An Integrated Algorithm for Combined Placement and Libraryless Technology Mapping [*]

Yanbin Jiang † Sachin S. Sapatnekar ‡
† Cadence Design Systems, Inc., San Jose, CA 95134
‡ Department of ECE, University of Minnesota, Minneapolis, MN 55455

Abstract

This paper presents a new solution for combining technology mapping with placement, coupling the two into one phase. The original aspects of our work are the use of libraryless mapping and a state space search mechanism that is used to find the best solution. Several heuristics are presented for speeding up the search. Comparisons with a more conventional approach show that these strategies provide improvements of about 20%, with reasonable CPU times, on benchmark circuits.

1 Introduction

Traditionally, interconnect wire delays have only been considered at physical layout level, and only coarsely at the technology mapping level. The traditional design strategy performs technology mapping and physical layout design as two separate steps. The move to deep submicron technologies, where interconnect delays dominate, undermines the value of a stand-alone technology mapping/placement methodology, and a closer interaction between logic synthesis and physical design is now considered to be important.

Techniques that are used for technology mapping must also be examined anew. Traditional library-based design becomes limiting as it constrains the circuit designer to the limited design space defined by the library. A more complete exploration of the available design space can be made possible by the use of a wider range of possible gates, including complex static gates, which necessitates the use of dynamically generated libraries. The maturation of module generation techniques in layout synthesis has made the use of dynamic libraries more feasible, and their use in an industrial setting has been published in [3].

Several methods for merging technology mapping and physical layout design have been published in the past. One approach performs technology mapping based on the estimated placement information [1, 2, 8], possibly even modeling the effects of incremental updates to the placement without actually reperforming the placement [8]; we refer to this kind of method as placement-based mapping. Another trend is to perform logic resynthesis based on the placement information [3, 5, 10, 12]; this kind of method is called remapping. Our work differs from these in that we more completely merge the technology mapping and placement procedures by performing logic optimization considering the placement information.

[*]This research was supported in part by the National Science Foundation under grant NSF-9796305, and by the Semiconductor Research Corporation under grant 98-DJ-609.

Figure 1 illustrates the evolution of the interaction between technology mapping and physical layout design and places our method into this context. The first column shows the traditional divide-and-conquer strategy that completely separates these two steps. The second column shows the two trends of placement-based mapping and remapping described in the previous paragraph, and the third column shows our work. Another example that lies in the same column is the work in [7, 9], which uses a combination of the DAGON algorithm [6] and Yannakakis' algorithm to perform linear placement and technology mapping for the library-bound standard cell environment. This method differs from ours in that we use libraryless mapping, and that we apply our placer to the entire circuit in each iteration; the work in [7] applies Yannakis' algorithm locally but does not consider the effect of the new placement on existing matches during its transformations.

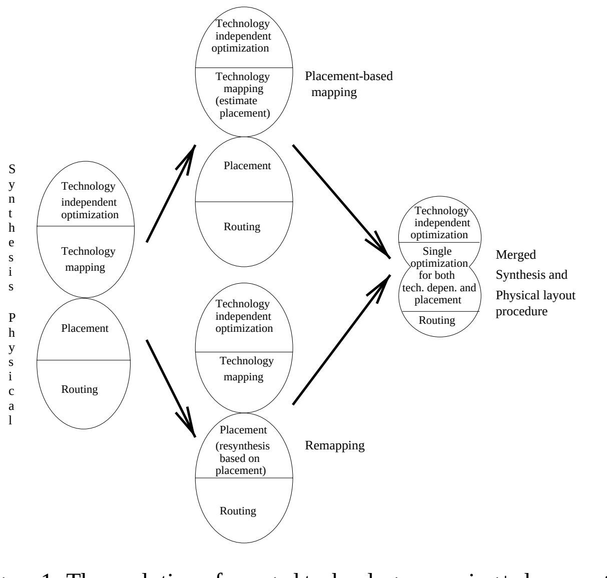

Synthesis
physical
Technology independent optimization
Technology mapping
Placement
Routing
Technology independent optimization
Technology mapping (estimate placement)
Placement-based mapping
Placement
Routing
Technology independent optimization
Technology mapping
Placement (resynthesis based on placement)
Remapping
Routing
Technology independent optimization
Single optimization for both tech. depen. and placement
Routing
Merged Synthesis and Physical layout procedure

Figure 1: The evolution of merged technology mapping+placement

Both remapping and placement based mapping are likely to suffer from inaccuracies in modeling the placement since the physical placement is likely to change by a great amount under resynthesis operations. Therefore, even though these techniques are faster than a method that performs placement again after each resynthesis step, they are likely to be inaccurate since their assumptions may be quite inconsistent with the final placement. The approach proposed in our work integrates placement with resynthesis operations, using incremental methods where possible to update the placement.

2 Interconnect Wire Timing Model

We represent interconnect wires using the π-model. A wire with the length l can be modeled as a resistance $R_{unit} * l$ connected by two grounded capacitance $\frac{C_{unit} * l}{2}$, where R_{unit} and C_{unit} are the interconnect wire resistance and capacitance, respectively, per unit length. Gates are also modeled using RC elements.

Global gate collapsing technique [4] is a technique used for technology mapping under a dynamic library that overcomes the limitations of a fixed library at the technology mapping stage by dynamically using complex gate structures generated on the fly. The global gate collapsing procedure structurally generates the complex gates, based on a simple topological technique, called the OTR (Odd-level Transistor Replacement) method, that permits subcircuits with to be collapsed into a single complex gate.

The work in [4] presented a dynamic programming algorithm that used the OTR procedure for gate collapsing. The algorithm here modifies the procedure to better incorporate interconnect wire information. In particular, instead of collapsing a set of gates together, it may be preferable to keep these gates separate to act as repeaters on the wire. This is illustrated in the example in Figure 2. Consider a medium-length or long interconnect wire whose R and C are both proportional to the length of the interconnect wire. For such a wire of length L, the delay is proportional to L^2. If gates B, C, D and E are not collapsed, then the gates divides the wire into four parts with length $l_1, \cdots, l_4$. If these are sufficiently small, then the delay grows linearly, rather than quadratically, with L. In such a case, one may prefer not to combine the gates and instead use them to perform the same function as repeaters on this wire.

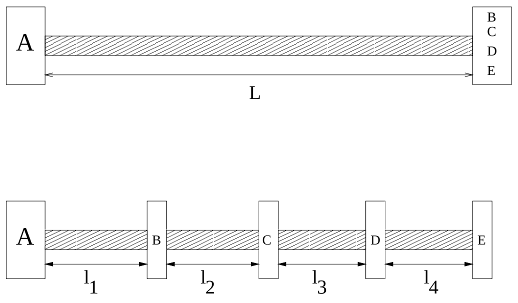


Figure 2: A situation where collapsing is not preferred

3 Merging Technology Mapping and Placement

Before commencing, we would like to point out that any placer and technology mapper can be used in the design methodology proposed in this work; in our experiments, we used our implementation of the placer GORDIANL [11] and the technology mapping algorithm (libraryless) in [4]. This technique can result in a solution that can be optimized for various objectives such as minimizing the circuit delay, the circuit area, or the power dissipation.

The initial circuit is decomposed into 2-nand gates and inverters and the placement algorithm is executed for this initial circuit. The decomposition has the advantage of allowing the placer to choose simple gates as repeaters; it is well known that complex gates do not make effective repeaters. Based on these placement results, we apply the gate collapsing algorithm to determine the recommended sets of gates to be combined into complex gates. This would result in a scenario of the type shown in Figure 3, where the dashed lines enclose the set of gates that are recommended to be collapsed.

However, the placement impact of gate collapsing is very significant, and if we choose to perform all of the recommended gate collapsings at the same time, the solution could result in an entirely different placement for which this set of gate collapsing operations would be suboptimal. In other words, the purpose of having placement guide the logic optimization process would be lost. Therefore, we perform gate collapsing and placement iteratively and intelligently. In each iteration, we collapse one set of gates to create a complex gate, update the placement, and based on this new physical layout, generate a new set of gates to be collapsed. Since each

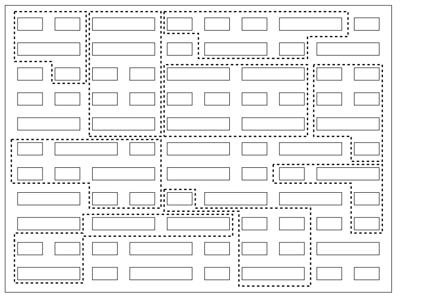
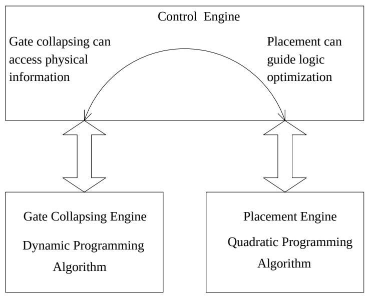


Figure 3: All possible collapsings based on placement result

such step represents a small change in the total layout, we may use a fast incremental engine to update the placement; in this case, we have adapted GORDIAN-L for incremental placement.

We design a control strategy to combine the placement engine and gate collapsing engine, as shown in Figure 4. The control engine alternately calls upon the gate collapsing engine and the placement engine, using information from each step to guide the other.



Control Engine
Gate collapsing can access physical information
Placement can guide logic optimization
Gate Collapsing Engine
Dynamic Programming Algorithm
Placement Engine
Quadratic Programming Algorithm

Figure 4: Overview of the strategy

We represent the space of possible gate collapsing choices by a search tree. We denote a state as a configuration of the circuit after collapsing one set of gates and updating the placement. The solution space can then be organized as a state space tree, as shown in Figure 5. In this figure, the root represents the initial input circuit that is decomposed into 2-nand gates and inverters. Each node in level $i+1$ is the result of collapsing one set of gates from its parent circuit at level i. Each node v, it has n_v children, corresponding to the total number of sets of possible gates to be collapsed based on the placement corresponding to node v. The leaves of the state space tree are the states that correspond to the final circuit in which no further gate collapsing is possible.

The optimal solution can be found using a *branch-and-bound* technique to explore the solution space.

For the purposes of the branch-and-bound exploration, we define the following terms. A *live node* is one that has been generated during the search process without any child node yet to be generated. A *dead node* is a generated node that is not to be expanded further. An *E-node* is a live node whose children are currently being generated.

4 Maximum Benefit (MB) Search

In both LIFO and FIFO branch-and-bound search, the selection rule for the next E-node does not allocate any specific preference to a node that has a very good chance of taking the search to the optimum solution. In our approach, we develop the maximum benefit (MB) search method to explore the state space to get the optimum solution.

The MB search assigns an intelligent benefit function $b(\cdot)$ for each live node, and selects the next E-node on the basis of this

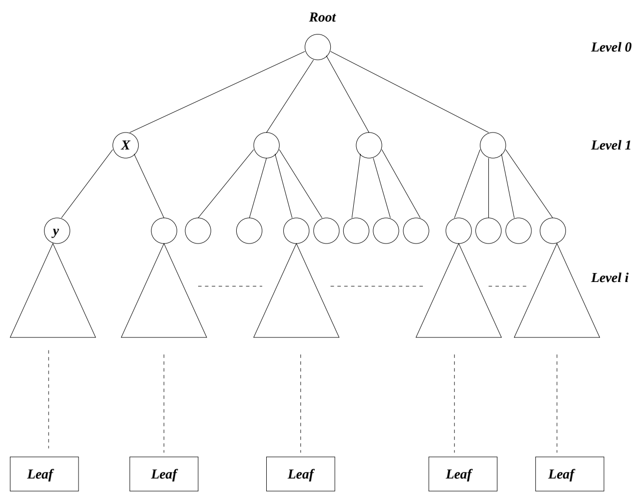


Figure 5: The state space tree

ranking function. The motivation for this is that a good choice of E-node could lead to a better bounding function for the remaining nodes.

If $f(x)$ is the objective function, then the ideal way to assign the benefit function for each node x would be on the basis of the how much improvement, $\Delta f(x)$, in the objective function has already been achieved, using the root node of the search tree as a basis, and how much potential reduction, $\Delta f_{pot}(x)$, can be achieved in the future from the descendants of node x; while the former is a known quantity, the latter can only be computed speculatively. For any node x, let $\hat{g}(x)$ be an estimate of the $\Delta f_{pot}(x)$ that can be achieved. Node x is assigned a benefit function $\hat{b}(x) = \Delta f(x)+\hat{g}(x)$. Roughly speaking, $\hat{b}(root)$ is equivalent to an estimate of the benefit function value for the best leaf node descended from node x.

Each node x has a benefit $\hat{b}(x)$ associated with it that is to be maximized. The function $\hat{b}(\cdot)$ is used to estimate an upper bound on the best solutions obtainable from any node x.

If $lower$ is a designated lower bound on the benefit solution, then all live nodes x with $\hat{b}(x) < lower$ may be killed (i.e., converted to dead nodes) as all solutions reachable from x have $b(x) < \hat{b}(x) < lower$. Variants of the search algorithm can be obtained from various choices of the parameter $lower$. If we set the bounding function $lower = bf(x) = \max[\hat{b}(y_i)]$, where y_i are the children of node x, then the algorithm is reduced to the greedy algorithm. On the other hand, if we set $lower = bf(x) = -\infty$, then the algorithm will explore the entire state space since no nodes can be killed by the bounding process. Note that greedy approach is not guaranteed to be optimal since if $\hat{b}(y_i)$ (with y_i defined as before) were a precise estimate of the residual benefit function, then the solution obtained by this approach at a leaf node is exact; however, as $\hat{b}(y_i)$ is an estimate, the approach is heuristic in nature.

The manner in which we select the benefit function, $\hat{b}(x) = h(x)+\hat{g}(x)$, and the bounding function, $bf(x)$, is crucial to the success of the procedure. The function $h(x)$ is easily computed as the value of $f(x)$ at node x based on the partial decisions made so far. The determination of $\hat{g}(x)$, the estimate of the potential benefit from the leaf nodes that are descendants of x, calls for a detailed explanation. By definition, we can see that $\hat{g}(x)$ is the best estimate of $f(x) - f(leaf)$. Given the gate collapsing structure at node x, we can estimate the result of collapsing all of the candidates *immediately* without further exploring the search tree, and that result gives us an estimate of $f(leaf)$, which is used to calculate $\hat{g}(x)$. Note that since this corresponds to an actual possible layout, it is guaranteed that the best possible value of the benefit function is no worse than this value. For the bounding function $bf(\cdot)$, we use $\varepsilon \times \max \hat{b}(y_i)$, where $\max \hat{b}(y_i)$ is the maximum benefit among all the children of node x, n_x is the total number of children of node x, and $0 \leq \varepsilon \leq 1$ is a user-settable parameter.

The pseudo code for the algorithm is as follows:

```
Algorithm State_Space_Search
MB Search(netlist *initial_circuit)
/* Initial_circuit is decomposed into 2-nand gates + inverters */
   queue_head = initial_circuit
   while(queue_head $\neq$ NULL) {
      $x$ = E-node = queue_head
      perform gate_collapsing
      for eachpossible set of gates recommended
               to be collapsed (i.e., each child $y$ of $x$ ) {
         perform collapsing
         update the placement incrementally
         find $\hat{b}(y)$
      }
      find max $\hat{b}(y)$
      for each $y$ {
         if $\hat{b}(y) \geq \varepsilon \times \max \hat{b}(y)$ add_queue
      }
      remove queue_head
   }
```

5 Variants of the Algorithm

5.1 The Direct Algorithm

The direct method is a very simple heuristic that takes the set of gates that were recommended to be collapsed after the initial placement, and collapses them all at once. This is then followed by the placement procedure to obtain the final physical layout. The direct algorithm corresponds to the simplest extension to the conventional method of performing technology mapping.

5.2 The Complete Search Algorithm

The complete search procedure explores the entire state space tree and does not use a lower bounding function to reduce the search space. This algorithm is obtained by applying Algorithm State_Space_Search with ε set to 0.

In practice, however, the run time of this approach is far too large. Our experiments showed that there was no appreciable difference between an ε value of 0.9 and 0.6, and therefore our presented results correspond to an ε value of 0.618, which is the golden section ratio ($= \frac{\sqrt{5}-1}{2}$).

5.3 The Greedy Algorithm

In the procedure of iteratively performing placement and gate collapsing, a greedy approach may be applied to find the solution. In each iteration, from the current state x, we choose the child node that can have the maximum benefit as a *live node*. The procedure is repeated until a leaf node of the search tree is reached, where no further gate collapsing can improve the benefit function. This algorithm is obtained from Algorithm State_Space_Search by setting $\varepsilon = 1$.

5.4 The 0/1 Algorithm

For each possible set of gates that are recommended for collapsing, there are two choices: either to collapse them or not. We denote the choice of collapsing them as '1', and that of not collapsing them as '0'. For N possible sets of gates to be collapsed, in theory, there are total 2^N configurations. Since the computation is expensive, we resort to a heuristic method to solve this problem.

Table 1: Experimental Results

Circuit	Complete Search Algorithm		Greedy Algorithm		0/1 Algorithm		Direct Algorithm	
	Min. Delay (ns)	CPU (s)	Min. Delay (ns)	CPU (s)	Min. Delay (ns)	CPU (s)	Min. Delay (ns)	CPU (s)
C432	42.56 (24.05%)	261	43.76 (21.91%)	31	45.26 (19.24%)	23	56.04	8
C499	61.27 (15.35%)	189	63.85 (11.79%)	31	66.13 (8.63%)	24	72.38	8
C880	49.83 (26.92%)	247	51.82 (24.01%)	32	52.79 (22.58%)	23	68.19	15
C1355	73.18 (20.15%)	325	75.27 (17.87%)	51	78.05 (14.84%)	32	91.65	17
C1908	135.25 (19.83%)	401	139.68 (17.21%)	77	144.69 (14.23%)	45	168.71	31
C2670	151.62 (21.44%)	1784	154.47 (19.97%)	158	171.73 (11.03%)	103	193.01	49
C3540	200.91 (20.08%)	1839	209.29 (17.84%)	221	231.81 (8.94%)	162	254.56	70
C5315	310.17 (14.97%)	2184	326.51 (10.49%)	601	331.96 (8.99%)	259	364.79	173
C6288	238.98 (20.64%)	3147	247.03 (17.97%)	613	264.37 (12.21%)	397	301.14	260
C7552	352.73 (21.49%)	3381	359.96 (19.88%)	872	404.47 (9.98%)	593	449.29	310

We begin with a configuration of $\underbrace{(0,0,0,\cdots,0)}_{N}$ where all gates are uncollapsed, and then perform N iterations. In ith iteration, we collapse the ith possible complex gate. In doing so, we get a new configuration, which is then compared to the best configuration so far. If the new configuration is better, then the best configuration is updated. Otherwise, we continue to the $(i+1)$th iteration.

6 Complexity of the Algorithms

The complexity of the placement engine is $O(mn\lg n)$, where n is the number of gates in the circuit and m is the number of iterations required by the conjugate gradient scheme. The complexity of global gate collapsing is as given in [4], but it is greatly dominated by the complexity of placement in practice and is not counted here in the complexity computations. In the complete search algorithm, the complexity is $O(2^n mn\lg n)$. For the greedy and 0/1 algorithms, the complexities are $O(mn^2\lg n)$. For the direct method, the complexity is simply $O(mn\lg n)$.

7 Experimental Results

All of the above algorithms were implemented in C on a SUN Sparc 1/170 workstation.

Tables 1 shows the results of each of the heuristics and the complete search algorithm. All of these results were generated with the intention of minimizing the delay of the final circuit. The technology parameters used here correspond to the 0.5μm technology parameters from MCNC.

The percentages shown inside the parentheses beside the delays are the delay improvements with respect to the delay of Direct Algorithm, which is the simplest extension to the conventional algorithm for placement with technology mapping. From the results, we note that all of the other three methods work well as compared with the direct algorithm. The most accurate approach we have listed is the complete search algorithm, and the greedy and 0/1 algorithm work well in comparison with this method. The reasons for that are because our gate collapsing sets are obtained on the basis of the physical layout information and our estimation function $\hat{g}(x)$ is a good estimate of the real benefit function.

8 Conclusion

A new family of approaches for combining technology mapping with placement is presented in this work. The method explicitly considers interconnect effects and uses them to guide the search process. A more conventional approach is implemented, and is referred to as the direct algorithm. This is the baseline method against with other techniques are compared. Two methods, the greedy and 0/1 algorithms, are proposed to overcome the limitations of the direct algorithm. Significant improvements are seen in the quality of the results, and the algorithm is also seen to perform well in comparison with the Complete Search Algorithm that makes the most complete exploration of the search space.

We expect similar gains to be made regardless of the mapper and placer by using our merging methods and more gains to be made by using our libraryless mapper instead of library-based mapper.

References

[1] S-C. Chang, K-T. Cheng, N-S. Woo and M. Marek-Sadowska, "Layout Driven Logic Synthesis for FPGAs," in *Proc. of ACM/IEEE Design Automation Conf.*, pp. 308–313, 1994.

[2] K. Chaudhary and M. Pedram, "A Near Optimal Algorithm for Technology Mapping Minimizing Area under Delay Constraints," in *Proc. of the ACM/IEEE Design Automation Conf.*, pp. 492–498, 1992.

[3] S. Gavrilov, A. Glebov, S. Pullela, S. C. Moore, A. Dharchoudhury, R. Panda, G. Vijayan and D. T. Blaauw, "Library-Less Synthesis for Static CMOS Combinational Logic Circuits," in *Proc. of the IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 658–662, 1997.

[4] Y. Jiang, S. S. Sapatnekar and C. Bamji, "A Fast Global Gate Collapsing Technique for High Performance Designs using Static CMOS and Pass Transistor Logic," in *Proc. of the IEEE Int. Conf. on Computer Design*, pp. 276–281, 1998.

[5] Y-M. Jiang, A. Krstic, K-T. Cheng and M. Marek-Sadowska, "Post-Layout Logic Restructuring for Performance Optimization," in *Proc. of the ACM/IEEE Design Automation Conf.*, pp. 662–665, 1997.

[6] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching," in *Proc. of the ACM/IEEE Design Automation Conf.*, pp. 341–347, 1987.

[7] J. Lou, A. H. Salek and M. Pedram, "An Exact Solution to Simultaneous Technology Mapping and Linear Placement Problem," in *Proc. of IEEE Int. Conf. on Computer-Aided Design*, pp. 671–675, 1997.

[8] M. Pedram and N. Bhat, "Layout Driven Technology Mapping," in *Proc. of the ACM/IEEE Design Automation Conf.*, pp. 99–105, 1991.

[9] A. H. Salek, J. Lou and M. Pedram, "A DSM Design Flow: Putting Floorplanning, Technology-Mapping and Gate-Placement Together," in *Proc. of ACM/IEEE Design Automation Conf.*, pp. 128–133, 1998.

[10] K. Sato, M. Kawarabayashi, H. Emura and N. Maeda, "Post-Layout Optimization for Deep Submicron design," in *Proc. of ACM/IEEE Design Automation Conf.*, pp. 740–745, 1996.

[11] G. Sigl, K. Doll and F. M. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?," in *Proc. of the ACM/IEEE Design Automation Conf.*, pp. 427-432, 1991.

[12] K. J. Singh, A. R. Wang, R. K. Brayton and A. Sangiovanni-Vincentelli, "Timing Optimization of Combinational Logic," in *Proc. of the IEEE Int. Conf. on Computer-Aided Design*, pp. 282–285, 1988.