A New Faster Algorithm for Iterative Placement Improvement

Moazzem Hossain, Bala Thumma, Sunil Ashtaputre
Compass Design Automation
San Jose, CA 95131

Abstract

We present a new faster design-style independent iterative placement improvement algorithm. Randomized simulated annealing based algorithms produce good result. But on very large designs, the inherently long runtime makes it prohibitive to use randomized algorithm. On the other hand deterministic improvement methods do not produce as good a result as the simulated annealing based algorithms. Moreover, none of the existing placement improvement techniques addresses the non row-based design style. In this paper, we combine the advantages of both the random and deterministic approach to develop a new faster placement improvement algorithm. Experimental results show that our algorithm performs much better than existing placement improvement algorithm. On some benchmarks, our algorithm is as much as 8x faster than that on Domino with a significant reduction in total net length.

1. Introduction

A circuit consists of a set of modules (also known as cells) and a netlist specifying the interconnections between the modules. The VLSI layout problem is to realize the circuit in a minimum layout surface while meeting the circuit specifications. High performance and minimum layout area are the two most important objectives for a VLSI design layout problem. Traditionally, due to the inherent complexity of the problem, the VLSI layout problem of a circuit has been divided into two sequential phases: placement and routing. The objective of the placement problem is to map the modules of a circuit on a layout surface while optimizing certain objective functions. Once the placement is complete, the routing problem is to realize the interconnections between the modules according to the specifications. The region on the layout area which is not occupied by modules can be used for routing.

The placement phase is very important in the overall physical design of a circuit. This is due to the fact that a poorly placed layout cannot be improved by high quality routing. In other words, the overall quality of a layout, in terms of area and performance, is mainly determined by the placement phase. The placement problem has been studied extensively [2, 3, 5, 6, 7, 8, 14, 15, 16]. The placement algorithms can be classified into two major groups: constructive placement and iterative improvement methods. The input to a constructive placement algorithm is a set of modules and the netlist defining interconnections between the modules. The algorithm finds the location for each module. On the other hand, the input to an iterative placement improvement algorithm is an initial placement. The algorithm iteratively modifies the placement in search of a better placement in terms of the given objective functions. As the VLSI design is moving towards deep submicron, the complexity of the placement problem is also increasing. As a result, the constraints on the placement problem is also increasing. And it is becoming almost impossible to meet all the constraints in one placement step. However, the long placement step makes it prohibitive to do the placement again, if certain constraints are not met. A fast iterative improvement algorithm can be used to improve the placement quality to meet those constraints.

Iterative placement improvement algorithms can be classified into two basic categories: randomized and deterministic. There are two basic randomized iterative placement improvement methods - Simulated Annealing [10, 13, 16] and Simulated Evolution [3, 12]. Randomized iterative placement improvement algorithms never reject better solutions, but also accepts intermediate inferior quality placements. Therefore, these algorithms have the ability to escape the local optima to obtain a solution close to the global optimum [13, 15]. Although it is possible to get a (close to) optimum solution, the converging time is extremely long for very large designs. As the VLSI technology is advancing rapidly, the complexity and the size of the designs has made the randomized placement improvement methods
unacceptable due to their inherently long runtime.

Although deterministic placement improvement methods [4, 8, 9] can model the objectives of the placement very well and is much faster than the randomized methods, these type of algorithm can get trapped in a local optimum. Also as the design gets larger, the complexity of the constraints and objectives also gets more complex, making the process slower. As a result, most of the deterministic algorithms address the problem by decomposing the problem into many subproblems, thereby, compromising the quality of the solution.

Since both randomized and deterministic methods have advantages and disadvantages, one might ask if it is possible to combine the two methods to overcome the limitations of both. The idea is to devise an algorithm that will, to some extent, deterministically modify the existing placement to get an improved placement. This motivates our research on iterative placement improvement for extremely large designs.

In this paper, we present a new algorithm which uses a deterministic algorithm to compute a target window for a module to be moved in order to optimize the overall objective functions. The algorithm can be used with or without simulated annealing. When used with simulated annealing, instead of selecting modules randomly to move or interchange, target windows are used to select modules and to move. The algorithm has been implemented in C++. The algorithm performs equally well on both row-based and non row-based design styles. Experimental results show a significant speed up in runtime over the traditional simulated annealing based algorithms, as well as over the deterministic placement improvement methods, with a significant reduction in total net length and layout area.

The rest of the paper is organized as follows. Section 2 presents some related definitions used in the paper. An overview of the algorithm is presented in Section 3. In Section 4, we present the linear time algorithm to compute the target window of a module. Section 5 gives the experimental results on some actual industry benchmarks. Section 6 contains the concluding remarks.

2. Definitions

The input to our algorithm is a netlist, a cell library, and the initial placement locations for all the modules. Let \( M = \{\mu_1, \mu_2, \ldots, \mu_m\} \) be the set of modules in a design and \( N = \{\nu_1, \nu_2, \ldots, \nu_m\} \) be the set of nets that specify the interconnections between the modules in \( M \). Then the netlist can be described as a binary relation \( R = N \times M \). A pin of a net \( \nu \in N \) on a module \( \mu \in M \) is represented by \( (\nu, \mu) \in R \). The set of modules connected by a net \( \nu \) is defined by \( M_\nu = \{\mu \in M | (\nu, \mu) \in R\} \). Similarly, the set of nets with a pin on module \( \mu \) is defined by \( N_\mu = \{\nu \in N | (\nu, \mu) \in R\} \).

For a net \( \nu \in N \), the bounding-box, \( B_\nu \), of \( \nu \) is defined by the minimum rectangle enclosing all the pins of \( \nu \). The net length of \( \nu \), \( L_\nu \), is estimated by half-perimeter length of \( B_\nu \). Since in most of cases, the interconnections between the modules are realized by rectilinear wires, the net length of a net can be considered as the summation of length in \( x \) and \( y \)-direction. The lengths in \( x \)- and \( y \)-direction are also called \( x \)-length and \( y \)-length, respectively. For a net \( \nu \), if \( L_\nu(x) \) and \( L_\nu(y) \) are \( x \)-length and \( y \)-length, respectively, then \( L_\nu = L_\nu(x) + L_\nu(y) \). For a net \( \nu \) and a module \( \mu \), if \( (\nu, \mu) \in R \), then we define \( B^\mu_\nu \) be the bounding-box of \( \nu \) without the pin \( (\nu, \mu) \). The total net length is the sum of the lengths of all nets in \( N \).

Let \( f(x) \) be a piece-wise linear unimodal function. We define the domain of minimum, \([x_l, x_r]\), of \( f(x) \) as the subset of the domain of \( f(x) \) such that \( f(x) \) reaches its minimum if and only if \( x_l \leq x \leq x_r \).

In our algorithm, we use the notion of multiset. Just like a set, a multiset is a collection of elements; however, unlike set, a multiset can have multiple occurrences of the same element. Let \( a_e \) and \( b_e \) be the number of occurrences of element \( e \) in multisets \( A \) and \( B \), respectively. The operation union of two multisets \( A \) and \( B \) results in a multiset \( C \) in which the occurrence of \( e \) is \( a_e + b_e \). The operation intersection of two multisets \( A \) and \( B \) results in a multiset \( C \) in which the occurrence of \( e \) is \( \min\{a_e, b_e\} \). Let \( C = A \cap B \) and \( c_e \) be the number of occurrences of element \( e \) in \( C \). The operation \( A - B \) results in a multiset \( D \) in which the occurrence of \( e \) is \( a_e - c_e \).

Given a placement of all the modules in \( M \), we are to find an optimal location for a module \( \mu \). The optimal location for \( \mu \) is a location on the layout area such that placing \( \mu \) in that location results in a minimum total net length for all the nets with a pin on \( \mu \). A target window, \( W_\mu \), for a module \( \mu \) is a rectangular region \((x_l, y_l, x_r, y_r)\) such that placing \( \mu \) at \((x, y)\) results in a minimum \( L_\mu \), if and only if \( x_l \leq x \leq x_r \) and \( y_l \leq y \leq y_r \). For \( \mu \), if \( W_\mu \) can be computed then \( \mu \) can be placed inside \( W_\mu \) to minimize the total net length.

3. Overview

As mentioned earlier that an iterative placement improvement method iteratively modifies the existing placement to get a better quality placement. Each iteration of the placement improvement may consist of
different passes, such as moving modules to a different location, swapping two or more modules, flipping modules. Our algorithm uses the following passes in each iteration.

1. **m-way Interchange Pass**: Let \( W^\mu \) be the target window for module \( \mu \in \mathcal{M} \). Also assume that \( (x_i, y_i) \) be the placement location of modules \( \mu_i \). Each interchange involves \( m \) \((m \geq 2)\) modules from \( \mathcal{M} \). We define an injective function \( \sigma_1 : \{1, 2, 3, \ldots, m\} \rightarrow \{1, 2, 3, \ldots, n\} \), for

\[
1 \leq I \leq \left( \begin{array}{c} n \\ m \end{array} \right),
\]

to select an ordered list of \( m \) modules \( (\mu_{\sigma_1(1)}, \mu_{\sigma_1(2)}, \ldots, \mu_{\sigma_1(m)}) \) from \( \mathcal{M} \). The \( m \)-way interchange problem is to compute a \( \sigma_1 \), for

\[
1 \leq I \leq \left( \begin{array}{c} n \\ m \end{array} \right)
\]

such that \( \mu_{\sigma_1(i)} \) is located inside \( W_{\mu_{\sigma_1(i-1)}} \) for \( 2 \leq i \leq m \) and \( \mu_{\sigma_1(1)} \) is located inside \( W_{\mu_{\sigma_1(m)}} \). Once such an ordered list of cells is found, \( \mu_{\sigma_1(i)} \) is placed in \( (x_{\sigma_1(i+1)}, y_{\sigma_1(i+1)}) \), for \( 1 \leq i \leq m - 1 \) and \( \mu_{\sigma_1(m)} \) is placed in \( (x_{\sigma_1(1)}, y_{\sigma_1(1)}) \). A special case of \( m \)-way interchange is known as pair-wise interchange where \( m = 2 \). In pair-wise interchange, two modules are selected to interchange their placement locations such that each one is within the target window of the other. The main difference in our approach and in the true randomized interchange is that of target window. Target window gives the desired location in where placing a module will result in a reduced net length.

2. **Move Pass**: In a random improvement method, each module is randomly moved into a location. If the move reduces the total net length or the objective function, then move is accepted. Otherwise, the move is rejected. We use target window to move a module. Target window of a module gives a region in the layout area such that placing that modules in any location inside that window will result in a reduced total net length. As a result, move pass is more deterministic in our algorithm than in a true random improvement method.

3. **Flip Pass**: Each module is flipped in the same location to reduce the total net length. For a module, if flipping that module results in a reduced net length then the flip is accepted, otherwise the flip is rejected. The flip pass in this algorithm is the same as a flip pass in a true random improvement method.

Our iterative improvement algorithm comprises of the above passes. The algorithm iterative executes each pass until no more improvement is obtained. Target window is used to select modules to interchange. In move pass, target window is used to move modules to the best location with respect to the placement of all other modules. Therefore, the most important part of the algorithm is the computation of the target window for each module. After each interchange or move, the target windows of all the affected modules are recomputed.

### 4 Target Window Computation

For a given module \( \mu \in \mathcal{M} \), we are interested in computing its target window \( W^\mu = (x^\mu_l, y^\mu_l, x^\mu_r, y^\mu_r) \). The target window is computed using the nets with a pin on \( \mu \). The \( x \)-span and \( y \)-span of the window are \( (x^\mu_l, x^\mu_r) \) and \( (y^\mu_l, y^\mu_r) \), respectively. The \( x \)-span is computed using the \( x \)-length. Similarly, the \( y \)-span is computed using the \( y \)-length. It is easy to see that the two can be computed independently. We only present of the computation of \( x \)-span of the target window. The computation of \( y \)-span is similar.

Let \( k (k \geq 1) \) be the number of nets with a pin on \( \mu \). Then \( N^\mu = \{v^\mu_1, v^\mu_2, \ldots, v^\mu_k\} \). Also let \( B^\mu_r = (l_i, b_i, r_i, t_i) \) be the bounding box of \( v^\mu_i \) without the pin \( (v^\mu_i, \mu) \). Let the placement of all the modules in \( \mathcal{M} - \{\mu\} \) be fixed and we need to find a placement location for \( \mu \). Let \( L^\mu (v^\mu_i, \mu) \) be the net length function for net \( v^\mu_i \) in the \( x \)-direction with respect to the placement location of \( \mu \). It is easy to see that \( L^\mu (v^\mu_i, \mu) \) is a piece-wise linear unimodal function. Then the domain of minimum of \( L^\mu (v^\mu_i, \mu) \) is \([l_i, r_i]\). If \( L^\mu (\mu) \) is the \( x \)-length function for all the nets with a pin on \( \mu \), then \( L^\mu (\mu) = \sum_{i=1}^{k} L^\mu (v^\mu_i, \mu) \). Since the sum of piece-wise linear unimodal functions is also piece-wise linear unimodal function, \( L^\mu (\mu) \) is a piece-wise linear unimodal function. Therefore, the domain of minimum of \( L^\mu (\mu) \) will give the \( x \)-span of the target window of \( \mu \).

Based on the above discussion, a straight-forward algorithm to compute the \( x \)-span of the target window of a module \( \mu \in \mathcal{M} \) can be computed by first computing the functions \( L^\mu (v^\mu_i, \mu) \), for \( 1 \leq i \leq k \) and then by algebraically adding \( L^\mu (v^\mu_i, \mu) \) and computing the domain of minimum of the resultant function. However, the time complexity of such an algorithm is \( O(k^3) \), where \( k \) is the total number of nets with a pin on \( \mu \).

We make use of the piece-wise linear unimodality of \( L^\mu (v^\mu_i, \mu) \) to develop our linear time complexity algorithm. For \( L^\mu (v^\mu_i, \mu) \) with domain of minimum \([l_i, r_i]\) and for a point \((x, y)\), we say that the domain of minimum of \( L^\mu (v^\mu_i, \mu) \) is on the left side of \( x \) if \( r_i \leq x \), and the domain of minimum of \( L^\mu (v^\mu_i, \mu) \) is on the right side of \( x \) if \( x \leq l_i \). Otherwise, we say that \( x \) is in the domain of minimum of \( L^\mu (v^\mu_i, \mu) \). We now prove the following important lemma:
Lemma 1 Given a module $\mu \in \mathcal{M}$, let $N_\mu = \{v^\mu_1, v^\mu_2, \ldots, v^\mu_k\}$. If for $1 \leq i \leq k$, $L_x(v^\mu_i, \mu)$ if the function of $x$-length of $v^\mu_i$ with respect to the placement of $\mu$, with domain of minimum $[l_i, r_i]$, let $S = \bigcup_{i=1}^k \{l_i, r_i\}$ be a multiset. Also let $L_x(\mu) = \sum_{i=1}^k L_x(v^\mu_i, \mu)$ with domain of minimum $[l, r]$. Then $l \in S$ and $r \in S$.

Proof: We only show that if $l \not\in S$ then $[l, r]$ is not the domain of minimum of $L_x(\mu)$. For the case that $r \in S$ can be shown analogously. Let us assume that $l \not\in S$. There must exist a net $v_i \in N_\mu$ and a net $v_j \in N_\mu$ such that $l_i < l < l_j$. If no such net exist in $N_\mu$, then the domain of minimum of all $L_x(v^\mu_i, \mu)$ is either to the right side or to the left side of $[l, r]$. In that case, moving $l$ either to the right or to the left towards the domain of minimum of $L_x(v^\mu_i, \mu)$ will result in a overall net length reduction. That contradicts the fact that $l$ is the left boundary of the domain of minimum of $L_x(\mu)$.

We now show that there are exactly equal number of nets in $N_\mu$ with domain of minimum to the left and to the right of $l$. Let, $k_1$ and $k_2$ be the number of nets with domain of minimum to the left and to the right of $l$, respectively. Also let $k_3$ be the number of nets for which $l$ is within the domain of minimum. Also let $x_l \in S$ is the largest in $S$ such that $x_l < l$ and $x_r \in S$ is the smallest in $S$ such that $x_r > l$. If $k_1 > k_2$, then for any value $l'$, $x_l \leq l' < l$, $L_x(\mu)$ will decrease, thus the assumption that $l$ is the boundary of the domain of minimum of $L_x(\mu)$ is not true. Similarly, if $k_1 < k_2$ then for any value $l'$, $l < l' \leq x_r$, $L_x(\mu)$ will decrease, thus $l$ is not in the domain of minimum of $L_x(\mu)$. Therefore, $k_1$ must be equal to $k_2$. That is, there are exactly equal number of nets in $N_\mu$ with domain of minimum to the left and to the right of $l$.

Since, $k_1 = k_2$, the value of $L_x(\mu)$ will be same in $l$ and in $x_l$. Now for any value $l' < x_l$, the net length of $k_2$ will increase, the net length for $k_1 - 1$ will decrease and the net length for $k_3 + 1$ nets will remain the same. Therefore, $L_x(\mu)$ will increase. Thus $x_l$ is the boundary of the domain of minimum of $L_x(\mu)$. Therefore, $l$ must be equal to $x_l$; as a result $l \in S$.

Similarly, we can show that $r \in S$. ■

The algorithm, **min_domain**, presented below is based on the above lemma. The input to the algorithm is the module $\mu$ for which the target window should be computed. For $k$ nets ($k \geq 1$) connected to $\mu$, their domain of minimum with respect to the placement of $\mu$ is computed first. A multiset $S$ is constructed from the boundary domain of minimum of $L_x(v^\mu_i, \mu)$, for $1 \leq i \leq k$. The $k$th and $(k + 1)$th value in $S$ gives the domain of minimum of $L_x(\mu)$. We now formally present the algorithm.

**Algorithm min_domain ($\mu$)**

1. compute $N_\mu$. Let $N_\mu = \{v^\mu_1, v^\mu_2, \ldots, v^\mu_k\}$.
2. Let $S = \emptyset$.
3. for $i = 1$ to $k$ do
   3.1. compute $B^\mu_{v^\mu_i} = (l_i, b_i, r_i, t_i)$.
   3.2. $S = S \cup \{l_i, r_i\}$.
4. compute $k$th smallest element in $S$.
   Let $l = k$th smallest element.
5. compute $(k + 1)$th smallest element in $S$.
   Let $r = (k + 1)$th smallest element.
6. return $[l, r]$.

According to [1], the $k$th smallest element in $S$ can be computed in $O(k)$ time. Since the bounding box of nets are computed incrementally, we can assume that the bounding box is already computed and stored in the data structure. Thus the overall time complexity of the algorithm **min_domain** is dominated by the computation of the $k$th and $(k + 1)$th smallest element in $S$. Therefore the time complexity of the algorithm **min_domain** is $O(k)$ where $k$ is the number of nets connected to $\mu$.

Following theorem shows that the interval $[l, r]$ computed by the above algorithm is the domain of minimum of $L_x(\mu)$, for a given module $\mu$.

**Theorem 1** The interval $[l, r]$ computed by the algorithm **min_domain** is the domain of minimum of $L_x(\mu)$ for a given module $\mu$.

**Proof:** Directly follows from the proof of lemma 2. ■

Similarly, we can compute the domain of minimum of $L_y(\mu)$ using the algorithm **min_domain**. In case of $L_y(\mu)$, we just need to consider the $y$-length functions for all the nets connected to $\mu$. Since the domain of minimum of $L_x(\mu)$ and $L_y(\mu)$ gives the $x$-span and $y$-span of the target window of module $\mu$, we can compute the target window for $\mu$ in $O(k)$ time, where $k$ is the number of nets connected to $\mu$.

**Theorem 2** Given a module $\mu$, the target window of $\mu$ can be computed in $O(k)$ time, where $k$ is the total number of nets with a pin on $\mu$.

5. Experimental Results

We have implemented the algorithm on C++ and tested on many industry designs. As mentioned earlier that our algorithm can be used for both row-based and non row-based design styles. Since Domino is the only iterative placement improve algorithm and only works in row-based design styles, we compare our results with the results obtained by Domino [4] in row-based design styles. In this case, the initial placement was obtained
Table 1. Specifications of Benchmark Examples

<table>
<thead>
<tr>
<th>circuit</th>
<th>#cells</th>
<th>#nets</th>
<th>#pads</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>1204</td>
<td>1309</td>
<td>134</td>
</tr>
<tr>
<td>c2</td>
<td>6372</td>
<td>7268</td>
<td>158</td>
</tr>
<tr>
<td>c3</td>
<td>14115</td>
<td>14930</td>
<td>220</td>
</tr>
<tr>
<td>c4</td>
<td>14332</td>
<td>15339</td>
<td>149</td>
</tr>
<tr>
<td>c5</td>
<td>14640</td>
<td>14921</td>
<td>192</td>
</tr>
<tr>
<td>c6</td>
<td>34965</td>
<td>36643</td>
<td>349</td>
</tr>
<tr>
<td>c7</td>
<td>27838</td>
<td>28977</td>
<td>259</td>
</tr>
</tbody>
</table>

Table 2. Net length comparison with Domino in non simulated annealing mode

<table>
<thead>
<tr>
<th>circuit</th>
<th>initial net length ((\lambda))</th>
<th>Domino net length ((\lambda))</th>
<th>Our algorithm net length ((\lambda))</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>(1.87 \times 10^6)</td>
<td>(1.67 \times 10^6)</td>
<td>(1.65 \times 10^6)</td>
</tr>
<tr>
<td>c2</td>
<td>(6.98 \times 10^5)</td>
<td>(5.46 \times 10^6)</td>
<td>(5.43 \times 10^6)</td>
</tr>
<tr>
<td>c3</td>
<td>(2.78 \times 10^7)</td>
<td>(2.03 \times 10^7)</td>
<td>(2.14 \times 10^7)</td>
</tr>
<tr>
<td>c4</td>
<td>(2.09 \times 10^7)</td>
<td>(1.74 \times 10^7)</td>
<td>(1.71 \times 10^7)</td>
</tr>
<tr>
<td>c5</td>
<td>(1.45 \times 10^7)</td>
<td>(1.28 \times 10^7)</td>
<td>(1.30 \times 10^7)</td>
</tr>
<tr>
<td>c6</td>
<td>(4.94 \times 10^7)</td>
<td>(4.58 \times 10^7)</td>
<td>(3.87 \times 10^7)</td>
</tr>
<tr>
<td>c7</td>
<td>(6.27 \times 10^7)</td>
<td>(5.9 \times 10^7)</td>
<td>(5.88 \times 10^7)</td>
</tr>
</tbody>
</table>

Table 3. Run time comparison with Domino in non simulated annealing mode

<table>
<thead>
<tr>
<th>circuit</th>
<th>Domino (cpu) (sec)</th>
<th>Our algorithm (cpu) (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>100</td>
<td>14</td>
</tr>
<tr>
<td>c2</td>
<td>756</td>
<td>112</td>
</tr>
<tr>
<td>c3</td>
<td>1348</td>
<td>812</td>
</tr>
<tr>
<td>c4</td>
<td>718</td>
<td>372</td>
</tr>
<tr>
<td>c5</td>
<td>5227</td>
<td>563</td>
</tr>
<tr>
<td>c6</td>
<td>15836</td>
<td>8631</td>
</tr>
<tr>
<td>c7</td>
<td>12755</td>
<td>1943</td>
</tr>
</tbody>
</table>

by Gordian [11]. Table 1 gives the specifications of different circuits of row-based styles (standard cell design) on which the experiments were conducted.

We have tested our algorithm on both simulated annealing and non simulated annealing mode. In non simulated annealing mode, only the positive moves, interchanges and flips are accepted. Table 2 and gives the comparison between Domino and our algorithm in non simulated annealing mode. Also in Table 4, we present the experimental results on our algorithm in simulated annealing mode. The initial placement was the same as in Table reftable2.

As can be seen from the tables, in non simulated annealing mode, our algorithm is as much as 8x faster than that of Domino. The net length is also as much as 4% better than Domino. In simulated annealing mode, our algorithm is slower than Domino; however, the total net length is as much as 8% better than Domino. We have also tested our algorithm on non row-based gate-array designs. We get 3% to 10% improvement in net length over the initial placement net length.
6. Conclusion

We have presented a new improved iterative placement improvement algorithm that combines the advantages of both deterministic and random placement improvement techniques. The algorithm works equally well on both row-based and non row-based design styles. This is the first algorithm to address the placement improvement in non row-based design styles. Experimental results on actual industry benchmarks show a significant improvement in runtime as well as placement quality achieved by our algorithm. Our algorithm can easily be extended to handle timing constraints in the design. We are currently in the process of evaluating the placement quality on the timing driven layouts.

References


