This paper appeared in: Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASPDAC) 2013, 453-460

BonnCell: Automatic Layout of Leaf Cells

Stefan Hougardy, Tim Nieberg, Jan Schneider
Research Institute for Discrete Mathematics
University of Bonn
Lennéstr. 2, 53113 Bonn
Email: {hougardy, nieberg, schneid}@or.uni-bonn.de

Abstract— In this paper we present BonnCell, our solution to compute leaf cell layouts in VLSI design. Our placement algorithm allows to find very compact solutions and uses an accurate target function to guarantee routability. The routing algorithm handles all nets simultaneously using a constraint generation MIP based approach. Finally, yield and electromigration properties are improved in a post-processing phase. Our approach considers design rules already during placement and routing, is able to treat gridless technologies, and easily adapts to new design rules and future technologies as for example double patterning in 14nm and beyond.
The experimental results on current 22nm designs of our industry partner show significant improvements both in terms of design quality and turnaround time compared to manual designs done by experienced designers.

I. Introduction

In a hierarchical design of a complex chip the *leaf cells* are the functional units at the lowest level of the hierarchy. A leaf cell (which is also called *cell* or *circuit*) realizes simple logical functionality and is built from a small number of transistors, usually not more than 30.

Most of the leaf cells used for the design of a chip are taken from a leaf cell library that is created in advance. However, at several points in the design process of a chip it may turn out that highly optimized special purpose cells are needed which are not contained in the library. In these cases a new leaf cell has to be created or an existing leaf cell has to be redesigned.

So far an experienced designer is able to craft leaf cell layouts of higher quality than automatically generated layouts. However, with each new technology the need for high quality automatic leaf cell layout generators increases. This is due to the fact that design rules (DRC) become more and more complex and the number of different leaf cells used in modern designs is growing steadily. Moreover, the manual layout of a complex leaf cell can take several days, making this process a severe bottleneck in turn around time.

In this paper, we present our solution BonnCell for the automatic generation of leaf cell layouts, both for placement and routing. Our tool provides solutions that are optimal in terms of area consumption and reduces the need for manual interaction significantly in practice. While many existing leaf cell layout tools require gridded technology, our tool also allows to handle non-gridded instances. Moreover, we consider many complex design rules as for example line end spacing and specific via requirements already during the placement and the routing phases. This is a crucial requirement for the current 22nm technology and beyond as design rule cleanness can no longer be achieved by simple local postprocessing operations.

A leaf cell consists of several field effect transistors (*FETs*). Each FET connects three nets—source, drain, and gate—which are on top of the diffusion area. Figure 1 shows a FET in cross section. A FET can have more than one gate, and we refer to the number of gates as the number of *fingers* of the transistor.

The *leaf cell layout problem* can be described as follows. As input an image of the cell is given, i.e. an area with predefined vertical power tracks, where a given set of FETs has to be placed. The electrical connectivity of the FETs is described in a netlist. The task is to decide how many fingers a FET should use and to assign to each FET a location within the image subject to the design rules of a given technology. Here, the height of the image is the most important optimization criterion as this determines the area of the cell on the chip. Given a placement of the FETs, the goal is then to find an embedding of rectilinear Steiner trees which realizes the given netlist. This has to be done meeting the DRC constraints, as well. As overall goal in routing, we minimize the weighted netlength, especially making the topmost available layer M2 more expensive in line with the designers needs.

A crucial point to obtain high quality leaf cell layouts is that the placement algorithm has a very good estimate how much free area is needed in the routing step. If the placement is too pessimistic this will result in a waste of space. We have designed a target function for the placement step that very accurately estimates the quality of a placement with respect to

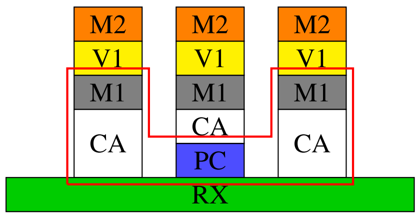


Fig. 1. Cross section of a field effect transistor (FET) and leaf cell: the diffusion area is denoted by RX, PC gives the layer of the gates, CA (together with wires on M1) form the source and drain contacts, and M1/M2 are the metal layers of the leaf cell. CA and V1 denote via layers.

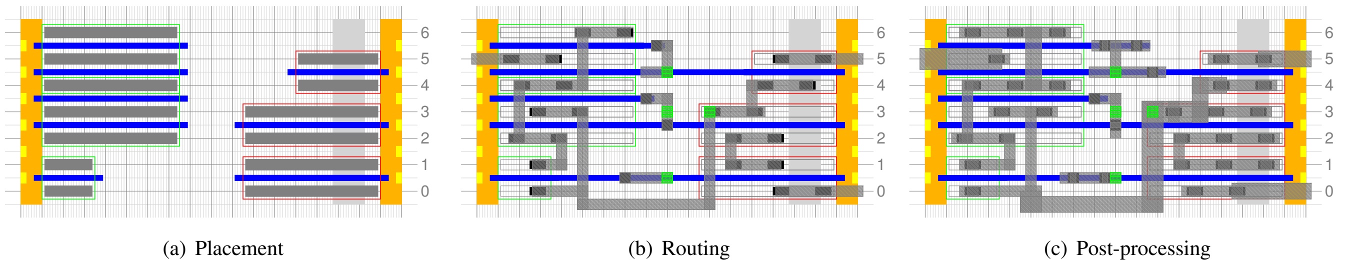


Fig. 2. Example of BonnCell outputs. The orange areas to the left and right represent the power structures of a circuit column, the green rectangles represent external connections of the cell. Metal on PC is depicted in blue, M1 wires are gray. (Note that the leaf cells of our industry partner are rotated by 90 degree compared to the leaf cells of most other manufactures.)

later routablity (see Section A). Our placements turned out to be routable in more than 95% of all considered instances. In many cases we can even prove that our placement solution is *optimal* with respect to our target function.

A drawback of our very compact placements is that routing becomes much more difficult. Thus a standard sequential rip up and reroute approach turned out to fail for most of our instances. Instead we use an approach that allows to route all nets *simultaneously* and consider many design rules already while building up the nets. The latter is required because only few design rule violations can be fixed afterwards due to the limited routing space in our compact placements.

In the remainder of this paper, we present and discuss our approaches to place and route a leaf cell (see also Figure 2). In Section II, we discuss the placement algorithm, followed by the routing solution in Section III and postprocessing in Section IV. Section V reports the results of our implementation with current leaf cell designs at the 22nm technology node from our industrial partners at IBM.

A. Related Work

Most previous work on leaf cell layout focuses on restricted versions of the placement problem only (e.g. [1, 2, 3, 4, 5, 6]), and the goal considered is area minimization. There are only few works applying existing routing strategies to a placed leaf cell given in [7, 8, 9].

In [2], a MIP based placement engine is presented and the FETs are placed such that their gates can be connected by direct wires. Focusing on diffusion region sharing, [1] presents an algorithm to stack transistors. The choice of folding fingers is not considered explicitly, but left as input to be done by the designer prior to the given algorithm. An enumeration with integrated partitioning of the cell is used for placement in [3], and [7, 5] apply several heuristics to enumerate Eulerian paths that connect nearby gates and contacts.

After an exhaustive search based placement, the routing part in [9] uses a greedy channel routing algorithm to connect gates. If the routing step fails, additional space is added at the ends of the cell. The authors also state that the results from this approach in terms of quality is unacceptable and introduce several pre- and post-processing steps to control the quality of their routing better.

Poirier [8] uses a sequential shortest path approach to route a placed leaf cell, combined with cost-based ripup and reroute. In case the routing step does not succeed, empty space is added to the cell to mitigate congestion and also different orders of the nets are tried.

In [10] the routing problem is formulated as a SAT problem by considering a set of possible connections for each pair of terminals of a net. Pairs being in conflict or redundant pairs are pruned to reduce the size of the SAT instance which is then solved using a standard SAT solver.

Note that all of these works consider so-called gridded technologies. Especially with respect to DRC constraints for wiring on a leaf cell, using a coarse wiring grid requires (almost) no consideration of these rules. In newer technologies, wires may be placed arbitrarily and the number and complexity of DRC constraints has increased. Moreover, they now have to be taken into account both during placement and routing, making the problems significantly more complicated to solve in practice.

II. Placement

The input of the leaf cell placement problem is a set $\mathcal{F}$ of FETs, a set $\mathcal{N}$ of nets and a large number of technology-specific constraints. A *FET* is characterized by a tuple $(w, I, n_g, n_s, n_d, v, \delta)$, where $w \in \mathbb{N}$ is the total width of the FET (which is roughly proportional to the amount of current that can flow through the device), $I = [f_{\text{min}}, f_{\text{max}}]$ is the interval of allowed finger widths, $\{n_g, n_s, n_d\} \subset \mathcal{N}$ denote the nets attached to the ground, source, and drain contacts, $v \in \mathcal{V}$ is a VT level and $\delta \in \{N, P\}$ is the type.

FETs can be realized in several different ways, so solving the placement problem does not only include the assignment of locations to each transistor. The total width w of a FET F can be distributed to an arbitrary number of *fingers*. Using only 1 finger, the physical width of the FET's underlying RX area will be equal to its total width while spending 3 fingers results in a FET that uses two more tracks in the vertical direction but requires only a third of the width. In general, if F is realized with ϕ fingers it uses $\phi + 1$ tracks and requires a *finger width* of $\frac{w}{\phi}$. The only restriction on the number of fingers is that $f_{\text{min}} \leq \frac{w}{\phi} \leq f_{\text{max}}$ must hold. A FET with several fingers connects source and drain alternately between the fingers. The placement algorithm is also allowed to swap FETs. In this case, the source and drain contacts of F exchange their places. Figure 3 shows the same FET realized in different ways.

The output of the placement algorithm consists of locations

$(x, y) : \mathcal{F} \to \mathbb{R}^2$, finger numbers $\phi : \mathcal{F} \to \mathbb{N}_{>0}$ and swaps $s : \mathcal{F} \to \{\text{yes}, \text{no}\}$. This information is then passed on to the routing algorithm (see Section III).

Although the 22nm design rules give much freedom in the placement of FETs, a large fraction of the real-world layouts employ a specific structure. In these cells the transistors are arranged in two *stacks* in the space between two power rails. The one stack is built from the cell's N-FETs and is connected to the first power rail, while the other stack contains the P-FETs and is connected to the other power rail. While BonnCell makes no assumption on the number and location of the power rails, it is designed to generate 2-stack cells having this commonly used structure.

While many design rules become irrelevant when restricting BonnCell to 2-stack placements, there are still some important constraints the placement tool has to obey. The most important ones are:

- A function d specifies the minimum size of gaps between FETs in the following way: If F_2 is the upper neighbor of F_1 on one of the stacks, then there must be at least $d := d(F_1, \phi(F_1), s(F_1), F_2, \phi(F_2), s(F_2))$ unused tracks between them. If $d = 0$, then the FETs can abut and if $d = -1$, then the topmost contact of F_1 may overlap the bottommost contact of F_2. In the latter case the diffusion regions overlap as well and the contact is used simultaneously by both FETs.
- The horizontal distance between the two stacks must be large enough to guarantee the local routability. This distance is not a constant but depends on the nets which access the FETs on a given coordinate and might vary from track to track.
- While the main algorithm of BonnCell computes the track number for each FET (i.e. the y-coordinate), the horizontal placement cannot be ignored altogether. Specific sequences of finger widths on subsequent tracks are forbidden, as they would result in a placement for which no feasible x-coordinates exist.

A. Target Function

BonnCell aims to find placements which are small but as "routable" as possible. To do so, we use an efficiently computable model to measure the routability of a placement $P = (x, y, \phi, s)$ involving the following three values:

- Let $h(P)$ be the *height* of the placement, i.e. the number of non-empty tracks.
- The *gate-gate netlength* is defined as $\text{ggnl}(P) := \sum_{n \in \mathcal{N}} \text{ggnl}(n)$, where $\text{ggnl}(n)$ is the height of the bounding box of all gate contacts in n.
- The *weighted netlength* is defined as $\text{wnl}(P) := \sum_{n \in \mathcal{N}} w_n \cdot \text{nl}(n)$, where $\text{nl}(n)$ is the height of the bounding box of all contacts in n (gate, source, and drain) and w_n are user-defined net weights.

We can now define a partial ordering $\leq_P$ on all placements as the lexicographic ordering of the triple $(h(\cdot), \text{ggnl}(\cdot), \text{wnl}(\cdot))$. A very similar target function was already used in [6].

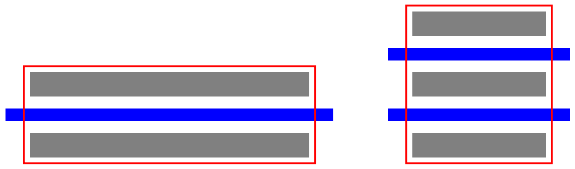


Fig. 3. The same FET realized with 1 and 2 fingers

This method to evaluate the quality of placements turns out to be a very good indicator for the routability of real-world 22nm cells. The first-order criterion is the cell height because this is what the user usually tries to minimize and that determines the footprint of the cell on the chip area. The second-order criterion, the gate-gate netlength, comes from the fact that the layer on which gates are accessed has a very high resistance and connections on this layer should be as short as possible. Additionally, on this scale, netlength in general is a good measure of the routability and our experiments show that a cell which is optimal within this quality measure is legally routable in almost all cases (see Section V).

B. Placement Algorithm

BonnCell's placement algorithm, as outlined in Algorithm 1, implements a recursive enumeration of all possible placements that backtracks as soon as the current (partial) solution cannot be part of a placement that is better than the best placement that was found so far. This branch & bound method runs in two phases: The first one is faster but finds only solutions with a specific structure. The second one is not restricted but is more likely to fail on complex instances. If that happens, BonnCell returns the result from phase 1.

In the first phase, BonnCell introduces a vertical line that separates both FET stacks. It then requires that every FET is placed completely on one side of this line. From the possible x-coordinates it chooses the one that leads to two stacks which are as balanced as possible. It then computes height-optimal placements for both sides independently in order to find the number of tracks used by an optimal solution, say h_{opt}. This is done using a single-stack subroutine which is explained in Section C.

After h_{opt} is known, BonnCell starts to look for an optimal placement in terms of the previously defined target function. We use the single-stack placer to enumerate all legal arrangements of the N-FET stack with height h_{opt} and for each of these placements we use a modified version of the single-stack placer to find the best placement of the second stack. The secondary and ternary optimization targets are used in this modified placer to prune branches of the search tree: Both single-stack placers maintain intervals for each net that represent the already placed contacts of the nets. The size of these intervals form lower bounds for the final net length. BonnCell can bound when their sum exceeds the best known netlength. We also incorporate the still unplaced contacts of nets in this computation, as each unplaced contact induces, under certain circumstances, a further increase of the length of that net.

Algorithm 1 TWOSTACKPLACEMENT

```
1: Phase 1: Determine vertical line separating N and P stack
2: Compute minimum heights $h_N, h_P$ of restricted stacks
3: for every placement of stack N with height $\max\{h_N, h_P\}$ do
4:     Enumerate all possible placements of P stack with height $\max\{h_N, h_P\}$ and save the best 2-stack placement
5: end for
6: Phase 2: Remove vertical line and FET width restrictions
7: $t_b \leftarrow$ type of the bigger stack, $h_b \leftarrow$ min. height of $t_b$
8: $t_s \leftarrow$ type of the smaller stack
9: for $h = h_b, h_b + 1, \ldots$ do
10:     for every placement of stack $t_b$ with height $\leq h$ do
11:         Enumerate all possible placements of $t_s$ which are not higher than $t_b$ and save the best 2-stack placement
12:     end for
13:     Exit loop if legal placement of height $h$ exists or timeout
14: end for
15: Return best placement
```

The outcome of phase 1 is an optimal placement with the restriction imposed by the vertical line. Phase 2 drops this restriction and finds a global optimum. The main problem here is that we do not know in advance how many tracks a smallest legal placement of the cell needs because both stacks compete for the same placement space. A very compact first stack which uses a minimum number of tracks will most likely not leave enough free area for the second stack to be placed. Since the minimum height of the larger stack is a lower bound for the height of the finished cell, this height is used for the first try. If this does not suffice, the cell height is increased by 1 track and BonnCell repeats the process until a placement is found.

The complexity of phase 2 is much higher because the flexibility in the number of fingers each FET may have is significantly higher. However, there are also new ways to bound:

- When placing the second stack, we bound if two FETs overlap or if the space between horizontally neighboring FETs does not suffice for local routability.
- Each time the first stack is fully placed, BonnCell analyzes the free space per track and compares it to the FETs that need to be placed in the second stack. In many cases this is enough to see that the FETs in the smaller stack do not fit in the remaining placement space.

C. Placing a Single Stack

An important subroutine of the BonnCell placement method is ENUMERATESTACK. Its input is a set $\mathcal{F}$ of unplaced FETs, usually of the same type (N or P), and its output is a list of all possible 1-stack placements of minimum height.

The outer loop of the method (Algorithm 2) distributes fingers to the FETs. In a first step, it uses for all FETs their respective minimum number of fingers, then all possibilities to add 1 finger to the set of FETs are enumerated, then 2 fingers and so on. We can exit the loop as soon as the total number of fingers exceeds the number of tracks used by the best solution that was found so far. Before we run the inner loop, we compute an upper bound h_{ub} for the minimum height of a placement using FETs of the current finger numbers. To do so, a very fast and resonably good heuristic approach is applied.

Algorithm 2 ENUMERATESTACK($\mathcal{F}$)

```
1: for $\kappa = 0, \ldots, \kappa_{\max}$ do
2:     for all possibilities to add $\kappa$ fingers to $\mathcal{F}$ do
3:         Compute upper bound $h_{\text{ub}}$ for final stack height
4:         SINGLESTACKRECURSION($\emptyset$, $\mathcal{F}$)
5:     end for
6: end for
```

The inner loop (Algorithm 3) recursively generates the stack from bottom to top. Given a partial solution, it takes every yet unplaced FET and places it at the smallest legal y-coordinate on top of the partial solution – once unswapped and once swapped. In each step, a lower bound for the best height of a complete stack which extends the current partial placement is computed as follows:

$$h_{\text{lb}} := h(\mathcal{F}_p) + \textstyle\sum_{F \in \mathcal{F}_u} \phi(F) + \max\{0, n_{\text{odd}} - 1\},$$

Here $\mathcal{F}_p$ and $\mathcal{F}_u$ denote the set of placed respective unplaced FETs, $h(\mathcal{F}_p)$ is the height of the stack formed by $\mathcal{F}_p$, and n_{odd} is the number of nets that connect a bottommost or topmost contact of a FET in $\mathcal{F}_u$ for an odd number of times.

The second summand reflects the fact that every unplaced finger enlarges the current placement by at least one track. The third summand exploits the fact that FETs can only overlap (i.e. $d = -1$) if the topmost contact of the bottom FET connects the same net as the bottommost contact of the upper FET, otherwise $d \geq 0$ must hold. Consequently, if a net occurs an odd number of times as bottommost or topmost contact of an unplaced FET, this net will account for at least one additional track in the finished placement. For this consideration it is irrelevant if unplaced FETs will be placed swapped or unswapped, as swapping a FET changes the number a net occurs in this sum by an even number One has to subtract 1 because one such net can be compensated by placing it at the top of the stack.

If h_{lb} is larger than the best solution the algorithm has found up to that point, or the upper bound which was determined to initialize the inner loop, we do not have to investigate this branch further. In the end, the algorithm reports a placement of minimal height.

If only a single height-minimal placement is required, as e.g. in line 2 of Algorithm 1, a faster version of ENUMERATESTACK is used that branches only when $h_{\text{lb}} < \min\{h_{\text{best}}, h_{\text{lb}}\}$ and the partial solution has a specific block structure.

D. Additional Features

The behavior of the 2-stack placer can be manipulated with several parameters. A layout of minimum height can be hard to route, so the *additional tracks* feature allows the user to specify a number of tracks that can be used "for free" thereby increasing the routability.

Larger leaf cells are typically not forced between two power rails but are allowed to occupy several neighboring circuit

Algorithm 3 SINGLESTACKRECURSION($\mathcal{F}_p$, $\mathcal{F}_u$)

```
1:  if $\mathcal{F}_u \neq \emptyset$ then
2:      for all $F \in \mathcal{F}_u$ do
3:          Place $F$ on the bottommost possible track
4:          Compute lower bound $h_{\text{lb}}$ for final stack height
5:          if $h_{\text{lb}} \leq \min\{h_{\text{best}}, h_{\text{lb}}\}$ then
6:              SINGLESTACKRECURSION($\mathcal{F}_p \cup \{F\}$, $\mathcal{F}_u \setminus \{F\}$)
7:          end if
8:          Unplace $F$
9:          Swap $F$ and repeat the previous steps
10:     end for
11: else
12:     Save placement, $h_{\text{best}} \leftarrow$ height of current placement
13: end if
```

rows. Such *"multi-bit cells"* can also be placed with BonnCell, although we do not guarantee any kind of optimality in this case. To place a cell with $2k$ stacks, we first compute assignments of FETs to the stacks, evaluate these assignments and solve the most promising assignments with k copies of a variant of TWOSTACKPLACEMENT.

III. ROUTING

In order to solve the routing problem on a placed leaf cell, we use a mixed integer programming (MIP) approach. The formulation is based on a model for packing Steiner trees in graphs and is extended to produce a problem specific formulation. Note that we do not create a (huge) model and leave the rest to a commercial MIP solver, but have a more sophisticated approach (Section C).

Next to the input already given for the placement problem (Section II), the leaf cell routing problem expects the locations of each FET from the placement. Furthermore, external connections may be present for a net, together with a desired location for this external input and output.

A. Mixed Integer Programming Formulation

The leaf cell area for routing is represented by a three-dimensional grid structure, where the pitch on a layer is predetermined by the coordinates of the gates and the free area between the two stacks is divided according to fractions of the minimum wiring pitch. Note that we are working in so-called gridless technologies, where wires need not be placed in a regular grid where the distance between parallel lines is equal to the minimum wire width plus the minimum spacing. With respect to these notions, we use a half-grid on each plane.

The resulting grid defines the grid graph $G = (V, E)$ with vertices at the crossing of lines and edges between two vertices connected horizontally or vertically. Two vertices on different planes are connected if a via can connect these, creating a 3-D structure. On this graph, we are seeking a Steiner tree packing with a tree connecting the gates, contacts, and external pins of each net subject to additional constraints.

The placed FETs are then converted to terminals in the grid graph G. The contacts and external pins of a net become terminals of Steiner trees: Vertices representing gates (on PC) are directly given by the placement and vertices representing source and drain contacts follow certain rules with respect to the number of vias needed to connect RX to M1. Considering the RX area, there is some freedom in one direction on where to place a corresponding number of vias, and thus where to place a metal rectangle which then represents the terminal. In the following, we use the word terminal for all these contacts.

A decision variable x_{ij}^k is introduced for each net $k \in \mathcal{N}$ possibly using an edge $\{ij\} \in E$ of the graph G. Terminals of each net are combined to give terminals of a Steiner tree T_k. For ease of notation, we combine the usage of each edge also to a single variable x_{ij}. With these decision variables, we can formulate a core MIP for the Steiner tree packing problem in this graph based on a 2-D formulation in [11] as follows:

$$\begin{array}{lrll}
\min & \sum\limits_{\{ij\}\in E} c_{ij}x_{ij} & & \\
\text{s.t.} & x_{ij} - \sum\limits_{k=1}^{n} x_{ij}^k & = 0 & \forall\,\{ij\} \in E \\
& x_{ij} & \leq 1 & \forall\,\{ij\} \in E \\
& \sum\limits_{\{ij\}\in E \mid i\in W, j\notin W} x_{ij}^k & \geq 1 & \forall\, W \subset V, W \cap T_k \neq \emptyset \\
& & & (V \setminus W) \cap T_k \neq \emptyset, \forall k \\
& x_{ij}^k & \in \{0,1\} & \forall\,\{ij\} \in E, \forall k
\end{array}$$

Here, the first set of constraints links the usage of an edge $\{ij\} \in E$ by a net $k \in \mathcal{N}$ to the general variable x_{ij}, and the second set of constraints ensures that at most one net actually uses the respective edge. The third set of constraints ensures connectivity of each Steiner tree T_k, $k \in \mathcal{N}$, by introducing Steiner cut constraints. A Steiner cut is given by the edges of a partition $W \subset V$ of the vertices such that both sets W and $V \setminus W$ contain at least one terminal.

Instead of using such an exponential number of Steiner cut constraints to ensure connectivity, we extend the formulation for some nets by an alternative set of constraints based on network flow formulations as follows. Looking only at a single Steiner tree T_k, $k \in \mathcal{N}$, there are several MIP formulations to solve the Steiner tree problem [12]. For such a net, we combine our Steiner tree packing formulation with these single Steiner tree formulations.

A classical formulation is given in [13] that identifies one terminal as a source and the remaining ones as sinks. Then, for each source-sink-pair, a desired flow of one unit is sent from the source to the sink given by the flow variables f^t. This is done maintaining flow conservation amidst intermediate vertices of the now directed graph $G = (V, A)$. Figure 4 gives this formulation together with the additional constraints linking the flow f_{ij}^t to our edge based decision variables x_{ij}^k at each edge $\{ij\} \in E$, where we combine the two directed arcs of the directed formulation. This formulation, for each terminal, introduces only a constant number of constraints per edge and vertex of the grid graph.

In this basic formulation, no additional constraints, especially with respect to distances between wires resulting from the Steiner tree, are taken into consideration. Furthermore, two nets may actually share the same vertex, but not the same edge. Ensuring correctness in this sense, and also for the additional DRC constraints, further constraints presented next are enforced.

$$
\begin{aligned}
f^t(\delta^+(i)) - f^t(\delta^-(i)) \quad &= \quad \begin{cases} 1 & \text{for } i = r \\ -1 & \text{for } i = t \\ 0 & \text{else} \end{cases} \\
&\qquad\qquad \forall\, i \in V, \forall\, t \in T_k \setminus \{r\} \\
f^t_{ij} \quad \leq \quad x^k_{ij} & \quad \forall\, \{ij\} \in E, \forall\, t \in T_k \setminus \{r\} \\
f^t_e \quad \geq \quad 0 & \quad \forall\, e \in A, \forall\, t \in T_k \setminus \{r\}
\end{aligned}
$$

Fig. 4. Flow-based formulation for single Steiner tree $k \in \mathcal{N}, r \in T_k$ [13].

B. Mapping DRC Constraints

For the wiring within a leaf cell, the design rules fall into two basic categories: *diff-net rules* require a certain minimum distance between wires that belong to different nets, and *same-net rules* are in place to avoid geometric configurations with features below the lithographic capabilities and resolution, and to reserve space for optical proximity correction (OPC). While diff-net rules are most important, same-net rules have become more and more important with each new technology. Especially in routing, it is particularly important to obey all these rules within the algorithm because of the limited space available for fixing errors that require additional space by post-processing.

The given placement already induces forbidden edges in the grid graph which are immediately mapped to the respective variables. The basic distance rules are mapped as follows. Suppose that if an edge $e \in E$ is taken by the wiring of net $k \in \mathcal{N}$, then a neighboring edge $e' \in E$ cannot be used by another net. The inequality

$$x^k_e + \textstyle\sum_{i \in \mathcal{N} \setminus \{k\}} x^i_{e'} \leq 1$$

prohibits this situation. All basic diff-net distance rules can be modeled this way, including via distances and the interlayer via rules that prescribe minimum distances between vias in adjacent via layers. We also add some of these distance constraints for segments of the same net, as there are also same-net rules for spacing between non-adjacent segments of the same net.

Increased spacing may be required in specific situations, for example at a so-called *line-end* which is a polygonal edge between two convex corners of a metal polygon closer than some threshold to one another. In our application, such line-ends prominently occur at the end of a wire when we leave the respective plane with a via or at terminals (see Figure 5).

Looking at a vertex, let x_e, x_n, x_w, x_s denote the usage of the respective four planar edges leaving it for some net k. If the edge leaving the vertex to the west is solely used ($x_w = 1$), e.g. by having a via at the vertex, a line-end would occur and some additional spacing is required. Let e' denote an edge that no longer can be used, then

$$x_w - (x_n + x_e + x_s) + x^i_{e'} \leq 1$$

models this situation for each net $i \in \mathcal{N}$. It is easy to check that the constraint is non-binding if more than one or none of the edges x_e, x_n, x_w, x_s are used. If $x_w = 1$ and $x_n = x_e = x_s = 0$, then $x^i_{e'} = 1$ results in a violated constraint for any net $i \in \mathcal{N}$.

Every connected metalized polygon on a layer must have a certain *minimum area*, independent of its actual shape. Since each such polygon starts and ends in either a terminal or a via, we can force enough segments to be metalized in the area round these structures.

C. Constraint Generation Algorithm

The flow of the routing part is given in Algorithm 4. Due to the large number of constraints, we do not construct a MIP with all constraints from the beginning, but use a constraint generation technique that works as follows.

Generally speaking, the idea here is to start the algorithm by constructing a MIP that only contains a subset of the constraints. This reduced MIP is set up with constraints to introduce the terminals and constraints that block edges violating distances to existing shapes, e.g. power rails and RX areas. It also includes basic distance constraints such that along parallel edges the required distances are kept. We thus specifically allow nets to cross at a vertex, but not to overlap along any edge.

Once this reduced MIP is solved, all constraints are checked for feasibility, including connectivity and DRC. If there are specific constraints violated, these are added to the MIP, and we resolve based on the current solution again. Note that in this context the connectivity check does not involve verifying all exponentially many Steiner cut constraints, but a simple DFS algorithm suffices. Moreover, the result of such a reachability algorithm immediately gives violated Steiner cut constraints that can be added. Net connectivity can be achieved by either these additional Steiner cuts, or by the introduction of flow formulations for the net. In case the net is *almost* connected, i.e. there is only a short distance to close it, we add respective Steiner cuts to close the connection. Otherwise, a flow formulation is added.

D. A Sweep Line Based Improvement

Especially in large leaf cells, taking all nets into consideration at the same time when solving the packing problem results in very high runtime for the MIP solver even with the above approach. As the placement is done so that the terminals of a net are close to each other by minimizing the gate-gate-netlength, many nets only span a small part of the entire leaf cell in vertical direction. This can be used to create the following heutistic, however at the cost of optimality.

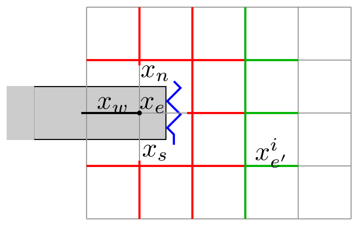


Fig. 5. Line-end (blue): edges already forbidden due to basic distance constraints are drawn red, additionally forbidden edges due to line-end DRC constraints are green. In this example, $x_w - (x_n + x_e + x_s) + x^i_{e'} \leq 1$ forces $x^i_{e'}$ to zero.

Algorithm 4 Constraint Generation LC Routing Flow

```
1: Construct MIP with limited but useful subset of constraints
2: Solve MIP
3: while nets not connected or DRC violations present do
4:     Add Steiner cut or flow-based single tree constraints
5:     Add constraints forbidding current violations
6:     Re-Solve MIP
7: end while
```

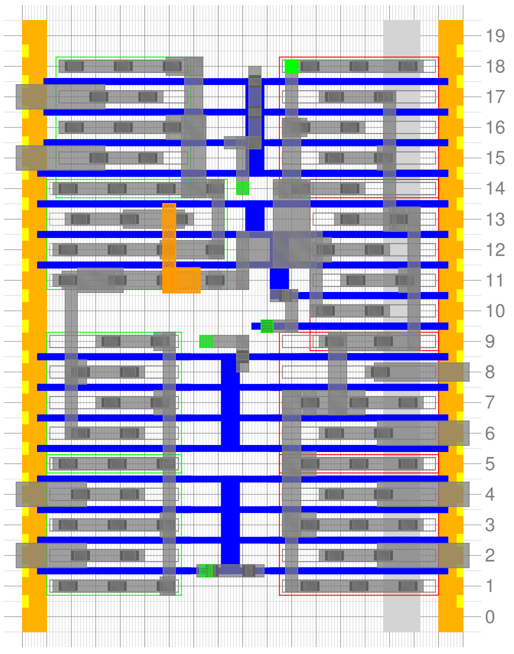


Fig. 6. Example for finished BonnCell output. The layout required a total single-threaded runtime of 2:22 hours.

Suppose that a sweep line is a horizontal line starting at the bottom of the placed cell, and that it is given together with a lookahead radius of a few tracks. We then start the constraint generation algorithm with only the additional DRC constraints within the window starting at the sweep line and ending at the height of the look ahead radius.

Once the solution returned by the constraint generation algorithm is correct for all wiring within and below this sweep line window, we raise the sweep line and thereby consider the resulting next constraints for the MIP solver. Everything below the sweep line is kept fixed, and thus these DRC constraints are met without further optimization.

IV. Post-processing

The output of the routing tool is a complete routing with wires of minimum width. While most important design rules, e.g. the line-end rules, are directly addressed in the algorithm, others are not encoded in the MIP formulation because they would increase its size too much. There are also some soft constraints which are desired rather than required. For this reason, several post-processing steps are used to decrease the number of design rule violations and increase the overall quality of BonnCell's output. These include specific rules for additional metal above and below vias, improvements for electrical robustness by increasing wire widths where needed, and centering of vias in the RX areas.

BonnCell also tries to add via redundancy by checking the proximity of existing vias for possible locations for redundant copies and choosing the location that requires the least amount of additional wire.

Cell	$\|\mathcal{F}\|$	$\|\mathcal{N}\|$	$\|T\|$	Placement h	Placement time	Routing time
1	9	12	57	15	0:04	2:55
2	7	10	32	10	0:01	0:12
3	6	9	41	10	0:01	0:13
4	12	13	57	14	0:43	27:45
5	6	10	25	7	0:01	0:08
6	14	16	39	10	0:01	0:33
7	4	9	47	11	0:01	3:26
8	5	9	34	13	0:01	0:23
9	15	23	47	14	0:01	1:49
10	8	12	46	11	0:01	38:27
11	2	6	32	8	0:00	2:10
12	2	6	56	14	0:01	5:08
13	16	16	51	14	0:01	14:33
14	5	8	16	4	0:00	2:24

TABLE I
RESULTS 22NM TESTBED: $|\mathcal{N}|$ NUMBER OF NETS, $|T|$ NUMBER OF TERMINALS (ACTIVE GATES, CONTACTS, AND EXTERNAL PINS), h HEIGHT IN TRACKS, TIME IS [MM:SS]

V. Experimental Results

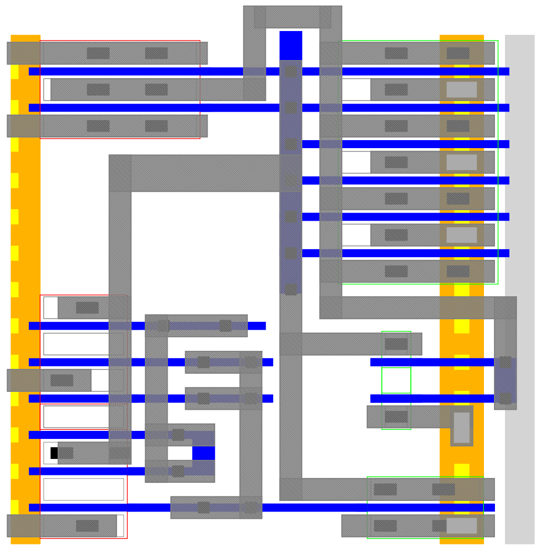
The described algorithms are implemented as part of BonnTools [14] of the University of Bonn within the scope of our cooperation with IBM and its customers. Table I reports on the runtime and quality of BonnCell, split into placement and routing part, for several characteristic current 22nm cells. All experiments were done on a 3.47GHz Intel Xeon machine, and we used CPLEX 12.2 as MIP-solver. The post-processing takes less than a second on all instances.

We would like to point out that our main goal in BonnCell is not only runtime of the tool, but also the quality and direct applicability of the resulting layout for actual designs. The tool is used in the creation of custom leaf cells by IBM, and resulted in a turn around time reduction of 50% in the core array design for a current chip.

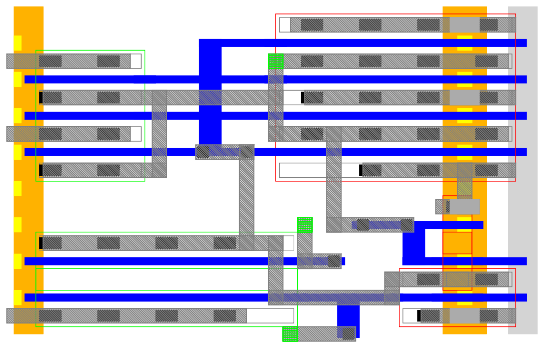
Leaf cell layouts generated by BonnCell are usually significantly smaller than layouts of the same cell that were designed manually and have a much shorter netlength. The simple target function of the placement algorithm turns out to be a very good indicator of the routability as only 2 placements in a testbed of 107 cells could be proven to be unroutable.

VI. Extensions and Future Work

In this paper, we introduced BonnCell, our solution to automatically layout leaf cells. We presented approaches that especially take DRC constraints both during placement and routing



(a) Handcrafted layout (15 tracks)



(b) BonnCell layout (9 tracks)

Fig. 7. Comparison between a handcrafted layout and a BonnCell layout for a typical small leaf cell. The runtime of the BonnCell run, including placement, routing, and post-processing, was 39 seconds.

into account, and that also address soft constraints like electrical robustness. BonnCell creates leaf cell layouts that significantly reduce the need for manual interaction, and is successfully used on current industrial VLSI chips.

The algorithms presented can be adapted to work with double patterning lithography in a straightforward way by introducing the new rules within the placement step e.g. to ensure colorability, and by introducing and linking two routing grids. This is part of current research to enhance the tool for upcoming technologies.

Acknowledgment

The authors would like to thank our cooperation partners at IBM, in particular Gerhard Hellner, Karsten Muuss, Raphael Polig, and Tobias Werner. Moreover, we would like to acknowledge the contributions of the students of the BonnCell team, in particular Katrin Kölker and Thomas Weyd.

References

[1] B. Basaran and R. Rutenbar, "An $o(n)$ algorithm for transistor stacking with performance constraints," in *Proc. DAC'96*, 1996.

[2] A. Gupta and J. Hayes, "Clip: Integer-programming-based optimal layout synthesis of 2d CMOS cells," *ACM Trans. on Design Automation of Electronic Systems*, vol. 5, no. 3, pp. 510–547, 1996.

[3] E. Malavasi and D. Pandini, "Optimum CMOS stack generation with analog constraints," *IEEE Trans. CAD of IC and Systems*, vol. 14, no. 1, pp. 107–122, 1995.

[4] Y. Lin, M. Marek-Sadowska, and W. Maly, "Layout generator for transistor-level high density regular circuits," *IEEE Trans. CAD and Systems*, vol. 29, no. 2, pp. 197–210, 2010.

[5] T. Uehara and W. van Cleemput, "Optimal layout of CMOS functional arrays," *IEEE Trans. on Computers*, vol. 30, no. 5, pp. 305–312, 1981.

[6] R. Bar-Yehuda, J. A. Feldman, R. Y. Pinter, and S. Wimer, "Depth first search and dynamic programming algorithms for efficient cmos cell generation," in *Proceedings of the fifth MIT conference on Advanced research in VLSI*. MIT Press, 1988, pp. 215–228.

[7] C. Lazzari, C. Santos, and R. Reis, "A new transistor-level layout generation strategy for static cmos circuits," in *Proc. ICECS'06*, 2006, pp. 660–663.

[8] C. Poirier, "Excellerator: Custom CMOS leaf cell layout generator," *IEEE TCAD*, vol. 8, no. 7, pp. 744–755, 1989.

[9] S. Wimer, R. Pinter, and J. Feldman, "Optimal chaining of CMOS transistors in a functional cell," *IEEE TCAD*, vol. 6, no. 5, pp. 795–801, 1987.

[10] N. Ryzhenko and S. Burns, "Standard cell routing via boolean satisfiability," in *Proc. DAC'12*, 2012, pp. 603–612.

[11] M. Grötschel, A. Martin, and R. Weismantel, "The Steiner tree packing problem in VLSI design," *Math. Prog.*, vol. 78, pp. 265–281, 1997.

[12] M. X. Goemans and Y. Myong, "A catalog of Steiner tree formulations," *Networks*, vol. 23, pp. 19–28, 1993.

[13] J. Beasley, "An algorithm for the Steiner problem in graphs," *Networks*, vol. 14, pp. 147–159, 1984.

[14] B. Korte, D. Rautenbach, and J. Vygen, "BonnTools: Mathematical innovation for layout and timing closure of systems on a chip," *Proc. of the IEEE*, vol. 95, pp. 555–572, 2007.