Transistor Level Micro Placement and Routing for Two-Dimensional Digital VLSI Cell Synthesis

by

Michael A. Riepe, Karem A. Sakallah

CSE-TR-364-98

THE UNIVERSITY OF MICHIGAN

Computer Science and Engineering Division
Department of Electrical Engineering and Computer Science
Ann Arbor, Michigan 48109-2122
USA

Transistor Level Micro Placement and Routing for Two-Dimensional Digital VLSI Cell Synthesis

Michael A. Riepe, Karem A. Sakallah

Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan 48109-2122

June 23, 1998

Abstract

Previous research into the problem of cell library synthesis for digital VLSI design has concentrated mostly on relatively simple 1-dimensional cell topologies for static CMOS designs. Recent interest has emerged in less constrained 2-dimensional topologies to support more complex non-dual circuits such as latches and flip flops, as well as high performance circuit families such as CVSL, PTL, and domino CMOS. We discuss a CAD methodology which supports a generalized placement and routing approach to the realization of mask geometry for such complex circuits. We explore the options available within this methodology, show how the transistor level placement and routing problems at the transistor level differ from those at the block level, and present some results for a prototype tool, TEMPO, which adopts this methodology.

1 Introduction

Library cells make up the lowest level of the digital VLSI design hierarchy. Clearly, the quality of the cells will have a direct impact on the quality of the final design. The cells must be designed to be compact and fast, with minimized power and parasitics, and with careful attention to requirements on the physical appearance of the cells as viewed by the higher-level placement and routing tools. Automated synthesis techniques have found limited application at the cell level because existing tools are unable to match the quality of human designed cells. For this reason cells are often designed by hand, requiring a significant investment in manpower.

An additional difficulty lies in the fact that the lifetime of a typical cell library may be as short as one or two years. Compaction techniques may be used to migrate a cell library to a new process technology if little more than a linear shrink is required, but this is unlikely to extend the lifetime for more than one or two generations before the loss in performance necessitates a complete redesign of the library. These problems are only becoming worse as device geometries shrink into the deep submicron regime.

In order to account for deep submicron effects, ever closer interaction is required between front end synthesis tools and back end placement and routing tools, power and delay optimization tools, and parasitic extraction tools. In order to enable this interaction, cell libraries must become ever more flexible. Multiple versions of each cell with different drive strengths are required. It may even be necessary to support versions of cells in different logic families with different power/delay trade-offs.

In addition to the need for families of cells which are parameterized in terms of their electrical behavior, it has been demonstrated [1] that standard-cell placement and routing tools are able to obtain significantly higher routing quality if they have the ability to choose from multiple instances of cells with a wide variety of pin orderings. Over five benchmarks circuits, an average reduction in the number of routing tracks of 10.8% was demonstrated by the authors.

It seems clear that as the number of cells in a typical cell library grows from the hundreds into the thousands, a dramatic increase in designer productivity will be required, necessitating a move toward more automated cell synthesis techniques. The authors of [1], in fact, advocate a move completely away from static cell libraries as we know them, toward a system which permits the automated synthesis of cells on demand. This would permit logic synthesis tools to request specific logic decompositions, doing away with the traditional technology mapping step; standard-cell and datapath placement and routing tools to request cells with an exact pin ordering; interconnect optimization tools to request cells with specific input and output impedance values; and power optimization tools to request cells, perhaps from one of several different logic families, with specific power/delay trade-offs.

Such an on-demand cell synthesis system will require effort on many fronts:

1. Automated transistor schematic generation: constraint driven logic family selection, netlist creation, and transistor sizing.
2. Automated cell geometry synthesis.
3. Automated cell testing and characterization.
4. Development of enabling logic synthesis, placement and routing, and power/delay optimization technology.

In this paper we address the second item in the above list: the fully automatic synthesis of library cell mask geometry. The input specification consists of a sized transistor-level schematic, a process technology description (design rules, parasitics, etc.), and a description of the constraints imposed by the higher-level placement and routing environment. We refer to this last item as the *cell template*. A list of common cell template constraints are enumerated in [1].

The CMOS cell synthesis problem has a rich history going back approximately 15 years. Most of this research has centered on a formulation of the problem which was referred to as the “functional cell” in a seminal paper by Uehara and VanCleemput [2]. In this style, an example of which is given in Figure 1, the transistors take on a very regular structure. They are arranged in a linear fashion so as to minimize the number of breaks in the diffusion strips (so called “diffusion breaks”). We will refer to layouts in this style as 1-dimensional, or 1D, layouts.

The synthesis of 1D layouts can be formulated as a straightforward graph optimization problem: the location of a minimal dual Euler-trail covering for a pair of dual series-parallel multigraphs. Uehara and VanCleemput developed an approximate solution technique for this problem, while Maziasz and Hayes [3] presented the first provably optimal

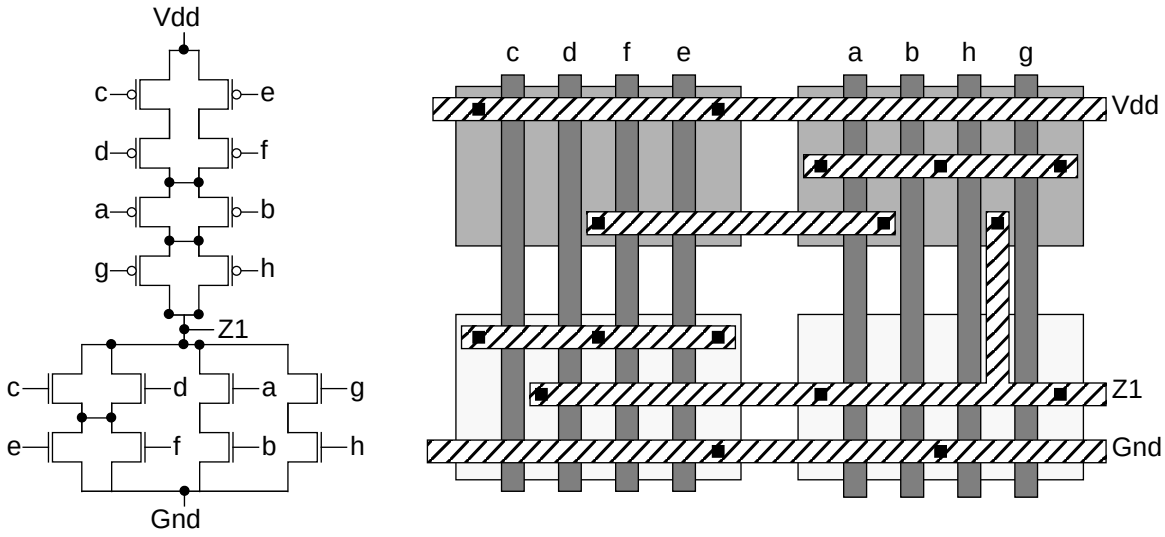


Figure 1: An example of a cell designed in the "functional cell" style of Uehara and VanCleemput [2].

algorithms.

A major drawback of the 1D layout style is that it directly applies only to fully complementary non-ratioed series-parallel CMOS circuits. Several significant systems have extended this style to cover circuits with limited degrees of irregularity. Among these are *Excellerator* [4], *LiB* [5], and *Picasso-II* [6]. Dynamic CMOS circuits, if the p-channel pull up transistors are ignored, can be optimized using a single-row 1D formulation. A good summary is presented by Basaran [7].

Despite the elegant formulation presented by the one-dimensional abstraction, it must break down at some point. When designing aggressive high-performance circuits the designer may call upon logic families such as domino-CMOS, pseudo-NMOS, Cascode Voltage Switch Logic (CVSL), and Pass Transistor Logic (PTL). These provide the designer with different size/power/delay trade-offs than are available with a static CMOS implementation. However, these non-complementary ratioed logic families often result in physical layouts which are distinctly 2-dimensional in appearance.

An example of such a circuit is shown in Figure 2. The example is a hand designed mux-flipflop standard cell implemented in a complementary GaAs process [8]. This example demonstrates a number of properties which deviate from the standard 1-dimensional style:

1. It is highly irregular. Some regularity is present in the rows, or "stacks" of merged transistors, but these stacks are of non-uniform size and are not arranged in two simple rows.
2. There are instances of complex geometry sharing, such as the "L" shaped structure in the upper-left corner.
3. The transistors are given a wide variety of channel widths.
4. The routing is non-trivial.
5. The port structure required by the back end placement and routing tools must be taken into account.

A variety of approaches have been taken to address the 2D cell synthesis problem. Tani et. al. [9] and Gupta and Hayes [10] discuss a style in which 2-dimensional layouts are formed from multiple 1-dimensional rows. The former presented a heuristic technique based on min-cut partitioning while the latter presented an exact formulation, called *CLIP*, based on integer linear programming. To distinguish this style from non row-based 2-dimensional styles, we refer to it as being 1-1/2 dimensional.

Xia et. al. [11] developed a method for BiCMOS cell generation. They group MOS transistors into locally optimal chains which behave as fixed blocks in the design. Bipolar transistors are treated individually and are given a fixed

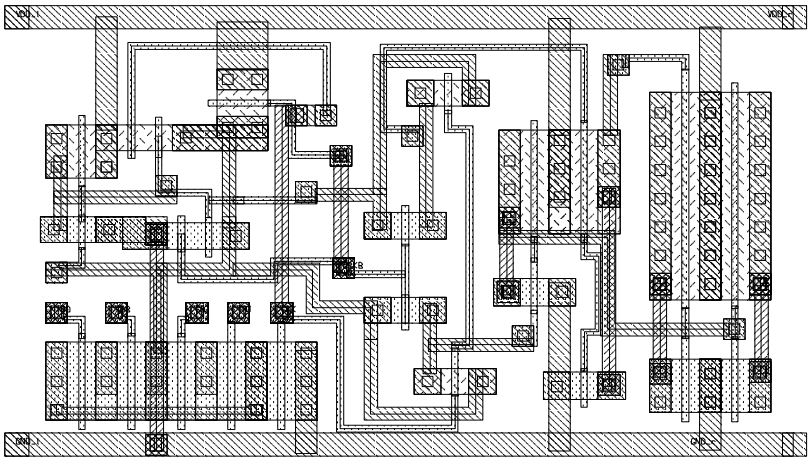


Figure 2: A manually designed cell showing complex two-dimensional layout structure

area with a flexible aspect ratio. A branch and bound algorithm is used to explore a slicing tree based floorplanning model of the circuit to find a placement of minimal area.

Fukui et. al. [12] developed a system for 2-dimensional digital transistor placement and routing. A simulated annealing algorithm is used to find good groupings of transistors into diffusion-shared stacks and a greedy exploration of a slicing structure is performed to find a 2-dimensional virtual grid floorplan. A symbolic router is used to perform detailed routing, and a final compaction step is used to permit transistors within stacks to slide into locally optimal positions.

In a more recent work by the same group, Saika and Fukui et. al. [13] present a second tool which operates by statically grouping the transistors into maximally sized series chains and then locating a high quality 1-dimensional solution in order to form more complex chains. Then a simulated annealing algorithm is used to modify this linear ordering by placing the diffusion connected groups onto a 2-dimensional virtual grid. Routing is done by hand.

It is also relevant to discuss work in analog circuit placement and routing in this context. One such system is *Koan/ Anagram* by Cohn et. al. [15]. This system uses simulated annealing to find a placement of analog components, simultaneously seeking to optimize device connectivity through arbitrary geometry sharing, while satisfying design rule constraints as well as analog constraints such as device symmetry and matching. It then uses a custom area router with aggressive rip-up and re-route capability to make the remaining required connections.

In this work we present the architecture for a 2D cell synthesis system which targets complex non complementary digital MOS circuits. We approach the problem as a general transistor level placement and routing problem, and show how existing placement and routing models and algorithms can be tailored specifically to digital transistor-level design.

A common theme among existing 2D cell synthesis systems [11,12,13] is that they attempt to capture localized regularity by grouping transistors into locally optimal transistor chains, or “stacks”. However, in all three systems this grouping is performed statically before placement. *Koan* [15], which is targeted at analog synthesis, performs no static clustering or stack formation, but instead allows these structures to form dynamically during placement. When applied to digital cells we found that Koan’s dynamic cluster formation was not able to discover the large regular clusters which are characteristic of digital designs (see Figure 13) but we take inspiration from its flexibility. An important theme in this work will be an examination of methods which permit the placement step to dynamically alter the transistor groupings during placement.

In Section 2 we introduce our proposed cell synthesis framework. Section 4 discusses the implementation of our experimental system, which is named *TEMPO*. Section 5 presents some experimental results and Section 6 summarizes our conclusions and discusses our plans for future work.

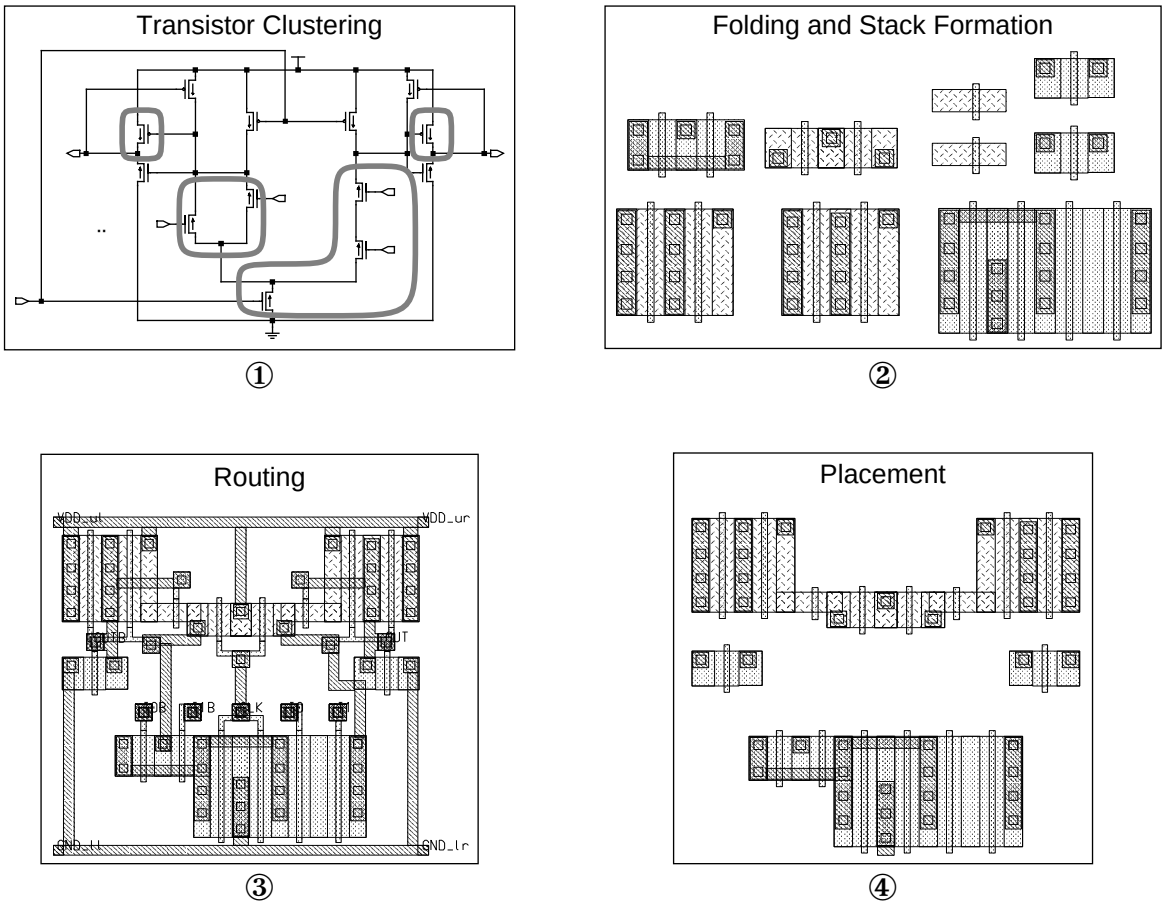


Figure 3: Proposed cell synthesis flow

2 Proposed Methodology

As our top level framework we adopt the flow of steps diagrammed in Figure 3, which was first suggested in [7]. We begin with a sized transistor netlist as well as the relevant design rules and cell template information. In the first step, clusters of transistors are formed, each representing a set to be composed into a single merged structure, or *transistor stack*. Clusters of size one represent degenerate stacks, or individual transistors. In the second step, large transistors may be *folded*, or split into two or more parallel-connected transistors of equivalent width. An ordering for the transistors in each stack is selected, and the geometry for each stack is generated. In the third step, a placement of the stacks is found that assigns them unique locations. Such a placement must satisfy the design rules and template constraints, should minimize some cost function, and leave enough empty space so that a feasible routing is guaranteed to exist. The fourth and final step generates the routing geometry to make the remaining electrical connections which were not made through direct connection by abutment. In general, it is not required that these four steps be performed sequentially. An algorithm may merge two or more steps, or iterate between them.

In the remainder of this section we discuss each step in more detail, outline the choices which we have adopted in our current implementation, and discuss how our methods extend those of previous authors.

2.1 Transistor Clustering

In the first step, transistor clustering, the task is to determine the optimal number of transistor stacks and which transistor(s) belongs in each stack. At this stage very little information is available, only the transistor connectivity

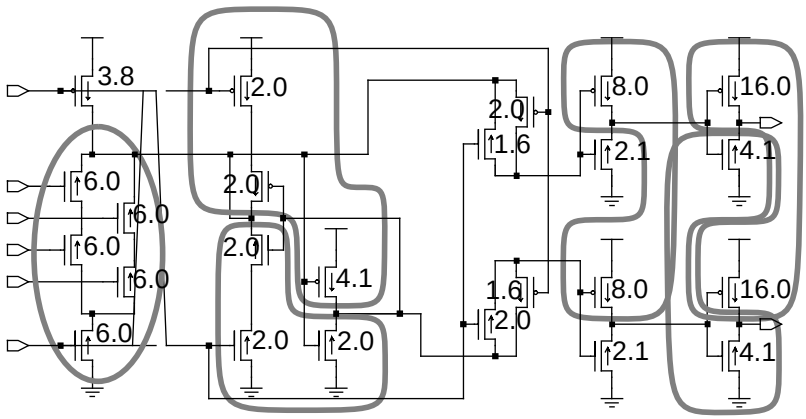


Figure 4: The clustering step. The schematic implements the CGaAs Mux-FlipFlop of Figure 2, and shows the partitioning used in the manual design. All transistor lengths are 0.5 microns.

and the sizes of each transistor, so the choices can only be heuristic in nature. Figure 4 shows the clustering used in the manually designed CGaAs Mux-FlipFlop circuit of Figure 2.

Authors in previous works have used a wide variety of heuristics at this stage. In [12] the authors use simulated annealing. In [13] only simple stacks made up of maximum length series chains were created and larger stacks are formed during the 1D optimization step. It is also fairly common, especially in 1-dimensional tools, to approach clustering by performing logic gate recognition, as in [6].

It is our opinion that, because of the heuristic nature of the decisions made at this stage, it makes little sense to devote a large amount of effort to determining, a-priori, any sort of "optimal" static clustering. Instead we have implemented a hybrid approach which allows the placement step to dynamically modify the clustering solution. During the clustering step we simply partition the netlist into maximally sized sets of diffusion connected same-polarity transistors. These sets are called *super-clusters*. After an ordering is assigned to the transistors in these super-clusters, the diffusion breaks are used to further partition the stacks into a number of *sub-clusters*. The corresponding sub-stacks become the atomic units available during the placement step, and orderings of the transistors in the super-stacks can be used to induce different clusterings in the individual sub-stacks. We will elaborate on this further in Section 2.3.3.

2.2 Transistor Folding and Stack Generation

In the second step, the clusters which were formed in step 1 are realized into physical geometry. To begin, large transistors may be "folded" in order to give individual transistors a more square aspect ratio and better allow stacked transistors to match the width of other transistors in the same stack. A folded transistor is formed by composing two or more parallel connected transistors to form a total channel width equivalent to the original.

After appropriate foldings are selected for each transistor, an ordering is selected for the transistors within each stack. For individual stacks it is known that the minimum width solutions correspond to solutions with a minimum number of covering Euler trails. It is also known that there are exponentially many Euler trails for any given stacking [7]. For example, in Figure 5 we show two different stackings for the same transistor netlist [3]. Both stacks have one diffusion break, but stack (2) requires one fewer horizontal routing track than stack (1). Stack (2) is also slightly narrower because the shared node between transistors c and b can be made without a contact. We should point out that, in this case, stack (2) was actually obtained by modifying the schematic—the top-level series chain was reordered. The two circuits will have the same logical behavior, but may not be electrically equivalent. Stackings which change the schematic should only be explored if permitted by the designer.

In order to establish the ordering of the transistors within the transistor stacks we have adopted the algorithms in [7] which were developed for 1D linear transistor array width minimization. As shown in Figure 6, a diffusion graph is constructed to represent the transistor connectivity. The graph vertices represent the electrical nodes in the circuit and the edges represent a transistor whose channel connects two of these nodes. It is well known that an Euler trail

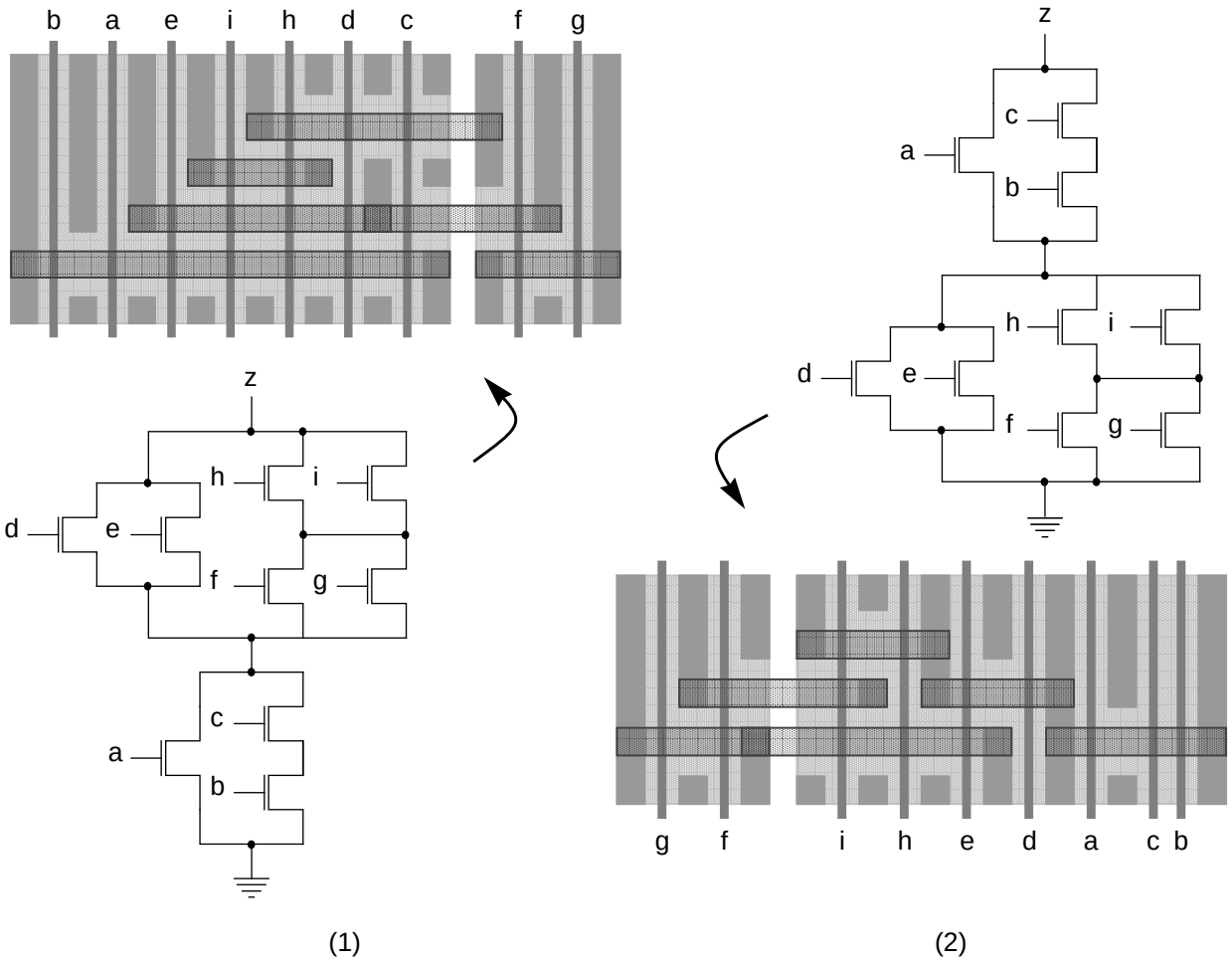


Figure 5: Two different transistor stackings generated for the same schematic.

can be embedded in a DAG if and only if the graph has either zero or two odd-degree vertices. In order to make the graph Eulerian, an artificial vertex (the "super" vertex) is added and edges are drawn connecting it to all odd degree vertices (there must be an even number of these). An algorithm from [23] is used to locate an Eulerian trail on this graph. The trail must begin and end at the super-vertex, though any internal vertex may be used if the input graph was already Eulerian. The order in which the graph edges are visited corresponds to the ordering of the transistors in the stack, with the super-edges corresponding to diffusion breaks.

As a final task within step 2, the physical geometry for each of the transistor stacks is realized. We implement this through a set of *generators* that create the design rule correct geometry, in the target technology, for stacks and for individual transistors. In Figure 7 we show the output of our stack generator for a fairly complex stack. Our generator currently supports stacks with internal diffusion breaks, as well as adjacent uncontacted transistors. It aligns oddly sized transistors along the bottom edge, and places ports to all external nets along this row. As shown in Figure 5, intra-stack routing is performed using a greedy left-edge algorithm, beginning from the second track and working upward. Only if no external ports are blocked will the first track be used for routing. We also attempt to place external ports in the top track of each transistor, if possible, to enable an escape path in both directions.

2.3 Stack Placement

It is the task of the stack placement step to assemble the transistor stacks into a configuration that is both design

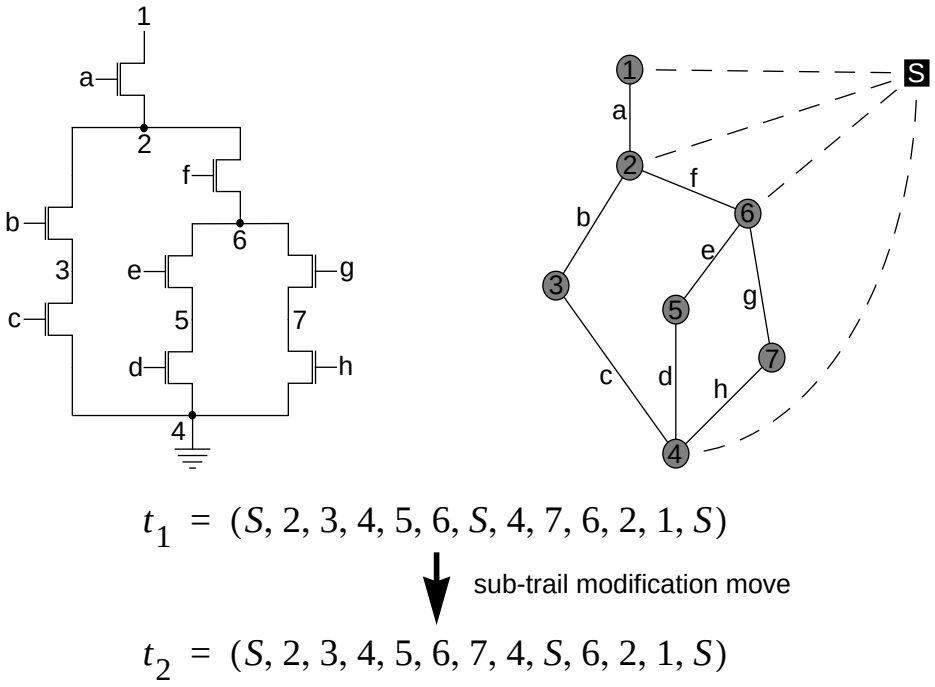


Figure 6: Euler trail formation and the sub-trail modification move [7]

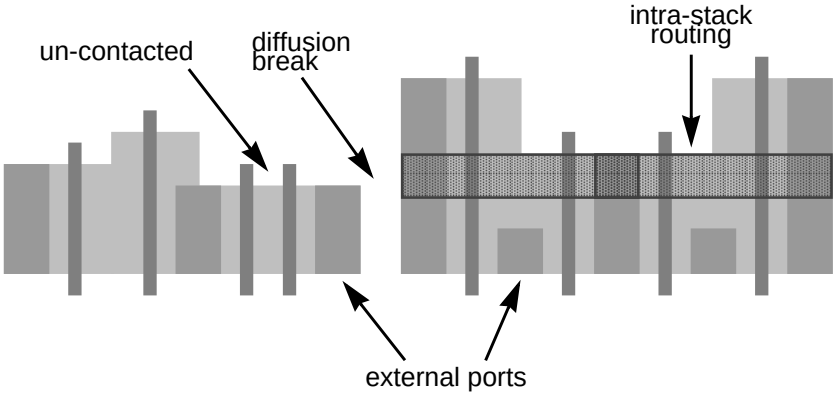
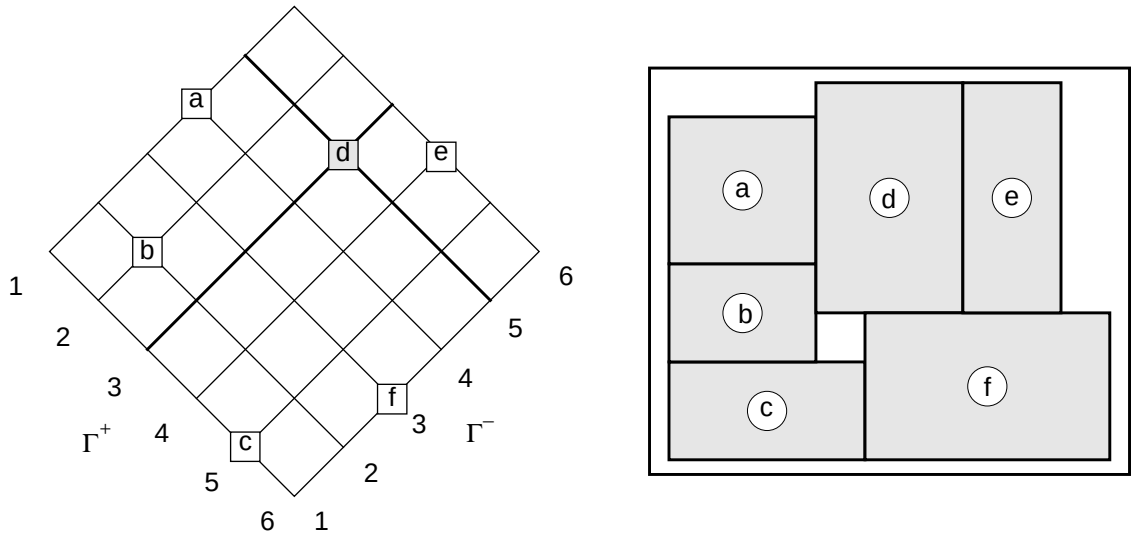


Figure 7: Layout produced by the stack generator

rule correct and which optimizes the desired cost function. In our current implementation this cost function is a weighted combination of the placement cost (area, perimeter, aspect ratio violation, etc.) and an estimate for the routing cost:

$$\text{cost} = w_1 \cdot \text{placement} + w_2 \cdot \text{routing} \quad (1)$$

In the implementation of the placement phase it is possible to borrow a great deal from the field of macro-block placement. However at the transistor level, a problem which we refer to as *micro-placement* [14], several differences manifest themselves. We list them here and elaborate on each one in the following sections.



$$(\Gamma^{+}, \Gamma^{-}) = (\text{abdecf, cbfade})$$

Figure 8: An example demonstrating the Sequence Pair representation from [18].

1. Modeling of the placement search space
2. Support for arbitrary geometry sharing
3. Support for transistor stacking
4. Routing model
5. Electrical considerations
6. Presence of cell template constraints

2.3.1 Modeling of the Placement Search Space

Of particular importance is the selection of a method with which to model the placement search space. In [16], a distinction is made between *direct* and *indirect* placement models. Direct models, such as those used in Timberwolf [17] and Koan [15], represent the placement as a set of objects, each with a specific *x* and *y* coordinate. Indirect models, on the other hand, represent the placement *symbolically* in an attempt to reduce the size of the search space. This reduction is achieved by collapsing many direct placements into equivalence classes which can be regarded as identical in some sense with respect to the cost function. Two examples of indirect models are the *Slicing Tree*, a method popular in the floorplanning literature which was adopted in [11] and [12], and the *Sequence Pair* introduced by Murata et. al [18].

We have chosen to base our placement engine on the Sequence Pair symbolic representation. This choice was made because we conjecture that symbolic methods provide a more efficient and rigorous framework with which to traverse the search space. In particular we chose the Sequence Pair over Slicing Trees because, as we will show, any placement reachable within a direct model can be collapsed into an equivalence class of reachable placements in the Sequence Pair representation. Some valid placements, on the other hand, are not representable as Slicing Trees.

The Sequence Pair model is based on the constraint graph formulation of the two-dimensional compaction problem [19,20,21]. Instead of explicitly representing the exact *x* and *y* coordinates for each object, this model simply keeps track of the relative position which will be enforced for each pair of objects in order to prevent them from overlapping. For two objects *A* and *B*, we simply need to know if *B* is above, below, to the right of, or to the left of *A*. This partial ordering implies a design rule constraint, in the specified direction, between the two objects. When all such

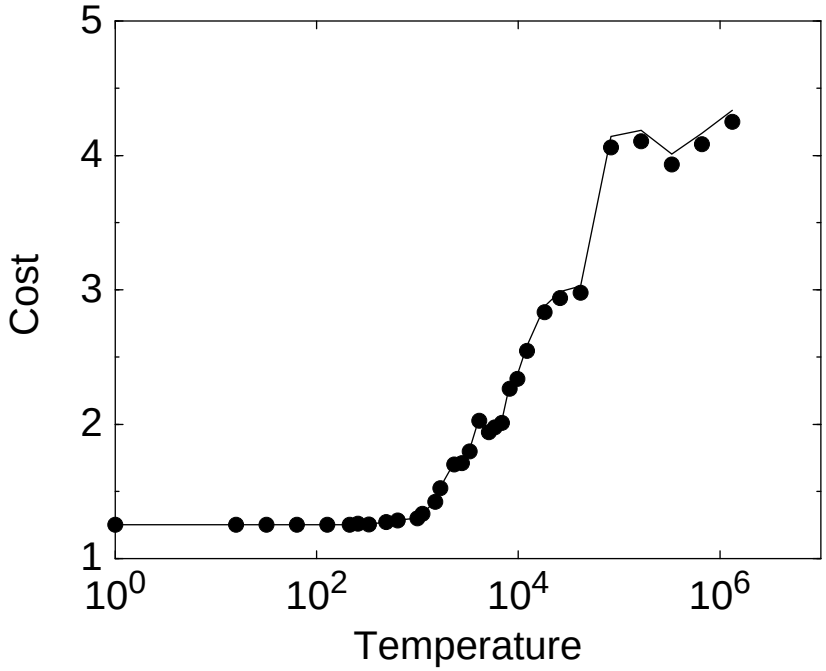


Figure 9: Simulated Annealing algorithm cooling data for a typical run

pairwise relationships are resolved, a constraint graph can be constructed and solved to yield the actual non-overlapping, design rule correct, coordinates for each object.

A Sequence Pair represents the partial ordering among placement objects as a pair of sequences, Γ^+ and Γ^-. The non-overlap constraint which is enforced between each pair of objects is implied by their relative positions in the two sequences. This is best visualized with the use of an oblique grid whose axes are labelled by the two sequences, as shown to the left in Figure 8. For example, object *e* appears after object *d* in both sequences, placing it in the right quadrant of the grid relative to *d*. This implies that a right-of constraint will be enforced between *d* and *e*. The final compacted placement for these objects, taking into account the constraints implied by the sequence pair and the sizes of the objects, is shown to the right.

It has been proven [18] that the set of packings representable with the Sequence Pair is *p-admissible*, meaning that it is precisely equal to the set of all feasible packings. However, each "packing" may actually represent a large set of equivalent placements obtained using a direct placement mode. When the constraint graph is solved, all objects on the *x* and *y* critical paths will have fixed positions, but all other objects will have some *slack* in their positions. These objects may move around within their range of slack while maintaining the same total area. We currently choose the simplest of these equivalent placements by compacting all objects toward the lower left, but one could conceivably optimize for some secondary criterion such as routing length.

In order to systematically explore the Sequence Pair solution space, as in [18] we use simulated annealing. This space is defined by the set of all permutations of the sequences Γ^+ and Γ^-, and all rotations and mirrorings of each object. The following set of moves are used by the simulated annealing engine:

1. Swap a pair of objects in either Γ^+, or Γ^- or both
2. Translate one object to a different position in either Γ^+, or Γ^- or both
3. Rotate and/or mirror one object into a different orientation
4. Re-order a selected super-stack or sub-stack (explained in Section 2.3.3)

Our simulated annealing engine makes use of a standard adaptive cooling schedule, automated initial temperature selection, range limiting, and statistical move selection [22]. In Figure 9 we show a plot of the cost function during a

Figure 10: An example of a second-order shared structure

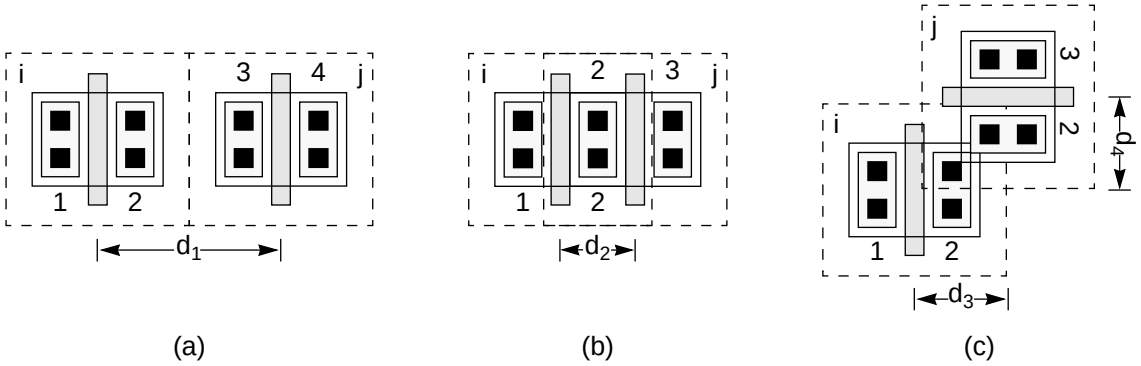


Figure 11: Three examples demonstrating a pair of transistors in different configurations with respect to geometry sharing.

typical run of the simulated annealing algorithm as the temperature is reduced.

2.3.2 Support for Arbitrary Geometry Sharing

The most noticeable difference between block placement and transistor-level placement is the fact that the objects being placed are no longer forbidden from overlapping. If two neighboring transistor stacks of the same diffusion type share a common electrical node, it will be desirable to allow the objects to merge. This will often reduce the cell area, and is also beneficial from a performance perspective, as the parasitic source and drain capacitances will be reduced.

Our methodology supports geometry sharing at two different levels. The formation of transistor stacks takes advantage of a great deal of the potential geometry sharing which is present in the circuit. This will be discussed in the following section. We also allow pieces of geometry to merge during the placement step, permitting larger merged structures to form and taking advantage of less obvious patterns of connection. We call these *second-order shared structures*. An example in Figure 10 shows two small transistors merging with a single larger transistor.

In order to support geometry merging at the transistor level we use a fairly simple mechanism, as shown in Figure 11. If two objects are in the proper configuration such that they have electrically compatible ports facing each other, as in Figure 11(b), the design rule constraint δ_1 can be relaxed to a smaller value, δ_2 to allow those ports to overlap. This may result in design rule violations in the final placement if the ports do not line up precisely, but this can be repaired in a post-processing step. This form of geometry sharing is formulated to encourage the formation of second-order source/drain shared transistor stacks, and complex stacks such as Figure 10. It does not currently support more complex shared structures such as Figure 11(c). This structure would require a *y* constraint as well as an *x* constraint to ensure design rule correctness.

2.3.3 Support for Transistor Stacking

As we mentioned in Section 2.1, it is very difficult to determine an optimal static clustering of the transistors into diffusion connected stacks. This is because the clustering step has no information about the relative positions of the stacks in the final placement. Traditional stack optimization algorithms can locate a stacking solution with minimum width and minimum internal routing cost, but such stackings may not be globally optimal when external area and wiring costs are taken into account. It is thus not clear *a-priori* to which stack a particular transistor should be assigned, and which stack ordering should be chosen.

Instead of investing a great deal of effort making static clustering choices prior to placement, we have adopted a methodology which allows the placement step to dynamically alter the clustering choices and stack orderings in order to find a configuration that minimizes global area and wiring costs. As described in Section 2.1, the clustering step is merely responsible for locating maximal sized sets of same-polarity diffusion connected transistors. These *super-stacks*, after being formed into linear transistor arrays during the stack generation phase described in Section 2.2, are broken at the diffusion breaks and the resulting *sub-stacks* are passed to the placement engine as the atomic placeable objects.

It is easy to see that there will be many Euler trail coverings for a given super-stack, and that all of them will have the same number of diffusion breaks. Thus, if we break the super-stacks at the positions of the diffusion breaks, and treat each sub-stack as a separate placeable object, the number of objects being placed will remain constant. If a new Euler trail is found for the super-stack, individual transistors may move from one sub-stack to another, and the individual sub-stacks may grow or shrink in size depending on the number of edges traversed in each sub-trail. The sub-stacks are free to be placed in any two-dimensional arrangement that optimizes the area and the external routing. Note that we are not attempting to optimize the height of these transistor stacks, as is traditional in the literature. In fact, the height of the stack is fixed by the input schematic. However, the stack geometry generator will report the number of internal stack nets which cannot be routed over top of the stacks, and these are added to the global routing cost. Thus, the sub-stacks are automatically optimized such that their internal routing cost is taken into account as well as the resulting global routing cost.

In order to unify the transistor clustering and stack formation steps with the transistor placement engine, two new moves were added to the simulated annealing move-set. The first corresponds to the *sub-trail modification move* described by Basaran [7]. One of the super-stacks is selected, and a random sub-trail within the current Euler trail is eliminated and replaced with a different randomly generated sub-trail with the same endpoints. It can be shown that this move is complete, and can thus be used to construct every possible Euler trail which can be embedded in the selected graph. The second move was added to increase the efficacy of the algorithm at low temperatures, when large sub-trail modification moves can be very disruptive and are thus rarely accepted. The second move, *the sub-stack modification move*, simply selects one sub-stack and locates a new ordering for its constituent transistors.

2.3.4 Routing Model

The most critical difference between block-level macro placement and transistor-level micro placement is the routing model which must be supported. A good routing model is critical because of the serialized split we have made between placement and routing. Good estimates for the routing cost must be made in order to encourage placements that are compact but also easily routeable. In addition, the placement step must leave sufficient empty space between the objects to accommodate the wiring.

Routing costs in large standard cell blocks can be accurately estimated using statistical means based on Steiner trees [24], however our experiments with these techniques at the transistor level showed that they tended to underestimate the routing cost and did not encourage final placements which were easily routeable. At the transistor level, it is our observation that multi-terminal nets are generally connected in a more point-to-point fashion. We have chosen to estimate the routing cost metric using a minimum spanning tree (MST) approximation.

Even more difficult than computing the wiring cost is the calculation of the additional area required for routing space. While we can generally assume that the router will be able to find some nearly direct path between terminals which approximates its minimum spanning tree *cost*, routing *space* calculations must be able to guess exactly which path that will be (without requiring a complete detailed routing). We have explored three techniques which are dia-

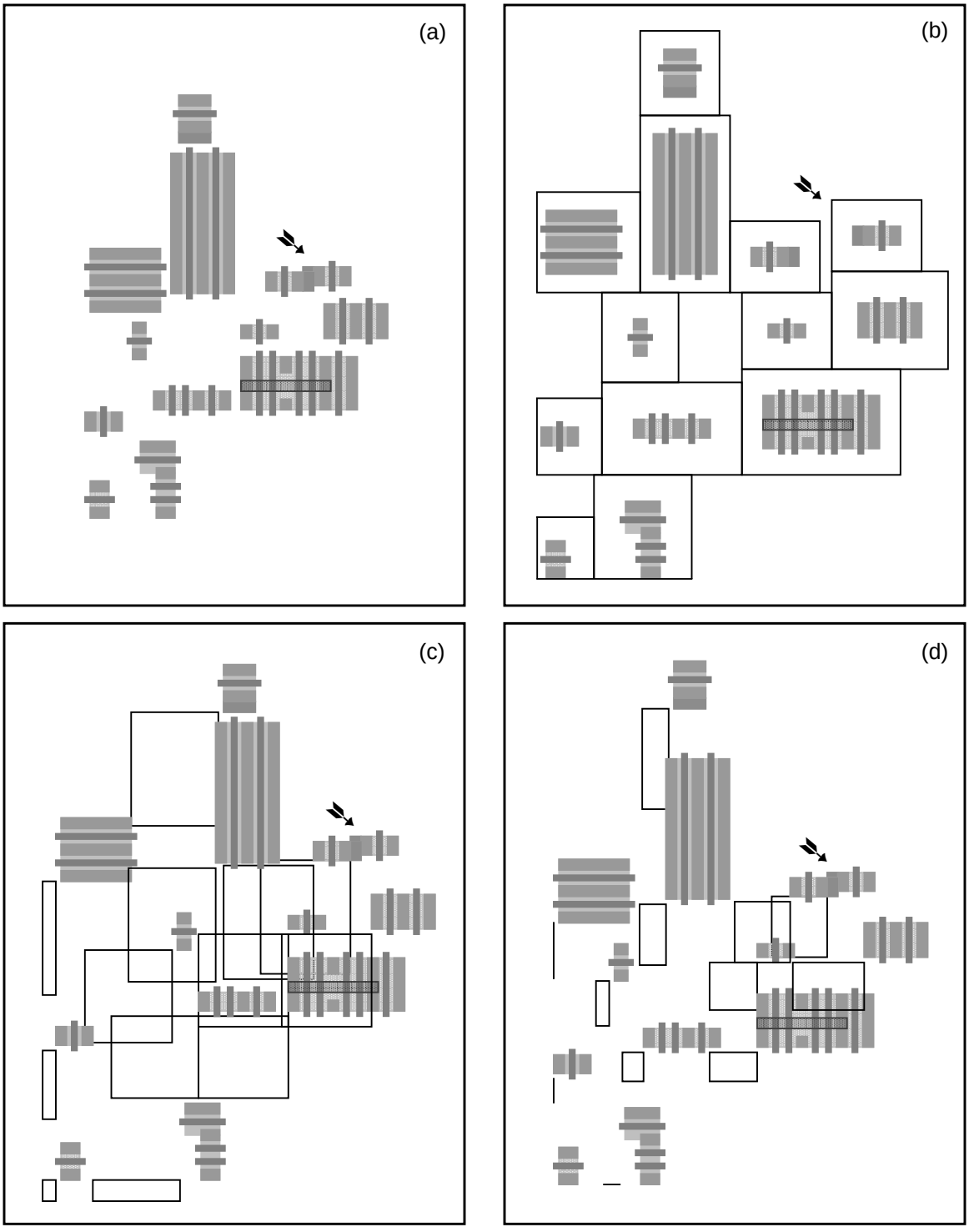


Figure 12: A demonstration of different routing space estimation techniques. Example (a) has no extra space reserved for routing, example (b) uses the method from [18], and example (c) uses the method from [15]. Example (d) uses our modified version of the method in [15].

grammed in Figure 12.

Figure 12(a) shows an intermediate placement without the addition of routing space. It is clear that some nets will be unroutable. In Figure 12(b) we use a technique from the *Koan* analog placement tool [15]. Here, each object's bounding box is increased to include an extra routing "*halo*". This halo is proportional in size to the number of I/O ports and inversely proportional to the perimeter of the object:

$$\text{halo}_i = \alpha \cdot \frac{\text{\# I/O ports}_i}{\text{perimeter}_i} \tag{2}$$

where α is an experimentally determined scaling factor for tuning purposes. We have found that this method is difficult to tune and tends to leave more space than necessary between most blocks, requiring post-routing compaction. In addition the halos tend to make the objects more uniform in size, due to its inverse relationship to the perimeter, resulting in placements which are more regular in appearance than manual designs.

An additional problem with the halo-based technique is that the non-uniform object growth tends to break apart second-order shared structures that are formed through design rule relaxation as described in Section 2.3.2. An example of this is marked with an arrow in the figures.

In Figure 12(c) we use a technique for routing space insertion due to Murata [18], which is an elaboration of a technique described by Onodera [21]. Here each object is translated from its original coordinates (x_i, y_i) to a new set of coordinates (x_i', y_i') based on an estimate of the routing resources which will be required below it and to the left. The new position is calculated as follows:

$$x_i' = x_i + \beta T \left(\frac{\sum_{i \in N_{x_i}} H_i}{H} \right) \qquad y_i' = y_i + \beta T \left(\frac{\sum_{i \in N_{y_i}} W_i}{W} \right) \tag{3}$$

where T is the routing pitch, N_{x_i} and N_{y_i} are the set of nets whose bounding boxes begin before object i in x and y, H_i and W_i are the height and width of net i's bounding box, and H and W are the height and width of the original placement. Again, β is an experimentally determined scaling factor. In the figure (x_i, y_i) correspond to the coordinate of the lower-left corner of each object. The additional boxes demonstrate the movement of each object—their lower left corners mark (x_i, y_i) and their upper-right corners mark (x_i', y_i'), for the objects i appearing at their upper-right corners. This method, designed for block placement, uses a greedy heuristic which assumes that every horizontal (vertical) net makes exactly one vertical (horizontal) jog at the far left edge of its bounding box. We have found that this method introduces a nonuniform bias in the routing, concentrating the routing space toward the lower left corner, leaving objects closer to the upper and right edges crowded together.

The final technique which we have explored is shown in Figure 12(d). This method, attempting to correct some of the problems which we found with Murata's method, makes different use of the method of Onodera [21]. We simply solve Equation (3) for $x_{\max}'$ and $y_{\max}'$, the upper right corner of the design, and assigns new coordinates to the blocks based on the following:

$$x_i' = x_i + (x_{\max}' - x_{\max}) \frac{x_i}{x_{max}} \qquad y_i' = y_i + (y_{\max}' - y_{\max}) \frac{y_i}{y_{max}} \tag{4}$$

This method approximate the total number of horizontal and vertical routing tracks which will be required, assuming that each net occupies one horizontal and one vertical track, and distributes these evenly throughout the design. As can be seen in the example, the allocation of routing space appears more uniform than in example (c), without the strong bias toward the lower-left corner.

As a final observation, note that the latter two expansion-based techniques based on Onodera's method do not break apart second-order shared structures (marked with an arrow) which were present in the original un-expanded

placement. This is explicitly supported in the expansion algorithm by identifying instances of geometry sharing and assigning all shared structures the same expansion factor as the most lower-left shared object.

We have chosen to adopt the final technique described above, which we term *proportional expansion*, for the reasons stated above. However, we still feel that these techniques are overly heuristic in nature and we are actively researching new alternatives.

2.3.5 Other Issues

We also listed two other issues which are of concern when adopting block level placement models to transistor level placement: electrical considerations, and the presence of cell template constraints.

An important aspect of transistor level synthesis is the optimization of the electrical performance of the circuit. Parasitic capacitances are minimized through geometry sharing, but one may also wish to analyze other properties such as crosstalk and the noise immunity of sensitive dynamic nodes. These may be incorporated into the cost function or as constraints in the move set.

Finally, one must account for a large number of possible constraints resulting from the cell template specification, as discussed in Section 12.1. Some of these may be complex and non trivial to account for in the placement and routing models. For our experiments we have adopted a relatively simple datapath style cell template specification which we explain in Section 4.

3 Routing

The final step of the synthesis design flow is routing. In our implementation, the placement step is responsible for ensuring that the routing problem is feasible. The routing step as we view it does not directly affect the cost function (except perhaps through routing parasitics.) For the final placement, either a feasible routing exists or it does not. Much research has been done in the field of detailed maze routing, and we make use of an existing tool: Anagram-II, an analog maze router described in [15].

4 Implementation

In this section we discuss a prototype tool called *TEMPO* (Transistor Enabled Micro Placement Optimization.) We have been using this tool as a framework to explore the ideas discussed in the previous sections. TEMPO is implemented in C++ and has been tested under the Sun Solaris and Linux operating systems.

As discussed in Section 2.2.3., we have implemented a generic placement engine based on the symbolic Sequence Pair model. This has been adapted to transistor level placement through the representation of placeable transistor and stack geometry primitives, dynamic transistor clustering and stack re-organization, and an adjacency analysis step which collapses the design rule constraints in order to allow arbitrary geometry sharing. A simulated annealing optimization algorithm has been implemented with a standard adaptive cooling schedule, range limiting, and statistical move selection [22].

The input is a Spice netlist file and a technology file specifying the design rules. Generators have been implemented to construct the transistor stack geometry, making use of optional static clustering and stacking order specifications supplied by the user through annotations in the Spice file. Routing length estimates are made using a minimum spanning tree model, and three different methods for performing routing space estimates have been implemented. We make use of the *Anagram-II* router for post-placement routing. Output mask geometry is produced in Berkeley Magic format.

We have adopted a datapath-style cell template for the current set of experiments. Internal routing is performed in poly and metal-1. Control inputs are assumed to arrive vertically in metal-2 and data inputs arrive horizontally in metal-3. We place these inputs as overcell routing tracks in a position which is as close as possible to the center of the associated net's bounding box.

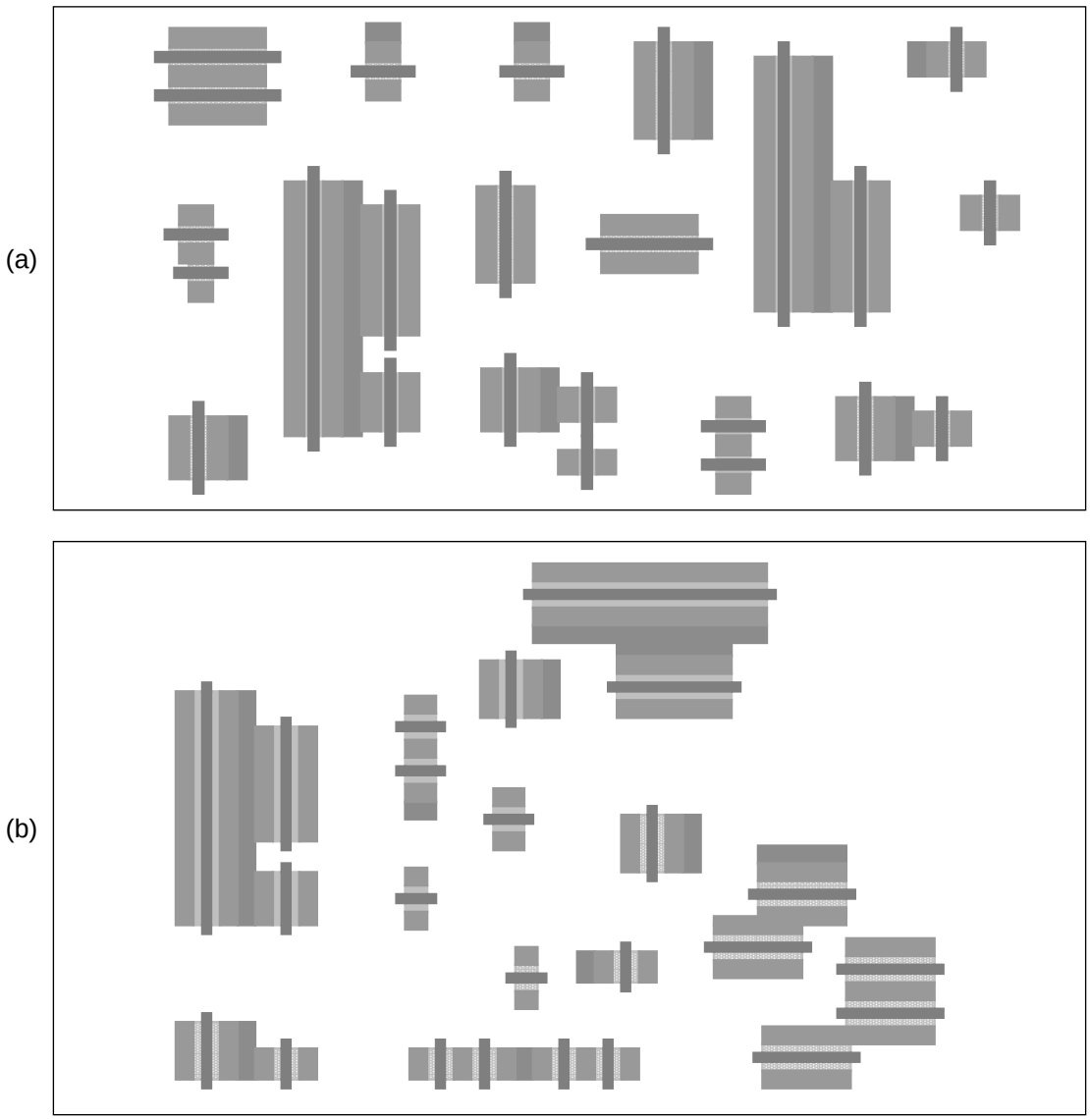


Figure 13: The mux-flipflop placement produced by: (a) the *Koan* analog placement system [15], (b) TEMPO with transistor stacking disabled.

5 Experiments

The lack of a standard set of 2D circuit benchmarks makes quantitative comparison among 2D cell synthesis tools difficult. In this section we discuss several experiments which qualitatively illustrate the value of several aspects of our methodology.

In Figure 13 we show an experiment conducted on the mux-flipflop circuit, our running example from Figure 2. Figure 13(a) shows the placement as realized by the *Koan* analog placement system [15]. Figure 13(b) shows the output of *TEMPO* when clustering and dynamic transistor stacking have been disabled. Here all geometry sharing results from the merging of adjacent geometry. It closely resembles the output of *Koan*. Without integrated transistor stacking, *Koan* and *TEMPO* are not able to discover the long transistor chains which are clearly visible in the manual design.

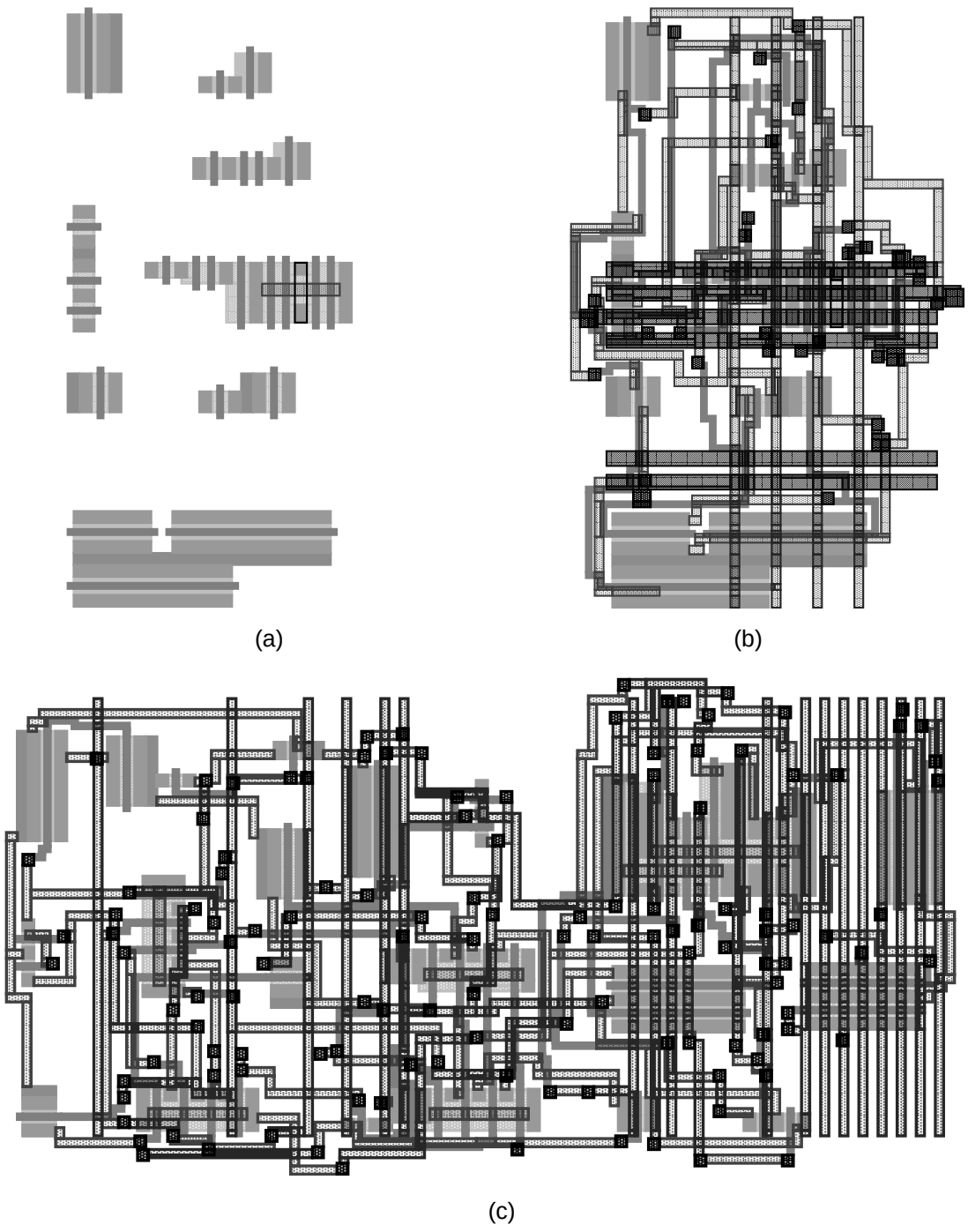


Figure 14: Two placed and routed example circuits. Example (a) is the mux-flipflop circuit from Figure 2. Example (b) is a CVSL carry-save adder with muxed latching inputs from [25].

Two complete placed and routed examples are shown in Figure 14. Figure 14(a) demonstrates a placement of the mux-flipflop circuit realized by TEMPO with clustering and dynamic transistor stacking enabled, while Figure 14(b)

shows the final routing solution realized by the *Anagram-II* router. The second circuit, shown in Figure 14(c), is the "CSA-MUX" circuit from [25]. Both circuits were implemented in the MOSIS 0.35 micron CMOS process HP14B. The former used the transistor sizes specified in the original CGaAs design, and was optimized for minimum area, while the latter was sized by us using HSpice and was optimized to find its minimum width given a fixed height.

In Figure 14(a) the mux-flipflop example has dimensions (h=67μ, w=44.3μ). This compares favorably with the hand-designed example in Figure 2, which has dimensions (h=35μ, w=60μ). However, a direct comparisons is impossible—the MOSIS CMOS process used in TEMPO has tighter metal geometries than the CGaAs process used in the manual design.

6 Conclusions and Future Work

In this paper we have formulated a general four step framework for 2-dimensional cell synthesis based on transistor-level placement and routing. Our methodology is targeted at complex digital designs in non-dual logic families such as CVSL, PTL, and domino CMOS, for which existing 1-dimensional methods are inadequate. We have explored a number of options available at each step of this process, and described the choices we have made in the implementation of our prototype tool framework, *TEMPO*. We have presented a number of examples which illustrate the main features of our model: a non-gridded 2D symbolic placement engine, dynamic clustering and stack formation through transistor stack reorganization and through arbitrary geometry sharing, and integrated routing space estimation.

We are currently researching the wire space estimation problem in search of more accurate techniques, and we are compiling a set of benchmark circuits for 2-dimensional cell placement problems. These benchmarks will be obtained from the literature as well as from industrial sources, and we intend to release these to the community in order to encourage more accurate comparison between competing tools.

As a long term goal, we would like to explore methods for linking on-demand cell synthesis to front-end logic synthesis and back-end buffer sizing, power reduction, and wire optimization techniques.

Acknowledgments

The authors would like to thank Dr. Rob Rutenbar and Phiroze Parakh for many interesting discussions.

References

[1] M. Lefebvre, D. Marple, C. Sechen, "The Future of Custom Cell Generation in Physical Synthesis," in proc. 1997 Design Automation Conference, pp. 446–451.

[2] T. Uehara, W.M. VanCleemput, "Optimal Layout of CMOS Functional Arrays," IEEE Transactions on Computers, C-30(5), May 1981, pp. 305–312.

[3] R.L. Maziasz, J.P. Hayes, "Layout Minimization of CMOS Cells," Kluwer Academic Publishers, Boston, 1992.

[4] C. Poirier, "Excellerator: Custom CMOS Leaf Cell Layout Generator," IEEE Transactions on Computer Aided Design, 8(7), July 1989, pp. 744–755.

[5] Y. Hsieh, C. Huang, Y. Lin, Y. Hsu, "LiB: A CMOS Cell Compiler," IEEE Transactions on Computer Aided Design, 10(8), August 1991, pp. 994–1005.

[6] M. Lefebvre, D. Skoll, "PicassoII: A CMOS Leaf Cell Synthesis System," in proc. 1992 MCNC International Workshop on Layout Synthesis, vol. 2, pp. 207–219.

[7] B. Basaran, "Optimal Diffusion Sharing in Digital and Analog CMOS Layout", Ph.D. Dissertation, Carnegie Mellon University, CMU Report No. CMUCAD-97-21, May 1997.

[8] B. Bernhardt et al, "Complementary GaAs (CGaAs™): A High Performance BiCMOS Alternative", in proc. 1995 GaAs IC Symposium, pp. 18–21.

[9] K. Tani et. al., "Two-Dimensional Layout Synthesis for Large-Scale CMOS Circuits", in proc. 1991 ICCAD, pp. 490–493.

[10] A. Gupta, J.P. Hayes, "CLIP: An Optimizing Layout Generator for Two-Dimensional CMOS Cells," in proc. 34th Design Automation Conference, 1997, pp. 452–455.

[11] H. Xia, M. Lefebvre, D. Vinke, "Optimization-Based Placement Algorithm for BiCMOS Leaf Cell Generation", IEEE J. Solid State Circuits, 29(10), Oct. 1994, pp. 1227–1237.

[12] M. Fukui, N. Shinomiya, T. Akino, "A New Layout Synthesis for Leaf Cell Design", in proc. 1995 ASP-DAC, pp. 259-263.

[13] S. Saika, M. Fukui, N. Shinomiya, T. Akino, “A Two-Dimensional Transistor Placement Algorithm for Cell Synthesis and its Application to Standard Cells”, IEICE Trans. Fundamentals, E80–A(10), Oct. 1997, pp. 1883–1891.
[14] R. Rutenbar, private communication.
[15] J. Cohn, D. Garrod, R. Rutenbar, and L. R. Carley, “Analog Device-Level Layout Automation”, Kluwer Academic Publishers, Boston MA., 1994.
[16] R. Rutenbar, “Simulated Annealing Algorithms: An Overview,” IEEE Circuits and Devices Magazine, January 1989, pp. 19–26.
[17] C. Sechen, A. Sangiovanni-Vincentelli, “The TimberWolf Placement and Routing Package”, IEEE Journal of Solid State Circuits, SC-20(2), Apr. 1985, pp. 510–522.
[18] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani, “Rectangle Packing Based Module Placement,” in proc. 1995 ICCAD, pp. 472–479.
[19] M. Schlag, Y. Liao, et al., “An Algorithm for Optimal Two-Dimensional Compaction of VLSI Layouts,” Integration 1(2,3), 1983, 179-209.
[20] S. Sutanthavibul, E. Shragowitz, J. Rosen, “An Analytical Approach to Floorplan Design and Optimization,” in proc. 27th Design Automation Conference, 1990, pp. 187–192.
[21] H. Onodera, Y. Taniguchi, and K. Tamaru, “Branch-and-Bound Placement for Building Block Layout,” in proc. 28th Design Automation Conference, 1991, pp. 433–439.
[22] J. Cohn, “Automatic Device Placement for Analog Cells in KOAN,” Ph.D. Dissertation, Carnegie Mellon University, CMUCAD-92-07, 1992.
[23] C. Papadimitriou, K. Steiglitz, “Combinatorial Optimization Algorithms and Complexity,” Prentice-Hall, 1982, p. 413.
[24] C. Cheng, “RISA: Accurate and Efficient Placement Routability Modeling”, in proc. 1994 ICCAD, pp. 690–695.
[25] J. Kowaleski et. al. “A Dual-Execution Pipelined Floating-Point CMOS Processor”, in proc. 1996 ISSCC, pp. 358-359.