TIMING DRIVEN PLACEMENT IN INTERACTION WITH NETLIST TRANSFORMATIONS

Guenter Stenz[1] *Bernhard M. Riess*[2] *Bernhard Rohfleisch*[2] *Frank M. Johannes*[1]

[1]Institute of Electronic Design Automation
Technical University of Munich
80333 Munich, Germany
Stenz@regent.e-technik.tu-muenchen.de

[2]Siemens AG
Semiconductor Group
81617 Munich, Germany
Bernhard.Riess@hl.siemens.de

ABSTRACT

In this paper, we present a new approach that performs timing driven placement for standard cell circuits in interaction with netlist transformations. As netlist transformations are integrated into the placement process, an accurate net delay model is available. This model provides the basis for effective netlist transformations. In contrast to previous approaches that apply netlist transformations during placement, we are not restricted to local transformations like fanout buffering or gate resizing. Instead, we exploit global dependencies between the signals in the circuit. Results for benchmark circuits show excellent placement quality. The maximum path delay is reduced up to 33 % compared to the initial timing driven placement of the original netlist and up to 18 % compared to the results obtained by consecutive optimization of the netlist and timing driven placement of the optimized netlist. This delay reduction is achieved with almost no increase in chip area.

1. INTRODUCTION

1.1. Motivation

Performance requirements are steadily increasing. Therefore, timing optimization has become one of the most challenging problems faced in the design of today's highly integrated circuits. Consequently, timing optimization techniques have been developed for all stages of the design process. However, interaction between the stages has to be improved.

In Figure 1a), the traditional design flow is shown. On the logic synthesis stage, timing optimization is guided by crude path delay models. They are mainly based on gate delays and rough estimations of the net delays as layout information does not exist in this early design stage. The net delays are often estimated by using the individual output resistances and input capacitances of the gates provided by the library. The fact that the delay of a net mainly depends on its length is ignored. Other approaches use statistical data derived from previously generated designs for net delay estimation. These statistical data often do not represent the net delays of the current design very well. As net delay models lack sufficient accuracy during logic synthesis, the subsequent placement and routing stages have to optimize the net delays in the most critical paths.

Since VLSI geometries are scaled down more and more, net delays become dominant compared to gate delays. Consequently, suboptimal decisions may be performed during logic synthesis as only rough estimations for the most dominant part of the path delays are available. Nevertheless, the subsequent placement stage assumes this suboptimal netlist to be static. This problem will become even more important for future design processes.

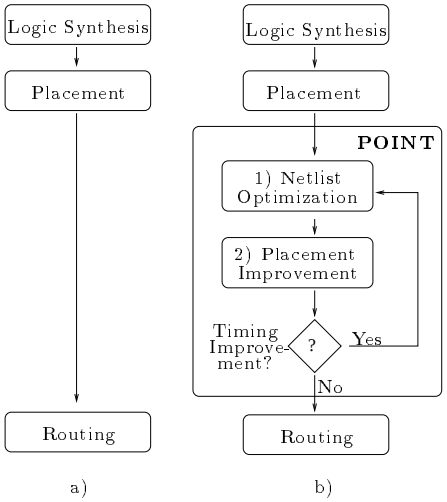

Figure 1. Comparison of Design Flows: a) Traditional, b) Deep-Submicron

1.2. Outline of our approach

To overcome this problem, we propose a new approach that exploits the optimization potential of netlist transformations during the placement process. As netlist transformations are performed *during* the placement process, they are based on more accurate estimations of the net delays.

We propose a design flow as shown in Figure 1b). After logic synthesis, a placement is generated. We use this placement as initial placement for our new approach POINT (**P**erformance **O**ptimization by **I**nteracting **N**etlist Transformations and **T**iming Driven Placement). POINT performs a sequence of netlist transformation and placement improvement phases which are closely coupled. To transform the logic netlist, we have adopted a method that effectively reduces the delay of mapped circuits [1]. Accurate estimations of the net delays are obtained from the current placement to guide the selection of the netlist transformations. After the netlist has been modified, the placement is

updated by a placement improvement phase. For this purpose, we developed a new timing driven iterative placement improvement algorithm.

1.3. Previous work

The problem of combining logic and layout synthesis procedures has been addressed in several papers in recent years. The existing techniques can be divided into logic synthesis approaches that use a *companion placement* [2, 3, 4] and placement procedures that apply local netlist transformations after initial placement and update the placement accordingly [5, 6, 7].

The approaches of the first group use the companion placement to guide the synthesis process by providing more accurate estimations on area, delay, or routability. This companion placement is dynamically updated during the logic synthesis process. Pedram and Bhat [2] present a technique that considers net lengths during both, area and delay oriented technology mapping. The key idea is to estimate the interconnection dependent contributions to circuit area and delay by referring to a dynamically updated global placement [2, 8] of the unmapped network. The same authors [3] extend this idea to logic restructuring and technology decomposition. Vaishnav and Pedram [4] propose an effective fanout optimization algorithm that generates fanout trees that are free of internal edge crossings resulting in reduced chip area after final routing. In this approach, the companion placement is used to get an order restriction on the fanouts of the gates. All these approaches suffer from the fact that predicting post-layout net lengths during logic synthesis is extremely difficult.

The approaches of the second group are placement algorithms that apply *local* netlist transformations after initial placement. Kannan et al. [5] start from an initial placement followed by timing optimization using fanout buffering and gate resizing transformations. Estimations of the net delays based on the initial placement are used for selecting the most useful transformations. A linear programming approach has been applied for resizing and relocating of critical gates by Chuang and Hajj [6]. Liu et al. [7] propose to resynthesize the logic in the most congested regions of the chip to reduce routing area. The approaches of the second group are able to estimate net lengths accurately since they operate on a placed circuit. However, they only apply local netlist transformations like fanout buffering and gate resizing which strongly restricts the optimization potential.

POINT combines the advantages of previous approaches. It operates on a placed circuit which allows more accurate net delay estimation. Additionally, our new approach is not restricted to local netlist transformations like fanout buffering or gate resizing. Instead, global dependencies between the signals in a circuit are exploited to improve the circuit's performance. Nevertheless, our experiments show that the netlist transformations perturb the placement only moderately such that the perturbed placement can be used as a good initial solution for the subsequent placement improvement phase. Results for benchmark circuits show excellent placement quality. The maximum path delay is reduced up to 33 % compared to the initial timing driven placement of the original netlist and up to 18 % compared to the results obtained by consecutive optimization of the netlist and timing driven placement of the optimized netlist. This delay reduction is achieved with almost no increase in chip area.

The remainder of this paper is organized as follows. The next section presents our delay model. We describe our new approach of timing driven placement in interaction with netlist transformations in Section 3. In Section 4, experimental results are presented and discussed.

2. DELAY MODEL

POINT is independent from the choice of a specific delay model. We use a model that was recently proposed for analytical timing driven placement [9]. In this section, we will give a short review of this delay model.

We model the total delay of a path as the sum of the gate delays and the sum of the net delays on this path. The gate delays represent the intrinsic delay of a gate and are considered to be independent from the fanout of the gate. Instead, fanout dependencies are modeled by the net delays. For net delay estimation we use the Elmore delay [10].

As the final routing of a net is not available during placement, a net model must be used to estimate the net delay. We use the following star model: Assuming given coordinates of the pins connected by a net, the star point is computed as the center of gravity of all pins of the net. The star point is connected to the source pin and all sink pins of the net. The star model for a four–pin net is shown on the left of Figure 2. Let k denote the number of pins of a

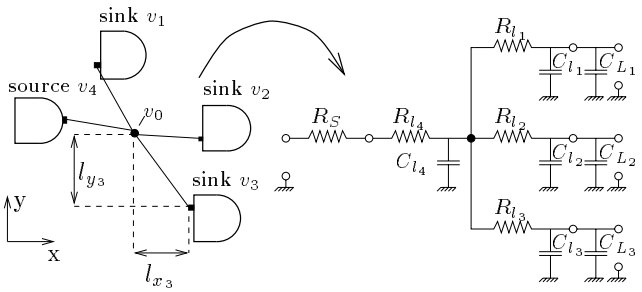

Figure 2. Star model and derived electrical model

net, v_0 denote the star point, $v_1, \ldots v_{k-1}$ the sink pins, and v_k the source pin of a net. Let l_{x_i} and l_{y_i} denote the distances from the star point to pin v_i in x– and y–direction, respectively. From this net model, a corresponding electrical model can easily be derived, which is used for net delay estimation. On the right of Figure 2, the electrical model of the four–pin net is given. It consists of an output resistance R_s, a series resistance R_{l_i} and a parallel capacitance C_{l_i} for each edge of the star and a load capacitance C_{L_i} parallel to each sink pin. The lumped resistances and capacitances R_{l_i} and C_{l_i} are defined as

$$R_{l_i} = r_x \cdot l_{x_i} + r_y \cdot l_{y_i} \quad \text{and} \quad C_{l_i} = c_x \cdot l_{x_i} + c_y \cdot l_{y_i}, \quad (1)$$

where r_x, r_y, c_x and c_y denote the resistances per unit length and capacitances per unit length in x– and y–direction, respectively.

Now, we are able to compute the Elmore net delay $nd(v_k \rightarrow v_q)$ from the source pin v_k to the sink pin v_q:

$$\begin{aligned} nd(v_k \rightarrow v_q) &= (R_S + R_{l_k})(C_{l_k} + \sum_{i=1}^{k-1}(C_{l_i} + C_{L_i})) \\ &\quad + R_{l_q}(C_{l_q} + C_{L_q}) \end{aligned} \quad (2)$$

To evaluate the circuit's timing behavior, we assign arrival times to all pins of the circuit. An arrival time of 0 is assigned to all primary inputs of the circuit and the arrival times of all other pins are computed by traversing the netlist starting from the primary inputs towards the primary outputs in a breadth-first-search manner. The *maximum path delay* is given as the largest arrival time of all

primary outputs. Tracing back from the primary outputs with the largest arrival times to the primary inputs, the critical paths of the circuit are identified.

3. THE NEW APPROACH

In this section, we will first give an outline of POINT. Then we will discuss the netlist transformation and placement improvement phase in more detail.

3.1. Outline of POINT

Our approach POINT is illustrated in Figure 1b). The algorithm starts from an initial standard cell placement of a logic netlist. It consists of two closely coupled phases (phases 1 and 2). In phase 1, transformations of the logic netlist are performed to reduce the maximum path delay. To select a delay reducing transformation, the critical paths have to be determined. For that purpose, we use the current placement to obtain the net delays as described in Section 2. Of course, only netlist transformations that do not change the circuit's logic behavior are performed. Due to the netlist transformations, gates may be removed as well as new gates may be added. For each gate that is added to the netlist we immediately determine a tentative location allowing temporary gate overlaps. Therefore, all gates have a location at any time. The critical paths of every intermediate netlist are determined based on these coordinates. The netlist transformations aim at shortening these critical paths. It should be noted that in general the placement is not legal after netlist transformations have been performed.

After a specified number of netlist transformations has been performed, phase 2 of our approach is entered. Now, the perturbed placement is legalized and improved by applying a placement improvement phase. In this phase we first compute an overlap–free placement and then refine this placement iteratively by rearranging subsets of gates in certain regions of the chip area. The two phases are repeated until no further improvement can be obtained.

In the following, we will discuss the netlist transformation phase and the placement improvement phase in more detail.

3.2. Timing Optimization by Structural Netlist Transformations

To transform the netlist in phase 1 of our approach, we use the concept of signal substitutions, which has been shown to be effective for timing optimization of mapped logic netlists [1]. In the following, we will give a brief review and an illustrating example of this method. In this context, the same label is used for a gate and for the signal at its output pin.

An *output substitution* OS2(a,b) is a netlist transformation that substitutes the stem signal a by signal b.

An *input substitution* IS2(a_1,b) is a netlist transformation that substitutes the branch signal a_1 by signal b.

In the example shown in Figure 3a), the gate a is connected to the two fanout gates d and e. For the output substitution OS2(a,b), both fanout gates of a are disconnected from gate a and connected to gate b instead (Figure 3b)). After the substitution, the output of gate a is floating and therefore gate a and all gates in its fanout-free region can be removed from the netlist. If an input substitution $IS2(a_1, b)$ is carried out, only one of a's fanout gates (gate d) is re-connected, as shown in Figure 3c). For input substitutions, no gates can be pruned. Nevertheless, they are valuable for timing optimization. Furthermore, substitutions with inverted signals are also considered. In this case, an inverter has to be added to the netlist.

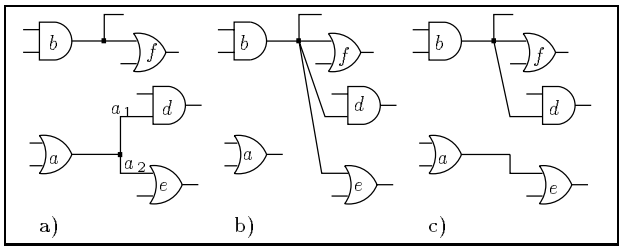

Figure 3. a) Original netlist, b) Output substitution OS2(a,b), c) Input substitution IS2(a_1,b).

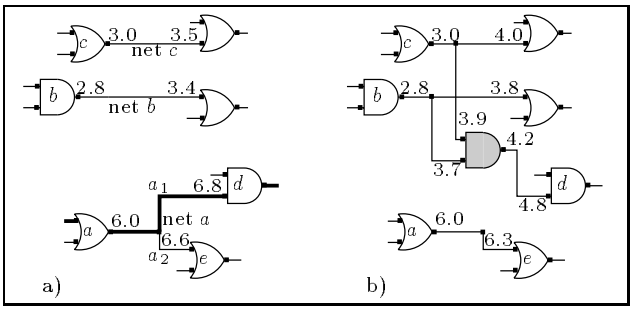

Figure 4. a) Original netlist with critical path and b) Input substitution IS3(a_1,b,c).

The concept of substitutions can be extended to substitutions involving more than only two signals. The following 3-signal substitutions provide more powerful methods to transform the netlist.

An *output substitution* OS3(a,b,c) is a netlist transformation that substitutes the stem signal a by the output signal of a new gate driven by b and c.

An *input substitution* IS3(a_1,b,c) is a netlist transformation that substitutes the branch signal a_1 by the output signal of a new gate driven by b and c.

An input substitution IS3(a_1,b,c) with an AND-gate is illustrated in Figure 4. For the newly inserted gate, all types of 2-input gates can be considered.

Intuitively, the transformations performed on the logic netlist must not change the circuit's logic behavior. Transformations that preserve the logic behavior are called *permissible*. Permissible substitutions can be computed using well developed ATPG-methods. For details of the computation of permissible substitutions we refer to [1].

We use an example to illustrate how substitutions can be used to shorten the critical paths. In the following, all nets driven by a gate on a critical path are called *critical nets*. Assume that the bold-printed path shown in Figure 4a) is critical. The numbers next to the sink and source pins of nets a, b, and c denote the arrival times at the respective pins. The branch signal a_1 is part of a critical net and therefore we want to substitute it. Assume that the part of the netlist shown in Figure 4 is embedded in the whole netlist in such a way that the substitution IS3(a_1,b,c) with an AND-gate is permissible. To perform the substitution, we add an AND-gate driven by b and c. The output of the AND-gate is used to substitute the critical branch signal a_1 as shown in Figure 4b). We initially place this AND–gate in the center of gravity of all pins it is connected to. The critical input of gate d has an arrival time of 6.8 before the substitution. The substitution reduces the arrival time at this input to 4.8. In the example of Figure 4, a branch signal contained in a critical path has been substituted. Note that

a substitution of the branch signal a_2, which itself is not contained in a critical path, would also decrease the arrival time at the input of gate d as the fanout of a decreases. Therefore, we also consider substitutions of branch signals not contained in a critical path but connected to a critical net. After a transformation we perform timing analysis to identify the new critical paths, which are then targeted by the next substitution. To reduce the maximum path delay we compute all substitutions for all signals connected to critical nets and select the one with highest gain.

This concept of substitutions is particularly suitable for incorporating it into timing driven placement. Assuming that net delays are known, they can be easily considered during the netlist transformations as shown in the example above. Furthermore, the substitutions are incremental netlist transformations. This is especially important if the transformations are performed during the placement process. A consistent integrated optimization is only possible if the transformations of the logic netlist do not lead to a totally different placement. Please note that IS2-transformations which contribute more than 50% to the total number of performed transformations, do not perturb the placement at all. This supports the convergence of the integrated optimization process.

3.3. Timing Driven Iterative Placement Improvement

The subsequent placement phase that legalizes this perturbed placement must fulfil two requirements. First, the placement procedure must start from the perturbed placement including the tentative locations for the newly added gates. If the new netlist would be placed starting from scratch, convergence between the netlist transformation and the placement procedure could not be expected at all. Second, the modifications of the perturbed placement should not be too small. If the perturbed placement is transformed into a legal placement with as few changes to the placement as possible, the subsequent netlist transformation phase will find no more netlist transformations. To meet the first requirement, ECO placement techniques are usually used. However, ECO placement techniques do not meet the second requirement. In this section, we will discuss a placement improvement procedure that meets both requirements.

It has been shown theoretically as well as experimentally that the linear assignment method combined with appropriate net models can be applied successfully to determine high-quality placements in terms of area and wire length. A typical method of this kind was implemented in the iterative placement program DOMINO [11]. Due to its good results in non-timing-driven placement, we adopt DOMINO's basic strategy for timing driven placement. In the following, we will first outline the new placement improvement approach and then discuss its differences and improvements compared to the DOMINO procedure [11].

3.3.1. Outline of the Placement Improvement Procedure

The algorithm as outlined in Figure 5 consists of the following three steps: First, an overlapfree placement is computed from the given perturbed placement. Next, the procedure iteratively generates a sequence of intermediate placements. In each iteration, an improved legal placement is generated from the current placement. Finally, we perform an intra-row permutation of gates. During this optimization, gates are rearranged within their row only. Except of the first step, new placements are only accepted if the maximum path delay has been decreased.

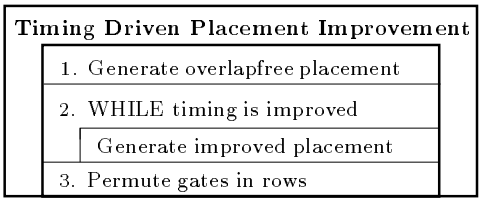

Figure 5. Overview of the timing driven iterative placement improvement procedure

In all steps of our algorithm, the chip area is covered by an array of regions. To each region we assign the subset of gates currently placed inside that region. We obtain subproblems, which consist of rearranging the gates currently placed inside a region. Solving subproblem by subproblem from the left to the right border of the chip yields an improved placement. Figure 6 illustrates a subproblem. For the regions on the left of the border line we have already computed an improved placement. The rearrangement process takes the gates in the current region shown in grey and assigns them onto locations on the right of the border line and within the lateral borders of the region. The three steps of the procedure shown in Figure 5 mainly differ in the choice of the regions. In the first and second step, a region contains gates from more than one row as shown in Figure 6. Therefore, gates can move from one row to another. As regions overlap, gates can also move from one region to the neighboring region. In the third step, a region only contains gates from one row.

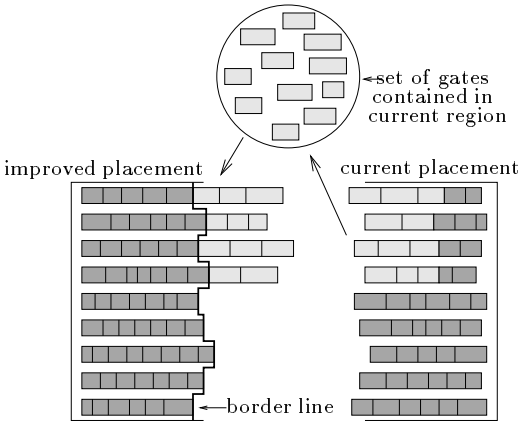

Figure 6. Generation of an improved placement by rearranging the gates in a region of the chip

Rearranging the gates contained in one region, i.e. solving one subproblem, can be formulated as a transportation problem. Gates are simultaneously transported to locations in an overlap–free manner by minimizing the transportation cost. This transportation problem can be efficiently solved by transforming it into a minimal-cost maximum flow problem [11]. The cost of assigning a gate a to location λ is denoted by $c_{a\lambda}$ where

$$c_{a\lambda} = c_{a\lambda}^{wl} + c_{a\lambda}^{delay} \tag{3}$$

The cost in terms of wire length $c_{a\lambda}^{wl}$ is computed according to a net model similar to the half perimeter bounding box of all pins of a net. For details of this net model and

the formulation of the transportation problem we refer to [11]. To take timing into account, we add the cost in terms of path delay $c_{a\lambda}^{delay}$ as it will be discussed in the following section.

3.3.2. Transportation Costs for Gates in Terms of Path Delay

Since during a placement phase the sum of the gate delays in a critical path is fixed, the entire path delay can only be reduced by shortening the delays of the nets contained in the critical path. As can be seen from equation 2, not only the gates contained in critical paths but all gates which are connected to a critical net contribute to the delay of the net. Consequently, in the following we consider all gates connected to a critical net as critical.

To minimize the length of a critical net we compute target coordinates $P_a^t = (x_a^t, y_a^t)$ for each gate a incident to the net. We compute the cost in terms of path delay $c_{a\lambda}^{delay}$ proportional to the distance $d_{a\lambda}^t$ of location λ to the target position of gate a, where s is the gradient:

$$d_{a\lambda}^t = |x_a^t - x_\lambda| + |y_a^t - y_\lambda| \quad (4)$$

$$c_{a\lambda}^{delay} = \begin{cases} d_{a\lambda}^t \cdot s & \text{if } a \text{ is critical} \\ 0 & \text{if } a \text{ is not critical} \end{cases} \quad (5)$$

Thus, we obtain a function with a cost of 0 at the target position and increasing cost with increasing distance to the target position. The gradient s is chosen such that the costs in terms of path delay are much higher than the costs in terms of wire lengths. Using these additional costs for the critical gates, these gates are pulled to their target positions with high priority. The non-critical gates are assigned to the remaining locations by minimizing the wire length.

Figure 7 illustrates how we compute the target coordinates P_a^t for a critical gate a. It shows a region as well

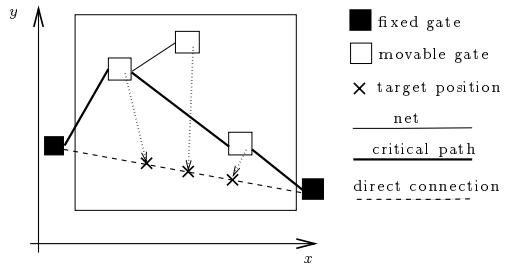

Figure 7. Target coordinates

as a part of a critical path entering the region through the left border and leaving the region through the right border. Since only gates inside the region can be moved, all gates outside the region are fixed. The fixed gates contained in the critical path are drawn in black. The movable critical gates are shown in white.

To reduce the net delays of the critical paths, we try to place the critical gates on the direct connection between the fixed gates outside the region. Intuitively, placing the critical gates onto these direct connections will minimize the length of the critical nets and therefore provide reduced path delays. The target coordinates of the movable gates contained in critical paths are equidistantly placed on these direct connections. This allows to consider the ordering of the gates in the path. The target coordinates of the gates are marked by crosses in Figure 7. Target coordinates of gates that are not contained in a critical path, but connected to a critical net are placed in the middle of the source and the critical sink of the net. This procedure allows to connect all gates of this net with minimal net length. Therefore it results in low net delay. Dotted lines in Figure 7 indicate the assignment of the critical gates to their computed target coordinates.

Using these target coordinates we are able to calculate the additional transportation cost in terms of path delay and obtain the total cost for assigning a gate to a location.

4. EXPERIMENTAL RESULTS

To evaluate POINT, we used combinational ISCAS–85 and ISCAS–89 benchmark circuits. The circuits were optimized and mapped to the SCMOS2.2 library by SIS using the delay oriented synthesis script `script.delay`. For the nets we assumed a capacitance per unit length of $242\,pF/m$ and a resistance per unit length of $25.5\,K\Omega/m$ [9]. Output resistances vary between 1 and $3\,K\Omega$ and input capacitances range from 0.07 to $0.14\,pF$. Typical gate delays are between 0.27 and $1.1\,ns$.

To get a fair comparison, we performed the following experiments on a set of 15 benchmark circuits. First, we generated a placement for the original logic netlist with the timing driven placement tool SPEED [9] which is the timing driven version of the well known placement procedure GordianL. We use this placement as the initial solution for evaluating POINT. For comparison, we generated a placement according to the traditional design flow consisting of netlist optimization followed by timing driven placement of the static, optimized netlist. For this purpose, we applied the netlist transformation procedure proposed in Section 3.2 to the original netlist generated by SIS, but without guidance by the placement. In this run, all terms in the Elmore Delay formula that depend on net lengths were set to zero as they are not known. This optimized netlist was then placed by the placement tool SPEED. All our experiments were performed on a DEC 250 4/266 workstation.

	SP	CONS		POINT			
circuit	t_{max} [ns]	t_{max} [ns]	CPU [s]	t_{max} [ns]	Imp. SP	Imp. CONS	CPU [s]
alu2	23.6	18.4	68	15.9	32.6	13.6	380
C1355	18.1	17.4	23	16.6	8.3	4.6	73
C880	17.9	17.7	4	16.6	7.3	6.2	56
C1908	26.1	24.5	41	23.2	11.1	5.3	215
alu4	27.9	27.4	37	22.8	18.3	16.8	544
vda	18.1	18.3	212	16.2	10.5	11.5	898
misex3	19.8	19.8	90	17.7	10.6	10.6	430
dalu	22.3	23.0	106	20.4	8.5	11.3	237
C3540	34.5	31.9	193	29.7	13.9	6.9	669
frg2	17.6	15.1	92	14.6	17.0	3.3	520
i8	20.2	20.4	97	18.9	6.4	7.3	346
pair	21.6	19.5	255	17.0	21.3	12.8	843
C5315	33.7	34.6	203	28.2	16.3	18.5	1 913
i10	40.4	37.3	1 362	33.0	18.3	11.5	2 652
C6288	80.1	79.9	1 295	76.7	4.2	4.0	3 125
AVG					13.6	9.6	

Table 1. Results of POINT compared to SPEED and CONS

In Table 1, the placement results obtained by SPEED (denoted by SP), the consecutive design flow (denoted by CONS), and POINT are compared. Column 2 shows the maximum path delay after the placement of the original netlist by SPEED. Columns 3 and 4 of Table 1 show the maximum path delays and CPU–times of the consecutive design flow CONS as described above. Finally, columns 5 to 8 present the maximum path delay, the relative improvement compared to the placement result of SPEED

circuit	# gates	area POINT	percentage compared to SPEED	CONS
alu2	271	1847 × 753	97.2	97.9
C1355	367	2023 × 885	100.0	100.0
C880	429	2012 × 899	101.7	102.2
C1908	535	2534 × 1119	108.4	109.2
alu4	598	2683 × 1389	99.7	101.6
vda	604	2903 × 1524	106.5	101.8
misex3	805	3148 × 2015	95.9	105.7
dalu	924	3292 × 1842	98.9	103.6
C3540	948	3141 × 1911	98.4	100.5
frg2	1 041	3136 × 1708	97.3	103.3
i8	1 069	3453 × 2204	99.1	103.0
pair	1 507	4102 × 2583	102.9	102.9
C5315	1 890	4526 × 3281	104.2	106.5
i10	2 325	4752 × 4231	103.6	104.1
C6288	2 650	5628 × 3330	102.2	112.6
AVG			101.1	103.7

Table 2. Chip area obtained by POINT compared to SPEED and CONS

and CONS and the CPU–times of POINT. On the average, POINT outperforms SPEED by 13.6% and the consecutive design flow by 9.6% in terms of maximum path delay. Please note that in the technology we used for our experiments net delays only contribute 37 % to path delays on the average. In modern technologies, net delays may contribute more than 60 % to path delays. Therefore we expect that results in these new technologies are even better as the influence of the net delays is larger than in our technology. Furthermore, we expect that considering net delays in larger circuits is even more important since die size and therefore maximum net lengths are larger.

The CPU–times for POINT are larger than for CONS. This is caused by the intermediate placement improvement phases and by a larger number of performed modifications. After the placement update, new promising netlist modifications may occur. Thus, alternating placement and netlist transformation phases allow each other to escape from local minima which leads to a larger amount of found transformations.

Table 2 compares the chip areas obtained by SPEED, CONS, and POINT after routing with TimberWolf 1.3. Column 2 shows the number of gates of the original netlists and column 3 the chip area obtained by POINT. As shown in columns 4 and 5, the placement obtained by POINT requires only 1.1% larger chip area than SPEED and 3.7% larger chip area than CONS on the average.

The results also show that the main limitation of POINT in its current implementation is the circuit size that can be handled due to high CPU requirements. The determination of permissible transformations can be performed for circuits containing several thousands of gates. However, in layout synthesis circuits with more than 100 000 gates have to be placed. Nevertheless, this is no fundamental limitation of our approach. Performance of a circuit is determined by quite small parts of the circuit containing the critical paths. To transform the logic netlist, only parts of the circuit that contain critical paths, have to be optimized. First experiments show that the CPU requirements can be reduced drastically by using this fact.

5. CONCLUSIONS

We presented a new approach to timing driven placement in interaction with netlist transformations. Starting from an initial placement and a logic netlist, our new approach iteratively performs netlist transformation and placement improvement phases, which are closely coupled. As the netlist is transformed *during* the placement process, accurate net delay models can be applied. Our research leads to the following conclusions:

Allowing netlist transformations during placement offers additional degrees of freedom to optimize the circuit's performance compared to placement techniques that operate on a static netlist. Since net delays can be estimated more accurately during placement, performing netlist transformations during placement leads to better results than performing them before placement only.

POINT reduces the maximum path delay up to 33 % compared to the timing driven placement of the original netlist and up to 18 % compared to results obtained by consecutive optimization of the netlist and timing driven placement of the optimized netlist. This delay reduction is achieved with almost no increase in chip area.

REFERENCES

[1] B. Rohfleisch, B. Wurth, and K. Antreich, "Logic clause analysis for delay optimization," in *32nd ACM/IEEE Design Automation Conference (DAC)*, pp. 668–672, June 1995.

[2] M. Pedram and N. Bhat, "Layout driven technology mapping," in *28th Design Automation Conference (DAC)*, pp. 99–105, 1991.

[3] M. Pedram and N. Bhat, "Layout driven logic restructuring/decomposition," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 134–137, 1991.

[4] H. Vaishnav and M. Pedram, "Routability-driven fanout optimization," in *30th ACM/IEEE Design Automation Conference (DAC)*, pp. 230–235, 1993.

[5] L. N. Kannan, P. R. Suaris, and H. Fang, "A methodology and algorithms for post-placement delay optimization," in *31st ACM/IEEE Design Automation Conference (DAC)*, 1994.

[6] W. Chuang and I. N. Hajj, "Delay and area opimization for compact placement by gate resizing and relocation," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 145–148, 1994.

[7] S. Liu, K. Pan, M. Pedram, and A. M. Despain, "Alleviating routing congestion by combining logic resynthesis and linear placement," in *European Conference on Design Automation (EDAC)*, pp. 578–582, 1993.

[8] W.-L. Lin, M. Sarrafzadeh, and C. K. Wong, "The reproducing placement problem with applications," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 686–689, 1994.

[9] B. M. Riess and G. G. Ettelt, "Speed: Fast and efficient timing driven placement," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 377–380, 1995.

[10] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal delay in rc tree networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD*, pp. 202–211, 1983.

[11] K. Doll, F. M. Johannes, and K. J. Antreich, "Iterative placement improvement by network flow methods," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD*, vol. 13, pp. 1190–1200, Oct. 1994.