KOAN/ANAGRAM II: New Tools for Device-Level Analog Placement and Routing

John M. Cohn, *Member, IEEE*, David J. Garrod, *Student Member, IEEE*, Rob A. Rutenbar, *Senior Member, IEEE*, and L. Richard Carley, *Senior Member, IEEE*

Abstract—**This paper describes KOAN and ANAGRAM II, new tools for device-level analog placement and routing. A block place-and-route style from macrocell digital IC's has recently emerged as a viable methodology for the automatic layout of custom analog cells. In this *macrocell* style, parameterized module generators produce geometry for individual devices, a placer arranges these devices, and a router embeds the wiring. However, analog layout tools that merely apply known digital macrocell techniques fall far short of achieving the density and performance of handcrafted analog cells. KOAN and ANAGRAM II differ from existing approaches by employing general algorithmic techniques to find critical device-level layout optimizations rather than relying on a large library of fixed-topology module generators. New placement algorithms implemented in KOAN handle complex layout symmetries, dynamic merging and abutment of individual devices, and flexible generation of wells and bulk contacts. New routing algorithms implemented in ANAGRAM II handle arbitrary gridless design rules in addition to over-the-device, crosstalk avoiding, mirror-symmetric, and self-symmetric wiring. Examples of CMOS and BiCMOS analog cell layouts produced by these tools are presented.**

I. Introduction

Two recent trends have exposed custom analog layout as a bottleneck in the path from specifications to silicon for analog cells. The first trend is the growing importance of mixed-signal ASIC's. Although an increasing fraction of ASIC designs requires integration of analog and digital components, the analog standard cell libraries used to implement them often cannot supply all the analog cells necessary for a given design. The second trend is the emergence of cell-level analog circuit synthesis tools [1]–[4] which can quickly transform circuit specifications into sized schematics for some important classes of cells. In both of these design scenarios, custom analog cell layout is required, which can be a time-consuming, critical bottleneck.

The problem of custom analog layout has generated considerable interest in the last few years. The critical problems involve handling analog-specific constraints that render the layout much more sensitive to low-level geometric choices than digital cells of similar size. Work to date on analog layout has included knowledge-based approaches [4]–[6], algorithmic techniques for placement [7]–[11], routing [7]–[13], compaction [7], [14], procedural device and module generation [3], [9], [15]–[17], and performance constraint generation [18], [19]. Our interest is algorithmic placement and routing for custom analog cells. A block place-and-route approach, which we refer to as the macrocell layout style [3], [7], [8], has emerged as a popular candidate for recent analog cell layout tools: from a netlist, critical primitives (such as matched devices or complex folded structures) are produced by parameterized module generators; these primitive blocks are placed, and then routed. This style is derived from techniques for layout of digital IC's. Unsurprisingly, many existing analog cell layout tools borrow heavily from the digital macrocell style, adapting digital layout ideas to analog problems.

Our central argument is that certain attributes of this digital layout style limit its ability to achieve high-quality analog cell layouts. Some common assumptions that help manage the complexity of large digital layouts, e.g., a slicing-style placement [20], the restriction that signal routing be confined to channels between placed objects, and the exclusive emphasis on minimizing wire length and area, are not essential for attacking analog device-level layout. In our experience, these assumptions actually interfere with the type of low-level optimizations common in manual analog cell layout. For example, much of the creativity displayed by analog layout experts involves shaping, folding, placing, and merging individual devices to achieve dense layouts. In such high-quality layouts, many connections are achieved by abutment rather than explicit wires, and some fraction of the remaining wires is routed directly over devices. These optimizations reduce not only layout area, but more importantly, the device parasitics themselves. In current analog macrocell systems, such optimizations appear inside procedurally generated subcircuits, but not between the modules involved in placement and wiring. Indeed, these systems usually require a large library of device generator programs, each implementing some common arrangement of basic devices, to achieve even moderately dense layouts. None of the analog layout systems of which we are aware can support the more free-form style of device layout characteristic of expert designs.

This paper presents an alternative macrocell layout style that permits more of the low-level layout optimizations described above. Specifically, we have designed new device placement and routing algorithms to support the following

Manuscript received August 1, 1990; revised November 9, 1990. This work was supported in part by the National Science Foundation under Grants MIP-8657369 and MIP-8451496, by the Semiconductor Research Corporation under Contract 90-DC-068, and by Harris Corporation. J. M. Cohn was supported by the IBM Resident Study Program.

The authors are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213.

IEEE Log Number 9041735.

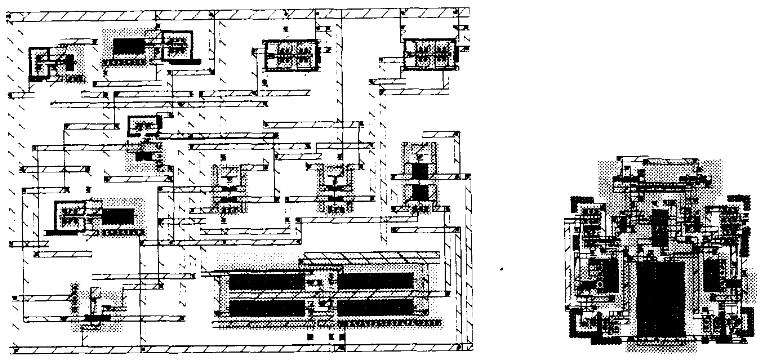


Fig. 1. Comparison of (a) ANAGRAM I comparator layout versus (b) KOAN/ANAGRAM II layout.

for analog cells:

- *Layout symmetries*: we handle both symmetric placement and symmetric detailed routing for differential circuits, including those with nonsymmetric components. This supports, for example, the layout requirements for a symmetric differential signal path and its associated nonsymmetric biasing circuitry.
- *Device merging / abutment*: we permit individual devices to be merged and abutted during placement. This not only increases cell performance, because of reduced parasitics, but also increases layout density, because the placer can now arrange devices into complex merged structures that are unlikely to be present in a library of module generators.
- *Well merging and bulk contacts*: we generate merged wells, and well and substrate contacts. This is required since the placed no longer assumes that these structures are completely fixed at the time of device generation.
- *Over-the-device wiring*: we do not restrict signal wires to be routed in channels between placed devices; instead, we allow wires to traverse devices at designer discretion. The router can handle arbitrary design rules on wires, use portions of placed devices as wiring, and does not require electrical terminals to be restricted to the perimeter of devices.
- *Crosstalk avoidance*: we model elementary capacitive coupling between signal nets, including simple shielding effects, and use these in the router to coerce embedded wires to avoid potentially damaging crossings or adjacencies.
- *Integrated rip-up / reroute*: we allow the router to rip up existing paths to improve wiring in densely placed, highly merged/abutted cells. This significantly increases the router's reliability by nearly eliminating its sensitivity to the order in which nets are routed.

We arrived at this set of layout requirements based on fabrication experiences with high-performance CMOS cells designed using our first-generation analog layout system, ANAGRAM I [8], [21]. These features are implemented in a new placement tool called KOAN, and a new router called ANAGRAM II. The layout results in Fig. 1 illustrate most clearly the differences between these first- and second-generation tools. Shown are two cell layouts for the same CMOS comparator design, one done by ANAGRAM I, which we regard as fairly typical of first-generation macrocell-style tools, and the other done by KOAN and ANAGRAM II. The layout generated by our new place-and-route tools is approximately one third the size of the earlier layout.

The remainder of the paper describes the architecture of these new layout tools. Section II describes the new device generation and placement algorithms used in KOAN. Section III next describes the new routing and path optimization algorithms used in ANAGRAM II. Some program implementation details for these tools are given in Section IV. Example results, including automatically generated CMOS and BiCMOS cells, appear in Section V. Finally, Section VI offers some concluding remarks.

II. Analog Device Placement in KOAN

A. Basic Architecture

KOAN is a device-level analog placement tool that supports symmetric placement, device merging, and abutment routing. KOAN consists of a set of procedural device generators, a device placer, and a well/substrate generator. In this, it architecturally resembles other analog macrocell layout systems [3], [4], [7]–[9]. It differs from these tools, however, in its ability to selectively overlap modules to reduce parasitic capacitance and cell area by appropriate sharing of geometry. Our strategy contrasts with other schemes in which such geometry sharing can only occur within the boundaries of procedurally-generated modules. This allows KOAN to exploit a potentially richer variety of geometry sharing alternatives than the few static topologies embodied in a typical procedural module generator library.

This strategy also strongly affects the design of the placement algorithms in KOAN: details about design rules, electrical connectivity, parasitic minimization, wells, etc., must be dealt with during placement, since they are not fixed during procedural device generation. Our placement strategy, like some device placers [7], [8], is based on simulated annealing. However, the annealing formulation is considerably more complex, given the new density and electrical performance optimizations we require it to support. KOAN employs a post-placement well generation strategy based on elementary computational geometry methods. These are described below.

B. Procedural Device Generation

KOAN has a library of procedural device generators, distinguished mainly by its small size. The library currently consists of two generators, one for folded FET's and the other for nonprecision capacitors. In an attempt to match the performance of hand crafted layouts, the common subcircuit layout structures that make up the bulk of typical generator libraries, e.g., cascode structures, matched differential pairs, and so forth, are created, *during* placement, by combining primitive devices. Since device generators are tedious to construct and maintain, we hope this approach will reduce the size of generator libraries, and encourage designers to implement new generators only for a smaller set of special-purpose structures unlikely to be found algorithmically, e.g., interdigitated cascode structures [7].

For each given set of electrical requirements, our generators produce several layout variants. For example, MOS devices are generated with a varying number of folds, varying contact locations, etc. The placer then chooses the variant that best fits the geometric and electrical requirements of the evolving layout.

Allowing the placer to merge and abut individual devices has several consequences on the design of device generators. First, the resultant device layouts are no longer *opaque* to the placer; they cannot be represented as simply a rectangle with terminals on its perimeter. Our generators supply to the placer detailed geometry for all electrical terminals that may subsequently participate in merge/abutment decisions. These include, for example, MOS drain, source, gate, and well contact geometry. Another consequence is that device generators cannot generate fully specified wells. If these are fixed at generation time, they adversely limit the flexibility of the placer to merge and abut devices. Hence, we generate those features that do not compromise the placer, for example, abutting substrate contacts on well-tied or rail-connected devices. The wells themselves are handled separately, in a post-processing phase after device placement or routing as described in subsection D.

C. Placement by Annealing

KOAN borrows from ANAGRAM I the idea of using a flat, nonslicing annealing model [22], [23] for device placement. Alternative approaches [3], [7] that adopt a slicing constraint have employed existing algorithms to handle block placement [24] and block shape (generated device variant) selection [25]. However, the slicing assumption is undesirable in our application for two reasons. First, it limits the set of reachable layout topologies, an effect most easily seen when the layout consists of many small objects, or objects with a wide variation in size [8]. Second, and more importantly, it prohibits us from reaching layouts where devices overlap in desirable ways. In the flat annealing style, intermediate states of the evolving layout can have arbitrary overlaps among movable devices. We use this as the mechanism to discover arbitrarily complex merged structures. One consequence of the flat style is that our placer produces an absolute, and not a relative (topological) placement. Hence, the placer is responsible for ensuring that there is sufficient wiring space for the router.

Simulated annealing [26] is a general optimization strategy based on iterative improvement with controlled hill climbing. This hill climbing allows annealing strategies to avoid many local minima in a complex cost surface, and reach better global solutions. To characterize KOAN's device placement strategy, we need to describe the four components of any annealing-based optimizer: 1) the *representation* for intermediate states of the layout visited during iterative improvement; 2) the set of allowable *moves* that transform one intermediate state of the layout to the next; 3) the *cost function* used to evaluate the quality of each intermediate layout; and 4) the *cooling schedule* used to control hill climbing.

Because it does not assume a slicing structure, KOAN uses a simple representation for evolving layouts. KOAN manipulates a set of rectilinear objects moving among arbitrary locations in a two-dimensional plane. Placeable objects can overlap in arbitrary ways as annealing proceeds. A bin hashing scheme is used to improve the performance of the overlap calculation as in [23].

One trade-off in any annealing-based layout algorithm is whether the responsibility for "good" layouts is embedded primarily in the choice of moves, called the *moveset*, or in the choice of cost function. KOAN relies mostly on its cost function to find good analog layouts. We do not employ moves that specifically seek to merge devices, to abut them, to share well contacts, etc. In our preliminary experiments, move sets that emphasized such specialized moves were consistently inferior to schemes that favored more random moves and a sophisticated cost function. The one important exception is that device symmetry and matching are supported directly in the move set. KOAN supports three classes of moves: *relocation* moves, *reshaping* moves, and *group* moves.

Relocation moves can translate, rotate, mirror, or swap devices. Three classes of analog constraints are maintained during all relocation moves.

1) Symmetry constraints reduce the effect of parasitic mismatch in differential circuits. Devices with symmetry constraints are always relocated to new symmetric positions. Single devices with symmetry constraints must slide along a symmetry line that bisects the evolving placement. Pairs of devices with symmetry constraints are always relocated in mirrored positions about this symmetry line. Such constraints have been handled elsewhere by adopting a slicing style [7]. However, because of its flat annealing formulation, arbitrary symmetries on arbitrary devices are especially easy to handle in KOAN.

2) Matching constraints force a common gate orientation (and overall device shape; see the discussion on reshaping moves below) on different devices. These help to reduce the effect of processing-induced mismatches.

3) Topological constraints allow the circuit designer to fix some aspects of the placement, while the placer handles the remainder. For example, individual device locations can be fixed in either one or both dimensions, or constrained to one of the edges of the layout or to its symmetry line. Likewise, cell terminals can be fixed in location, allowed to slide along one of the layout edges, or constrained to be symmetric to allow the resulting layouts to be abutment routed when placed side to side.

Reshaping moves replace one procedurally generated variant of a device with a different variant. This is how malleable devices are refolded, terminal contacts realigned, and so forth. To our knowledge, these sorts of alterations have only been used to date for cell area minimization [3], [7], [25]. In contrast, we allow the iterative improvement process to select any variant, not just the area-minimizing variant, because such perturbations may improve many different aspects of the layout. For example, a reshaped device might better merge with another device, thus reducing a critical nodal capacitance. As with the relocation moves, symmetry and matching are also enforced. If one device in a matched or symmetric group is reshaped, its companion devices are similarly reshaped.

Group moves extend the idea of moves for individual devices to moves for complex merged structures incorporating an arbitrary number of devices. Because we seek to encourage formation of dense merged/abutted device groups, it is essential that merged objects, once formed, be free to participate as a single unit in the same placement transformations as individual devices. It is also important to be able to dissolve such structures, by relocating one or more constituent devices sufficiently far to prevent interaction. The rationale here is the need to explore many different possible group mergings to prevent suboptimal groups from freezing too early in the annealing process. Group moves and single-device moves are handled uniformly. The process can be summarized as follows:

1) Randomly select a placed device D_i. Check to see if D_i is part of a larger group structure. Any set of devices $G = \{D_1, D_2, \cdots, D_i, \cdots, D_k\}$ that can be reached by traversing merged or abutted geometry constitutes such a group.
2) Randomly select one of the possible relocation moves {*translate*, *rotate*, *mirror*, *swap*}, or the reshaping move.
3) If reshaping was selected, replace device D_i with one of its generated variants. If D_i was part of a group, make a random binary decision either to a) align the new device variant so that it preserves the same overlap with its previously merged neighbors, or b) make no local adjustment.
4) If instead a relocation move was selected, randomly choose a subset of devices in the group, $M \subseteq G$. All devices in M participate in this move. Note that M might contain only device D_i, it might contain several devices, or it might be all of group G.
5) Apply the selected relocation to all devices in M. Note, if a *swap* was selected, identify another device E_i, and repeat step 4 to find a target set of devices with which to swap M. Maintain symmetry/matching if any members of the group have these constraints.

Step 3 is the one exception to the general rule that moves do not target specific optimizations. Since reshaping a device in a merged structure is highly disruptive, we have found that occasional attempts to properly align the new device in its local environment are more likely to cause beneficial shape changes to be accepted. This allows variants with similar contact placement to be interchanged without disturbing existing merges.

Much of the sophistication required to reach dense highly merged placements is embodied in KOAN's cost function, which is given by the following weighted sum:

$$cost = w_0 Overlap + w_1 Area + w_2 AspectRatio + w_3 NetLength + w_4 Proximity + w_5 Merge$$

where the w_i are experimentally chosen weights. The annealing process searches among different layout configurations to minimize this cost function. Some of these terms are familiar from digital macrocell placers. However, other terms are new, and all have been reformulated to handle analog-specific concerns.

The *Overlap* term penalizes illegal overlaps (measured as overlap areas) among devices. Recall that the placer has access to detailed geometry in each movable device. Part of each generated device is a protection frame that determines how closely distinct devices can be placed. The protection frame for each generated device accounts for the design-rule distances that must be maintained around the perimeter of each device. Space for wires to be embedded is also maintained, using a simple variant of the adaptive halos mechanism from [23]. Wiring space is allocated around each device depending on its size and number of wiring terminals. As devices are moved during annealing, an illegal overlap occurs if protection frames overlap, i.e., two pieces of electrically distinct geometry overlap, or are closer than design rules or wire space estimates allow. This term of the cost function must be driven to zero to ultimately produce a feasible layout. The *Area* term penalizes the total bounding box area of the placement. The *AspectRatio* term penalizes deviation from the desired aspect ratio R, and has the form $(R_{current} - R_{desired})^2$.

The *NetLength* term is the familiar sum of estimated lengths for each net. However, it is critical to use the right length estimator. Note that the detailed geometry representing terminals of nets to be wired may be large relative to individual devices, or the overall cell itself. Hence, estimators such as the half-perimeter of the least bounding rectangle of these terminals, or the length of a minimum rectilinear spanning tree connecting the centers of these terminals, can be extremely inaccurate. Instead, we construct the minimum rectilinear spanning tree that touches any piece of geometry in each electrically distinct terminal, and use its length as our estimator. This estimator also has the essential property that when the placement process merges the geometry of previously distinct terminals, the predicted net length to connect those terminals is zero.

The *Proximity* term allows designers to improve matching by encouraging arbitrary (possibly unconnected) devices to cluster. Devices in such proximity groups are modeled as having a dummy net connecting their respective centers. The *Proximity* term is the weighted sum of these dummy net lengths.

The *Merge* term is perhaps the most interesting part of the cost function. It can be regarded as the complement of the *Overlap* term: whereas *Overlap* penalizes overlaps that cause electrical violations, *Merge* rewards overlaps that are electrically beneficial. KOAN supports a flexible model of what geometry can be merged: roughly speaking, if two pieces of geometry on different devices are electrically common, and on *compatible* layers, they can be overlapped as far as the other prevailing design rules allow. Note that the notion of compatible layers is technology specific, and must be specified in the technology file. Fig. 2 shows examples of possible geometry sharing for a typical BiCMOS process.

The *Merge* term is formulated as

$$Merge = C_{area}(TotalMergeableArea - TotalMergedArea) + C_{perim}(TotalMergeablePerimeter - TotalMergedPerimeter).$$

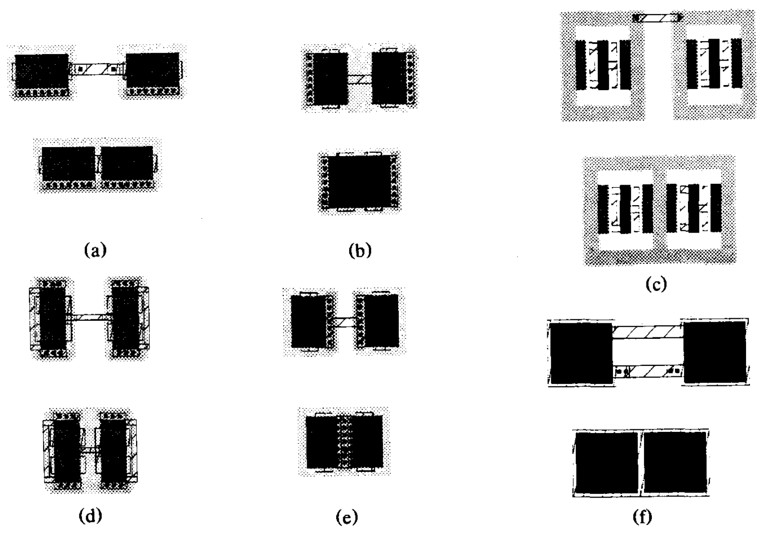


Fig. 2. Geometry sharing supported for typical BiCMOS process. (a) MOS gate abutment. (b) MOS diffusion sharing. (c) BJT guard-ring merging. (d) MOS metal abutment routing. (e) MOS substrate contact sharing. (f) Capacitor contact sharing.

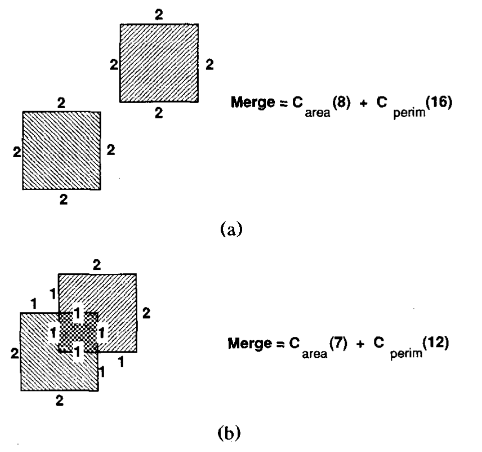
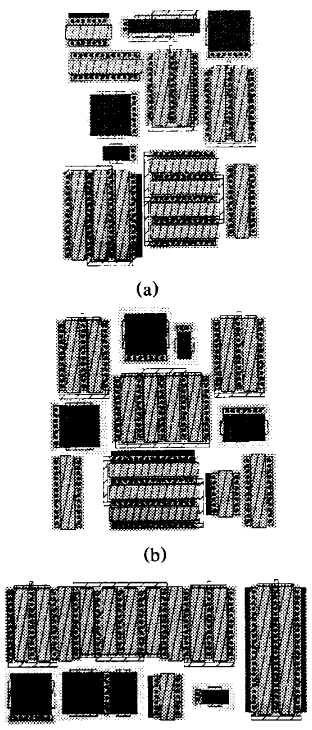

2
2
2
2
2
2
2
2
Merge = C area(8) + C perim(16)
(a)
2
1
1
1
1
1
2
2
1
1
1
2
Merge = C area(7) + C perim(12)
(b)

Fig. 3. Merge metric calculation. (a) Unmerged geometry, with more area and larger perimeter. (b) Merged geometry, with smaller area and perimeter.

Overlaps and abutments among compatible and electrically connected pieces of geometry do not count toward the *Overlap* penalty, since they are not electrically illegal. The total area of these mergable electrical terminals is the constant *TotalMergeableArea*. Similarly, the total perimeter of these terminals is the constant *TotalMergeablePerimeter*. *TotalMergedArea* and *TotalMergedPerimeter*, in contrast, change as devices are relocated to share differing amounts of geometry. Note that in the presence of no merges or abutments, *TotalMergedArea* and *TotalMergedPerimeter* are each zero, hence the *Merge* term is simply a constant overhead on the cost function. However, when merges occur, the terms measuring merged area/perimeter increase, decreasing the *Merge* penalty, and the overall cost of the layout. This is illustrated in Fig. 3. The overall effect of minimizing the *Merge* term is to maximize the sharing of geometry by maximizing allowable overlaps and shared perimeter. Since parasitic capacitance is directly proportional to such overlaps and perimeters, this term has the direct effect of minimizing such parasitics, when the per-unit-area (C_{area}) and per-unit-perimeter (C_{Perim}) capacitance scaling factors are set appropriately.

The cooling schedule is the policy that controls hill climbing during annealing. The placement must be "heated" to a high temperature to allow many random placements to evolve, then carefully "cooled" to allow the desired structure of the placement to freeze out. In the hot regime of annealing, moves that substantially increase the cost function are tolerated (again, to jump out of the local minima), but as the placement is cooled, fewer disruptive moves are permitted. There has been considerable progress of late in automating the decisions involved during cooling. We employ the automatic schedule from [27]. It is worthwhile to note that the individual terms in the overall cost function each dominate a different phase of the cooling process. In other words, the cost function itself is fixed, but the terms comprising it tend to freeze at different temperatures. Early in the annealing, the *Overlap* term forces illegal random overlaps to disappear. Later, the *NetLength* and *Proximity* terms encourage a good relative arrangement of devices. As the placement cools further, the *Merge* term starts to coerce desirable sharing of geometry, and finally the *Area* term causes the overall layout to shrink as much as possible.

Fig. 4 shows how all the new features of KOAN come into play. Three KOAN layouts are shown for a differential op amp with 11 devices and 12 nets. Fig. 4(a) shows a placement generated without symmetry, and with merging disabled. Although fairly dense, the nonsymmetric placement of the input-stage devices would likely increase the offset voltage. Fig. 4(b) shows a placement with symmetry, but again no merging. Notice, however, that not all components had symmetry constraints. This is not much smaller than the previous layout, but essential symmetries are now enforced for those devices that require it. This result is typical of other



(a)

(b)

(c)

Fig. 4. Impacts of symmetry, device merging in a simple op amp. (a) Symmetry and device merge/abutment disabled. (b) Symmetry enabled, device merge/abutment disabled. (c) Both symmetry and device merge/abutment enabled.

macrocell systems [8]. Fig. 4(c) shows a placement generated when both symmetry and merging are enabled. This layout is appreciably smaller (about 10%) and also much more typical of a handcrafted placement. In particular, the dense area at the top left is a symmetric, folded, five-way merged group of devices, which is a very compact solution. We believe it is unlikely that such a complex structure would be available as a single parameterized cell in a typical macrocell library. Nevertheless, KOAN was able to discover and optimize this structure automatically.

D. Well Generation

Correct handling of wells and associated geometry presents problems during both device generation and device placement. In most analog placement systems, wells and associated geometry are created when devices are generated. For example, in ANAGRAM I, each device was created with an appropriate bulk contact, well, and guard ring. In addition to being overly conservative, this approach wasted space, created extra bias routing, and precluded dynamic device merging. Hence, we have adopted an alternative approach in which most well geometry is generated after devices have been placed or routed. An exception to this is well- and bulk-connected devices. By examining the input netlist during device generation, we detect when the source of an FET

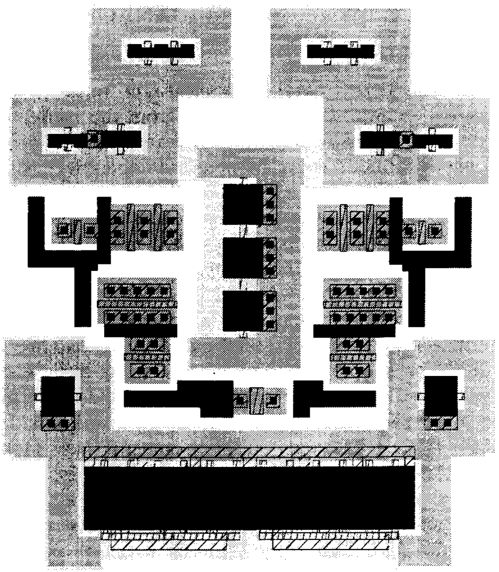


Fig. 5. Symmetric comparator with generated wells and bulk diffusions.

is connected to the same potential as its bulk. For these devices, we generate appropriate bulk contacts and abut them to the device's source. In this way, no additional routing is required for bulk biasing. We then allow bulk contacts to participate in contact merging during placement. This allows bulk contacts on different devices to merge into one contact shared by the merged devices. While merging bulk contacts does not present a significant parasitic advantage, it does improve layout density and simplify device wiring. Note also that while we do not force a segregation of n- and p-channel devices during placement (as is done in some other layout systems [3]), this type of segregation tends to occur automatically because of the device connectivity.

Well generation proceeds as a series of simple computational geometry steps such as shrinks, expands, unions, and intersections on the appropriate layers. For example, wells are produced approximately as follows. First, we expand the geometry of the well-bound devices (e.g., the n devices in a p-well process), find the union of these expanded shapes, and intersect this unioned region with an expansion of the geometry of the devices outside this type of well. The complement of this intersection defines all well regions; wells merge wherever possible in this scheme, which is essentially that used in [28]. Any well regions that remain floating after this shapes processing are contacted during routing by ANAGRAM II. To reduce the series resistance from the bulk contacts to other parts of the well and substrate, similar sequences of steps fill all unused space with the appropriate low-resistance diffusion straps. It is worth noting that although wells themselves can be generated before or after routing, we have found it beneficial to generate these low-resistance diffusion straps after routing, since they can interfere with wire paths that must be routed in polysilicon. Fig. 5 shows a placement for a high-speed CMOS comparator, along with its automatically generated wells and substrate contacts.

III. Analog Wire Routing in ANAGRAM II

A. Basic Architecture

ANAGRAM II is a detailed general-area router for analog cells. It borrows two critical ideas from the router in its predecessor, ANAGRAM I: a general-area routing strategy instead of a channel-routing model, and capacitive crosstalk penalty functions to encourage intelligent path decisions about net crossings and adjacencies. Although the sparser placements produced by first-generation layout tools could accommodate a channel-routing style (e.g., consider Fig. 1), the dense, highly merged/abutted placements produced by KOAN completely preclude such a stylized approach. Our area-routing strategy incorporates models of capacitive coupling, including simple shielding effects, in its basic evaluation mechanism for paths, allowing path selections to be coerced by possible interactions with other wired nets.

ANAGRAM II also has supports several new features missing in ANAGRAM I, and the other analog area routers of which we are aware. The first is the use of a tile-plane representation [29] for the to-be-routed layout, instead of a simple coarse grid. This representation supports essentially arbitrary wiring rules, in particular, over-the-device wiring. Second, new algorithms have been devised for line-expansion wire routing in this framework. The major contribution here is an algorithm for embedding geometrically matched, crosstalk-avoiding symmetric paths for differential signals. Third, a new integrated ripup/rerouting strategy has been designed. During the search for individual segments of wiring paths, the router can choose at any time to remove an existing wire. For the dense, highly merged/abutted placements produced by KOAN (or manually by layout experts), this integrated rip-up and rerouting turns out to be essential: without the ability to remove embedded nets on demand, it is often the case that there is no way to embed the next net. We describe below the tile-based representation used in ANAGRAM II, the new line-expansion algorithms for wire embedding in this framework, and the integrated rip-up/rerouting strategy.

B. Tile Representation

ANAGRAM II represents all placed devices, free wiring space, and embedded wires in a single-tile-plane data structure [29]. The tile representation frees us from some of ANAGRAM I's unnatural grid-based limitations on the location of devices and their terminals, and on the width and pitch of individual wires. Because there is no difference between unused space, wires, and device geometry, the router can "see" the internal details of devices. This has several advantages: over-the-device wiring incurs no overhead; pieces of devices can now themselves be used for wiring paths; electrical terminals can appear as arbitrary collections of geometry; and the same crosstalk penalties that accrue to wire segments can be applied to pieces of placed devices. A layout representation that supports careful over-the-device wiring is essential for routing dense placements. For example, terminals in some KOAN-generated placements can appear inside complex merged structures, and simply cannot be reached without extending a wire over some device geometry.

We use a single tile plane in which each individual tile represents a unique combination of mask layers. This facilitates many operations required by the ANAGRAM II's routing algorithm, most notably the enumeration of all geometric objects within a small region to check for design-rule and crosstalk violations. During initialization, ANAGRAM II constructs its tile-plane representation from an input device placement. It then determines the precise location of all routing terminals. Tile planes, which naturally support connectivity propagation, prove useful here by allowing the designer to tag only a small piece of a terminal and rely on ANAGRAM II to maximally expand the terminal to all connected geometry. During actual wire routing, the tile plane is employed primarily as a database, i.e., ANAGRAM II continually queries the tile plane for information about geometric objects within small regions. During routing, new wires are embedded and existing wires deleted from the tile plane as needed.

C. Line-Expansion Routing

ANAGRAM II is a line-expansion router. For each net to be embedded, a set of partial paths is maintained, sorted by cost. The least-cost path is selected, and expanded by probing incrementally outward from the head of the path. Each such probe results in a new partial path with a new cost, which is inserted back into the data structure used to sort the paths, in our case a *heap* [30]. The cycle of selecting, expanding, evaluating, and saving partial paths continues until all electrical terminals of the net have been connected, and no partial path can be expanded to make the same set of connections with less cost. The two critical components of the ANAGRAM II router are its expansion strategy and its path cost function.

Our expansion scheme relies heavily on the underlying tile-plane representation. ANAGRAM II always deals with exact wiring geometry, hence, each new path segment must conform to the prevailing design rules concerning width, pitch, interaction, via spacing, electrical interaction, etc. Again, the advantage of the tile plane is that it permits us to support essentially arbitrary wiring rules easily. However, the tile plane is used mainly for design-rule and electrical connectivity checking. Partial paths themselves are not embedded in the tile plane as they evolve (in contrast to [31]); they are simply stored in the heap. Only when a final path is found is the tile plane itself modified to embed the found path.

Another important algorithmic decision is the length of each probe attempted during path expansion. Long probes improve efficiency by allowing large distances to be traversed in a single step. However, long probes require much more complex computations to determine the optimal probe length, especially when an arbitrary number of crosstalk-interacting wiring layers is allowed. (An efficient scheme for whole-chip tile-based routing in this style is discussed in [32].) Short probes are less time efficient, but easily handle the small jogs, bends, etc., usually needed to reach dense terminals and avoid crosstalk problems. Our experiments with both styles favored short, unit-distance probes because of their simpler computational requirements, and the fact that successful probes in dense device placements tend to be short anyway. The following algorithm summarizes the expansion process for finding a path from a set of *source* terminals to one *target* terminal:

```
find_path(source, target)
{
  initialize heap to EMPTY;
  add source geometry to heap;/*begin with sources as
      partial paths*/
  /*main routing loop*/
  While (least-cost path in heap does not contact target) {
      expand(path);
  }
}
expand(partial_path)
}
  /*expand in all connected layers*/
  foreach (L in {connected, legal routing layers}){
    foreach (D in {each of 3 nonbackward probe
    directions}){
      p_new =partial_path + new geometry to expand it
              one unit in direction D on layer L;
      if (p_new is design rule correct){
        compute cost of p_new;
        add p_new to heap;
      }
    }
  }
  /*expand possible contacts*/
  foreach (C in {connected, legal contact layers}){
    p_new =partial_path + new geometry to add
              a contact to layer C;
    if (p_new is design rule correct){
      compute cost of p_new;
      add p_new to heap;
    }
  }
}
```

Multipoint nets are handled in the usual fashion by decomposing them into a minimum spanning tree of two-point nets. The full geometry of the current, partially routed multipoint net serves as the *source* for the expansion cycle to find the next two-point connection.

The cost of a partial path P has two components:

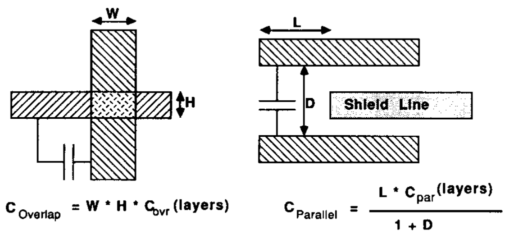
$$PathCost(P) = \sum_{segments\ s \in P} SegmentCosts(s) + Distance(s\ to\ target).$$

The Σ $SegmentCost(s)$ term sums the costs of P's constituent rectangular segments. The *Distance* term estimates distance from the head of the path P to the final target geometry, and is used to bias the expansion preferentially toward the target in the manner of conventional best-first search approaches [33]. After each expansion, the cost of the evolving partial path is updated by adding the cost of the newly expanded segment, and recomputing the *Distance* estimate.

Each *SegmentCost* has the form

$$Segment\_Cost = Wire + Direction + Crosstalk + Rip\text{-}up.$$

The *Wire* term is proportional to the area of the probed segment and the designer-specific layer cost. The effect is to favor short (low area) segments on preferred wiring layers, e.g., metal instead of polysilicon. The *Direction* term can be used to enforce preferred directions on individual wiring layers. Some routers use this to reduce congestion; our experience suggests this can be done more effectively with integrated rip-up and reroute. The *Crosstalk* term is unique to both ANAGRAM II and its predecessor, ANAGRAM I. By penalizing partial paths that contain undesirable crosstalk, we can force the router to search for less costly paths without undesirable net interactions. Both KOAN and ANAGRAM II allow the designer to specify arbitrary *compatibility classes* for electrical nodes. These classes specify which subsets of nets, when wired, may cross or be closely adjacent. For example, designers can define different classes of sensitive nodes (such as charge storage nodes) and noisy nodes (such as clocks) that should not interact. In addition, designers can specify which nodes can serve as potential shields for these unwanted interactions, for example, supply lines and some dc bias lines. This generalizes the simple noisy/sensitive/neutral classification scheme introduced by ANAGRAM I to handle arbitrary classes of net interactions. The crosstalk cost estimates the capacitance from a given probe segment to other *unshielded* interacting nodes. Simple capacitance models are used to penalize overlapping nets and non-shielded parallel runs, as shown in Fig. 6. The result of using this crosstalk cost during routing is that nets will attempt to take detours that avoid expensive crosstalk violations. The final path selected is the one which properly balances crosstalk cost and wiring cost. The *Rip-up* term is used during rip-up and reroute path optimization, and is described in the following section.


W
H
L
D
Shield Line
$C_{Overlap} = W * H * C_{ovr}(layers)$
$C_{Parallel} = \frac{L * C_{par}(layers)}{1 + D}$
Fig. 6. Crosstalk models for routing cost function.

The router also has the unique capability to find symmetric paths for differential signals with symmetrically placed terminals, even in the presence of arbitrary asymmetric blockages. Although we are aware of approaches that can support perfectly symmetric device placement, e.g., [7], we are unaware of any routers that can complete these placements with perfectly symmetric differential wiring. We model a differential net as two wires, each of which is exactly mirrored about an assumed symmetry line bisecting the layout. Embedding such nets is accomplished by routing one net of the symmetric pair using the same line-expansion algorithm described above, but constraining the search process to consider only partial paths that would also be legal if reflected across the symmetry line. A differential net is expanded twice during the probe process: once for the existing net's proposed probe, and once for the mirror reflection of the net and the probe on the opposite side of the symmetryline. Of course, the router must be sensitive not only to design-rule violations on the symmetric net's reflection, but to crosstalk violations as well. Thus, when routing a symmet-

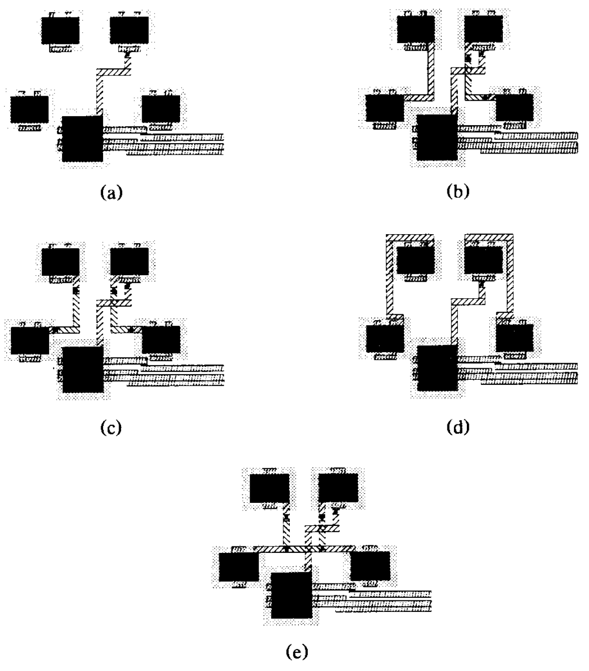


Fig. 7. Symmetric crosstalk-avoiding routes. (a) Four symmetric devices, with one asymmetric device and wire. (b) Routed with no symmetry, no crosstalk avoidance. (c) Routed with symmetry, but no crosstalk avoidance. (d) Routed with symmetry and crosstalk avoidance. (e) Routed as a single self-symmetric net, no crosstalk avoidance.

ric net, the crosstalk cost of a segment becomes the sum of the crosstalk cost of that segment and the crosstalk cost of its reflection across the symmetry line. Particularly good results can be achieved when all critically interacting nets are symmetric because any crosstalk violations that cannot be eliminated through detours in the routing will be identically matched on both sides of the signal path. Other approaches, such as [13], attempt to balance crossing of differential lines by adding equal-area crossings where necessary. However, we believe it is likely that superior matching occurs when the crossings/adjacencies are geometrically identical.

To further exploit the advantages of symmetric crossings. we have also added a new class of net called *self-symmetric*. A self-symmetric net is a single net whose pattern of pins is identical on both sides of the symmetry line. For example, this is frequently true of the clock nets in sampled-data circuits. With the knowledge that a net is self-symmetric, the router can then apply the symmetric routing algorithm to find a symmetric route for the individual halves and can connect the two halves in a symmetric manner by treating the symmetry line itself as a pin. Thus, the benefits of symmetric routing can be maintained for a single net whose function is identical on both sides of the symmetry line.

Fig. 7 shows how all the above features of ANAGRAM II come into play. A simple routing problem with a differential signal path (two pairs of symmetric terminals) is shown, along with an asymmetric noisy blocking wire. If no differential symmetry or crosstalk avoidance is enabled, the unbalanced paths in Fig. 7(b) result; note that one wire changes layers, while its companion does not. If differential symmetry is enabled, the matched paths in Fig. 7(c) are produced; both wires now make identical layer changes, one to avoid an obstacle, the other to match it to its companion. If both symmetry and crosstalk avoidance are enabled, the matched paths in Fig. 7(d) result; now both wires make a long detour to avoid the crosstalk violation that results if the rightmost wire crosses the noisy obstacle wire. Fig. 7(e) shows all four pins routed as a single self-symmetric net. Note how the two halves of the net are routed symmetrically and are connected at the center line.

D. Integrated Rerouting

Routers that route one net at time are often highly sensitive to the order in which nets are routed. For example, the currently evolving route cannot predict whether it is using space that will critically impact an unrouted net, nor can it determine that a small change in a previously embedded net might greatly help the current routing task. This is especially critical for analog routers that strive to avoid unwanted parasitic interactions among wired nets. Different net ordering schemes have been tried, e.g., routing short sensitive signals before clock nets, etc., but in general these techniques are highly unreliable, both in achieving 100% net completions, or perfect crosstalk avoidance. ANAGRAM II instead avoids this problem by relying on an aggressive iterative improvement strategy for routing. The router in ANAGRAM I routed all nets once, then randomly removed and rerouted nets until no further improvement was seen. In contrast, ANAGRAM II integrates net rip-up directly into the path-search mechanism used during line-expansion routing. The ability to remove any existing net as a new net is being routed proves to be essential for embedding wires in dense, KOAN-generated placements. In our experience, such placements are only wirable after a considerable amount of net rip-up and rerouting.

This is implemented as follows. Suppose an existing partial path P is being expanded. Suppose also that a probe with new wire segments s creates a design-rule violation with a previously embedded net N. Without any rip-up capability, we would simply reject segment s, since the new partial path $P + s$ is infeasible. With integrated rip-up, we allow the path $P + s$, but add to it a cost, *Rip-up*(N), associated with removing the obstacle net N. This is the *Rip-up* term in the *SegmentCost* for segment s. All partial paths that evolve from path $P + s$ are penalized by this amount, since they all include this segment s. Note that a partial path accrues a cost *Rip-up*(N_i) once for each net N_i it needs to remove. In particular, if some descendant of path $P + s$ also needs to use space occupied by net N, no additional rip-up penalty is assessed because it is assumed that the net has already been removed. The search otherwise proceeds exactly as previously described. During the search of individual wiring paths, a wide variety of different net rip-ups may be evaluated. However, only those associated with the final chosen path are actually removed. When a final path is determined, the nets that must be deleted to embed this path are removed from the tile plane, and scheduled for later rerouting. After their removal, the newly found path is embedded in the tile plane.

At any time, nets can be in one of two states: unrouted or routed. Unrouted nets are ordered in a queue. Nets are routed (or rerouted) in the order they are appear in the

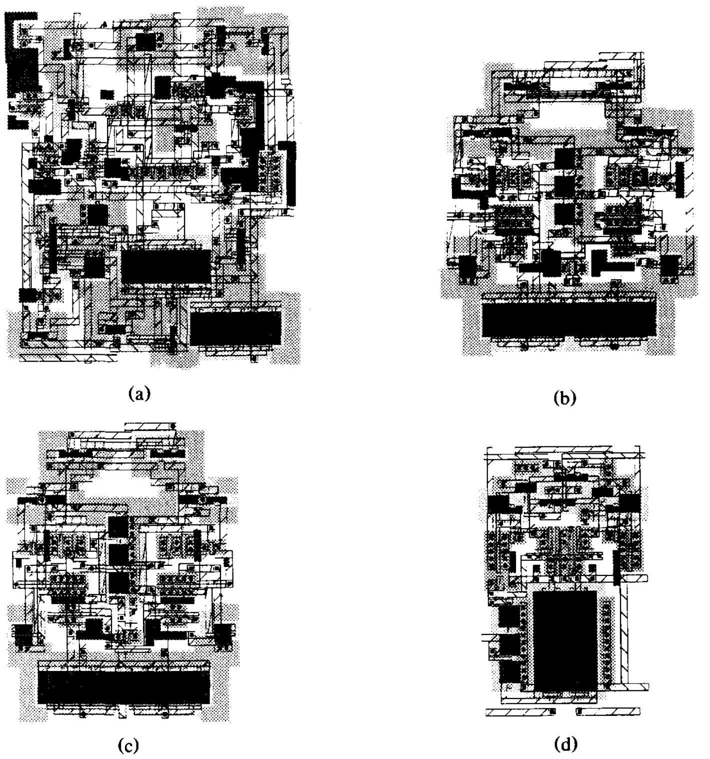

(a) (b) (c) (d)

Fig. 8. CMOS comparator layouts. (a) Automatic layout with no placement or routing optimization. (b) Automatic layout with placement optimization only. (c) Automatic layout with both placement and routing optimization. (d) Manual layout.

queue. Initially, the queue is filled with all nets to be routed, usually ordered with critical nets at the front of the queue. As nets are routed, the queue empties. As newly routed nets remove embedded nets, the ripped-up nets are reinserted in the queue. The *reroute policy* determines whether such nets are replaced at the head or tail of the queue. If an *early* policy is used, ripped nets are placed at the head of the queue, and thus immediatly rerouted. This has the effect of allowing a small set of interacting nets to negotiate their final placement, rerouting themselves in tight loop. If a *late* policy is used, ripped nets are placed at the end of the queue, and thus all other unrouted nets are routed first. This has the effect of allowing a removed net to reroute itself against the background of space and crosstalk constraints imposed by currently routed nets. Our experience favors the late scheme when many crosstalk interactions and terminal blockages are present in the placement, although with particularly critical nets it can be desirable to apply the early policy to these critical nets to ensure that they remain routed.

Because the routing/rerouting process might never terminate, the cost *Rip-up*(N) associated with removing net N cannot remain constant, but must increase as the routing cycle proceeds. This is handled by associating with each net N an aging factor $A(N)$. Each time net N is ripped up, its rip-up cost is multiplied by its aging factor $A(N)$. As net N ages—by being ripped up—its *Rip-up* cost increases, and it becomes more costly to remove. This suffices to guarantee eventual termination of the routing process, even when no final solution is found. In the beginning of the routing cycle, net rip-up costs are low, since nets embedded early in the routing process are more likely to cause congestion problems for future nets. In difficult routing problems a small set of critically interacting nets usually emerges and the route optimization process iterates among these nets for several cycles. It is during these critical negotiation cycles that the importance of individual net aging factors becomes apparent: the net with the larger aging factor emerges from this cycle with the favorable route, while the other net(s) must compromise.

Route blockages usually occur very near a net's source or target. ANAGRAM II's integrated rip-up/reroute scheme can quickly find a path out of a blocked source pin: after a small amount of search, the router is forced to accept a path which removes the net, despite its *Rip-up* cost. However, a net which blocks the target pin causes difficulties. Here, a *horizon effect* [33] problem forces the router to explore a nearly endless variety of slightly less desirable paths before choosing one that removes the relatively expensive blocking net(s); the router can easily waste a great deal of time or run out of memory. Fortunately, the designation of source and target is arbitrary. Thus, ANAGRAM II solves its horizon effect problem by reversing these assignments. Because this horizon effect often dominates search time, ANAGRAM II will actually make three attempts at routing a net. The first will terminate after a very small amount of search under the assumption that a horizon effect is impeding the search. The second attempt will reverse the terminals and search until

TABLE I
CMOS COMPARATOR PERFORMANCE COMPARISON
Speed is measured as decision time with overdrive corrected for offset.

Comparator Layout	KOAN Place Optimized	ANAGRAM II Route Optimized	Area	Speed (at 3 mV Overdrive)	Systematic Offset
Automatic (Fig. 8(a))	No	No	34 272 μm^2	25 ns	+3.5 mV
Automatic (Fig. 8(b))	Yes	No	24 768 μm^2	21 ns	+2.7 mV
Automatic (Fig. 8(c))	Yes	Yes	22 100 μm^2	20 ns	−220 μV
Manual (Fig. 8(d))	—	—	15 092μm^2	23 ns	−680μV

the maximum search depth is reached. If this fails, a final full depth search using the original terminal order is performed.

IV. Implementation Details

KOAN and ANAGRAM II together comprise approximately 20 000 lines of C code. The only input to the tools is 1) a common SPICE deck netlist [34] with annotations to control place/route options, and 2) a process-specific technology file. The SPICE netlist annotations are in the form of one-line comments which specify device and net symmetries, matchings, and sensitivities. The technology file is a text file containing line-by-line keyword/value specifications of layer-wise spacings, connectivities, extensions, merge compatibilities, etc., and is used commonly by both KOAN and ANAGRAM II. All communication of layout information between the tools is in MAGIC format [28], which allows designers to examine or modify intermediate layout results. Results presented in the following section were run on a DECstation 3100 under ULTRIX. Typical placement times for KOAN average 1 to 45 minutes of elapsed time, depending on the number of devices and amount of device merging optimization. Typical routing times for ANAGRAM II average 1 to 45 minutes of elapsed time, depending on the amount of crosstalk interaction to be managed, and the density of placed device terminals.

V. Layout Results

To demonstrate the effectiveness of KOAN and ANAGRAM II in custom analog cell layout, we present four sets of placed and routed layout examples produced by these tools. The first example is a high-performance CMOS comparator design. This circuit is particularly difficult to lay out, and was chosen to illustrate how the tools can optimize for electrical performance. The second and third examples are typical CMOS op-amp designs that illustrate how circuits with many large malleable devices can be aggressively optimized for density by our tools. The final example is a BiCMOS op amp, which illustrates how the tools can be easily retargeted to different technologies.

Example layouts for the comparator appear in Fig. 8. Table I summarizes these layouts and the results after parasitic extraction and simulation with HSPICE^{tm} [35]. The circuit is a high-speed regenerative comparator designed in MOSIS 2-μm p-well CMOS. The circuit has 26 devices and 21 nets, and is difficult to lay out because it has many small devices (not much smaller than wires themselves), a relatively large number of interconnections, and many potential

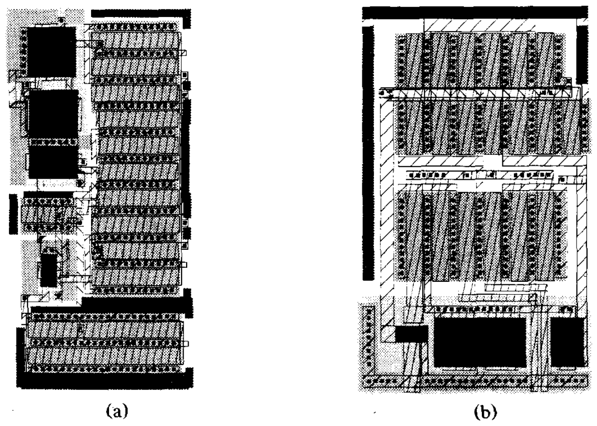


Fig. 9. Small CMOS op-amp layout. (a) Automatic layout with placement and routing optimization. (b) Manual layout.

crosstalk interactions between clocks and sensitive nodes. Fig. 8(a) shows a very poor automatic layout that makes no use of the optimization features in either KOAN or ANAGRAM II. No symmetry, merging, or abutment was encouraged during placement, and no symmetry or crosstalk avoidance was attempted during routing. As expected, the result is a slower comparator with a large systematic offset. Fig. 8(b) shows a better automatic layout. The device placement is highly optimized, because the placer was allowed to enforce symmetry and matching, and to merge/abut devices as necessary. However, the routing is as before—no symmetry, no crosstalk avoidance. The result is better: speed is improved considerably because of device merges, but there is still a large systematic offset due to asymmetric routing. Fig. 8(c) shows a fully optimized result. This is the same placement as Fig. 8(b), but with fully symmetric, crosstalk-avoiding routing. This result is the best in terms of speed, and has negligible systematic offset due to careful routing. Fig. 8(d) shows a comparable manual layout. The manual layout was aggressively optimized for density, and is 32% smaller than our best automatic layout. Nevertheless, the best automatic layout has somewhat higher performance due to well-chosen device merges and careful routing. This example illustrates the need to consider detailed electrical optimizations during both device placement and routing.

Layouts for a small CMOS op-amp example appear in Fig. 9. The circuit is a differential op amp designed in MOSIS 2-μm p-well CMOS. The circuit has 11 devices and 12 nets.

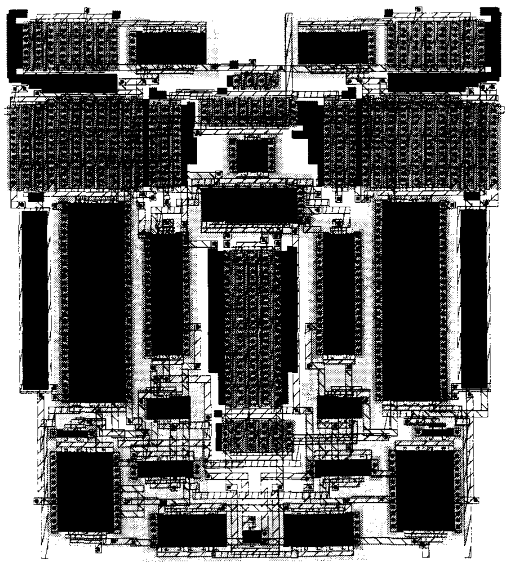


Fig. 10. Larger CMOS op-amp automatic layout.

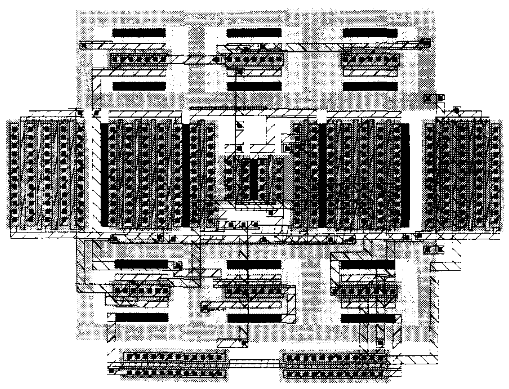


Fig. 11. Automatic BiCMOS op-amp layout with placement and routing optimization.

Fig. 9(a) shows an automatic layout exploiting all the capabilities of the placer and router. Fig. 9(b) shows for comparison a manual layout done by an industrial layout technician. Both layouts have essentially identical performance after extraction and simulation. However, the automatic layout made a different set of choices for device merging, abutment, and over-the-cell wiring, resulting in a cell that is actually 20% smaller than the manual design. The areas of the automatically generated cell and the manually generated cell were 25 872 and 32 430 μm^2 respectively.

An automatic layout for a much more complex op amp appears in Fig. 10. The circuit is also a differential op amp designed in MOSIS 2-μm p-well CMOS. However, this circuit has 31 devices and 24 nets. The automatic layout occupies 105 876 μm^2, and is only 13% larger than a high-quality manual layout.

As a final example, Fig. 11 shows an automatic layout of a folded-cascode op amp, now in a 2-μm n-well MOSIS BiCMOS process. The circuit has 16 devices and 15 nets. The current mirrors in this op amp are bipolar. A set of characterized bipolar device layouts was imported for use in this example. The layout tools can import arbitrary manual device layouts, and combine them with procedurally generated devices during placement and routing. We also altered the merging rules in KOAN to recognize and allow merging of the guard rings around each bipolar device. (Merging of collectors could also be supported, but does not provide any advantage in this layout.) Note the high degree of merging, both guard rings and MOS devices, in the final result. The layout occupies 41 888 μm^2.

VI. Conclusions

We have described new algorithms for analog device placement and routing, and their implementation in the tools KOAN and ANAGRAM II. Together, these tools support a more detailed model of analog device layout than other analog macrocell systems. Several layout capabilities are unique to these tools, in particular, their reliance on a very small library of procedural device generators, dynamic merging and abutment of devices during placement, over-the-device routing, mirror-symmetric and self-symmetric routing, and crosstalk avoiding area routing. Preliminary results are very encouraging. Our recent layouts are considerably smaller than our earlier attempts, and are beginning to approximate the density and aesthetics of expert manual designs. More importantly, our layout algorithms can reliably produce high-performance layouts. Moreover, we believe the flexible placement and routing models that underly these tools will allow us to target new technologies as they emerge. The extension from CMOS to BiCMOS cell layout, for example, required only the modification of a few technology rules in the placer and router. Our current efforts are focused on improving the communication between the placer and the router, since it is clear that division of the analog layout task into sequential placement and routing steps is artificial and often problematic in very dense layouts.

References

[1] M. G. R. DeGrauwe *et al.* "IDAC: An interactive design tool for analog CMOS circuits," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 6, pp. 1106–1116, Dec. 1987.

[2] R. Harjani, R. A. Rutenbar, and L. R Carley, "OASYS: A framework for analog circuit synthesis," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 12, pp. 1247–1266, Dec. 1989.

[3] H. Koh, C. Sequin, and P. Gray, "OPASYN: A compiler for CMOS operational amplifiers," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 2, pp. 113–125, Feb. 1990.

[4] A. H. Fung, D. J. Chen, Y. N. Lai, and B. J. Sheu, "Knowledge-based analog circuit synthesis with flexible architecture," in *Proc. IEEE Int. Conf. Computer Design*, Oct. 1988.

[5] M. Mogaki, N. Kato, Y. Chikami, N. Yamada, and Y. Kobayashi, "LADIES: An automatic layout system for analog LSI," in *IEEE Int. Conf. CAD*, Nov. 1989.

[6] M. Ayal, S. Piguet, M. Declercq, and B. Hochet, "An interactive layout generation tool for CMOS analog IC's," in *Proc. IEEE Int. Symp. Circuits Syst.*, June 1988.

[7] J. Rijmenants *et al.*, "ILAC: An automated layout tool for analog CMOS circuits," *IEEE J. Solid-State Circuits*, pp. 417–425, no. 2, Apr. 1989.

[8] D. Garrod, R. A. Rutenbar, and L. R. Carley, "Automatic layout of custom analog cells in ANAGRAM," in *Proc. IEEE Int. Conf. CAD*, Nov. 1988.

[9] E. Berkcan, M. d'Abreu, and W. Laughton, "Analog compilation based on successive decompositions," in *Proc. Design Automation Conf.*, June 1988.

[10] J. Trnka, R. Hedman, G. Koehler, and K. Lading, "A device level auto place and wire methodology for analog and digital masterslices," in *ISSCC Dig. Tech. Papers*, Feb. 1988.

[11] S. W. Mehranfar, "STAT: A schematic to artwork translator for custom analog cells," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 1990.

[12] R. S. Gyurcsik and J. C. Jeen, "A generalized approach to routing mixed analog and digital signals net in a channel," *IEEE J. Solid-State Circuits*, vol. 24, no. 2, pp. 436–442, Apr. 1989.

[13] S. Piguet, F. Rahali, M. Kayal, E. Zysman, and M. Declercq, "A new routing method for full custom analog IC's," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 1990.

[14] R. Okuda, T. Sato, H. Onodera, and K. Tamaru, "An efficient algorithm for layout compaction problem with symmetry constraints," in *Proc. IEEE Int. Conf. CAD*, Nov. 1989.

[15] H. Onodera, H. Kanbara, and K. Tamaru, "Operational amplifier compilation with performance optimization," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 1989.

[16] J. Kuhn, "Analog module generators for silicon compilation," *VLSI Design*, May 1987.

[17] R. J. Bowman, "Analog macrocell layout generation," in *Proc. 2nd Annual IEEE ASIC Seminar and Exhibit*, Sept. 1989.

[18] U. Choudhury and A. Sangiovanni-Vincentelli, "Constraint generation for routing analog circuits," in *Proc. Design Automation Conf.*, June 1990.

[19] J. Y. Lee, "Efficient pole zero sensitivity calculation using asymptotic waveform evaluation (AWE)," M.S. thesis, Dept. Electrical Comput. Eng., Carnegie-Mellon Univ., Pittsburgh, PA, May, 1990.

[20] R. H. J. M. Otten, "Automatic floor-plan design," in *Proc. 19th ACM/IEEE Design Automation Conf.*, June 1982, pp. 261–267.

[21] L. R. Carley, "ACACIA, The CMU analog design system," CMUCAD Tech. Rep. CMUCAD-89-64, Carnegie Mellon Univ., Pittsburgh, PA, Nov., 1989.

[22] D. W. Jepsen and C. D. Gelatt Jr., "Macro placement by Monte Carlo annealing," in *Proc. IEEE Int. Conf. Computer Design*, Nov. 1984, pp. 495–498.

[23] C. Sechen, "Chip-planning, placement and global routing of macro/custom cell integrated circuits using simulated annealing," in *Proc. 25th ACM/IEEE Design Automation Conf.*, June 1988, pp. 73–80.

[24] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," in *Proc. 23rd ACM/IEEE Design Automation Conf.*, June 1986, pp. 101–107.

[25] L. Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," *Inform. Contr.*, vol. 59, pp. 91–101, 1983.

[26] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[27] M. D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing," in *Proc. 1986 IEEE Int. Conf. CAD*, Nov. 1986.

[28] J. K. Ousterhout *et al.*, "Magic: A VLSI layout system," in *Proc. 21st ACM/IEEE Design Automation Conf.*, June 1984.

[29] J. K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, 1984.

[30] E. M. Reingold. J. Nevergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*. Englewood Cliffs, NJ: Prentice Hall, 1977.

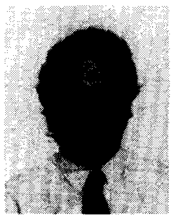
[31] A. Margarino *et al.*, "A tile-expansion router," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no 4, Apr. 1987.

[32] M. H. Arnold and W. S. Scott, "An interactive maze router with hints," in *Proc. 25th ACM/IEEE Design Automation Conf.*, June 1988.

[33] A. Barr and E. Feigenbaum, Ed., *The Handbook of Artificial Intelligence, Vol. I*. Los Altos, CA: W. Kaufman, 1981.

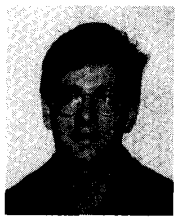
[34] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," ERL Memo ERL-M520, Univ. of Calif., Berkeley, May 1975.

[35] *HSPICE User's Manual*, Meta-Software Inc., Campbell, CA, 1987.

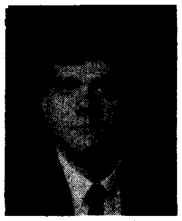


John M. Cohn was born in New York City in 1959. He received the B.S.E.E. degree from the Massachusetts Institute of Technology, Cambridge, in 1981.

From 1981 until the present he has worked at IBM's General Technologies Division, Essex Junction, VT, in the area of analog CAE tool development. In 1988 he was admitted to the IBM Resident Study Program, which has allowed him to attend Carnegie Mellon University, Pittsburgh, PA. He is currently a Ph.D. candidate in the Department of Electrical Engineering. His current research is in the area of analog layout automation. He is the author of the KOAN analog placer.



David J. Garrod was born in New York City in 1963. He received the B.S.E.E. degree from the University of California at Davis in 1985 and the M.S.E.E. degree from Carnegie Mellon University, Pittsburgh, PA, in 1987, where he is currently completing work for the Ph.D. degree in electrical and computer engineering. His research has concentrated on the area of layout algorithms for analog circuits. Specifically, he is the author of ANAGRAM I and II.



Rob A. Rutenbar (S'77–M'84–SM'90) received the B.S. degree in electrical and computer engineering from Wayne State University, Detroit, MI, in 1978, and the M.S. and Ph.D. degrees in computer engineering (CICE) from the University of Michigan, Ann Arbor, in 1979 and 1984, respectively.

In 1984 he joined the faculty of Carnegie Mellon University, Pittsburgh, PA, where he is currently an Associate Professor of Electrical and Computer Engineering, and of Computer Science. His research interests include VLSI layout algorithms, parallel CAD algorithms, and applications of automatic synthesis techniques to VLSI design, in particular, synthesis of analog integrated circuits, and synthesis of CAD software.

Dr. Rutenbar received a Presidential Young Investigator Award from the National Science Foundation in 1987. At the 1987 IEEE-ACM Design Automation Conference, he received a Best Paper Award for work on analog circuit synthesis. In 1989, he was Guest Editor of a special issue of *IEEE Design & Test*. In 1990 he received the Benjamin Teare Award for Excellence in Teaching from the College of Engineering at CMU. He is a member of ACM, Eta Kappa Nu, Sigma Xi, and AAAS.

L. Richard Carley (S'77–M'84–SM'90) received the S.B. degree from the Massachusetts Institute of Technology (MIT), Cambridge, in 1976 and was awarded the Guillemin Prize for the best EE undergraduate thesis. He remained at MIT where he received the M.S. degree in 1978 and the Ph.D. degree in 1984.

He has worked for MIT's Lincoln Laboratories and has acted as a consultant in the area of analog circuit design and design automation for Analog Devices and Hughes Aircraft among others. In 1984 he joined Carnegie Mellon University, Pittsburgh, PA, where he is currently an Associate Professor of Electrical and Computer Engineering. His research is in the area of analysis, design, automatic synthesis, and simulation of mixed analog/digital systems.

Dr. Carley received a National Science Foundation Presidential Young Investigator Award in 1985, and a Best Paper Award at the 1987 Design Automation Conference.