)}{\sum_{n_j \in P_w(n_i)} RS(n_j, s)}$$

$Sen(n_i)$ varies with the different values of s. In our results below, we set s=0.7.

3. New Placement Techniques

3.1 Placement Flow

The goal of this placement work is to improve timing performance by exploiting the sensitivity of the nets to placement changes. We build on the popular Gordian placement[5] technique. We use iterative timing analysis and the sensitivity information from Section 2 to improve the timing performance.

The first stage in timing optimization is to obtain an initial global placement of the circuit. We chose Gordian for this task, since it is fast, well known and produces excellent wirelength results. We then use minimum spanning tree estimates of interconnect topologies and process parameters to estimate the resistance and capacitance of the net wires. We use the paths constraints produced by logic synthesis, which are expressed in the Standard Delay Format(SDF). We compute all of the path slacks using the SDF constraints and compute the path delay using AWE[3] for the wire delay and the gate delay models of Section 2.1. We define t_{ws} as the worst slack over all of the paths in the design. As we iterate through the placements, we select a placement if the t_{ws} of this placement is better than the best previously encountered t_{ws}. We then compute the new net sensitivities according to the Section 2.2 or 2.3. We use the net sensitivity and slack to compute the net weights for the next iteration according to section, 3.3. We continue iterating until $t_{ws} > 0$ or until no delay improvement is observed. We generally observe a convergence, but this is an area for further study.

3.2 Net weights

Gordian uses quadratic programming to minimize the total wirelength. For each net, n_i, with p pins, there are p(p-1)/2 edges with weight 1/p connecting each vertex in n_i. The weighted squared Euclidean distance between two movable gates, i and j is $w_{i,j}\left((x_i - x_j)^2 + (y_i - y_j)^2\right)$. This is expressed as a quadratic objective function where the objective function is to minimize the sum of the squared wirelength: $\sum w_{i,j} \times \left((x_i - x_j)^2 + (y_i - y_j)^2\right)$. In order to improve the timing, we bias the placement by changing the weight contributions, $w_{i,j}$, of each net.

3.3 Computing Net Weights

Since our goal is to improve the circuit timing, we want the nets that are on critical paths to get higher weights. For example, if path, p_3, is the most critical path then we would like to increase the net weights for the nets $N(p_3)$ to reduce the parasitics of these nets and improve $D(p_3)$. Previous work [1,2,4,6] has treated all of the nets in $N(p_3)$ equally ignoring that the nets have different sensitivity to placement changes. This is non-optimal, since decreasing some nets will increase the length of others. We use the sensitivity and slack information to generate new net weights. We would like to decrease the length of those nets which are most sensitive to length changes in order to have the maximum delay improvement.

In general we would like the net weight in timing iteration j, $W_j(n_i)$ to vary inversely with the net's worst slack, $P_w(n_i)$, and directly with the sensitivity, $S(n_i)$. In other words as the net slack is more positive the weight on that net should decrease. Our experiments showed that basing the weight only on these factors caused the weights to oscillate, resulting in poor overall placement. For this reason we base the new weight not only on $P_w(n_i)$, but also on previous weights. Our weighting function, therefor combines all 3 of these elements: $P_w(n_i)$, $S(n_i)$, and $W_{j-1}(n_i)$. These details follow.

3.3.1 Computing Slack based weights.

We compare the sensitivity based algorithm to a baseline algorithm which uses only the net slack for computing net weights. This model is similar to ones used in prior timing driven placement work. In this model all of the nets on the most critical path get the same weight increase, or criticality. The criticality of a net is based on the slack, $S(n_i)$ of the most critical path which traverses the net, $S(n_i) = S(P_w(n_i))$. We consider a net to be critical if its slack is negative.

3.3.2 Computing Sensitivity based weights.

Our sensitivity metric tells us relatively how much delay improvement we can get by decreasing the length of each net in the path. The net slack information tells us which nets are most critical. In the sensitivity based model, we combine this information to focus the new net weights on the most sensitive nets. If a net is very sensitive and on a critical path then it will get a high criticality.

In our experiments we found the sensitivity is prone to oscillations from iteration to iteration. A high criticality in iteration j would dramatically decrease the sensitivity in iteration j+1, which in turn would produce a high sensitivity in j+2.

The sensitivity based weight builds on the slack based model with two new factors, the sensitivity, $Sen(n_i)$, and additional damping to prevent the weight on critical nets from decreasing too rapidly. This damping is required because the sensitivity itself is highly impacted by the weight, i.e. for a net with high sensitivity, $Sen(n_i)$, a high weight, $W_j(n_i)$ usually results in a

low sensitivity for the net in iteration j+1, which results in oscillations.

4. Experimental Results

Table1: Benchmarks: wirelengths						
Circuit	#cells	#nets	#rows	Slack	Sensitivity	Ratio
Intel1	1945	2571	36	2890	3029	1.05
Intel2	2236	2964	43	3423	3600	1.05
Intel3	1794	2205	31	2039	2114	1.04
Intel4	2669	2901	36	3648	3540	0.97
Intel5	990	119	24	1024	1115	1.09

4.1 Industry Timing results

We used a set of industry circuits for comparison. Table 1 shows the number of cells, nets and rows for these circuits. On the industry test cases we compare our Sensitivity results to the Slack model given in section 3.3.1. This Slack model is similar to the model used in [1]. Table 1 gives the total wirelengths for each of the circuits, measured as the sum of the half perimeters of each net. Our sensitivity wirelengths are range from 3% better to 9% worse than the slack only model.

Table 2: Worst negative slack and improvement over density					
	Gordian	Slack		Sensitivity	
Circuit	nontiming	worst slack	improve	worst slack	improve
Intel1	-0.262	-0.262	0%	-0.162	38%
Intel2	-0.400	-0.211	47%	-0.174	57%
Intel3	-0.231	-0.182	21%	-0.102	56%
Intel4	-0.583	-0.265	55%	-0.316	46%
Intel5	-0.227	-0.212	7%	-0.143	37%
avg			**26%**		**47%**

Table 2 gives the timing results for Gordian with no timing-driven, the slack only timing driven version of Gordian discusses in section 3.3.1 and the sensitivity results. For the slack and sensitivity results we give the improvement over the non-timing driven Gordian.

We measure our results using the paths generated by logic synthesis. The results presented are for a 0.25 micron process.

We use two measurements to evaluate our timing results, the worst slack and the optimization potential as defined in [1]. The optimization potential is defined in the following manner. We compute the lower bound by setting all of the interconnect parasitics to 0 and computing all of the path slacks to find the worst slack. We then calculate the non-timing driven path slacks and find its worst slack. The difference between the worst non-timing driven path slack and the lower bound slack is defined as the optimization potential. The optimization exploitation is the percentage of the optimization potential that is realized by the timing driven placement algorithm. This optimization gives a relative metric of the improvement over the non-timing driven results.

Table 3 gives the optimization exploitations for each of the circuits for the slack and sensitivity algorithms. On average, our optimization exploitation is 60% compared to 33% for Slack model. On average our timing improvement is 47%.

Table3 Non timing and optimization results				
Circuit	lower bound[ns]	Gordian nontiming	Slack exploitation	Sensitivity exploitation
Intel1	-0.028	-0.262	0%	43%
Intel2	-0.142	-0.4	73%	88%
Intel3	-0.029	-0.231	24%	64%
Intel4	-0.036	-0.583	58%	49%
Intel5	-0.078	-0.227	10%	56%
avg			**33%**	**60%**

5. Conclusions

In this paper we have presented a new timing driven placement algorithm which significantly improves the optimization exploitation. By combining the interconnect parasitics with improved device modeling, we enable future designs at higher performance with fewer logic synthesis iterations, thereby improving productivity. Net sensitivity allows us to predict the impact of net topology changes on circuit timing and exploit this to improve global placement. In use at Intel, our algorithm has demonstrated excellent results by improving the optimization exploitation for Intel designs by 60% on average versus 33% for the previous approach.

6. ACKNOWLEDGMENTS

Our thanks to Frank Johannes of TUM for allowing us to work with the Gordian source code.

7. References

[1] H. Eisenmann and F. M. Johannes, “Generic Global Placement and Floorplanning,” ACM/IEEE DAC, 1998.

[2] William Swartz and Carl Sechen, “Timing Driven Placement for Large Standard Cell Circuits,” DAC, pp. 211-215, 1995.

[3] Lawrence T Pillage and Ronald A Rohrer, “Asymptotic Waveform Evaluation for Timing Analysis,” IEEE Transactions on Computer-Aided Design, pp. 352-366, 1990.

[4] A. Srinivasan, A K. Chaudhary, E. S. Kuh, “RITUAL: Performance Driven Placement Algorithm for Small Cell ICs,” ICCAD, pp. 48-51, Nov. 1991.

[5] Jurgen M. Kleinhans, Georg Sigl, Frank M. Johannes, and Kurt Antreich, “GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization,” IEEE Transactions on Computer Aided Design, Volume 10, No. 3 pp. 356-365, 1991.

[6] Bernhard M. Riess and Gisela G. Ellelt, “SPEED: Fast and Efficient Timing Driven Placement,” pp. 377-380, 1995.