TRANSISTOR LEVEL MICRO-PLACEMENT AND ROUTING FOR TWO-DIMENSIONAL DIGITAL VLSI CELL SYNTHESIS

Michael A. Riepe, Karem A. Sakallah
University of Michigan, Ann Arbor, MI 48109-2122
{riepe,karem}@eecs.umich.edu

Abstract

There is an increasing need in modern VLSI designs for circuits implemented in high-performance logic families such as Cascode Voltage Switch Logic, Pass Transistor Logic, and domino CMOS. Circuits designed in these non-dual ratioed logic families can be highly irregular with complex geometry sharing and non-trivial routing. Traditional digital cell layout synthesis tools derived from the row-based "functional cell" style break down when confronted with such circuit topologies. These cells require a full-custom 2-dimensional layout style which currently requires skilled manual design. In this work we define the synthesis of complex 2-dimensional digital cells as a new problem which we call transistor-level micro-placement and routing. To address this problem we develop a complete end-to-end methodology which is implemented in a prototype tool named TEMPO. Our primary focus in this work is the micro-placement problem. We explore techniques for the modeling and dynamic optimization of geometry sharing though transistor chaining and arbitrary geometry merging. Experiments conducted with a new set of benchmark circuits show promising results when TEMPO is compared to a commercial cell synthesis tool.

1. Introduction

Library cells make up the lowest level of the digital VLSI design hierarchy. Clearly, the quality of the cells has 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 that standard-cell placement and routing tools are able to obtain significantly higher routing quality if they have the ability to choose between multiple instances of cells with a wide variety of pin orderings. In one experiment [13], an average reduction of 10.8% in the number of routing tracks was demonstrated over five benchmarks circuits.

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. Several authors, 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 [3, 13]. This would permit: 1) logic synthesis tools to request specific logic decompositions, doing away with the traditional technology mapping step; 2) standard-cell and datapath placement and routing tools to request cells with an exact pin ordering; 3) interconnect optimization tools to request cells with specific input and output impedance values; and 4) 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 [13].

2. Motivation

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 [22]. 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 gaps in the diffusion islands (so called "diffusion breaks"). We will refer to layouts in

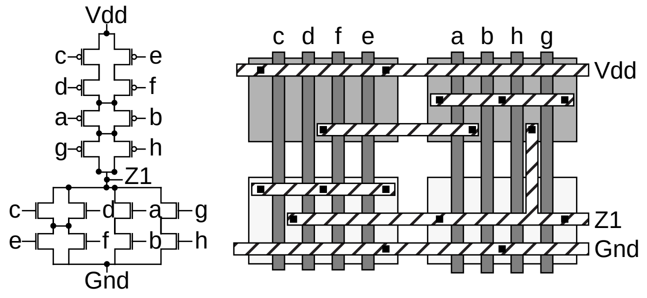


Figure 1: Example of a complex gate designed in the functional-cell style of Uehara & VanCleemput [22]

this style as 1-dimensional, or 1D, layouts.

The synthesis of 1D layouts can be formulated as a straightforward graph optimization problem: the identification 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 [14] presented the first provably optimal algorithms.

A major drawback of the 1D layout style is that it applies directly 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 *C5M* [3], *Excellerator* [17], *LiB* [10], and *Picasso-II* [12]. 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 [1].

Despite the elegant formulation presented by the 1D 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, such non-complementary ratioed logic families often result in physical layouts which are distinctly 2-dimensional (2D) 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 [2]. This example demonstrates a number of properties which deviate from the standard 1D style:

1. It is highly irregular. Some regularity is present in the rows, or “chains” of merged transistors, but these chains 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.

2.1. Previous Work

A variety of approaches have been taken to address the 2D cell synthesis problem. Tani et al [21] and Gupta and Hayes [9] discuss a style in which 2D layouts are formed from multiple 1D 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 2D styles, we refer to it as being 1-1/2 dimensional.

Xia et al [23] developed a method for BiCMOS cell generation.

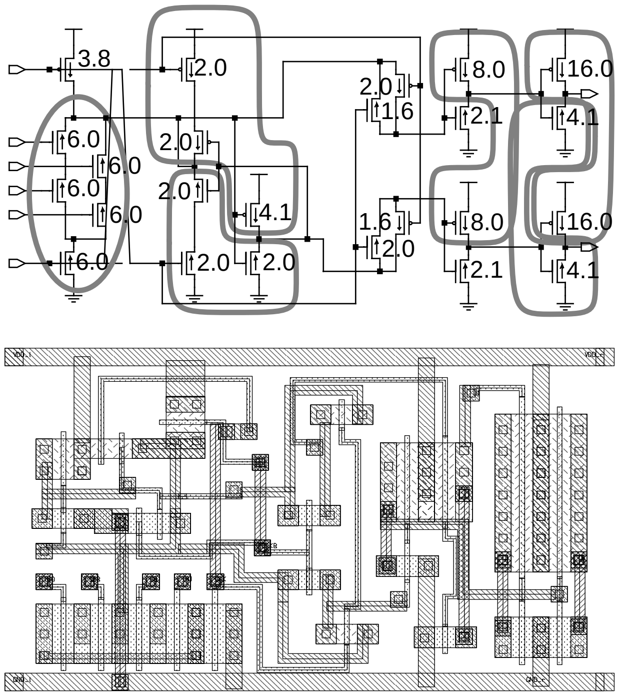


Figure 2: A manually designed cell showing complex two-dimensional layout structure

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

Fukui et al [8] developed a system for 2D digital transistor placement and routing. A simulated annealing algorithm is used to find good groupings of transistors into diffusion-shared chains and a greedy exploration of a slicing structure is performed to find a 2D virtual grid floorplan. A symbolic router is used to perform detailed routing, and a final compaction step is used to permit transistors within chains to slide into locally optimal positions.

In a more recent work by the same group, Saika and Fukui et al [19] present a second tool which operates by statically grouping the transistors into maximally sized series chains and then finding a high quality 1D 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 2D 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 [6]. 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.

2.2. Proposed Methodology

In this work we present the architecture for a 2D cell synthesis system which targets complex non-complementary digital MOS circuits. As our top level framework we adopt the flow of steps dia-

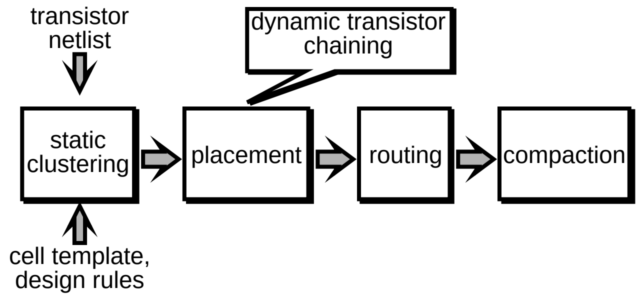


Figure 3: Proposed cell synthesis methodology

grammed in Figure 3. In the first step, clusters of transistors are formed, each representing a set to be composed into a single merged structure, or *transistor chain*. Clusters of size one represent degenerate chains of individual transistors. These clusters are passed to the placement step where they are assigned a position and an orientation and their chain ordering is optimized. The completed placement is routed and compacted to remove un-utilized space.

Central to our methodology are techniques for the modeling and optimization of transistor geometry sharing. When electrical connections are made through geometry sharing area is saved both because of the overlapping geometry and because of the elimination of wiring. In existing 2D cell synthesis systems [8, 19, 23] this optimization is performed as a static chaining step before placement. *Koan* [6], which is targeted at analog synthesis, performs no explicit chain formation, but instead allows arbitrary merged structures to form dynamically during placement. A unique aspect of the methodology in Figure 3 is that it adopts the strengths of both approaches. We make use of a late binding process in which explicit transistor chains are optimized *dynamically* during placement and arbitrary shared structures are allowed to form through dynamic geometry merging. During placement the chain optimization process thus has access to global placement and routing information which is not available in a static pre-processing step.

2.3. Outline

In the remainder of this paper we discuss each step of our methodology and discuss how our methods extend those of previous authors. In Section 3 we discuss the relevant aspects of transistor chaining theory. Section 4 introduces the details of our methodology. Section 5 discusses the implementation of our experimental system, which is named *TEMPO,* and presents some experimental results. Section 6 provides a summary and conclusions.

3. Transistor Chaining Theory

It should be clear from the layout shown in Figure 2 that most geometry sharing in digital circuits, even the complex digital circuits with which we are concerned, takes place within transistor chains. However, note that the chains do not normally appear in dual pairs as in the functional cell style of Figure 1. In this section we review the theory behind non-dual transistor chain optimization. We begin by establishing some basic terminology.

Definition 1: **Transistor Cluster**. A transistor cluster is an unordered set of same-polarity transistors which can be reached through an unbroken sequence of source or drain terminal connections. These are also often referred to as channel-connected components (CCCs).

Definition 2: **Transistor Chaining**. A transistor chaining is a mapping which assigns a unique ordering to the source-drain terminals of the transistors in a cluster.

Definition 3: **Transistor Chain**. A transistor chain is a transistor cluster which has been assigned a chaining. This is a well defined physical structure for which a design-rule correct layout can be generated.

Definition 4: **Diffusion Break**. A diffusion break is a position within a transistor chain at which two neighboring transistors do not share a common electrical terminal. Therefore these two terminals will not be able to share geometry and must be separated by the proper design rule distance.

Definition 5: **Transistor Sub-Chain**. A transistor sub-chain is a subset of a transistor chain which is free of diffusion breaks. We will make use of this definition in our dynamic chaining approach discussed in Section 4.2.

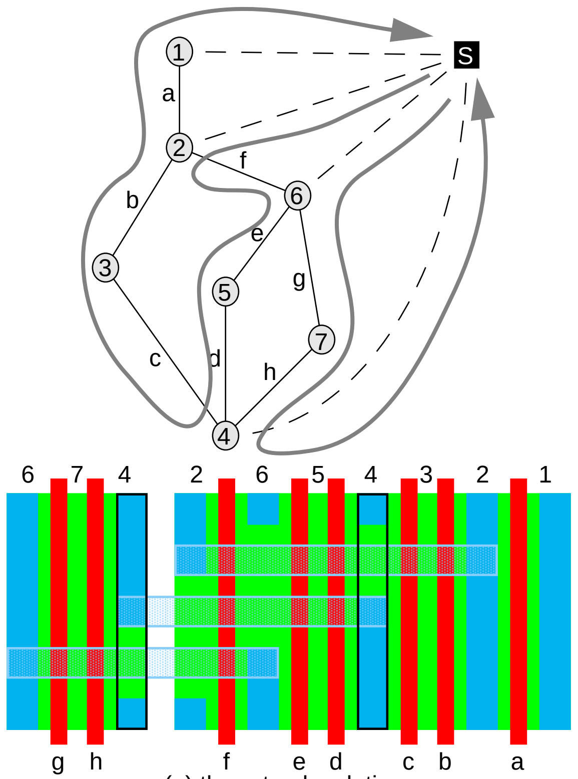
The process of transistor chain optimization involves two steps. First the transistors in the design must be partitioned into clusters. Second, a chaining must be found for each cluster. Chain optimization is the process of finding a chaining which minimizes the number of diffusion breaks, and hence the width of the chain, and at the same time optimizes the chain with respect to the global cell placement and routing.

Transistor chain optimization is traditionally posed as a graph problem. A graph called the *diffusion graph* is constructed for each cluster by associating a vertex with each electrical net. Edges represent transistors and span the corresponding source-drain terminal vertices. It is known that the minimum width solutions correspond to solutions with a minimum number of covering Euler trails of this graph. It is also known that there are exponentially many covering Euler trails for any given cluster [1]. For example, in Figure 4 we show two different chainings for the same transistor cluster. Both chains have one diffusion break, but note that chain b requires one fewer horizontal routing track than chain a.

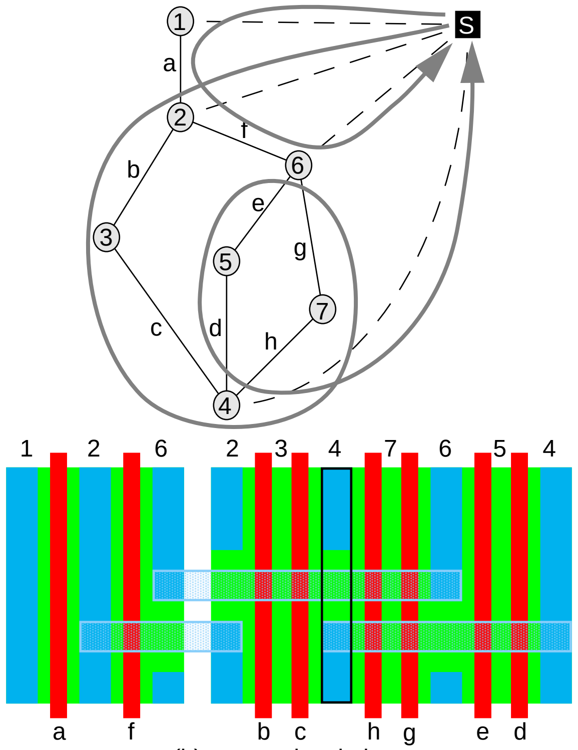
To establish the ordering of the transistors within chains we have adopted the algorithms in [1] which were developed for 1D linear transistor array width minimization. It is well known that an Euler trail can be embedded in a DAG if and only if the graph has either zero or two odd-degree vertices. In 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. The algorithm in Figure 5 is used to find an Euler trail on this graph beginning at the vertex v_α and ending at the vertex v_β. A covering 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 chain, with the super-edges corresponding to diffusion breaks. In Figure 4 we show the diffusion graphs associated with each chaining; the super-vertex is labelled “S”.

4. Cell-Level Transistor Micro-Placement

In order to attack the general 2D cell synthesis problem our methodology adopts a very general approach based on unconstrained placement and routing. However, there are a number of differences which distinguish this problem from traditional macroblock placement and routing. We refer to our problem as transistor-level micro-placement and routing. In the remainder of this section we discuss the implementation of each of the steps in the methodology outlined in Section 2.2.



(a) three-track solution



(b) two-track solution

Figure 4: Two chainings with different maximum route density

```
EULER($v_\alpha$,$v_\beta$)
    if $v_\alpha$ has no edges
        return $v_\alpha$
    starting from $v_\alpha$ create a walk of
    $G$, never visiting the same edge
    twice, until $v_\beta$ is reached again;
    let [$v_\alpha$,$v_1$],...,[$v_n$,$v_\beta$] be this walk;
    delete [$v_\alpha$,$v_1$],...,[$v_n$,$v_\beta$] from $G$;
    return (EULER($v_\alpha$),EULER($v_1$),...,
        EULER($v_n$),$v_\beta$)
```

Figure 5: Algorithm to find an Euler path in an Eulerian graph [16]

4.1. Static Transistor Clustering

In the first step, transistor clustering, the task is to determine the optimal number of transistor chains and which transistors belong in each chain. At this stage very little information is available, only the transistor connectivity and the sizes of each transistor, so the choices can only be heuristic in nature. Figure 2 shows the clustering used in the manually designed CGaAs mux-flipflop circuit.

Authors in previous works have used a wide variety of heuristics at this stage. In [8] the authors use simulated annealing. In [19] only simple chains made up of maximum length series chains are created and larger chains are formed during the 1D optimization step. It is also fairly common, especially in 1D tools, to approach clustering by performing logic gate recognition, as in [12].

We begin with an initial set of clusters formed from the set of all maximal channel-connected components (MCCCs). We then make use of a Fidducia-Mattheyses (FM) bipartitioning algorithm [7] to recursively partition these clusters until a user-supplied upper size bound is met. An optional feature allows the user to specify an upper limit on size variation among the transistors in a particular cluster.

4.2. Dynamic Transistor Chaining

As we mentioned in Section 4.1, it is very difficult to determine an optimal static clustering of the transistors into diffusion-connected chains. This is because the clustering step has no information about the relative positions of the chains in the final placement. Traditional chain optimization algorithms can locate a chaining solution with minimum width and minimum internal routing cost, but such chainings may not be globally optimal when external area and wiring costs are taken into account. It is thus not clear *a priori* to which chain a particular transistor should be assigned, and which chain ordering should be chosen.

The following technique allows the placement step to dynamically alter the chainings in order to find a configuration that minimizes global area and wiring costs. During clustering we form relatively large static clusters of transistors. During placement, when these large clusters are assigned a chaining, they are split at the diffusion breaks and the resulting *sub-chains* 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 chain, and that all of them will have the same number of diffusion breaks. Thus the number of sub-chains being placed will remain constant. When a new Euler trail is found for a chain, individual transistors may move from one sub-chain to another, and the individual sub-chains may grow or shrink in size. The sub-chains are free to be placed in any 2D arrangement that optimizes

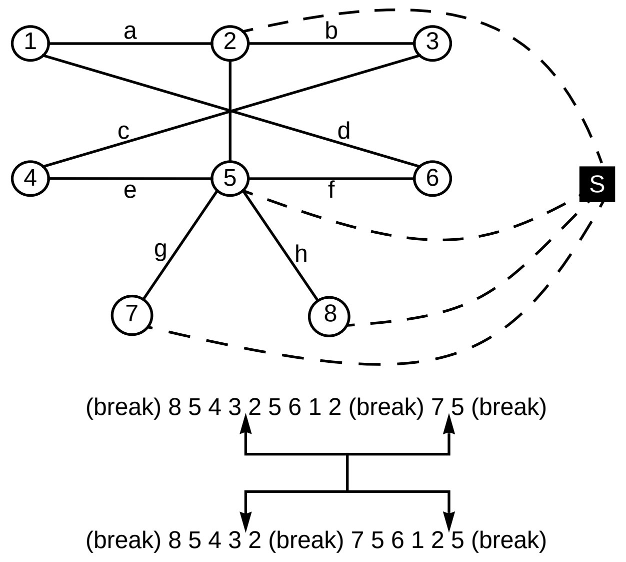

1
a
2
b
3
c
d
4
e
5
f
6
S
g
h
7
8
(break) 8 5 4 3 2 5 6 1 2 (break) 7 5 (break)
(break) 8 5 4 3 2 (break) 7 5 6 1 2 5 (break)

Figure 6: A chain modification performed with Basaran's iterative method [1]

the area and the external routing. Note that we are not attempting to optimize the height of these transistor chains, as is traditional in the literature. In fact, the height of the chain is fixed by the transistor channel widths in the input schematic. However, the chain geometry generator will report the number of internal chain nets which cannot be routed on top of the chains, and these are added to the global routing cost. Thus the sub-chains are automatically optimized such that their internal routing cost is taken into account as well as the resulting global routing cost.

It is the task of the placement engine to explore the universe of all possible chainings for each cluster. This corresponds to the set of all possible Euler trails in each cluster's diffusion graph. We make use of the following iterative technique due to Basaran [1] for generating one valid chaining from another. As shown in Figure 6, we select a random sub-trail from the current chaining and reset the edges of this sub-trail in the diffusion graph. We then call algorithm `Euler()` to generate a different sub-trail with the same two endpoints. Notice how the sizes of the two sub-chains changed as a result of this transformation. It can be shown that this operation is complete, and can thus be used to construct every possible Euler trail which can be embedded in the selected graph.

4.3. Support for Arbitrary Geometry Sharing

In addition to explicitly merged chains we also allow arbitrary pieces of geometry to merge during the placement step, thus permitting larger merged structures to form and taking advantage of less obvious patterns of connection. This is a trick often used by skilled human designers, and also is the primary method for geometry merging in the *Koan* [6] analog placement tool. We call the resulting merged objects *second-order shared structures*. An example in Figure 7 shows two small transistors merging with a single larger transistor.

In order to support arbitrary geometry merging at the transistor level we use a fairly simple mechanism. If two objects are in the proper configuration such that they have electrically compatible ports facing each other, 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.

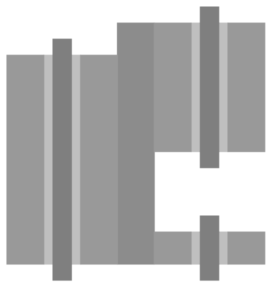


Figure 7: Example of a second-order shared structure

4.4. Modeling of the Placement Search Space

Of particular importance is the selection of a method with which to model the placement search space. We have chosen to base our placement engine on the symbolic Sequence Pair representation recently introduced by Murata et al [15]. This choice was made because we conjecture that symbolic methods provide a more efficient and rigorous framework with which to traverse the search space than direct representations of the object coordinates as used in *Timberwolf* [20] and *Koan* [6]. In particular we chose the Sequence Pair over symbolic Slicing Trees because some valid placements are not representable in the latter representation.

In order to systematically explore the Sequence Pair solution space, as in [15], we use simulated annealing. This space is defined by the set of all rotations and mirrorings of each object and the set of all permutations of two sequences, Γ^+ and Γ^-. This pair of sequences represent the relative x-axis and y-axis positions of the objects, respectively. The following set of moves are used by the simulated annealing engine:

1. Swap a pair of objects in Γ^+, Γ^- or in both
2. Translate one object to a different position in Γ^+, Γ^- or in both
3. Rotate and/or mirror one object into a different orientation
4. Re-order a selected chain as explained in Section 4.2.

Our simulated annealing engine makes use of a standard adaptive cooling schedule, automated initial temperature selection, range limiting, and statistical move selection [5]. The cost function is a weighted combination of the placement cost (area, perimeter, aspect ratio violation, etc.) and an estimate for the routing cost:

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

4.5. Routing Model

Because of the serialized split we have made between placement and routing, during placement we are forced to make use of computationally efficient estimates of the actual routing cost. The routing model which we use during placement accounts for routing effects in two ways. The routing component of the cost function in (1) is included to encourage the simulated annealing algorithm to find a placement, among all minimum area placements, which has low routing complexity. This is estimated by approximating the multi-terminal signal nets as rectilinear minimum spanning trees (RMSTs). RMSTs provide a fairly tight lower bound on the actual routing cost and can be computed in linear time.

In addition to the routing component of the cost function we

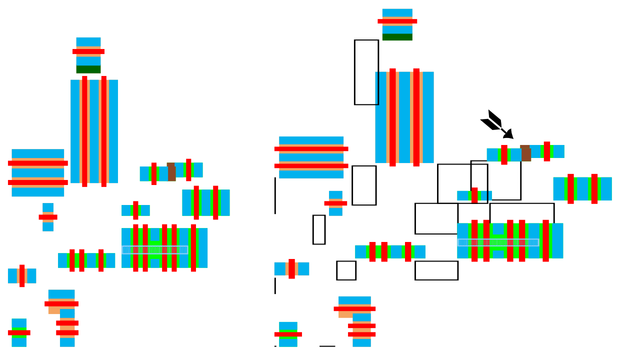

(a) before space insertion (b) after space insertion

Figure 8: Demonstration of routing space insertion

must also account for the extra space that this routing will require. Without inserting extra space between the transistors prior to routing we will most likely present the router with an infeasible problem. However, to do this precisely would require us to find a complete global routing. As an alternative we make use of the following approximate technique which is a variant of the technique used in [15].

Figure 8(a) shows an intermediate placement without the addition of routing space. It is clear that some nets will be unroutable. Figure 8(b) shows the same placement with the addition of extra routing space. Here each object is translated from its original coordinate (x_i, y_i) to a new coordinate (x_i', y_i') based on an estimate of the total routing resources which will be required below it and to its left. 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 objects i appearing at their upper-right corners.

This cell expansion is performed as follows. We first calculate a value for $(x_{\max}', y_{\max}')$, the new upper right corner of the placement bounding box, using the equation

$$x_{\max}' = x_{\max} + \beta T\left(\frac{\sum_{i \in N} H_i}{y_{\max}}\right) \qquad (2)$$

where T is the routing pitch, N is the set of all nets, and H_i is the height of net i's bounding box. β is a user defined scaling factor. We then assign new coordinates to the blocks based on the following equation:

$$x_i' = x_i + (x_{\max}' - x_{\max})\frac{x_i}{x_{\max}} \qquad (3)$$

(A similar pair of equations is used to calculate $y_{\max}'$ and y_i'.) This method approximates the total number of horizontal and vertical routing tracks that will be required, assuming that each net requires one horizontal and one vertical track, and distributes these evenly throughout the design. We therefore call this the *uniform expansion method*.

As a final observation, note that our method does not break apart second-order shared structures (one is marked with an arrow) which are present in the original un-expanded placement. This is explicitly supported in the expansion algorithm by identifying instances of geometry sharing and assigning the same expansion factor to all objects in the shared structure, namely that of the object occupying the lower-left corner of the structure's bounding box.

5. Experiment Results

To validate the approach outlined in the previous sections we have conducted an extensive series of experiments using a new set of high performance benchmark circuits. In the following we describe some details of our prototype implementation and benchmark circuits and report the results of our experiments.

5.1. Implementation

A prototype tool called *TEMPO* (Transistor Enabled Micro Placement Optimization) has been designed as a framework to explore the ideas discussed in this paper. *TEMPO* is implemented in approximately 48,000 lines of C++ code and has been tested under the Sun Solaris and Linux operating systems. To complete our end-to-end flow we make use of the *Anagram-II* [6] router for post-placement routing and the *Masterport* leaf cell compactor from Duet Technologies, Inc. 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/power/ground inputs arrive horizontally in metal-3. Prior to routing 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.

5.2. Benchmarks

The lack of a standard set of 2D cell synthesis benchmark circuits has made comparison with competing approaches difficult. For the following set of experiments we make use of a new set of 22 benchmark circuits which we have assembled from the solid-state circuits literature. These circuits are representative of the class of high performance and low power library cells for which this system is targeted. Included are circuits designed in domino and zipper CMOS, static and dynamic CVSL, as well as single- and double-ended PTL. The transistors were tuned with a heuristic non-linear gradient descent algorithm using models from a commercial 0.5µm CMOS process.

The characteristics of these circuits are described more completely in [18], and they can be obtained from the authors at the internet address in [11]. In addition to problems in transistor-level layout synthesis, we expect these benchmarks to prove useful on other problems in high-performance cell-level integrated circuit design automation. Other possible applications include: transistor-level circuit synthesis, transistor tuning, cell testing and characterization, and performance-driven technology mapping.

5.3. Results

We report the results of a series of experiments which compare *TEMPO* to the *LAS* synthesis tool [4] from Cadence Design Systems inc. *TEMPO* was run on a 300MHz Intel Pentium-II based workstation while *LAS* was run on a 170MHz Sun Ultrasparc system; both had 128 megabytes of RAM. We assume a standard 0.5µm CMOS process and make use of the MOSIS scalable submicron design rules (rev. 7.2) with a lambda value of 0.3µm. *TEMPO* cells were optimized for minimum area under our datapath style cell template. *LAS* is a 1-1/2 dimensional row-based sys-

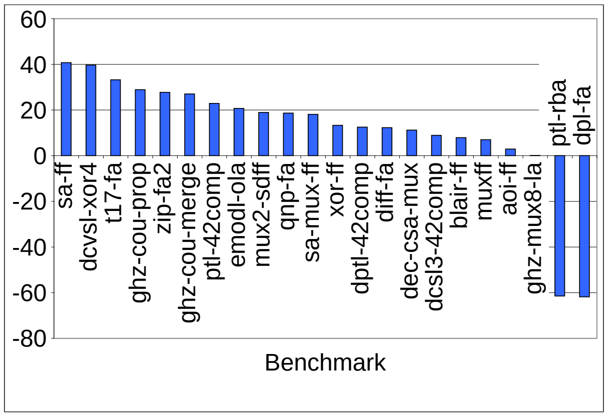


Figure 9: TEMPO vs. LAS percentage area improvement

tem which assumes the standard-cell style template. To minimize the overhead of the metal-1 power rails in *LAS* we forced these to have minimum width. *LAS* was run in full optimization mode to explore all possible transistor foldings and cell aspect ratios.

The improvement in benchmark cell area obtained with *TEMPO* is displayed in Figure 9. The results are encouraging: with two notable exceptions, the *TEMPO* layouts were consistently smaller than the *LAS* layouts. In 15 of the 22 circuits this improvement exceeded 10% and in 8 of the 22 the improvement exceeded 20%. Two representative layouts are shown in Figure 10. The upper cell is the dcvsl-xor4 benchmark which provides a visually appealing example of a 2D transistor arrangement with fairly sparse routing. This is in contrast to the row-based transistor arrangement and channel routing used by *LAS*. An area improvement of 39.69% was obtained for this circuit. Our second example, the ptl-rba circuit, is one of the two cases for which *LAS* produced a superior layout. In this case *LAS* was able to obtain an extremely efficient linear transistor arrangement with little wasted space, while *TEMPO* could not achieve a competitive placement. In this case the *LAS* area is 61.45% smaller than *TEMPO*'s.

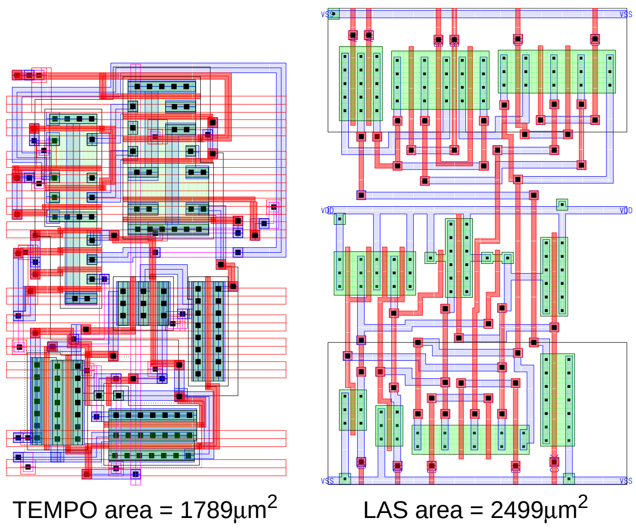
Because TEMPO makes use of a stochastic optimization algorithm it is likely that it will achieve a different result every time it is run. We were interested in observing the statistical spread in the final layout area obtained in this way. Our experimental results reported in Figure 9 use the smallest area obtained over 100 different trials. Figure 11 shows a histogram of the area spread observed for the sa-ff benchmark.

In Figure 12 we show some data indicating the run-time performance of *TEMPO*. Clearly, *TEMPO*'s simulated-annealing 2D placement algorithm, and the *Anagram-II* router's aggressive rip-up and re-route strategy, come at a substantial cost. This cost must also be multiplied by the number of trials conducted in the effort to obtain the best layout. Improving the predictability of the placement results to eliminate this high cost will be a significant topic of future work.

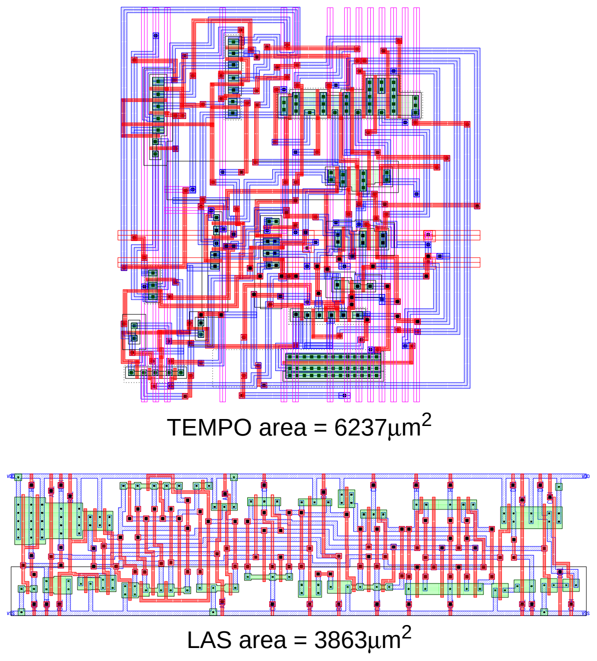
6. Conclusions

In this paper we have outlined a new problem in the field of cell level digital layout synthesis: 2D cell synthesis for high-performance non-dual digital VLSI cells. We have discussed a new methodology to address this problem based on an unconstrained 2D micro-placement and routing framework. Our philosophy emphasizes methods for modeling and optimizing geometry sharing among the transistors. One novel aspect of our approach is the late-binding of transistors to chains, allowing the chains to be dynamically adjusted during the placement step. This approach enables chain optimization to proceed with detailed knowledge of the placement and global routing cost function.

We have developed a prototype implementation of a transistor-level micro-placement tool based on these ideas, and assembled a complete end-to-end synthesis flow that makes use of third-party tools for detailed routing and post-routing compaction. An initial set of experiments comparing our system to Cadence LAS demon-



(a) dcvsl-xor4 benchmark



(b) ptl-rba benchmark

Figure 10: Two representative cell layouts produced by TEMPO and LAS (shown approximately to scale.)

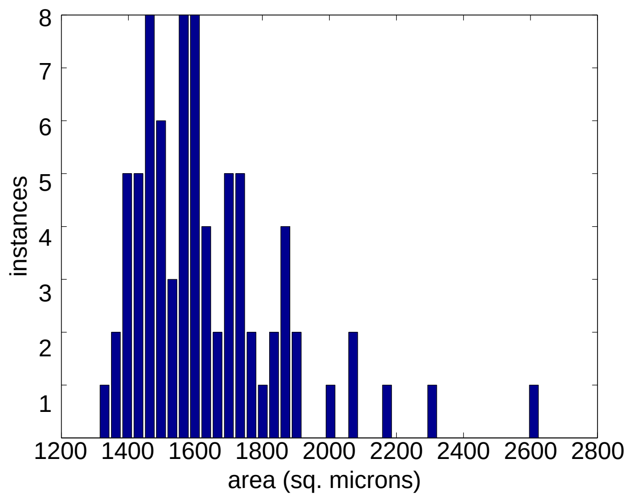


Figure 11: Layout area distribution for 100 experimental trials of the sa_ff benchmark

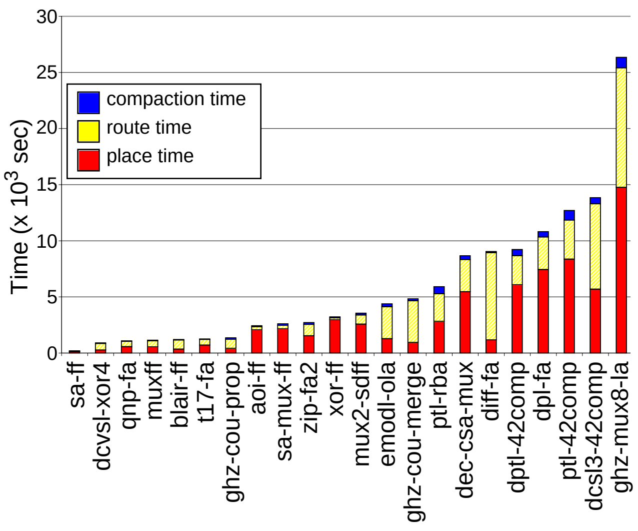


Figure 12: TEMPO run time statistics

strates encouraging results. However, the lack of predictability in the results produced by stochastic optimization result in a high computational cost that needs to be remedied in the future.

We are also pursuing further work in a number of additional areas: parasitic cost extraction, transistor folding, symmetric and partially symmetric chaining, well and well contact insertion, accurate routing area insertion, and congestion-aware routing cost estimation.

Acknowledgment

This research was supported by DARPA under contract DAAH04-94-G-0327.

References

[1] 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.

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

[3] J. Burns and J. Feldman, "C5M—A Control Logic Layout Synthesis System for High-Performance Microprocessors," *IEEE Trans. on CAD*, 17(1), Jan. 1998, pp. 14–23.

[4] S. Chow, H. Chang, J. Lam, and Y. Liao, "The Layout Synthesizer: An Automatic Block Generation System," in proc. *1992 CICC*., pp. 11.1.1–11.1.4.

[5] J. Cohn, "Automatic Device Placement for Analog Cells in KOAN," Ph.D. Dissertation, Carnegie Mellon University, CMUCAD-92-07, 1992.

[6] J. Cohn, D. Garrod, R. Rutenbar, and L. R. Carley, "Analog Device-Level Layout Automation", Kluwer Academic Publishers, Boston MA., 1994.

[7] C. Fiduccia, R. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," in proc. *19th DAC*, 1982, pp. 241–247.

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

[9] 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.

[10] 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.

[11] http://andante.eecs.umich.edu/tempo/ispd99_bench

[12] 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.

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

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

[15] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani, "Rectangle Packing Based Module Placement," in proc. 1995 ICCAD, pp. 472–479.

[16] C. Papadimitriou, K. Steiglitz, "Combinatorial Optimization Algorithms and Complexity," Prentice-Hall, 1982, p. 413.

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

[18] M. Riepe, "Transistor-Level Micro-Placement and Routing for Two-Dimensional Digital VLSI Cell Synthesis," Ph.D Dissertation, The University of Michigan, 1999.

[19] 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.

[20] 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.

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

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

[23] 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.