Direct Transistor-Level Layout for Digital Blocks

Prakash Gopalakrishnan, Rob A. Rutenbar
{prakashg,rutenbar}@ece.cmu.edu
Dept. of ECE, Carnegie Mellon University
Pittsburgh, Pennsylvania, 15213

Abstract - We present a complete transistor-level layout flow, from logic netlist to final shapes, for blocks of combinational logic up to a few thousand transistors in size. The direct transistor-level attack easily accommodates the demands for careful custom sizing necessary in high-speed design, and is also significantly denser than a comparable cell-based layout. The key algorithmic innovations are (a) early identification of essential diffusion-merged MOS device groups called *clusters*, but (b) deferred binding of clusters to a specific shape-level layout until the very end of a multi-phase placement strategy. A global placer arranges uncommitted clusters; a detailed placer optimizes clusters at shape level for density and for overall routability. A commercial router completes the flow. Experiments comparing to a commercial standard cell-level layout flow show that, when flattened to transistors, our tool consistently achieves 100% routed layouts that average 23% less area.

1. Introduction

Standard-cell based design methodologies have dominated layout generation of digital VLSI circuits. Standard cells have several notable virtues: they hide the increasingly unpleasant details of shape-level design rules; they arrange IO pins on individual gates in geometrically accessible locations; they assemble with relative ease into row-based blocks; they can be pre-characterized for timing and power. As a result, research in layout automation has split into two directions: (a) device-level layout for individual cells; (b) cell-level placement and routing for large digital blocks.

Unfortunately, the growing demands for transparent process portability, increased performance, and low-level device sizing to optimize block timing and power, are not easily handled in a fixed cell library. Libraries need to be large to achieve good logic synthesis results; today's best libraries comprise thousands of cell variants, enough to support multiple drive strengths and power/speed trade-offs. As a result, cell libraries carry an enormous inertia that resists porting, custom sizing, etc. This contributes to huge library maintenance costs.

We suggest direct transistor-level layout for blocks of digital logic as an alternative that can more easily accommodate the demands for device-level and shape-level flexibility. The recent emergence of efficient, accurate transistor-level timing estimators [5][16] mitigates the problem of timing characterization. Unfortunately, the layout quality of transistor-level algorithms proposed to date leaves much to be desired. We argue that the essential flaw in these prior attempts is an over-reliance on the methods and assumptions of large-scale cell-based layout algorithms. Individual transistors may seem simple, but they do not pack as gates do for purposes of optimal layout. Careful inspection of the internal layout of any well-designed library cell will reveal a wealth of shape-level optimizations that each save a bit of area or delay off the overall layout. These savings may seem negligible, but they are *amplified* enormously in any layout with thousands of cells. Algorithms that capture some of these geometric tricks usually rely on optimization frameworks that cannot scale beyond a few tens of devices. Algorithms that ignore these shape-level issues and pretend devices can be laid out as if they were gates suffer the consequences when thousands of devices are poorly packed.

In this paper we develop a novel set of algorithms for direct transistor-level layout that capture the essential shape-level optimizations, yet scale easily to netlists with thousands of devices. The key idea is early identification of essential diffusion-merged MOS device groups, and their preservation in an *uncommitted* geometric form until the very end of detailed placement. Roughly speaking, we extract essential groups *early* from the transistor-level netlist, place them *globally*, optimize them *locally*, and then finally *commit* each to a specific shape-level form while concurrently optimizing for both density and routability. We use a commercial detailed router to complete our flow. Results to date are encouraging: we can place/route 2000 devices in about 30 minutes, and we can consistently save 15-30% on area in comparison with a standard commercial cell-based flow.

This paper is organized as follows. Section 2 reviews existing transistor-level approaches and also builds the geometric foundation for our layout abstraction. Section 3 gives an overview of our methodology, highlighting its key differences from existing methods. Sections 4, 5 & 6 describe our algorithms in detail. Though our algorithms are circuit-style independent, our current focus is to demonstrate the viability of our approach for series-parallel static-CMOS circuits. We summarize our routing integration in Section 7, and present our experimental setup and results in Section 8. Finally, Section 9 offers concluding remarks.

2. Background and Previous Approaches

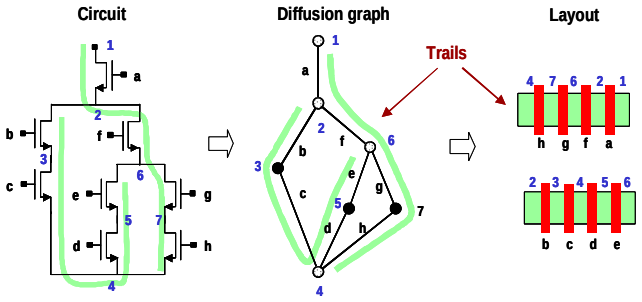
Prior work on transistor-level digital layout focuses on two targets: layout for individual library cells, and layout for larger blocks of arbitrary logic. We briefly look at each of these.

In the context of cell layout generation, Maziasz [18] provides a good review of the graph theoretical formulations and algorithms for one-dimensional cell layout synthesis. Some approaches, like Malavasi [17], LiB [10] and PicassoII [14], partition the circuit into locally optimal clusters of transistors and then place these clusters. However, it has been noted by Sadakane [22] and HCLIP [7] that techniques which generate intra- and inter-cluster layouts in two separate stages can yield layouts that are far from optimal. The concept of geometric options for clusters during final placement (Sadakane [22]) is one we will revisit—though we present a novel scalable formulation. Although HCLIP does explore inter-cluster

diffusion merging using a hierarchical ILP formulation, we believe that this must be exploited dynamically, in the context of overall physical placement and routability. Further, the maximum number of transistors that can be handled by any of these methods is about 100 to 200. We aim at circuits that are an order of magnitude larger.

Basaran [1] begins by transferring a MOS circuit into a *diffusion graph*. Formally, this graph has a node for each net and an edge for each MOS device channel. Any trail through the graph identifies a series of devices, all of whose drain/source diffusions may be shared if the devices are placed in a row, in the same order the path visits the edge associated with each device (see Figure 1). It was shown that iterative improvement methods involving trail reordering can then be used to search the solution space of min-width layouts for those that minimize routing. Riepe [21] uses Basaran's techniques to dynamically alter trails within a 2-D placement using a sequence pair formulation. More recent work, AKORD [27], uses pre-calculated optimal chaining orders to accept/reject random moves within an annealing framework, in the style of KOAN [4]. These chains of transistors, called *trails*, are a natural abstraction for layout optimization. However, as before, these algorithms do not scale well for larger circuits.

FIGURE 1. Trails in Diffusion Graph and Layout



It has been noted by Lefebvre and Chan [13] that for 2-D layouts, a solution with minimum wiring complexity is not necessarily of minimum width. This leads us to believe that constraining transistors too early into dense, min-width, locally optimal, clusters can hamper placement flexibility.

In the realm of block-placement at transistor level, Tani, *et al.* [25] partition the circuit into sub-circuits using a temporary cell library which they create. Each sub-circuit is a collection of cells in a row. They then optimize each row for one-dimensional placement. They have generated layouts with areas smaller than standard cell-based layouts—but only in an older channel-based routing style. Their saving came from allowing routing over the cell, which is the norm today.

Three recent industrial cell and block synthesis tools address related practical problems. Cellerity [8] employs annealing-based optimization for diffusion abutment, wirelength, channel density and gate alignments during leaf cell synthesis. C5M [2] develops hierarchical row-based macros for static CMOS logic. Schematic partitioning is combined with device-size-tuning and on-the-fly leaf-cell synthesis, accommodating multiple objectives and top-down constraints. However, our method aims to eliminate explicit cell generation. Nevertheless, we note that pin positioning and cell image flexibility are crucial to routability. LAS [3] produces flat symbolic layouts at block-level, which are then compacted. A major drawback is the preliminary partitioning into locally optimal clusters which are not thereafter dynamically altered in the context of the overall placement. We have observed empirically in our own tools that dense diffusion merging without careful routability consideration will require excessive white space to accommodate routing. This results in sub-optimal overall layouts.

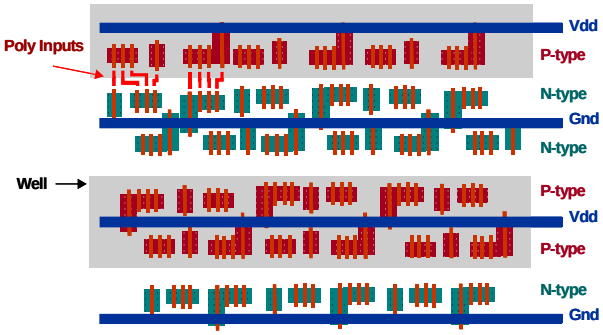
In the next section, we present our new strategy that overcomes the quality and scalability problems of these previous approaches.

3. Overall Strategy

Let us start with a fundamental question: *in a transistor-level layout tool, what exactly are we placing and routing*? Clearly, standard-cells offer a rather coarse abstraction as fundamental units of placement. Individual transistors, on the other hand, tend to be too small, especially if we want both shape-level optimization (which is very local) and accurate routability prediction (which requires a more global view). As mentioned in the previous section, *trails* (see Figure 1), which are chains of diffusion-merged transistors, are natural device groupings. In this section we will show that abstracting the netlist as groups of trails, we can effectively support geometric optimization from shape-level up to block-level.

For the static CMOS style, we restrict ourselves to a row-based geometric model that resembles a standard cell-based footprint. As illustrated in Figure 2, we have two rows of N-type trails alternating with two rows of P-type trails. There are two reasons for this choice: (a) transistors from alternating rows of the same diffusion type can share a common power rail and also the same well region; (b) transistors from neighboring rows of opposite diffusion type can have aligned polysilicon gates.

FIGURE 2. Our Geometric Model

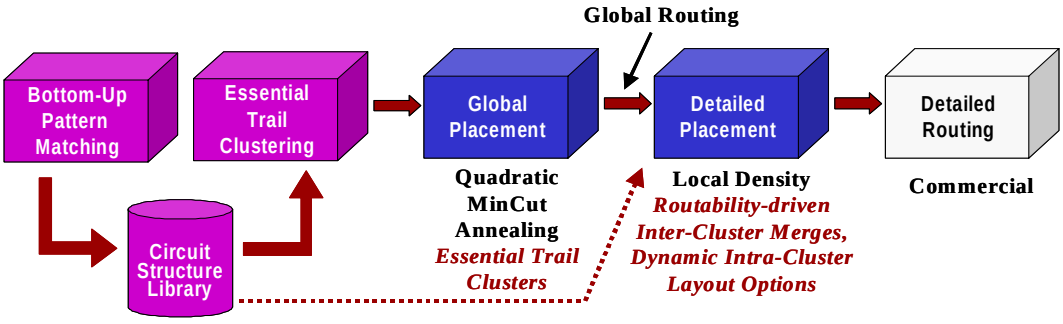


Transistor clustering has been a key step in many prior layout algorithms. However, the aggressive clustering needed to handle flat placement of thousands of devices invariably leads to locally-optimal, but globally sub-optimal groups that limit the shape-level optimizations we want to explore. We resolve this by initially forming only *essential* clusters of devices, which are mandatory for the given circuit style. For example, for static CMOS circuits, these are transistors that are strongly connected and constrained to be together due to polysilicon gate-alignment. A cluster is represented as a group of connected transistor trails. In our approach, "global" optimization focuses on these clusters as the atomic units; "detailed" optimization focuses on the trails inside a specific cluster.

Previous approaches to clustering either completely disallowed inter-cluster merges (like standard-cell methods), considered inter- and intra- cluster merges in two separate stages, or ignored global placement/routability issues during cluster layout optimizations. The key components of our trail-level approach are as follows:

- **Minimal Essential Clustering**: We initially form essential clusters to meet circuit style-imposed layout restrictions. We keep this minimal for maximum placement flexibility.

FIGURE 3. Our Complete Flow.



- **Circuit Structure Recognition**: We recognize structure in transistor netlists and store distinct circuit structures in a library for pattern matching. These circuit patterns are our basis for clustering. This creates a single level of hierarchical divide-and-conquer that lets us efficiently compute all feasible trail-level layouts for any cluster (pattern). This is efficient because we only need to compute essential trail formations for a few distinct circuit structures.
- **Global Placement**: We find optimal locations for our essential clusters while minimizing overall wirelength and congestion. The clusters, however, are not committed to any fixed layout option (any arrangement of transistor trails) during this phase.
- **Detailed Placement**: We next explore dense inter-cluster diffusion merging considering all feasible intra-cluster layout options, in the context of global placement and routability. This is where each cluster is finally bound to a physical shape-level trail layout. Global routing is updated dynamically during this optimization.

To complete the flow, we have integrated a commercial detailed router. We now describe each of these steps in detail.

4. Essential Clusters and Circuit Structure

Transistor netlists and their corresponding layouts have a significant amount of important local structure. When style-specific circuit constraints are imposed, there are not too many ways in which the transistors can be combined to produce routable layouts. For example, an important requirement for series-parallel static-CMOS circuit layouts is the alignment of polysilicon gates belonging to complementary transistors. Another performance-induced constraint is the level of strapping required on the source/drain diffusion nodes. Guan et. al. [6] describe techniques for creating layouts when partial/full diffusion strapping is required. It turns out that even for the most commonly occurring circuit structures, there may not be several different ways of combining the transistors to form trails while paying attention to these details. For the circuit shown in Figure 4(a), there is only *one* such way of combining the transistors so that the resulting placement is routable. For some other simple circuits like that in Figure 4(b) there are more trail formation options to choose from.

These complementary pairs of trails together form the *essential trail clusters* that *need* to be together because of circuit style imposed restrictions. We refer to the corresponding layout options for these clusters as *cluster layout options*. Notice however that these options are still modest in number (two for the simple NAND example in Figure 4(b)). Consequently, we argue that the cluster layout options should not be discovered on-the-fly, but should instead be dynamically re-arranged during placement.

Further, in large transistor-level netlists, there are often repetitive circuit structures. This is especially true of control logic and data-path circuits that are commonly synthesized by technology mapping to a (sized transistor-level) logic library. Our key idea is to identify these circuit structures bottom-up via pattern matching. We use graph isomorphism techniques for pattern matching. There have been similar approaches to pattern matching in netlist partitioning [20]. We use the Graph-Matcher library [19]. We then determine optimal trail formation options for these circuit structures and store them in a *Circuit Structure Library*. As shown in Figure 5, this provides a framework that allows us to use any of the known diffusion graph algorithms or other cell layout synthesis techniques to determine optimal trail options. This results in significant computational savings because we only need to determine optimal options for small, distinct circuit structures in the netlist.

Sadakane [22] uses clusters and template-options for clusters, but their method is extremely time-consuming and handles only up to 80 transistors. This could be attributed to their annealing-based intra-cluster option selection. We describe in the next two sections how we explore inter-cluster diffusion merges between all feasible intra-cluster layout options, exhaustively and deterministically, in the context of global placement and global/detailed routability.

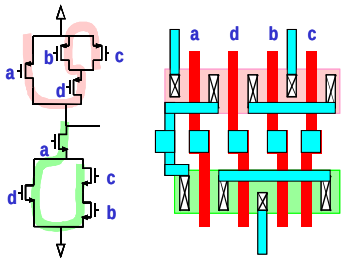


FIGURE 4. (a) Circuit with only one configuration for trails

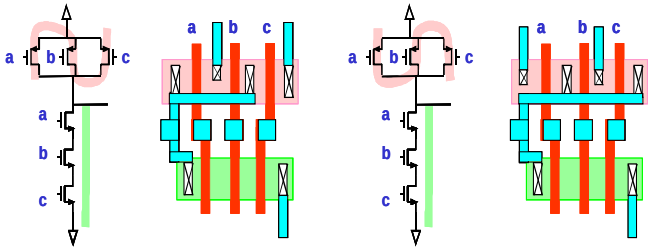


FIGURE 4. (b) Simple NAND circuit with two possible configurations for trails.

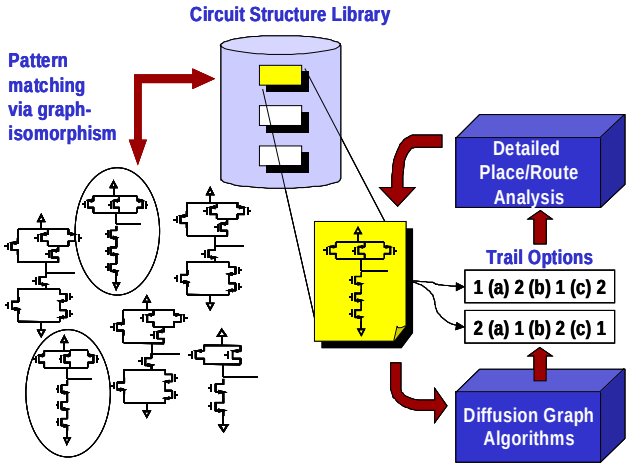


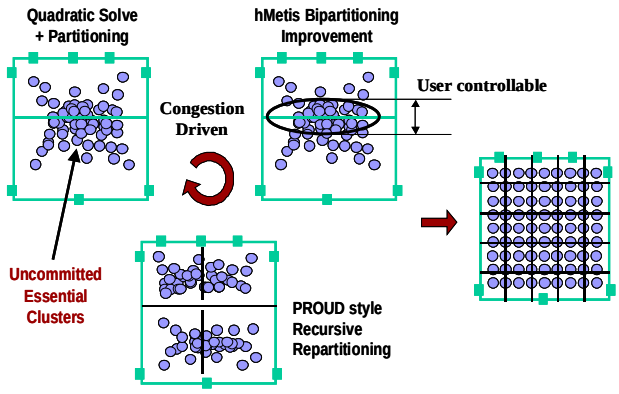
FIGURE 5. Circuit Structure Library

5. Global Placement

The placeable modules here are essential clusters that are not committed to any fixed trail-level layout option. We employ a recursive re-partitioning-based quadratic placement in the style of PROUD [26]. As a convenient simplification during quadratic optimization, we assume the input/output pins are located along the boundary of the block. A quadratic solve then gives us initial locations for the clusters. During the partitioning phase, a horizontal (or vertical) cut line is located, via sorting on Y (or X), so that about half the cluster area is on each side of the cut. A physical cutline is then placed at the actual center of the partition. For each partition thus formed, quadratic optimization is carried on, for clusters within the partition, after projecting the coordinates of all clusters/pins outside the partition onto the cutline. Applying this idea recursively (see Figure 6), we create a sequence of increasingly smaller (but increasing in number), wide and narrow regions in which we confine and re-place the enclosed modules.

Quadratic solves invariably result in many modules placed very close to the cutline, making it difficult for the partitioning step to make intelligent decisions. GORDIAN [12] introduced the idea of using bipartitioning to resolve which side of the cut these objects should be bound to. Hence, we formulate a similar bipartitioning problem that is allowed to relocate a specified fraction of the clusters on each side of the physical partition, that are near the cutline. This is also shown in Figure 6. The remaining clusters are assigned

FIGURE 6. Global Placement Strategy



to partitions based on their quadratic solve-based locations. Varying this fraction lets us control the emphasis on bipartitioning relative to that on pure quadratic placement. A fraction of about 0.6 to 0.7 gives us good results on average. For bipartitioning, we used hMetis [11]. As we show later in our results, this gives us excellent control over wiring congestion during global placement.

The final step in our global placement is a phase of simulated annealing [23] for legalization and local minimization of wirelength, congestion and layout row raggedness. Specifically, we perform annealing in the cold regime which favors significant downhill optimization.

Most prior transistor-level placers and essentially all standard-cell placers would stop here. These approaches do not permit *inter-cluster merging*. This is primarily because denser diffusion merges usually compromise routability. At block-level, standard cell designs are routable at least in part because at transistor level, all the internal sub-nets are fully pre-routed and sufficient pin spacings have been arranged for net accessibility. Because we do not wish to stop here, however, we must take the responsibility for these careful shape-level optimizations. This is the purpose of the next phase of placement.

6. Detailed Placement

At the end of global placement, we have arranged device clusters in rows (see again Figure 2), legalized these clusters, and minimized an estimate of wirelength and congestion. However, none of these clusters have been yet committed to a physical trail-level layout. By deferring this binding to a separate placement step, we are able to optimize each cluster carefully against a good global model of both wirelength and congestion. In this section, we discuss how we optimize individual clusters and the boundaries between clusters for density, and for routability. Attention to shape-level routability turns out to be extremely crucial here. We use simple global routing to predict macroscopic routability in this step. We then present a novel technique to generate dense layouts while handling routability considerations.

6.1 Intra-Cluster Optimizations

Our essential clusters are typically composed of a few transistor trails. The cluster is "uncommitted"—it has a nominal width in the row, but no specific layout. To create a palette of layout alternatives for a cluster, we must focus carefully on routability of individual trails, and groups of adjacent trails.

To begin, note that a transistor trail is fully routable *only if*:

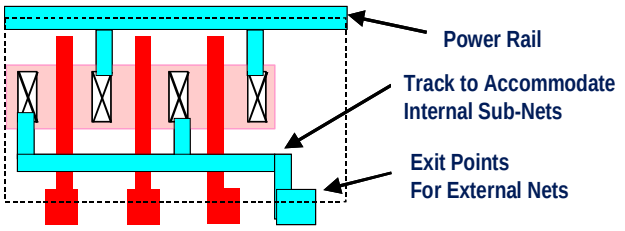
- All sub-nets internal to the trail have enough space to be completely routed within the trail.
- All nets connecting to the outside have enough space to exit from the trail through pin escapes.

Figure 7 illustrates this for a simple example. It is important to ensure that pins connecting to polysilicon gate inputs are spaced appropriately to allow accessibility on metal layers.

The same conditions apply when we need to route a set of trails. For example, a pair of vertically adjacent trails in a cluster typically represent NMOS and PMOS devices sharing some poly gates; choosing the placement so that poly gates can both align and escape to metal is critical. A pair of horizontally adjacent trails typically represent same-type MOS devices that may be able to share diffusion and merge. Again, the critical routability issue is ensuring enough tracks so that internal sub-nets can connect to these devices, while not compromising any polysilicon gate alignments.

Our first step in generating cluster layout options is to match the

FIGURE 7. Trail Routability



cluster circuit structure with patterns in the circuit structure library. As described earlier, this gives us the various feasible combinations of transistors to produce easily routable trails. Each of those trail formations then offers us layout alternatives for the cluster. We note here that while some small gates (like NAND, NOR) map one-to-one to essential clusters, other small gates (like AND, OR) and larger gates (like AOI, OAI) map to a handful of essential clusters. Relative geometric locations are then assigned to these trails while respecting shape-level routability issues we just discussed. This is then combined with transistor-size information. For big transistors, various fingering options can be incorporated. This can result in a richer set of cluster layout options, thereby supporting dynamic folding during local placement optimization.

6.2 Inter-Cluster Optimizations

Allowing adjacent clusters to merge, i.e., to share diffusion in the N- or P-type device rows, is a simple but remarkably powerful optimization. In particular, it is a local optimization that is *amplified* when we scale to thousands of devices. We illustrate this with a simple example. Figure 8 (left) shows the different intra-cluster layout options for a simple circuit cluster. Given a placed cluster, we show (on the right) different possibilities of inter-cluster merging between the layout options for the uncommitted cluster and the placed cluster. Notice in the figure that if neighboring clusters cannot merge (i.e., no suitable selection/orientation of the uncommitted cluster yields a legal geometric merge) then the resulting layout resembles a pair of standard cells: the clusters can only abut at their boundaries. But if a suitable merge can be found, the clusters can actually *overlap*. In this simple example, the best merging option allows diffusion sharing in the top device row (and slightly extends the diffusions if necessary for design rules), and replaces one small power rail connection with a single wider strap that is physically on the boundary between the clusters.

Notice the saving in area, even for a single inter-cluster merge, by choosing the *best* layout option. Also note that all merges we consider between neighboring clusters satisfy the detailed routability conditions: the resulting dense layout must have enough space within, to accommodate all internal sub-nets and enough space for external nets to escape out.

At block-level, these layouts are also subject to global routing congestion. This is irrespective of whether the layout was generated using standard cells or at transistor-level. We address this next.

6.3 Global Routability

In order to estimate global routing congestion after global placement, we replace each net with its simple Steiner representation. We then calculate the wiring demand at various locations in the layout using a coarse global routing grid. In regions that are heavily congested, we provide some relief by introducing extra vertical routing tracks while we consider inter-cluster neighbor merges in that region. This makes sense because ultimately the pin locations of the trails constrain global nets. The farther they are spaced, more the availability of routing tracks. Horizontal routing tracks are also introduced by adjusting row-spacings.

In the next sub-section, we present a linear-time heuristic to find the best inter-cluster merges considering all possible layout options for each uncommitted cluster in the context of global and detailed routability.

6.4 Local Placement Optimization

We present here a heuristic row-based optimization that traverses each row from left to right while finding the best cluster layout option for an uncommitted cluster, given the best layout option for the neighboring (left) cluster. Formally, the best layout op-

FIGURE 8. Best Option Merging

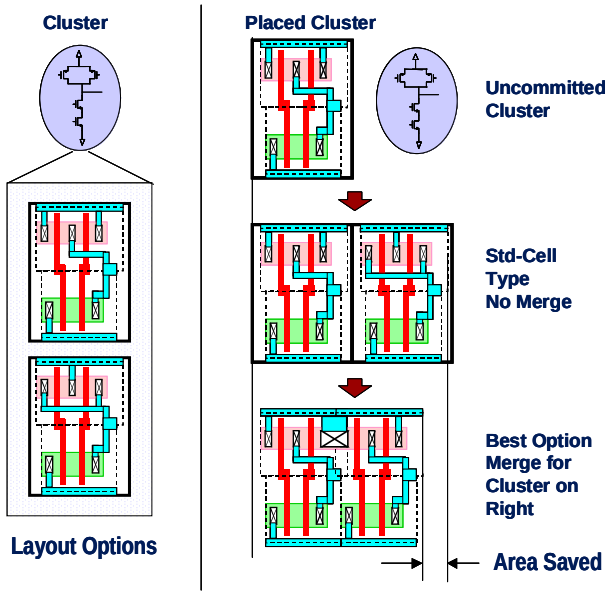
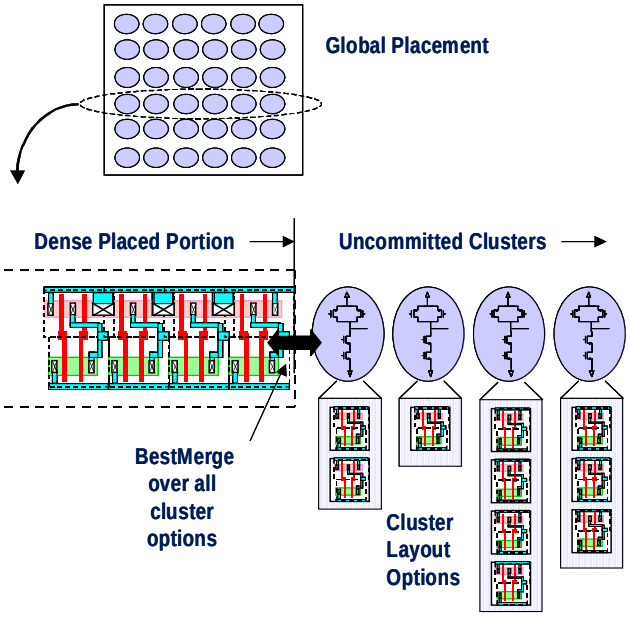


FIGURE 9. Local Placement Optimization



tion for cluster *i+1* is given by:

$$BestOption_{i+1} = k \qquad (k \in S_{i+1}) \qquad satisfying$$
$$MinWidth_{i+1} = MinWidth_i + BestMerge(BestOption_i, k)$$

where S_j is the set of all layout options for cluster *j*, $MinWidth_j$ is the minimum row layout width up to cluster *j*. *BestMerge(L,R)* is the densest layout generated by merging layout option *R* to the right of layout option *L*. Let *P* be the set of all trails belonging to layout option *R* that interact with *L*. Let $DiffX_p$ be the left-boundary of the diffusion region belonging to trail *p* ($p \in P$) and let $PinX_p$ be the location of the left-most pin belonging to *p*. The goal of *BestMerge* is to then:

$$\begin{pmatrix} Minimize \\ \forall (p \in P) \end{pmatrix} \{DiffX_p, PinX_p\} \qquad subject \quad to$$

- *Constraint A (Diffusion Merging)*

$$DiffX_p \geq MaxDiffX_L + DRCLength$$

where $MaxDiffX_L$ is the right-most diffusion boundary of the trail belonging to layout option L that interacts with p. DRCLength is the design-rule imposed restriction. In general, if the interacting trails can be merged, then (DRCLength < 0), else (DRCLength > 0)

- *Constraint B (Routability)*

$$PinX_p \geq MaxPinX_L + SubNetTracks + GlobalVertTracks$$

where $MaxPinX_L$ is the right-most location of a pin belonging to layout Option L. SubNetTracks are the number of extra vertical routing tracks required to completely route internal subnets belonging to the trail. GlobalVertTracks are the number of extra global vertical routing tracks required in this region.

- *Constraint C (Poly Alignment)*

In order to align the polysilicon gate inputs of neighboring N & P trails and also the pins, we formulate a quadratic wirelength minimization equation between the objects to be aligned [30]. Assuming that the distances between the gate inputs are fixed and so also the distances between the pins, we end up with a linear constraint:

$$a_1 \times PinX_p + \sum_{p \in AlignP} (a_p \times DiffX_p) = constant$$

where AlignP is the set of trails that needs to be aligned.

BestMerge thus computes the left-most positions for all trails belonging to layout Option *R* while satisfying polysilicon alignment and routability constraints. Note that in regions that are global-routing congested, *Constraint B* dominates whereas *Constraint A* dominates in other regions resulting in dense inter-cluster merges. This local row-based optimization is illustrated in Fig. 9. The overall algorithm optimizes each row in the layout by traversing from bottom to top, left to right. At the end of processing every row, we update all global nets that were affected. This lets us use the most recent information while accounting for global congestion. During this process, inter-row spacing is introduced to satisfy horizontal global routing demand.

The current implementation is greedy, but is very effective and very fast. (A more expensive dynamic programming approach, in the style of [9] is one obvious extension here; another would be to iterate multiple passes through the complete row-based local optimization and global routing update.) The implementation has a linear complexity of *O(K * n)*, where *K* is the maximum number of layout options for any cluster in the circuit and *n* is the number of clusters.

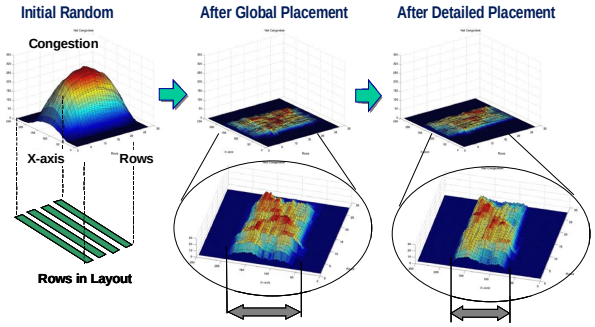


FIGURE 10. Congestion plots for circuit c432

7. Routing

To complete our flow, we have integrated the Cadence IC-Craftsman router [29] at the back-end. We route the power nets, intra-cluster and inter-cluster nets, in this order. In the next section, we present our experimental setup and results comparing fully routed layouts generated using our tool with those generated using a standard Cadence cell-based flow.

8. Experimental Results

We have implemented these ideas in a tool called *TrailBlazer*. Before we present our detailed experimental setup and results, we take a quick look at a simple example result illustrating various algorithmic aspects of our flow.

8.1 Simple Layout Example

We describe here the results obtained for a simple benchmark, the circuit c432 from [28]. To create a suitable netlist for us, we resynthesize c432 onto a simple target cell library that comprises INVERT, AND, NAND, OR, NOR gates up to 4 inputs. Assuming standard static CMOS, the circuit structure library has only 7 series-parallel circuit patterns in it, e.g., the AND gates end up decomposed into a combination of more simple NAND and INVERT patterns. Mapped onto this library, c432 has 191 gates; after flattening it has 836 devices and 456 nets; after essential cluster recognition it has 206 clusters which are what we place in our global placer. The technology is 0.35um HP CMOS with one poly and 3 metal layers. Placement (global and detailed) requires 3 minutes; detail routing takes another 20 minutes.

Figure 10 shows congestion after both global and detailed placement, estimated at each point as the number of net bounding boxes that overlap that point. Detailed placement reduces the width of the overall layout, without compromising overall congestion. Figure 11 shows the distribution of trails of various sizes, where the

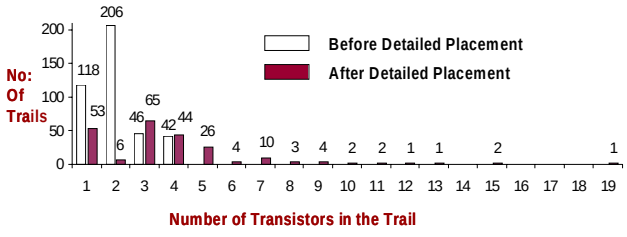


FIGURE 11. Trail histograms for circuit c432

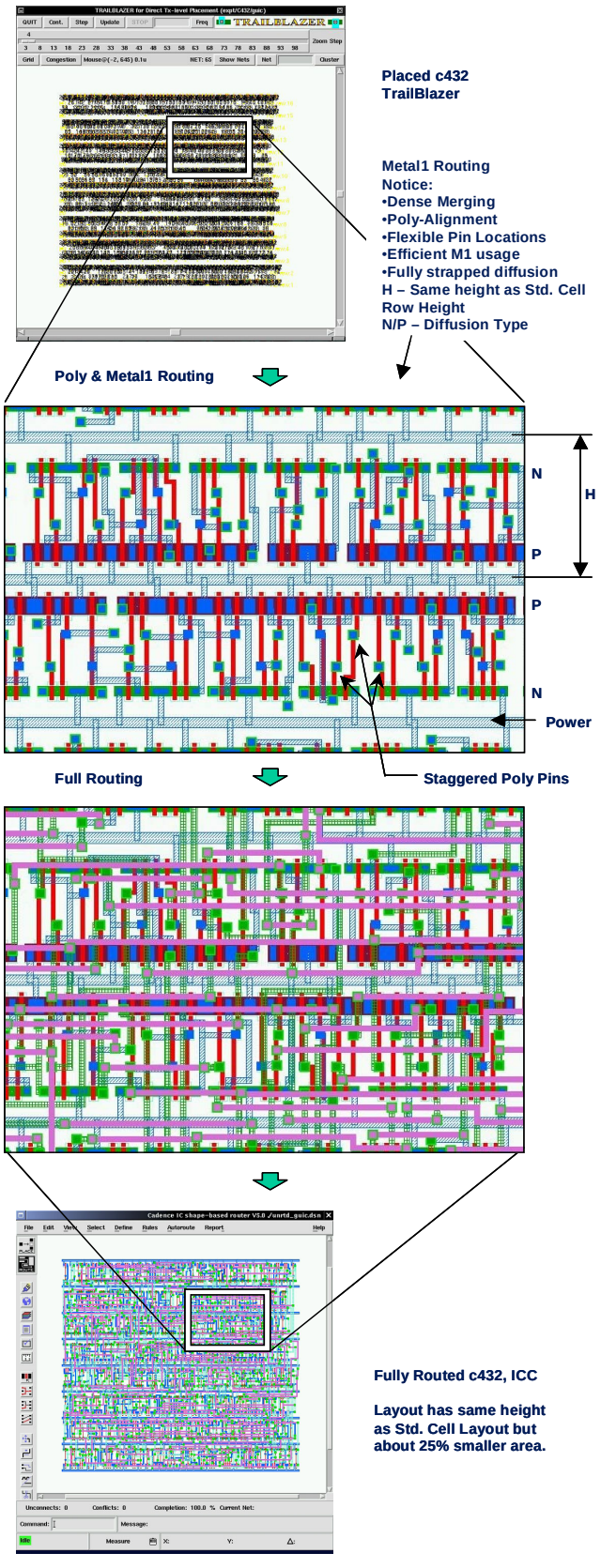


FIGURE 12. Different stages of evolution of the layout for c432 from placement to detailed routing

TABLE 1. Maximum Density Routed Results

Bench.	No. of Trans	Rows	Our Layout Area (um^2)	Std. Cell Lower Bound (um^2)	%Impr over Lower Bound	Silicon Ensemble Area (um^2)	%Impr over SE
frg1	454	6	6714.9	8934.3	**24.8%**	9452.7	**28.9%**
b9	464	5	7033.5	9153	**23.12%**	9510.8	**26%**
i2	686	7	8968.1	11321.1	**20.7%**	12606.3	**28.8%**
i4	764	7	11084.9	12360.6	**10.3%**	13768.7	**19.5%**
C432	836	8	12744.0	16200.0	**21.3%**	17172.0	**25.7%**
apex7	938	8	14850.0	18489.6	**19.7%**	19828.8	**25%**
example2	1140	9	18893.3	22404.6	**15.7%**	23874.8	**20.9%**
i6	2066	12	29224.8	34700.4	**15.2%**	35964.0	**18.8%**

size of a trail is the number of transistors in the trail. Detailed placement increases the number of "big" trails, i.e., it achieves significant diffusion merging across clusters. Figure 12 displays a snapshot of TrailBlazer showing the final placement for c432 and the various stages of detailed routing.

8.2 Layout Comparison Experiments

To quantify the potential advantages of transistor-based layout versus standard cells, we ran a series of logic netlists using our flow, and a standard Cadence Silicon Ensemble (SE) flow [29].We start with a logic description (in BLIF format) which is compiled into a functional schematic (in NETBLIF format) using logic synthesis (SIS [24]). We used several standard LGSynth91 benchmarks from [28]. The synthesis target library is again the simple INVERT/ AND/NAND/OR/NOR library from the previous section. It is worth noting that a new and open question is the correct choice for this intermediate library: in our flow it serves as a means to coerce synthesis to produce a netlist that uses "good" device-level structures. Larger groupings for devices are discovered during detailed placement. On the other hand, for the standard cell design, this *is* the set of placeable objects. For the most direct comparison, we use the same synthesized gate-level netlist for each layout, but flatten it to transistors for our own layout flow. However, we do ensure that all device sizes are the same for the both the transistor-level and standard cell layouts.

The technology is again HP 0.35um, and we used an existing cell library. There are four available routing layers. Poly is only used for gate-input connections. Metal1 is allowed to route in either direction, but Metal2 is mostly vertical and Metal3 mostly horizontal.

In our first set of experiments, we try to normalize away the effects of standard cell placement and routing and focus on the lowest-level optimizations we do for the transistor layout. We fix the height of our device rows to be the same as the standard cells, and fix their pitch at the rule-legal minimum. We then fix the overall heights of both the transistor and standard cell layouts to be equal. We describe each benchmark, and compare total area of final routed layouts, in Table 1. Note that we also compare against an *absolute lower bound*--the sum of the areas of the individual standard cells in the cell layout. Note that our layouts are 10-25% *better* than this absolute lower bound: this is because detailed placement allows clusters to overlap. We are also 19-29% better than the densest layouts generated using Silicon Ensemble (set to 99% row utilization, i.e., maximum placement density).

In our second set of experiments, we lay out a set of more difficult benchmarks that require more attention to routing congestion. To begin, we place our transistor-level netlists using TrailBlazer

TABLE 2. Routing Congestion-Aware Routed Results

Bench.	No. of Trans	Layout Height *(um)*	Our Layout Area *(um^2)*	Silicon Ensemble Area *(um^2)*	%Impr over SE
C880	1410	150	29595	38550	**23.2%**
term1	1682	162	34279	47142	**27.3%**
x4	1826	180	37620	46800	**19.6**
C1355	2164	196	47844	61603	**22.3%**
C1908	2552	203	51887	61692	**15.9%**
i9	3256	244	67344	83204	**19.1%**

and then route them in Cadence. We then fix the height of the cell-based layout to be the same as our successful device-level layout. We adjusted percent row utilization in Silicon Ensemble until the cell layouts were just fully routable. Table 2 shows the results of this experiment. We note again that the transistor-level layouts are from 16% to 27% smaller than the comparable cell layout. The detailed placement optimization carefully allocates both intra-row wiring space and inter-row wiring space, and correlates quite well with the space required to actually complete the routing. We find empirically that with these routing optimizations disabled, it is impossible to create routable transistor-level layouts. The layouts were also generated in reasonable time. The largest benchmark with 3256 transistors took about 9 minutes to place and another 30 minutes to route on a desktop workstation.

9. Conclusions

We presented a novel transistor-level layout flow, for blocks of combinational logic up to a few thousand transistors in size. The key algorithmic innovations are early identification of essential diffusion-merged MOS device *clusters*, but deferred binding of clusters to a specific shape-level layout until the very end of placement. A commercial router completes the flow. Experiments comparing to a commercial standard cell-level layout flow show that, when flattened to transistors, our tool consistently achieves 100% routed layouts that average 23% less area. Our approach shows that a good divide-and-conquer attack need not handicap the shape-level optimizations that have always been the distinguishing features of the best custom device-level layouts. Our emphasis on recognized circuit clusters means that natural groups of a few connected devices always have the optimal geometric arrangement. Global placement using only these clusters reduces the complexity of the problem. Detailed placement focusing on intra-cluster alternatives and inter-cluster merges allows us to do crucial shape-level optimizations, but scale to large netlists.

Our current work is focused on adding a timing-driven component to this flow [5][16], and on understanding the role played by logic synthesis and technology mapping when the target cell library is an artificial and intermediate construct, intended only to guide the creation of a well-structured transistor-level netlist.

Acknowledgment

We would like to thank Bill Halpin and Artour Levin of Intel and John Cohn and Jeff Burns of IBM for constructive comments and criticisms. This work was funded by the Semiconductor Research Corporation.

References

[1] A. Basaran and R. Rutenbar, “An o(n) algorithm for transistor stacking with performance constraints,” in *ACM/SIGDA Physical Design Workshop*, 1996.

[2] J. Burns and J. Feldman, “C5M-A Control Logic Layout Synthesis System for High-Performance Microprocessors,” in Proc. *1997 ISPD*, pp. 110-115.

[3] 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.

[4] J. M. Cohn, D. J. Garrod, R. A. Rutenbar and L. R. Carley, “Analog Device-Level Layout Automation”, Kluwer Academic Publishers, Boston MA., 1994.

[5] F. Dartu, L. T. Pileggi, “TETA: Transistor-Level Engine for Timing Analysis,” in Proc. *35th DAC*, June 1998, pp. 595-598.

[6] B. Guan, C. Sechen, “Efficient standard cell generation when diffusion strapping is required,” in Proc. *1996 CICC*., pp. 501 -504.

[7] A. Gupta and J. Hayes, “Near-Optimum Hierarchical Layout Synthesis of Two-Dimensional CMOS Cells,” *Proc. 12th International Conference on VLSI Design*, Jan 1999, pp. 453-459.

[8] M. Guruswamy, R. Maziasz, D. Dulitz, S. Raman, V. Chiluvuri, A. Fernandez and L. Jones, “CELLERITY: A Fully Automatic Layout Synthesis System for Standard Cell Libraries,” in Proc. 1997 DAC, pp. 327-332.

[9] Her, T. W. and D. F. Wong, “Optimal Module Implementation and its Application to Transistor Placement,” IEEE International Conference on Computer-Aided Design, 1991, 98-101.

[10] Y. Hsieh, C. Hwang, Y. Lin, Y. Hsu, “LiB: A CMOS Cell Compiler,” *IEEE Trans. on CAD*, 10(8), August 1991, pp. 994-1005.

[11] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, “hMetis, A Hypergraph Partitioning Package, Version 1.0,” Manuscript, December 1997.

[12] J. Kleinhans, G. Siegl, F. Johannes, K. Antreich, “GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization,” *IEEE Trans. CAD*, vol 10 no 3, March 1991.

[13] M. Lefebvre and C. Chan, “Optimal Ordering of Gate Signals in CMOS Complex Gates,” in Proc. 1989 CICC, pp. 17.5.1-17.5.4.

[14] M. Lefebvre and D. Skoll, “PicassoII: A CMOS Leaf Cell Synthesis System,” in Proc. *1992 MCNC Intl. Workshop on Layout Synth.*, vol. 2, pp. 207-219.

[15] M. Lefebvre, D. Marple and C. Sechen, “The Future of Custom Cell Generation in Physical Synthesis,” in Proc. *1997 DAC*, pp. 446-451.

[16] C. B. McDonald and R. E. Bryant, “Symbolic Functional and Timing Verification of Transistor-Level Circuits,” IEEE Transactions on Computer-Aided Design, March 2001.

[17] E.Malavasi, D. Pandini, “Optimum CMOS Stack Generation with Analog Constraints,” *IEEE Transactions on Computer-Aided Design*, Vol. 14, No. 1, January 1995, pp. 107-122.

[18] R.L. Maziasz, J.P. Hayes, *Layout Minimization of CMOS Cells*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1992.

[19] B.T. Messmer and H. Bunke, “Subgraph isomorphism in polynomial time,” Technischer Bericht IAM 95-003, Institut fur Informatik, Universitat Bern, Schweiz, 1995.

[20] M. Ohlrich, C. Ebeling, E. Ginting, L. Sather, “SubGemini: Identifying SubCircuits using a Fast Subgraph Isomorphism Algorithm,” Proc. *1993 DAC*, pp. 31-37.

[21] M. A. Riepe, “Transistor Level Micro Placement and Routing for Two-Dimensional Digital VLSI Cell Synthesis,” Ph.D. dissertation, University of Michigan, 1999.

[22] T. Sadakane, H. Nakao, and M. Terai, “A new hierarchical algorithm for transistor placement in CMOS macro cell design,” Proceedings of CICC-95, pp. 461-464.

[23] C. Sechen and A. Sangiovanni-Vincentelli, “The TimberWolf Placement and Routing Package,” *IEEE Journal of Solid-State Circuits*, Vol. SC-20, No. 2, April 1985, pp. 510-522.

[24] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, A. Sangiovanni-Vincentelli; SIS: A System for Sequential Circuit Synthesis; Dept. of EECS, University of California, Berkeley, 1992

[25] K. Tani, K. Izumi, M. Kashimura, T. Matsuda and T. Fujii, “Two-Dimensional Layout Synthesis for Large-Scale CMOS Circuits”, in proc. *1991 ICCAD*, pp. 490-493.

[26] R. S. Tsay, E. Kuh, C. P. Hsu, “PROUD: A Sea-Of-gates Placement Algorithm,” *IEEE Design & Test of Computers*, Dec 1988.

[27] T. Serdar, C. Sechen, “AKORD: Transistor-Level and Mixed Transistor/Gate Level Placement Tool for Digital Data Paths,” in Proc. 1999 *ICCAD*.

[28] http://www.cbl.ncsu.edu/benchmarks

[29] http://www.cadence.com

[30] Prakash Gopalakrishnan, “Direct Transistor-Level Layout for Digital Blocks,” Ph.D. Thesis, Carnegie Mellon University, 2001.