Optimal Transistor-Level Layout of Single- and Multi-Row Cells for 2nm and Beyond

STEFAN HOUGARDY, University of Bonn, Germany
BENJAMIN KLOTZ, University of Bonn, Germany
MALTE SCHÜRKS, University of Bonn, Germany
ARMIN SETTELS, University of Bonn, Germany
TOBIAS WERNER, IBM, Germany

The advent of 2nm gate-all-around (GAA) technology introduces significant challenges for standard-cell layout. Design rules have become increasingly complex, routing resources are severely constrained due to shrinking circuit row heights, and pin accessibility remains a critical bottleneck. Cells are nowadays often built as multi-row cells, which, together with transistor folding, significantly enlarges the search space for transistor placement. At the same time, layouts must meet stringent PPA requirements.

We present an automated layout flow that addresses all of these challenges simultaneously by integrating a SAT-based formulation for routing and pin accessibility into a branch-and-bound placement framework. Our method not only satisfies the full set of design constraints, but also provides a formal guarantee of optimality in a mathematically rigorous manner. For very large multi-row cells, such as local clock buffers that can contain more than 200 FETs, we propose a block-based floorplanning method. Our approach has been used to generate the layout of all cells in a library comprising several hundred standard cells, with sizes ranging from 2 to 160 FETs, and is deployed in the production of a high-end processor fabricated using 2nm GAA technology. We present experimental results based on the NS3K PDK, demonstrating that our algorithm achieves up to 30% area reduction while maintaining practical runtimes suitable for full-library generation. In addition, we propose a modification to the NS3K PDK that more closely reflects industrial 2nm nodes, and we provide a publicly available, extended benchmark suite of cells comprising up to 141 FETs.

CCS Concepts: • **Hardware → Electronic design automation**.

Additional Key Words and Phrases: standard cell design, cell layout, transistor placement, cell routing

1 Introduction

The 2nm GAA technology introduces substantial challenges for standard-cell layout. Design rules are significantly more complex than in previous technology nodes, and in-cell routing resources are severely constrained. In particular, only four signal tracks and at most three unidirectional routing layers are available within a single-row cell (see Fig. 1), making multi-row cells [GL25] increasingly prevalent. Consequently, constructing area-optimal layouts that are routable, satisfy all design rules, and provide

Authors' Contact Information: Stefan Hougardy, University of Bonn, Bonn, Germany, hougardy@or.uni-bonn.de; Benjamin Klotz, University of Bonn, Bonn, Germany, benkl@posteo.de; Malte Schürks, University of Bonn, Bonn, Germany, schuerks@dm.uni-bonn.de; Armin Settels, University of Bonn, Bonn, Germany, settels@dm.uni-bonn.de; Tobias Werner, IBM, Ehningen, Germany, tobias.werner@de.ibm.com.

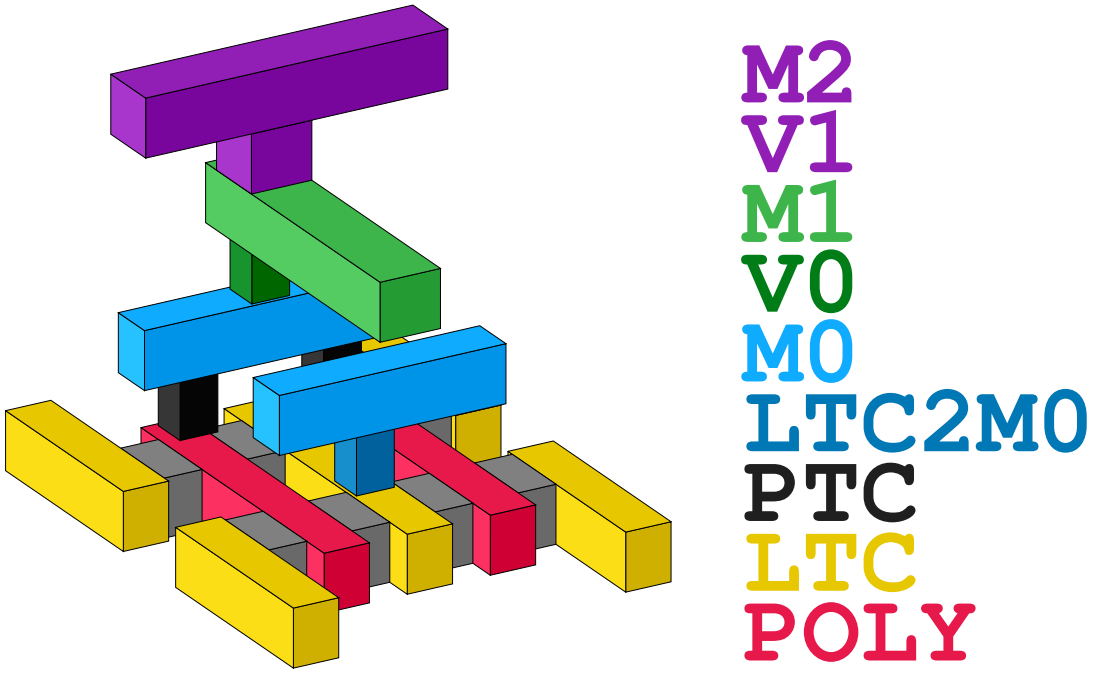


Fig. 1. Layers in the NS3K PDK

good pin accessibility has become a highly nontrivial task. In addition, design-for-manufacturability (DFM) and timing constraints must be considered. The growing complexity of the layout process renders manual design approaches impractical. At the same time, existing automated methods may fail on some instances [HKK+25, GL25, BK24, RF21] or yield suboptimal results [HHF+23, GL25, VCHSW20] if they are unable to address all constraints simultaneously.

In this paper, we present a fully automated flow for standard-cell layout that integrates transistor folding, routing, pin accessibility, and design-rule compliance directly into transistor placement. This tight integration enables our method to guarantee the construction of area-optimal layouts that satisfy all design rules, minimize (weighted) net length, and ensure pin accessibility. Furthermore, our routing framework naturally supports optimization for DFM and timing objectives.

The cell layout problem can be divided into two main steps: transistor placement and in-cell routing. Transistor placement involves the folding, row assignment, and placement of single fingers while in-cell routing must obey all design rules and guarantee pin-accessibility. Moreover, optimizing netlength and satisfying DFM constraints are part of the routing step. Several exact approaches for the transistor placement problem have been suggested. They are mainly based on SMT (satisfiability modulo theory), branch-and-bound, or ILP (integer linear programming) [CSC+24a, VCHSW20, CSC+24b, SLL26]. Many of these exact placement approaches use some kind of routing estimate to evaluate the routability of a given placement. Such approaches may fail as long as they do not exactly model *all* design rules. Fig. 8 gives an example showing that a routing estimate that does not take the M0-min-area rule into account fails to correctly predict the routability of a placement. Exact routing approaches have been

suggested; they are often based on SMT or ILP [SLL26, VCHSW20, BK24]. Fully integrating an exact routing algorithm into an exact placement algorithm is a challenging task, as the runtime becomes prohibitively large. For single-row cells, such an approach has been described in [VCHSW20]. However, their approach cannot guarantee the optimality of layouts for multi-row cells, as they precompute an assignment of the FETs to rows and then apply their single-row algorithm. Thus, to the best of our knowledge, our approach is the first to compute layouts of cells that guarantee optimality in the sense that provably no smaller layout can exist that satisfies all design rules and ensures pin accessibility. Our approach also allows the incorporation of additional constraints; for example, it can compute an area-optimal layout under the restriction that no M2 is used. A key component of our approach is a sophisticated branch-and-bound placement algorithm in combination with a grid-less SAT-based routability check. With this approach, we can, for example, build the 2-row latch `DFF_X2` with 28 FETs (see Table 2) in 60 seconds and simultaneously prove that it is netlength-optimal using 9 tracks.

For substantially larger cells (e.g. local clock buffers (LCBs) which may contain more than 200 FETs and are built using four or six rows) proving optimality is no longer feasible due to the significant increase in runtime. For such large cells we present a block-based branch-and-bound mode and a floorplanning mode. The latter is especially useful for optimizing the timing of an LCB.

The main contributions of this paper are as follows

- Overall flow
  - Full integration of a SAT-based routability test (including pin accessibility) into the placement algorithm
  - Largely technology-independent, easily adaptable, and thus well suited for design technology co-optimization
  - Support for multi-row cells
- FET placement algorithm
  - A branch-and-bound based placement algorithm that computes a global optimum, even for multi-row cells, while satisfying all design rules, routability, and pin accessibility.
  - Support for additional constraints, different objective functions, and FET folding
  - Block-based branch-and-bound and floorplan-based placement for industrial instances with up to 200 FETs
- Routing and pin accessibility algorithms
  - Shape-based, grid-less routing
  - Exact ILP formulation ensuring design-rule compliance
  - Fast SAT-based routability checking
  - Guaranteed simultaneous pin accessibility
  - Support for inner net splitting and DFM optimization
- New instances and more realistic design rules
  - Adapted NS3K PDK reflecting industrial 2nm nodes
  - 16 new benchmark cells with up to 141 FETs

2 Placement and Routing

2.1 Core algorithms

Most algorithms presented in later sections are based on two core concepts: an accurate ILP formulation of the routing problem and a fast branch-and-bound placement algorithm.

Our ILP formulation for the routing problem uses a variation of the multicommodity flow formulation for the Steiner tree packing problem [Won84]. New in our approach is that we do not use a grid graph derived from the metal layers, but we introduce the new concept of *routing shapes*. These allow an exact modeling of the design rules and are used to automatically derive the routing graph, see Section 2.3.1 for more details.

Our branch-and-bound placement algorithm follows [VCHSW20]. Given a cell width W, it determines whether a routable placement of width W exists. If so, it can optimize any objective that can be computed efficiently. A simple example for such an objective is the weighted bounding-box netlength (BBNL). In addition to this, we penalize misaligned gates by adding one gate pitch of netlength for each. Note that we do not force all gates to be aligned as for some cells (e.g. the `MUX_X1` in the NS3K data set) better solutions (in terms of area) can be obtained if not all gates are aligned. For placements with the same netlength value, we compare the *max cut value* which is the maximum number of nets that have to cross any vertical line through the cell. A small max cut value strongly correlates to the M2 usage of the final routing. The cell width W is optimized in an outer loop. Folding is integrated by splitting all FETs into single fingers before applying the algorithm.

During the branch-and-bound search for placements, we perform limited routability checks (taking some design rules into account) on partial placements and full checks (taking all design rules into account) on complete placements. These partial checks prune large parts of the search space, especially under complex design rules. We further prune using width bounds from [VCHSW20]. For a set $\mathcal{F}$ of FETs of the same type (P/N), these bounds give lower bounds for the number of gaps $\text{gaps}_1(\mathcal{F})$ needed to place them in a row, considering only diffusion sharing. These bounds extend naturally to additional technology constraints, such as sheet-width-dependent gap sizes.

2.2 Placement Algorithms

In the following sections we describe how we extend the branch-and-bound placement algorithm to multi-row cells (Section 2.2.1), and we present three additional algorithms (sections 2.2.2, 2.2.3, and 2.2.4), which are used to place large cells. Moreover, we explain inner net splitting in Section 2.2.5 and our placement post-optimization in Section 2.2.6.

2.2.1 Multi-Row Branch-and-Bound. We generalize the single-row branch-and-bound algorithm from [VCHSW20] to multi-row cells. The key observation is that there is an injective mapping ψ between placements in an n-row cell of width w and placements in a single-row cell of width $n \cdot w$ where transistor contacts are always allowed to share at positions that are a multiple of w (see Fig. 2). Based on this mapping, the enumeration of placements proceeds analogously to the single-row case, with the only differences being that diffusion sharing is permitted at positions corresponding to multiples of w and no FET is allowed to have its leftmost contact left and its rightmost contact right of such a position. Note that these modifications affect only the enumeration of partial placements; routability checks are still performed with respect to the original n-row cell.

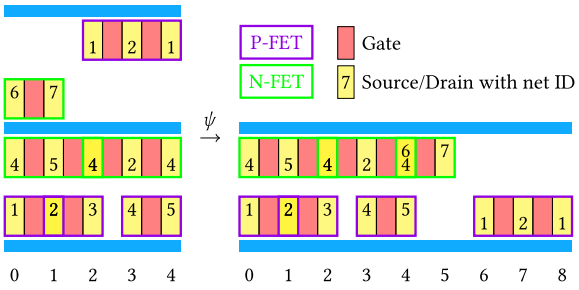


Fig. 2. Mapping of a 2-row placement of width 4 consisting of 4 P-FETs and 3 N-FETs to a 1-row placement of width 8. At width 4 in the 1-row cell, two contacts with different nets share, but this is allowed since the contacts do not overlap in the original 2-row cell.

As in the single-row branch-and-bound algorithm, we employ width lower bounds. However, these bounds cannot be applied directly due to the relaxed sharing constraints at multiples of w. Using ψ, we obtain that for a set of FETs $\mathcal{F}$ of the same type,

$$\mathrm{lbgaps}_n(\mathcal{F}) \coloneqq \max(0, \mathrm{gaps}_1(\mathcal{F}) - (n-1)) \tag{1}$$

is a valid lower bound on the number of gaps required to place $\mathcal{F}$ in n rows without overlap, except for allowed sharing.

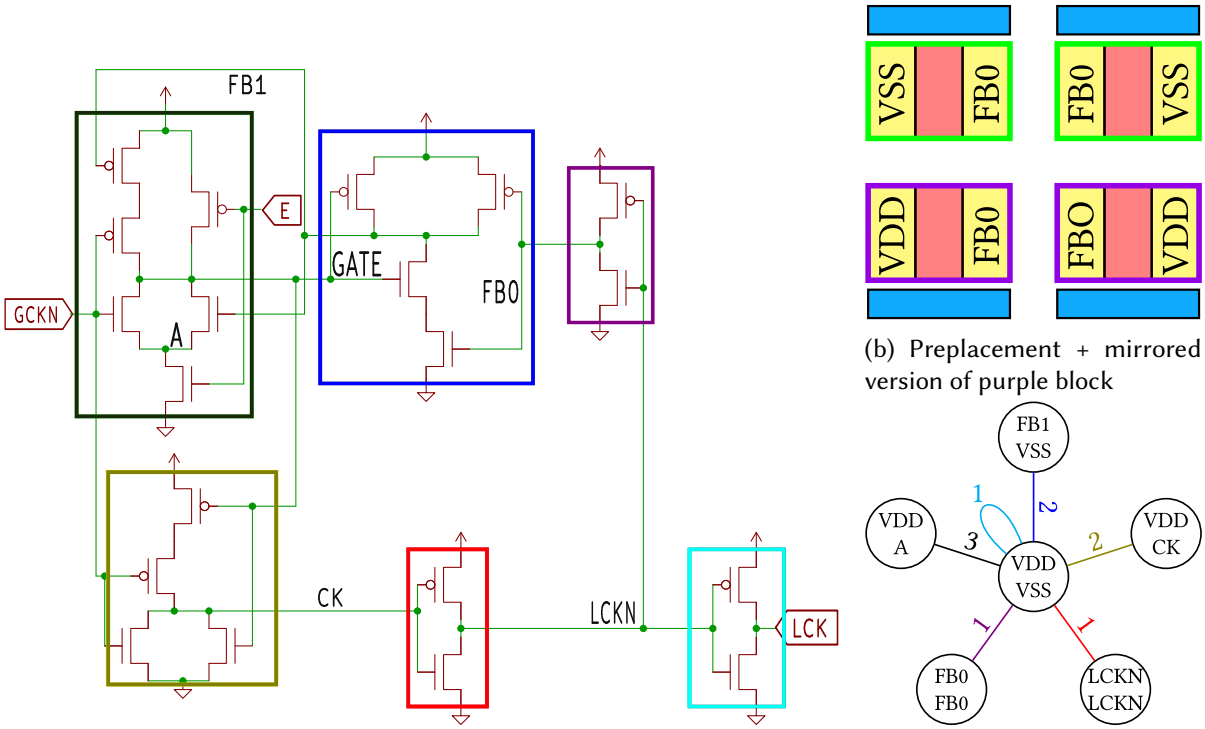
This approach can also be directly applied in SRAM design, where a circuit row may contain only N-FETs instead of being split into N- and P-FETs.

2.2.2 Linear Arrangement Placer. A common approach for handling larger cells is to heuristically partition them into smaller subcells that are then placed next to each other. In contrast, our linear-arrangement–based algorithm does not fix the full partitioning upfront, but determines the decomposition structure dynamically.

We first compute a minimum-cut linear arrangement of the FET/net hypergraph using dynamic programming [Ham19], motivated by the correlation between the maximum cut and M2 usage. Based on this ordering, we iteratively construct placements for prefixes ending at each FET F_i. Given placements up to each FET F_j with $j < i$, a placement up to F_i is obtained by combining a placement of the interval $[F_j, F_i]$ with the placement up to F_j. Interval placements are computed using our branch-and-bound placer with a small time limit. Using the generalization from Section 2.2.1, this approach naturally supports subcells spanning multiple rows.

This method is particularly effective for latches and flip-flops, where the linear arrangement reflects circuit stages. It can be further improved by heuristics, such as avoiding splits that break inverters or optimizing a soft-max over all cuts instead of only the maximum cut to reduce overall netlength.

2.2.3 Block-Based Branch-and-Bound. The width bound presented in Section 2.2.1 has one serious drawback that limits its performance on larger cells: It does not consider whether the FETs can be packed in an n-row cell of width W. Consider for example the N-FETs in Fig. 2. Two of them have width two, one has width one and one sharing gap is needed in a single-row placement. Nevertheless, they cannot be packed in two rows of width three.



(a) Block Decomposition

(b) Preplacement + mirrored version of purple block

(c) *SG* with weights

Fig. 3. Block decomposition and corresponding sharing graph

Our bottom-up approach to address this problem begins by decomposing the netlist into functional blocks according to [LM89] (see Fig. 3a). These blocks typically represent fundamental CMOS gates that should not be split in the layout. We use a modified version of their procedure to account for additional structures appearing in some industrial designs.

For each block, we compute a routable single-row preplacement using the FET-level branch-and-bound placer. These placements are fixed up to mirroring (see Fig. 3b), and we assume that the outer source/drain contacts are aligned between N- and P-FETs. Placement is then performed at the block level using branch-and-bound, enabling diffusion sharing between blocks while preserving routability.

Following the lower-bound techniques for odd-fingered FETs described in [VCHSW20], we construct a sharing graph *SG* that captures which blocks can share diffusion contacts. Each vertex encodes the interface properties of a block boundary (e.g. nets, sheet width, and VT level for both P- and N-FETs). For each block, we add an edge between the vertices representing its left and right boundaries, weighted by the block's preplacement width.

In practice, the sharing graphs of latches are often acyclic except for loops. This structure enables stronger lower bounds than those given by Eq. (1). For simplicity, we first consider the acyclic case and discuss the extension to loops later. We also assume a minimum gap of 1 between non-sharing blocks; the approach generalizes to larger gaps. Given the remaining space $R_1, \ldots, R_b$ in each row, deciding

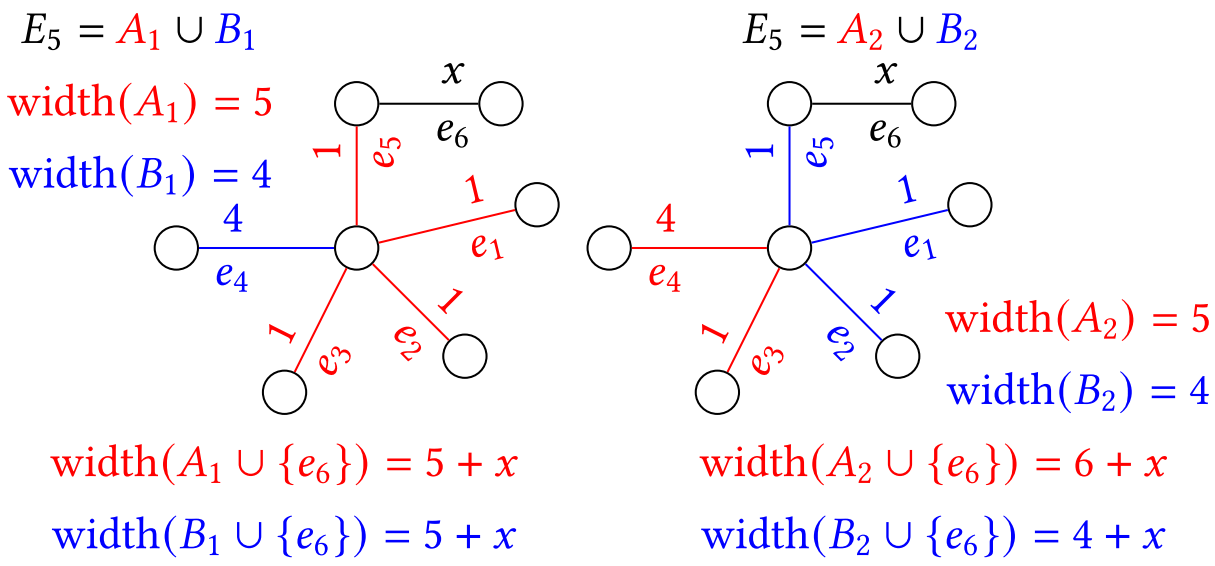


Fig. 4. Edge weights are indicated above the edges. Both partitions of the set $E_5 = \{e_1, \ldots, e_5\}$ need to be considered when adding e_6 due to different parities.

whether the blocks fit into the remaining space can be modeled as a graph-theoretic problem.

***b*-Rows Acyclic Lower Bound Problem** for $b \in \mathbb{N}$
Given: Acyclic graph $SG = (V, E), \omega : E \to \mathbb{N}, R_1, \ldots, R_b \in \mathbb{N}_0$,
Question: Is there a partition $P = \{P_1, \ldots, P_b\}$ of E such that

$$\text{width}(P_i) \leq R_i \text{ for all } i \in \{1, \ldots, b\}?$$

Here, $\text{width}(E')$ of a set $E' \subseteq E$ is $\omega(E') + \max(0, \text{mwp}(E') - 1)$, where $\text{mwp}(E')$ denotes the size of a minimum walk partition of E'. For $E' \subseteq E$, denote by $\text{odd}(E')$ the odd degree vertices in the graph (V, E'). For a partition P of E, we define $\omega(P)$, $\text{odd}(P)$, and $\text{width}(P)$ as the vectors obtained by applying the functions to each set of P. Since SG is acyclic, we have

$$\text{mwp}(E') = \frac{|\text{odd}(E')|}{2}, \tag{2}$$

Therefore, we can compute $\text{width}(P)$ from $\text{odd}(P)$ and $\omega(P)$.

Assume that E is ordered as $\{e_1, \ldots, e_{|E|}\}$, and define $E_j := \{e_1, \ldots, e_j\}$. Consider a partition P of E_{j-1} and the partition P' obtained by adding e_j to the k-th set P_k of P. Clearly, $\omega(P')$ is equal to $\omega(P)$ except for the k-th entry, and the same is true for $\text{odd}(P')$. We have $\omega(P')_k = \omega(P)_k + \omega(e_j)$ and $\text{odd}(P')_k = \text{odd}(P)_k \triangle e_j$, i.e. exactly the endpoints of e_j change their parity. In particular, the actual partition P is not required to compute the values, they only depend on $\omega(P)$ and $\text{odd}(P)$.

The dynamic program Algorithm 1 iteratively computes

$$W_j := \{(\omega(P), \text{odd}(P)) : P \text{ partition of } E_j\}. \tag{3}$$

by checking for all elements $(w, o) \in W_{j-1}$ how w and o change if the corresponding partition of E_{j-1} is extended to a partition of E_j by putting e_j into any of the b sets.

Algorithm 1 OrderedEdgesDynamicProgram

Input: Instance $(SG = (V, E, \omega : E \longrightarrow \mathbb{N}), b, R_1, \ldots, R_b)$ of the b-Rows Acyclic Lower Bound Problem
Output: Returns $W_{|E|}$

```
W_0 ← {((0, ..., 0), (∅, ..., ∅))}                    // b zeros and b empty sets
for e_j = e_1, ..., e_|E| do
    W_j ← ∅
    for (w, o) ∈ W_{j-1} do
        for i ∈ {1, ..., b} do
            (w', o') ← Update_ω_and_odd(SG, i, (w, o), e_j)
            if w'_i ≤ R_i then
                W_j ← W_j ∪ {(w', o')}
return W_|E|
```

Note that a vertex v is never contained in $\mathrm{odd}(P)$ for a partition P of E_j if no edge incident to v is present in E_j. Also note that if instead all edges incident to v are present in E_j, it cannot be removed from or added to $\mathrm{odd}(P')$ when adding e_{j+1} to some set of P. By storing an incrementally computed version of $\mathrm{width}(P)$ instead of $\omega(P)$, we can omit these entries from $\mathrm{odd}(P)$. With a well-chosen edge order, the number of entries of $\mathrm{odd}(P)$ stored in each iteration can be bounded by $(b-1)\log_2(|V(SG)|)$. For fixed b, this yields a pseudopolynomial runtime:

Theorem. *The b-Rows Acyclic Lower Bound Problem can be solved in time $O(|E(SG)||V(SG)|^{b-1} b \Pi_{i=1}^{b} R_i)$*

Since this algorithm is usually applied on cells with at most twenty blocks, this is fast enough as a lower bound computation.

This can be generalized to graphs that are acyclic except for loops. In this case Eq. (2) generalizes to

$$\mathrm{mwp}(E') = \frac{|\mathrm{odd}(E')|}{2} + |\{C : C \text{ is loop component of } (V, E')\}|,$$

where a loop component is a connected component that consists of a single vertex with at least one loop. Define

$$\mathrm{loops}(P)_{1 \le i \le b, v \in V}, \mathrm{edg}(P)_{1 \le i \le b, v \in V} \in \{0,1\}^{b \times V}$$

to indicate whether a vertex has an incident loop or non-loop edge in the corresponding set of the partition. Then, $\mathrm{width}(P)$ can be computed from $\omega(P)$, $\mathrm{odd}(P)$, $\mathrm{edg}(P)$ and $\mathrm{loops}(P)$. The update steps for $\mathrm{loops}(P)$ and $\mathrm{edg}(P)$ are similar to those for $\omega(P)$ and $\mathrm{odd}(P)$, allowing a slightly modified algorithm to be used to compute the bounds.

2.2.4 Floorplanning. For very large cells, such as the LCBs described in Section 3.2, enumerating all placement choices of each block becomes impractical. Moreover, blocks in LCBs often exhibit imbalances between P- and N-FETs, making a block-based branch-and-bound approach ineffective. To address this, our floorplan-based placement mode decomposes the problem into the following steps.

As in the block-based branch-and-bound approach, we first compute *preplacements* for each block. The algorithm dynamically partitions the cell into subcells to ensure that a placement is found within the time limit. Unlike in the block-based branch-and-bound algorithm, the exact preplacements have little impact on the final layout. They will only be used to provide estimates on block sizes and the required

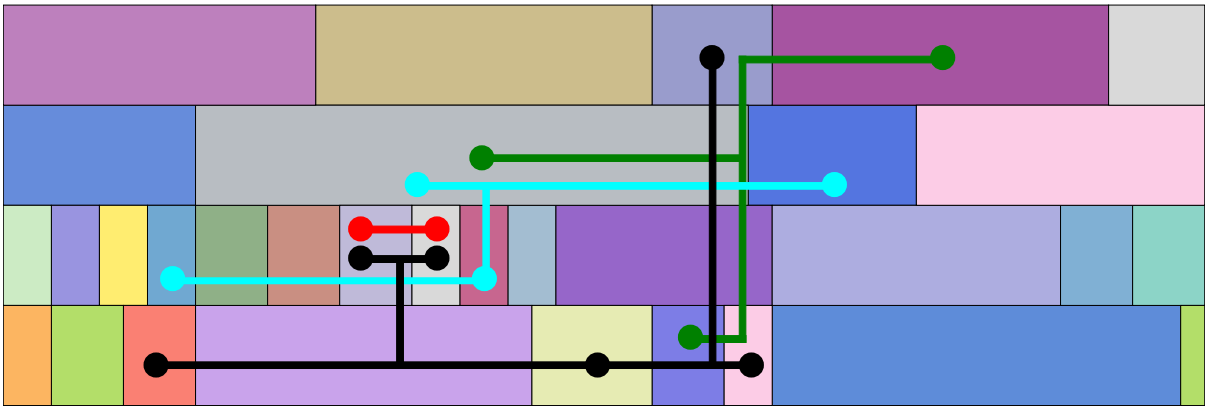


Fig. 5. Floorplan and global routing of selected nets of a local clock buffer. When solving the gray block (third row), the virtual pins for the blue, green, and black nets from below are fixed. New virtual pins are created to the top (black, green) and east (blue).

spacing between blocks. Next, we solve a *floorplanning* problem, formulated as a rectangle packing problem that places blocks while minimizing weighted bounding-box netlength. Net weights can be adjusted to reflect timing criticality. We solve this problem using a standard ILP formulation [SSR91] and a black-box ILP solver. In practice, most runtime is spent on this step. Since solving the ILP to optimality is infeasible for large instances, we allocate 75% of the total placement time budget to it. This is motivated by the observation that higher-quality floorplans significantly simplify subsequent steps and improve overall solution quality.

We then compute a *global routing* for the floorplan. This is a Steiner tree packing problem on a coarsened routing graph that considers only inter-block nets and ignores design rules. The resulting routing defines the overall structure of inter-block connections. Using a multicommodity flow ILP formulation [Won84], this step can typically be solved within seconds. The objective accounts for both global congestion and an estimate of local congestion due to intra-block wiring.

Once the overall structure of the placement and routing have been determined, we compute a DRC- and LVS-clean realization using a subcell-based flow. Subcells are solved sequentially for each row from left to right. Their boundaries are determined based on block sizes and adjusted dynamically to ensure feasibility within the time limit. To guarantee global connectivity, virtual pins are introduced at subcell and row boundaries according to the global routing (see Fig. 5). Each subcell is then solved using the branch-and-bound placer.

Since block widths used in the floorplan are only estimates, the combined routing of the subcells of a circuit row may deviate from the global routing, particularly in the positions of inter-row pins. To address this, we recompute the global routing after processing each circuit row to ensure consistency across rows.

2.2.5 Inner net splitting. A common layout optimization, e.g., in NAND gates, is to fold and interleave different FETs, creating multiple contacts of an internal net that are electrically equivalent even without direct connections. Removing such wiring reduces capacitance and improves routability (see Fig. 6). This generalizes to nets appearing on the source/drain of exactly two FETs of the same type and size.

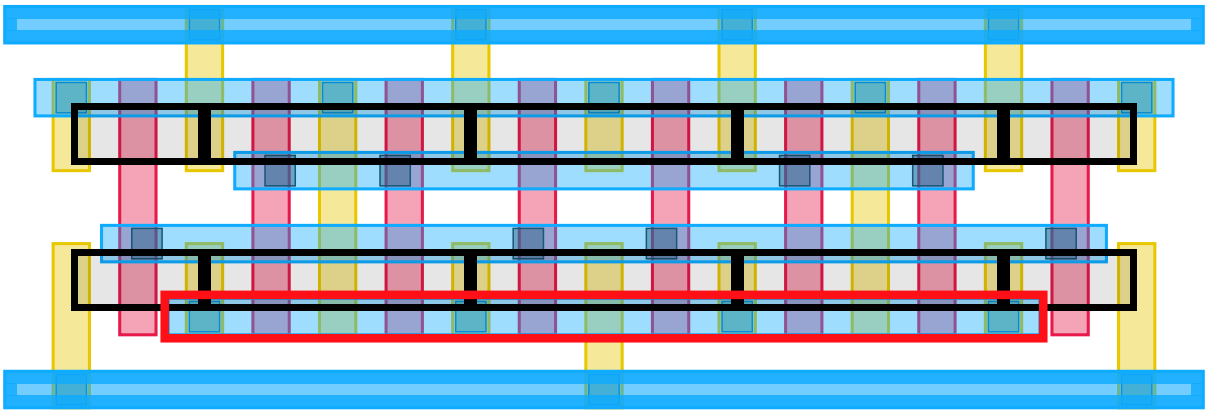


Fig. 6. Inner net splitting on NAND2: The highlighted M0 segment never carries any current and can be removed

Industrial LVS tools support this optimization.[1]

During placement, we promote such configurations by modifying the netlength objective: each eligible net is split into two-terminal connections between corresponding fingers such that the cost of a minimum perfect matching on these is minimized. Compared to using the bounding-box netlength this penalizes long internal connections and favors diffusion sharing.

During routing, we partition the source/drain terminals of the net into balanced components, i.e. components that do not need to be connected to each other. We modify our multicommodity flow formulation to allow for multiple roots and compute a Steiner forest connecting each of the balanced components. The balanced components are not required to be connected to each other, but may be connected if this reduces netlength.

2.2.6 Postoptimization by local search. Since routability can have hard-to-predict effects on subcell width (see Fig. 8), our subcell-based algorithms may produce suboptimal results in some cases. To mitigate this, we apply a local search step after the main placement phase to further reduce cell width. In each iteration, we generate up to 10 000 candidate placements by moving one or more groups of FETs and select the best routable solution among them. For these routing checks, all nets not incident to or near the moved FETs are kept fixed.

2.3 Routing

In the following sections, we describe our routing shape based graph and ILP formulation (Section 2.3.1) that encodes all connectivity, design rule (Section 2.3.2) and pin accessibility constraints (Section 2.3.5). In addition, we use SAT solvers to speed up the computation of routability checks (Section 2.3.3) and in practice near-optimal routing computations (Section 2.3.4).

2.3.1 Routing shapes. Our specialized routing framework based on routing shapes [Klo23] allows us to quickly adapt to changing design rules. A *routing shape* is a rectilinear polygon that represents a piece of wiring in the final routing (see Fig. 7). It has the following additional properties:

[1]For example, `short_equivalent_nodes()` in Synopsys ICV and "Split Gate Reduction" in Siemens Calibre

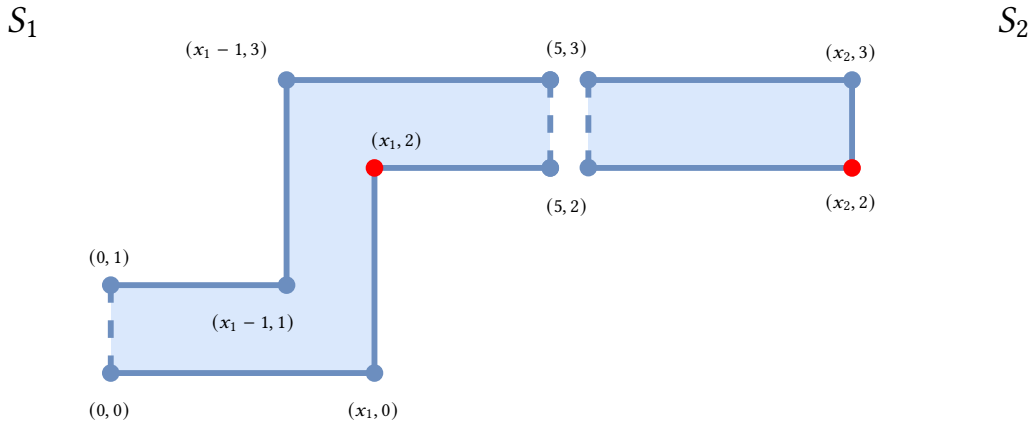


Fig. 7. Two routing shapes. S_1 is an S-shaped wire with a variable jog position (x_1); S_2 has a variable horizontal endpoint. Dashed edges are incomplete and can be merged.

- For each routing shape r, we add a binary activation variable x_r to the ILP, which indicates if the routing shape is part of the routing solution.
- Coordinates of its vertices may be expressed using additional ILP variables. This allows us to define flexible polygons that can change their exact shape based on design rule requirements.
- Edges of the polygons may be marked as incomplete. The final wiring will consist of multiple routing shapes that are merged together at their incomplete edges so that no incomplete edges remain. We require that the coordinates of incomplete edges are constants (i.e. they do not depend on ILP variables). This allows us to check if two routing shapes can be merged together before obtaining an ILP solution.

We automatically define a set of routing shapes $\mathcal{R}$ such that every routing satisfying all design rules can be represented. Conversely, any routing that cannot be represented violates at least one design rule. This guarantees that the ILP solver optimizes over the full space of DRC-clean routings. To reduce runtime and memory usage, we choose $\mathcal{R}$ as coarse as possible while maintaining this property [Klo23].

At the heart of our ILP formulation we utilize the multicommodity flow formulation for the Steiner Tree Packing problem on a routing graph $\mathcal{G} = (V, E)$ [Won84, VCHSW20]. This formulation ensures that an ILP solution meets all connectivity constraints. We automatically deduce $\mathcal{G}$ from $\mathcal{R}$ by setting $V \coloneqq \mathcal{R}$ and adding an edge between two routing shapes if and only if they can be merged together. A solution to the Steiner Tree Packing problem in $\mathcal{G}$ then corresponds to a set of routing shapes that, merged together, forms the final routing. To reduce the size of $\mathcal{G}$ and the ILP formulation, we can identify subsets $U \subset \mathcal{R}$ of routing shapes that pairwise contradict each other because of design rules. For such cases, we contract the contradicting vertices in $\mathcal{G}$ to a single vertex U with a binary activation variable v_U and add the constraint $v_U = \sum_{u \in U} x_u$.

2.3.2 Design Rules. Our design rule framework adds additional constraints to the ILP, so that if a routing shape r is active (i.e. $x_r = 1$), it fulfills all design rules. To enable a quick implementation of design rule changes, we have split this into two subtasks.

Query functions gather very small subsets of $\mathcal{R}$ that could potentially violate design rules. This approach is feasible, because most real-world design rules can be checked locally on a complete routing, i.e. on very small subsets of $\mathcal{R}$. For example, consider the two red vertices in Fig. 7. A design rule (e.g. "Inner vertex to line end spacing") may require the distance between these to be at least λ. A corresponding query function will enumerate small subsets of $\mathcal{R}$, like $\{S_1, S_2\}$, the two shapes shown in Fig. 7, which together could violate the design rule.

Subsequently, *checking functions* will process the results of the query functions, namely the small subsets of $\mathcal{R}$. To ensure DRC-clean routing, they will add additional constraints to the ILP. For the example above, a checking function will add a constraint $x_{S_1} \wedge x_{S_2} \implies x_2 - x_1 \geq \lambda$.

Splitting the design rule implementation into query and checking functions mimics the notation used in real-world design rule manuals, offering two major advantages:

- Users are able to explore how different design rules affect layout quality by easily modifying and recombining query and checking functions, allowing for effortless design technology co-optimization.
- The basic design rule notions (e. g. "inner vertex" or "distance to" in the example above) typically don't change much between different industrial technologies. As our query and checking functions closely follow this notation, we can adapt to new technologies very quickly. For example, the initial implementation of the NS3K rules was finished in a few hours, most of which was spent on the image definition and changes to the FEOL structure. Industrial nodes are typically more complex, but also for these the adoption time has been reduced from months to few weeks.

The system can easily be extended to support DFM rules. This is done by introducing slack variables into the corresponding constraints, and adding these variables to the objective function.

2.3.3 SAT-based routability checks. During the branch-and-bound placement algorithm we check many placements for routability, sometimes far more than 10^8. A large fraction of these can be excluded by only considering relatively basic rules, but it can still be necessary to perform more than 10^7 full routing checks. In [VCHSW20], these checks are performed using an ILP solver despite the fact that it is not necessary to compute a good routing: It is enough to show that some DRC-clean routing exists.

Following [PLK+20], our implementation uses a boolean satisfiability (SAT) solver to perform these checks instead, drastically reducing the placement runtime. To reduce maintenance effort and ensure consistency, we automatically generate the SAT instance from the ILP instance. For this, the few non-binary variables in our ILP are first replaced by a series of binary variables, each representing a power of two in the binary representation of the variable value. Finally, we apply the algorithm described in [ARMS02, Sec. 3.1] to obtain a SAT instance.

Since we check many, often similar, placements for feasibility, we use incremental SAT solving [BHvMW21] during placement. This requires us to modify our routing formulation so that the location of each FET is determined by variables. This increases the complexity of the ILP/SAT instance and hence the time for solving the instance once. However, the ability of the SAT solver to reuse learned clauses during further routing checks strongly outweighs this increase on all nontrivial instances.

2.3.4 Decision-guided routing. The routings computed by a standard SAT solver are fully DRC- and LVS-clean, but typically exhibit extremely large netlength, making them impractical beyond proving feasibility. ILP solvers generally fail to compute feasible routing solutions for larger cells in reasonable

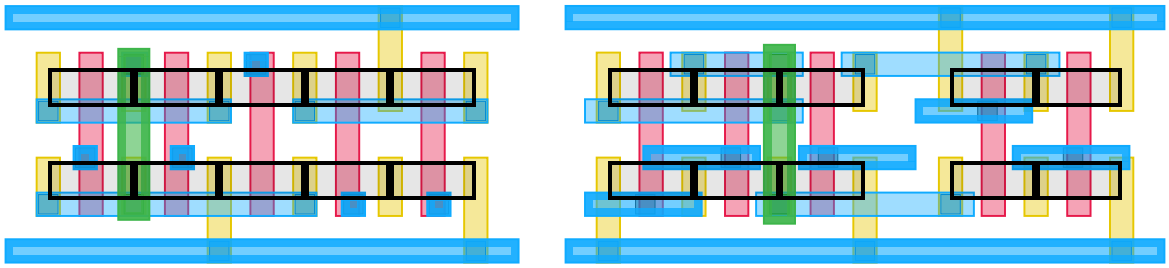


Fig. 8. Minimum-width AOI221 layouts: original NS3K PDK (left) and our modified NS3K-B PDK (right).

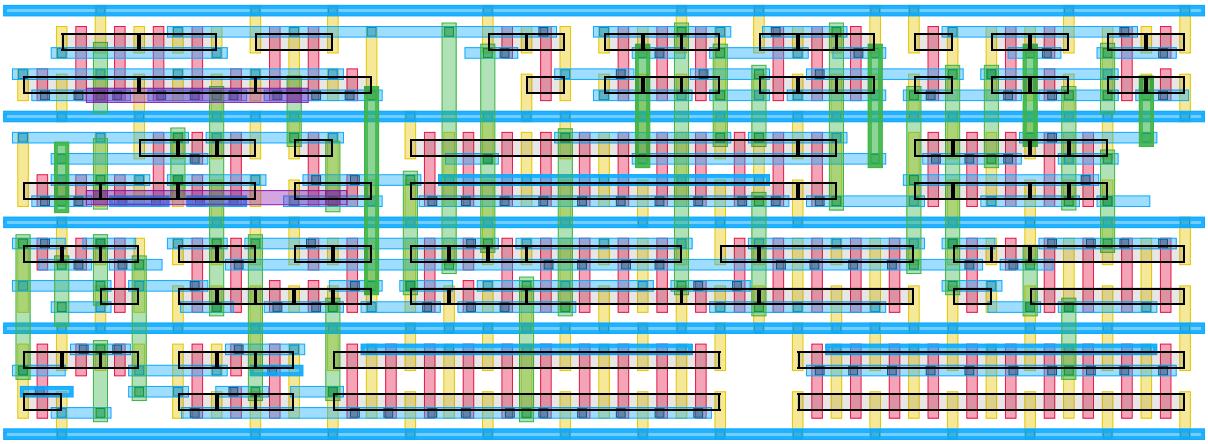


Fig. 9. Example layout of our LCBLES instance.

time. Ripup-and-reroute approaches also typically fail on these very dense layouts since they are unable to resolve the design rule conflicts. Decision-guided routing based on [Nad16] offers a compromise between ripup-and-reroute strategies and SAT solvers: The routing step is integrated into a CDCL-based SAT solver [BHvMW21] as a decision heuristic, and can provide the SAT solver with information about blocked cuts in the graph. The SAT solver's own conflict analysis acts as the ripup strategy. The completeness of the CDCL algorithm guarantees that a solution will be found in finite time if one exists. At the same time, the routing-aware decision strategy allows the solver to find short routes for many nets, while also speeding up the routing process in many cases. While the routings computed by this strategy are often not good enough to be used directly, they can serve as a starting point for further optimization. For example, it is possible in practice to reach near-optimal solutions by repeatedly optimizing small windows within the cell using an ILP solver.

Since the algorithm requires access to various internal functions of the SAT solver, we use a custom basic CDCL implementation instead of an existing SAT solver. The performance of the SAT solver does not seem to be overly critical for the overall routing runtime since most time is spent on Steiner tree computations.

2.3.5 Pin Accessibility. We ensure that all pins of our cells are accessible by requiring all external nets to be connected to M1 during routing. These temporary M1 connections are then removed before the final layout is written. Since M2 is only rarely used, this is typically sufficient to guarantee simultaneous accessibility of all pins of a cell by a detailed router.

3 Experimental Results

3.1 Technology nodes

We report three sets of experimental results. The first two are based on publicly available PDKs and netlists, as described below. The third set uses the Samsung Foundry 2nm technology [JKP+24] and we briefly report the results in the conclusion.

For the first set of experiments, we use the predictive NS3K technology node [KJW+23]. However, its design rules are overly optimistic compared to industrial nodes, making routing unrealistically easy. For example, NS3K uses minimum-area constraints equal to a square of minimum track width, which is significantly smaller than in practice. We therefore introduce a modified node, NS3K-B, with much stronger min-area, via spacing, and via overhang rules than in NS3K. In addition, we allow LTC to connect the N- and P-regions of a circuit row, a feature found in several industrial nodes. Fig. 8 illustrates example layouts of the same cell in NS3K and NS3K-B, showing that stricter M0 minimum-area constraints increase the minimum achievable cell width.

Although NS3K-B still employs simplified design rules compared to industrial technologies, we believe its routing complexity is representative of real-world nodes. The placement complexity remains somewhat lower, as the FET models support only a single sheet width and a single threshold voltage.

3.2 Test instances

The netlists used for the predictive technologies are drawn from three sources. First, we use the 49 netlists provided to us as part of the NS3K PDK. Second, we include all 208 netlists from the ASAP7 PDK [CVS+16], converting FET sizes by assuming the maximum number of fins per finger. Third, we add 14 netlists from prior work, including radiation-hardened latches [CNV96], low-power latches [WSB+25], and pulsed local clock buffers [WCH+14]. In terms of complexity, the additional latches are comparable to or slightly more challenging than those in ASAP7, while the LCBs are significantly larger. Our largest instance contains 141 FETs, 242 active fingers, and 89 nets across four circuit rows, with many misaligned FET sizes due to tight timing constraints.

Some cells are built at multiple heights, resulting in 52, 229, and 16 layouts from NS3K, ASAP7 and our new instances respectively. We categorize our testset into 215 logic cells, 49 latches/flip-flops, and 33 clocking-related cells. 240 of the cells are built as single-row cells, 52 as double-row cells, and 5 as quadruple-row cells.

Alongside SPICE netlists and GDS layouts, we provide JSON files describing transistor placements. The dataset is publicly available at https://doi.org/10.60507/FK2/9HPTQB.

3.3 Experimental setup

We apply different algorithms to different cell categories to benefit from the layout characteristics exhibited by the categories.

Logic cells are solved using our branch-and-bound algorithm. For multi-row cells, we apply the multi-row variant (Section 2.2.1). In NS3K-B, we recursively decompose the cell when the runtime becomes prohibitive due to the more complex design rules.

Latches/flip-flops are solved using three approaches: the linear arrangement placer (Section 2.2.2) with single-row and double-row subcells, and the block-based branch-and-bound method (Section 2.2.3).

Clocking cells are solved using the floorplanning approach (Section 2.2.4).

We allow folding only for the logic cells. All runs use the decision-guided router with ILP-based local post-optimization. Placement is limited to 1000 s (logic cells) and 8 h (latches and clocking cells), while routing is limited to 300 s and 4 h, respectively. For placements composed of multiple subcells, we apply the local search algorithm described in Section 2.2.6.

All experiments use 16 threads on an AMD EPYC 9684X with 2.3 TB RAM. ILPs are solved with CPLEX 22.1.1.0, SAT instances with CaDiCaL 1.5.2, and decision-guided routing with a custom solver. Each run is limited to 64 GB RAM.

All layouts are verified using Synopsys ICV to ensure compliance with LVS and DRC rules.

3.4 Results

Table 1 summarizes our results using NS3K rules on the cells from [KJW+23]. Since the logic cells are built by applying the multi-row branch-and-bound algorithm, their width is the smallest possible. With one exception, their placements are also provably optimal among the minimum-width placements w.r.t. the netlength metric described in Section 2.1 and Section 2.2.5. The algorithms used to build the latches do not provide optimality guarantees. However, running the multi-row branch-and-bound placer proved that no layout of smaller width exists for any of the latches. Notably, none of our layouts require the use of M2.

We compare our results with layouts reported in [KJW+23] and [SLL26]. Since [SLL26] does not cover all cells from [KJW+23] and uses a different number of rows for some, we exclude these cases from the comparison and retain the original number of rows from [KJW+23]. Compared to [KJW+23], we achieve an average cell-width reduction of 7%, with improvements exceeding 30% for some cells. For the subset considered in [SLL26], our method produces layouts that are one track narrower for two cells. Across all 52 instances, our solutions are provably width-optimal, and thus no further improvements are possible.

Under the stricter NS3K-B rules, the minimum feasible cell width increases by one track for 9 of the 52 cells from [KJW+23] due to increased routing complexity. For two cells (`XOR_X1` and `XNOR_X1`), routing requires the use of a single M2 track.

Table 2 reports the solution quality of the linear-arrangement placer with 2-row subcells and the block-based branch-and-bound placer on selected multi-row latches under NS3K-B rules. We also provide a lower bound on the cell width based on the width bound from [VCHSW20], showing that some of our solutions have optimum width. Both the bounds and the optimality guarantee assume that no FET folding takes place. For all 21 multi-row latches in our testbed, at least one algorithm finds an M2-free solution. The linear-arrangement placer with single-row subcells solves only 4 instances, while the 2-row variant solves all 21 instances. The block-based branch-and-bound algorithm solves 18 instances; and when successful, it is typically faster than the 2-row linear-arrangement placer and often yields more compact layouts.

Table 1. Results on the NS3K benchmark cells from [KJW+23]. # = number of cells. Cell widths are in gate pitches. Runtime is the wall-clock time for placement and routing. Latches (*) are solved using the block-based branch-and-bound algorithm.

		Avg. width		Time		Avg. width	
Category	#	NS3K	Ours	Ours	#	SLL26	Ours
AND,OR	7	7.43	**6.43**	20 s	6	5.33	5.33
AOI,OAI	8	5.00	**4.75**	10 s	6	5.00	5.00
BUF,INV	15	13.67	13.67	174 s	12	14.17	14.17
NAND,NOR	8	5.25	5.25	6 s	8	5.25	5.25
XOR,XNOR	4	6.00	**5.50**	19 s	2	6.00	6.00
DFF*	2	10.50	**9.00**	283 s	2	10.00	**9.00**
DL*	2	8.50	**6.50**	69 s	0	N/A	N/A
SDFF*	2	15.00	**12.00**	5 251 s	0	N/A	N/A
HA	1	8.00	**5.00**	30 s	0	N/A	N/A
FA	1	11.00	**8.00**	281 s	1	9.00	**8.00**
MUX	2	8.00	**6.00**	19 s	1	7.00	7.00
All	52	8.96	8.31		38	8.47	8.39

Table 2. Results for selected multi-row latches in NS3K-B. LB: lower bound on cell width. Bold: provably optimal. Superscripts denote the source: A(SAP7), N(S3K), B(onn).

		Cell width with / without M2		
Cell name	#FETs	2-row LA	Block-B&B	LB
SDFFQ_2DICE^{B}	60	23 / 31	– / –	19
XOR_SLAT^{B}	39	**13** / 14	**13** / 14	13
SDFF_X1^{N}	38	13 / 13	**12** / **12**	12
SDFFQ_2ROW^{B}	36	**13** / **13**	**13** / **13**	13
SDFLX4_2ROW^{A}	32	13 / 13	12 / 12	11
DFFASRHQNX1^{A}	32	12 / 14	11 / 12	10
SDFHX3_2ROW^{A}	32	13 / 12	12 / 12	11
DFF_X2^{N}	28	**9** / **9**	**9** / **9**	9

The results in Table 3 for selected large clocking cells demonstrate the effectiveness of our floorplanning mode. It successfully generates layouts for all clocking cells in the testbed, with runtimes that scale well with cell size. Although in these runs M2 usage is not explicitly optimized, placements based on low-congestion global routings typically require only a small number of M2 tracks. Fig. 9 shows the resulting layout of LCBLES.

4 Conclusion

Our standard-cell flow is the first to solve all NS3K instances width-optimally. It also generated a complete DRC- and LVS-clean industrial standard-cell library in Samsung SF2 technology [JKP+24],

Table 3. Results on selected clocking-related cells in NS3K-B

Cell name	#FETs	#Nets	Width	#M2	Runtime
LCBLESMICRO4	141	90	41	5	11.09 h
LCBLES	94	63	34	0	5.25 h
LCBP	73	56	22	0	5.22 h
ICGX2P67DC	56	32	21	2	4.14 h

comprising 195 logic cells, 47 latches, and 19 LCBs with up to 160 FETs in 15 hours. Results for the complete 261-cell SF2 library are similar to those in Table 1. The generated logic-cell and latch layouts have been used directly in industry without manual intervention. LCB quality is comparable to layouts produced by experienced designers over several weeks, although FET size tuning is still required to meet timing targets. Our tool supports this via an ECO mode that minimizes changes to parasitics. To foster future comparisons, we release the stricter NS3K-B design rules, benchmark cells with up to 141 FETs, and our corresponding layouts.

Acknowledgments

We are grateful to Taigon Song for providing the NS3K PDK and to Ting-Xin Lin for providing more detailed results of [SLL26].

References

[ARMS02] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Generic ILP versus specialized 0-1 ILP: an update. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '02)*, pages 450–457, 2002.

[BHvMW21] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition, 2021.

[BK24] Kyeonghyeon Baek and Taewhan Kim. CSyn-fp: Standard cell synthesis of advanced nodes with simultaneous transistor folding and placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(2):627–640, 2024.

[CNV96] T. Calin, M. Nicolaidis, and R. Velazco. Upset hardened memory design for submicron CMOS technology. *IEEE Transactions on Nuclear Science*, 43(6):2874–2878, 1996.

[CSC+24a] Handong Cho, Hyunbae Seo, Sehyeon Chung, Kyu-Myung Choi, and Taewhan Kim. Standard cell layout generator amenable to design technology co-optimization in advanced process nodes. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2024.

[CSC+24b] Sehyeon Chung, Hyunbae Seo, Handong Cho, Kyumyung Choi, and Taewhan Kim. Optimal layout synthesis of multi-row standard cells for advanced technology nodes. In *ICCAD '24: Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pages 38:1–8, 2024.

[CVS+16] Lawrence T. Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. ASAP7: A 7-nm finFET predictive process design kit. *Microelectronics Journal*, 53:105–115, 2016.

[GL25] Kairong Guo and Yibo Lin. Multi-row standard cell layout synthesis with enhanced scalability. In *2025 International Symposium of Electronics Design Automation (ISEDA)*, pages 317–323, 2025.

[Ham19] Thekla Hamm. Finding linear arrangements of hypergraphs with bounded cutwidth in linear time. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, pages 20:1–20:14, 2019.

[HHF+23] Chia-Tung Ho, Alvin Ho, Matthew Fojtik, Minsoo Kim, Shang Wei, Yaguang Li, Brucek Khailany, and Haoxing Ren. NVCell 2: Routability-driven standard cell layout in advanced nodes with lattice graph routability model.

In *ISPD '23: Proceedings of the 2023 International Symposium on Physical Design*, pages 44–52, 2023.

[HKK+25] Inhyeok Hwang, Ahreum Kim, Byungsu Kim, Seonil Choi, Handong Cho, Hyunbae Seo, and Taewhan Kim. 2nm GAA cell architecture exploration driven by automated standard cell layout generator for design-technology co-optimization. In *2025 IEEE 38th International System-on-Chip Conference (SOCC)*, pages 1–6, 2025.

[JKP+24] Dongchan Jeong, Seungkwon Kim, Seulki Park, Sang Hyeon Lee, Sada-aki Masuoka, Byungha Choi, Shincheol Min, Sanghoon Lee, Minseong Lee, Chang-Woo Sohn, Jaehun Jeong, Yuri Yasuda-Masuoka, and Ja-Hum Ku. Product performance aware 3rd generation GAA platform transistor design with extreme small local layout effect and transistor variation. In *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pages 1–2, 2024.

[KJW+23] Taehak Kim, Jaehoon Jeong, Seungmin Woo, Jeonggyu Yang, Hyunwoo Kim, Ahyeon Nam, Changdong Lee, Jinmin Seo, Minji Kim, Siwon Ryu, Yoonju Oh, and Taigon Song. NS3K: A 3-nm nanosheet FET standard cell library development and its impact. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(2):163–176, 2023.

[Klo23] Benjamin Klotz. *Cell Layout Routing*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, August 2023.

[LM89] Wen-Jeng Lue and L.P. McNamee. VLSI leaf cell design by understanding circuit structures. In *Proceedings of the 2nd International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE '89) – Volume 1*, pages 500–508, 1989.

[Nad16] Alexander Nadel. Routing under constraints. In *Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design (FMCAD 2016)*, pages 125–132, 2016.

[PLK+20] Dongwon Park, Daeyeal Lee, Ilgweon Kang, Chester Holtz, Sicun Gao, Bill Lin, and Chung-Kuan Cheng. Grid-based framework for routability analysis and diagnosis with conditional design rules. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):5097–5110, 2020.

[RF21] Haoxing Ren and Matthew Fojtik. Standard cell routing with reinforcement learning and genetic algorithm in advanced technology nodes. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ASPDAC '21, pages 684–689, 2021.

[SLL26] Meng-Yu Shih, Ting-Xin Lin, and Yih-Lang Li. A graph-based approach for optimizing pin access in nanosheet FET standard cell library synthesis. In *Proceedings of the 2026 International Symposium on Physical Design (ISPD '26)*, pages 75–82, 2026.

[SSR91] Suphachai Sutanthavibul, Eugene Shragowitz, and J.B. Rosen. An analytical approach to floorplan design and optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10:761–769, 1991.

[VCHSW20] Pascal Van Cleeff, Stefan Hougardy, Jannik Silvanus, and Tobias Werner. BonnCell: Automatic cell layout in the 7-nm era. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2872–2885, 2020.

[WCH+14] James Warnock, Yuen Chan, Hubert Harrer, Sean Carey, Gerard Salem, Doug Malone, Ruchir Puri, Jeffrey A. Zitz, Adam Jatkowski, Gerald Strevig, Ayan Datta, Anne Gattiker, Aditya Bansal, Guenter Mayer, Yiu-Hing Chan, Mark Mayo, David L. Rude, Leon Sigal, Thomas Strach, Howard H. Smith, Huajun Wen, Pak-kin Mak, C.-L. Kevin Shum, Donald Plass, and Charles Webb. Circuit and physical design of the zEnterprise™ EC12 microprocessor chips and multi-chip module. *IEEE Journal of Solid-State Circuits*, 49(1):9–18, 2014.

[Won84] Richard T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984.

[WSB+25] David Wolpert, Gerry Strevig, Chris Berry, Leon Sigal, Bill Huott, Mark Cichanowski, Matthias Pflanz, Tobias Werner, Philipp Salz, Nick Jing, Michael Romain, Iris Leefken, Richard Serton, Rajesh Veerabhadraiah, Dureseti Chidambarrao, Robert Arelt, Matt Angyal, Ben Trombley, Arvind Haran, Stefan Hougardy, Ben Klotz, and Rahul Rao. 37.1 IBM Telum II processor design-technology co-optimizations for power, performance, area, and reliability. In *2025 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 68, pages 606–608, 2025.