Efficient and Scalable Post-Layout Optimization for Field-coupled Nanotechnologies

Simon Hofmann *Student Member, IEEE,* Marcel Walter *Member, IEEE,* and Robert Wille *Senior Member, IEEE*

***Abstract*—As conventional computing technologies approach their physical limits, the quest for increased computational power intensifies, heightening interest in post-CMOS technologies. Among these, *Field-coupled Nanocomputing* (FCN), which operates through the repulsion of physical fields at the nanoscale, emerges as a promising alternative. However, realizing specific functionalities within this technology necessitates the development of dedicated FCN physical design methods. Although various methods have been proposed, their reliance on heuristic approaches often results in suboptimal quality, highlighting a significant opportunity for enhancement. In the realm of conventional CMOS design, post-layout optimization techniques are employed to capitalize on this potential, yet such methods for FCN are either not scalable or lack efficiency. This work bridges this gap by introducing the first scalable and efficient post-layout optimization algorithm for FCN. Experimental evaluations demonstrate the efficiency of this approach: when applied to layouts obtained by a *state-of-the-art* heuristic method, the proposed post-layout optimization achieves area reductions of up to $73.75\,\%$ ($45.58\,\%$ on average). This significant improvement underscores the transformative potential of post-layout optimization in FCN. Moreover, unlike existing algorithms, the method exhibits scalability even in optimizing layouts with over 20 million tiles. Implementations of the proposed methods are publicly available as part of the *Munich Nanotech Toolkit* (MNT) at https://github.com/cda-tum/fiction.**

I. Introduction

While the demand for computational power is experiencing continuous growth, fueled by the expansion of data centers managing vast digital ecosystems [1] and the development of large language models that require intensive processing for training on massive datasets [2], the limits of *Moore's Law* are becoming evident. Furthermore, projections indicate that, by 2030, the information and telecommunications sector could account for $51\,\%$ of global electricity consumption and $23\,\%$ of global greenhouse gas emissions [3]. Hence, suitable alternatives to conventional CMOS technologies are imperative.

A potential solution for the future of green computing at the nanoscale is *Field-coupled Nanocomputing* (FCN, [4]), which operates by leveraging the repulsion of physical fields instead of electric current. Recently, FCN has received a significant boost with several breakthroughs in fabrication including the successful experimental demonstration of a functional sub-$30\,\mathrm{nm}^2$ OR gate [5], [6], [7], [8].

This was achieved by utilizing *Silicon Dangling Bonds* (SiDBs, [5]) on a hydrogen-passivated silicon surface [6], [9]. These advancements have further contributed to the growing interest in FCN, leading to substantial investments, amounting to millions of dollars, in research enterprises like *Quantum Silicon Inc.*

With gates made out of SiDBs as basis, more complex functionality and, hence, entire logical circuits can be realized. To this end, gates have to be placed onto a layout and connected with each other. In addition to placing standard gates, routing wire segments that connect them is important, as they incur the same area and delay costs. This interdependence between gate placement and wire routing implies that the overall area, as well as the critical path and delay depend on two key factors: the positioning of standard gates and the total number of wire segments in a given layout.

Unfortunately, physical design algorithms for conventional CMOS cannot be seamlessly transferred to FCN due to peculiar technological constraints. In response, the design automation community has explored various innovative strategies. These include heuristic combinational methods [10], the use of SAT and SMT solvers [11], [12], bespoke manual techniques [13], and machine learning-based approaches [14], [15]. However, given the complex nature of these problems, as noted by their exponential characteristics [16], most solutions employ heuristics, as determining the optimal solution w.r.t. area is generally only practical for smaller functions. As a result, many of their generated layouts are suboptimal, mirroring issues faced in CMOS design where *post-layout optimization* is often necessary [17], [18], [19], [20].

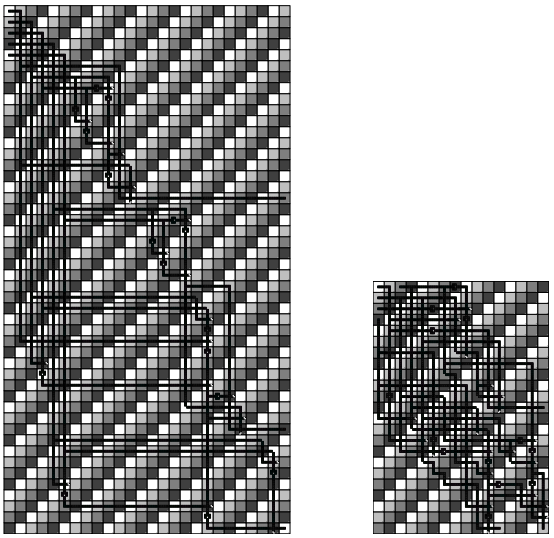
One effective strategy for layout optimization is gate relocation [21]. This involves optimizing the positioning of standard gates by removing the routing to adjacent gates, exploring alternative placements, and employing the A* search algorithm for rerouting [22].

Similarly, given that wire segments share the same area and delay costs as standard gates, another optimization tactic is wiring reduction [23]. This can be achieved by the selective removal of excess wiring in a layout, contingent upon the ability to restore functional correctness by realigning the resulting layout fragments.

Although gate relocation can lead to significant area savings and is an efficient optimization method, it is not particularly scalable. The complexity and feasibility of relocation escalate with both the number of gates and the overall layout area before optimization. In contrast, wiring reduction is more scalable, capable of handling layouts encompassing thousands of gates and millions of tiles, even though it typically results in less area reduction compared to gate relocation.

In this paper[1], we introduce gate relocation and wiring reduction, as well as a novel approach that merges these two optimization strategies into a single, comprehensive,

[1]Preliminary versions of this work have been published in [21], [23].



(a) Initial layout for the *cm82a* function created by the heuristic *ortho* with an area of $26 \times 48 = 1248$ tiles.

(b) After optimization, the area is reduced to $16 \times 23 = 386$ tiles.

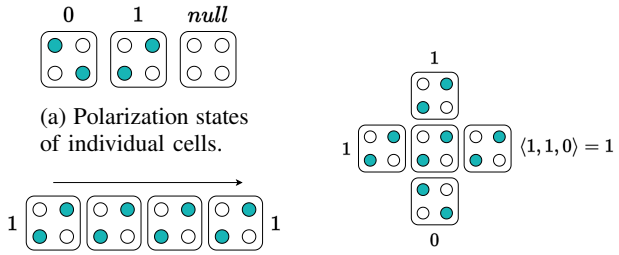
Fig. 1: Application of the post-layout optimization algorithm leading to an area reduction of $70.51\,\%$.

post-layout optimization algorithm. This algorithm combines the best of both worlds by being scalable as well as efficient, making it highly effective for preparing layouts for further processing stages such as physical simulation [24], [25] and fabrication [8]. Additionally, thanks to new insights linking Cartesian layouts commonly used in QCA with hexagonal layouts preferred for SiDBs [26], the proposed algorithm is also applicable to multiple FCN technologies. Furthermore, the algorithm is built on a gate-level abstraction, enabling it to be effortlessly adapted to accommodate newly emerging FCN technologies.

Experimental evaluations conducted for this work confirm the benefits of post-layout optimization for FCN. In fact, applied to layouts generated by a state-of-the-art heuristic method, called *ortho* [10], an area reduction of $45.58\,\%$ can be obtained on average. This is exemplified via the *cm82a* benchmark function [13], for which we achieved an area reduction of $70.51\,\%$, as illustrated in Fig. 1.

The development of layouts with minimal area overhead is imperative not only for reducing the computational complexity associated with simulation tasks but also for decreasing manufacturing costs. Consequently, this work represents a significant advancement toward the realization of energy-efficient FCN layouts, positioning FCN as an environmentally sustainable alternative to traditional CMOS technologies.

The remainder of this paper is structured as follows: Section II reviews the technical background on selected FCN technologies. Afterward, Section III reviews physical design algorithms for FCN. Section IV introduces the two optimization algorithms for efficient gate relocation and scalable wiring reduction, as well as the resulting post-layout optimization approach combining the best of both worlds. The results obtained by our experimental evaluations of the methods are summarized in Section V. Finally, Section VI concludes the paper.



(a) Polarization states of individual cells.

(b) Wire segment.

(c) Majority gate.

Fig. 2: Elementary QCA cells and compound structures.

An open-source implementation on top of the *fiction* framework [27] is available as part of the *Munich Nanotech Toolkit* (MNT, [28]).[2] Furthermore, the generated layouts have been included in the benchmark suite *MNT Bench* [29].[3]

II. Background

Field-coupled Nanocomputing (FCN), a promising class of post-CMOS technologies, presents a viable approach to meet the escalating demand for computational capabilities caused by countless applications like large language models, datacenters, or cryptocurrencies, while also addressing ecological impacts of high energy consumption leading to increased greenhouse gas emissions. FCN technologies enable circuit functionality at the nanoscale without depending on electrical current flow to transmit signals and perform computations, thereby reducing power consumption and lessening greenhouse gas emissions [4]. This section delivers essential background information required for understanding the rest of this work.

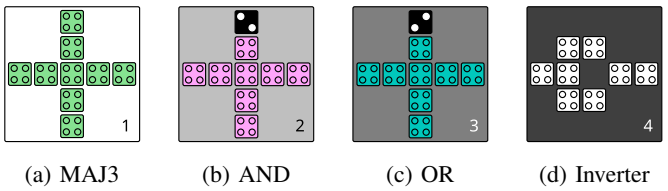
In the following, Section II-A examines the most thoroughly investigated FCN implementation, *Quantum-dot Cellular Automata* (QCA, [30]). Following that, Section II-B provides insights into recent significant advancements in the fabrication of *Silicon Dangling Bonds* (SiDBs, [5]), including the successful demonstration of a functional sub-$30\,\mathrm{nm}^2$ OR gate [5], [6], [7], [8], [31]. Finally, Section II-C outlines the technological limitations associated with FCN and outlines the differences in layout topologies of competing FCN implementations.

A. Quantum-dot Cellular Automata (QCA)

In the QCA technology, the fundamental building block, called the *cell*, has a similar significance to the transistor in CMOS as it is the elementary device. Each QCA cell comprises four *quantum dots* arranged in a square configuration on a substrate. The binary values 0 and 1 are encoded via polarization in the electron configurations, as illustrated in Fig. 2a, which generate electric fields that influence neighboring cells, aligning their polarization accordingly. This interconnectedness allows for the propagation of information and the execution of computations across multiple cells.

[2] Code is available at https://github.com/cda-tum/fiction.

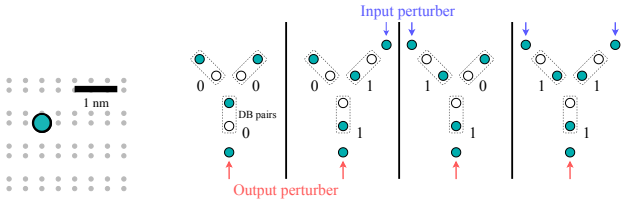
[3] https://www.cda.cit.tum.de/mntbench



(a) MAJ3 (b) AND (c) OR (d) Inverter

(e) Wire (f) Bent wire (g) Fan-out (h) Crossing

Fig. 3: The QCA ONE gate library [32].



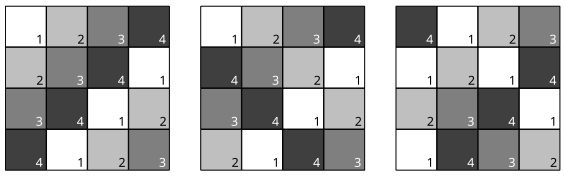
(a) H-Si(100)-2×1 surface structure. (b) Recreation of a binary-dot OR gate [8], adapted from [33].

Fig. 4: SiDBs on an H-Si(100)-2×1 lattice implementing logic.

For instance, arranging QCA cells in a straight line forms a binary wire segment, as depicted in Fig. 2b. Moreover, positioning a QCA cell adjacent to three input cells enables the implementation of the majority-of-three (MAJ3) function, which is then propagated to an output cell on the right, as shown in Fig. 2c. This arrangement provides the basis for building boolean complete gate libraries like *QCA ONE* [32], illustrated in Fig. 3, which can then be used to create more complex functions by connecting the outputs and inputs of gates with wire segments, similar to placement and routing of gates made out of transistors in traditional CMOS. Notably, as depicted in Figs. 3e to 3h, wire segments occupy the same area—a tile hosting cells on a 5×5 grid—and incur the same delay (one clock phase per tile) as standard gates, illustrated in Figs. 3a to 3d [32].

B. Silicon Dangling Bonds (SiDBs)

SiDBs are created by selectively removing hydrogen atoms from a passivated silicon (H-Si(100)-2×1) surface [9] using, e. g., a scanning tunneling microscope [5]. Recent advancements in the field [6], [34], [35], [36], [37] have enabled this fabrication process to produce atomically-sized, chemically identical quantum dots with unparalleled precision. Instead of four, SiDB cells require only two quantum dots [8]. On top of the possibility to implement the standard QCA cells using two SiDB cells, even smaller gates compared to the QCA gates illustrated in Fig. 3 can be created by only using single SiDB pairs to represent the binary values 0 or 1, depending on the position of the charge, therefore creating *Binary-dot Logic* (BDL) [8].



(a) *2DDWave* [41]. (b) *USE* [42]. (c) *RES* [43].

Fig. 5: Common clocking schemes for FCN technologies. The four distinct clock phases, labeled 1 through 4, are represented by white, light gray, dark gray, and black, respectively.

A schematic representation of an SiDB on the surface of an H-Si(100)-2×1 lattice can be seen in Fig. 4a, where the possible locations of the SiDB are colored gray and the actual SiDB is indicated by the green dot. Furthermore, the BDL concept facilitated the successful experimental demonstration of an SiDB OR gate with a footprint of less than $30\,\mathrm{nm}^2$ [8], which has never been achieved with QCA cells. A schematic illustration of this OR gate can be seen in Fig. 4b, demonstrating how the output changes based on the four possible input combinations of 0 and 1.

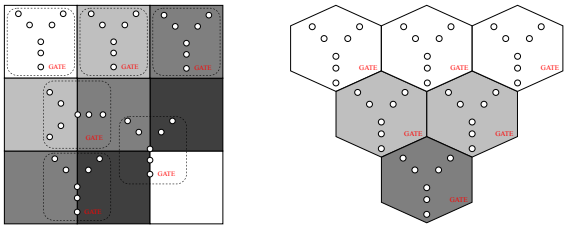
The *Bestagon* library [33] offers implementations of this OR and other standard gates, some of which are designed using reinforcement learning [38]. Efficient and accurate simulations of these gates can be conducted using physical simulators such as *SiQAD* [39], *QuickSim* [24], or *QuickExact* [25]. On top, standard gates with the lowest SiDB count, leading to less fabrication cost, can be determined using exact simulations [40].

C. Technology Constraints

In FCN technologies, numerous constraints create significant challenges for the design of circuit layouts. The requirement for *planarity* imposes strict limitations on crossing capabilities, complicating wire routing considerably [44]. Additionally, FCN circuits must be subdivided into uniform regions that are periodically activated by external fields. This concept, referred to as *Clocking* [44], [45], plays a critical role in maintaining signal stability and regulating information flow, which is essential for ensuring the proper functionality of both combinational and sequential circuits in the FCN domain. To ensure *signal synchronization*, wire paths have to be balanced to prevent information desynchronization [46], [47].

The standard clocking framework is based on four sequential clock signals that allow a pipeline-like progression of data through tiles governed by clock 1, followed by those controlled by clock 2, clock 3, and, ultimately, clock 4 before cycling back to clock 1 [44], [45].

The distribution of clock signals via buried electrodes within the substrate remains a subject of debate, leading to the development of various clocking schemes, as depicted in Fig. 5 [41], [42], [43]. The *2DDWave* clocking scheme in Fig. 5a is particularly useful for combinational logic, as it ensures unidirectional information flow from left to right and from top to bottom, thus facilitating strictly acyclic and linear data propagation. In this scheme, each gate can receive input signals from its top and left edges of the square tile they are



(a) 3×3 Cartesian layout. (b) 3×3 hexagonal layout.

Fig. 6: (a) When strictly connecting inputs to outputs, Y-shaped SiDB gates do not fit into the structure of Cartesian grids as elementary building blocks. (b) Hexagonal grids can host Y-shaped SiDB gates without modifications.

placed on, and transmit outputs through the right and bottom edges. These specific characteristics have given rise to custom heuristics that can efficiently handle the physical design of arbitrarily large logic networks with minimal computational overhead on the *2DDWave* scheme [10], [48].

Unfortunately, the direct substitution of QCA gates with SiDB gates leads to a geometric inconsistency on the Cartesian grid, arising from the disparity between the plus-shaped QCA gates and the Y-shaped SiDB counterparts, as illustrated in Fig. 6a. Y-shaped SiDB gates receive input signals from two adjacent gates positioned to the north and transmit output information southward. This setup inherently establishes a unidirectional data flow exclusively from top to bottom, as demonstrated in Fig. 6b.

The principal distinctions between clocking schemes on Cartesian layouts and clocking schemes on hexagonal layouts, which are suitable for SiDBs, include:

- The majority of QCA layouts utilize the *2DDWave* clocking scheme, as it imposes minimal overhead for implementing most combinational functions [11]. In *2DDWave*-clocked layouts, signal propagation is only permitted to the east and south, as illustrated by the configuration of clock phases in Fig. 5a.
- First approaches of using hexagonal layouts for SiDBs [33] use a *row-wise* clocking scheme to facilitate southward signal propagation, which aligns with the Y-shape of the *Bestagon* gates.

III. Related Work: Physical Design for FCN

The established FCN design flow is depicted in Fig. 7. After synthesizing the logic network from a truth table and applying logic optimization techniques and technology mapping, the network is further processed using logic optimization, then mapped to the specific technology before minimizing crossings, inserting fanouts and balancing. Afterward, the underlying clocking scheme to be used as the floorplan has to be chosen. Based on the selected clocking scheme, the next step is detailed placement and routing using one of the exact or heuristic physical algorithms, which is also the most complex step in the whole physical design flow. After the placement and routing step, the finished layout has to be checked for any design rule violations and functional equivalence to the truth table from the beginning. Only if all of these checks are passed, the resulting layout can be universally applied across various FCN technologies such as QCA, SiDBs (using a conversion technique described in Section III-C), or *Nanomagnet Logic* (NML) [49], by mapping all gates and wires to their respective cell-level implementations defined by a technology-specific gate library [32], [33], [50].

This section provides a comprehensive overview of physical design algorithms, spanning from exact methodologies outlined in Section III-A to heuristic strategies discussed in Section III-B. Additionally, it details the conversion technique that adapts Cartesian *2DDWave*-clocked layouts appropriate for QCA to hexagonal grids suitable for SiDBs in III-C.

A. Exact Approaches

Exact physical design algorithms, e. g., [11], [12], generate layouts from specifications that are optimal concerning specific cost metrics, typically the layout area. While one approach [12] extracts a symbolic formulation that encapsulates the design task of realizing a given function in terms of a QCA circuit, along with all necessary objectives and constraints from the given logic network, another approach [11] advances this by starting from the truth table of the underlying function. Passing the resulting formulation to a reasoning engine allows for the extraction of a solution that is optimal with respect to some cost metric, such as layout area.

These algorithms are clocking-scheme agnostic, although experimental results showed that, due to the different arrangement of clock zones, the resulting layout area can differ based on the underlying clocking scheme.

Example 1: In Fig. 8, the exact algorithm was used to create layouts for a multiplexer on different clocking schemes and each gate was then mapped to its respective cell-level implementation from the QCA ONE *gate library [32]. Due to the different arrangement of clock zones, the layout with the least amount of tiles on* 2DDWave *only requires* 12 *tiles, as seen in Fig. 8a, while the optimal solutions on* USE *in Fig. 8b and* RES *in Fig. 8c impose an additional overhead of* 3 *tiles with a layout size of* 15 *tiles.*

Indeed, experimental evaluations have shown that the *2DDWave* clocking scheme introduces the least area overhead on average [11].

Nonetheless, these algorithms are hindered by performance constraints due to the $\mathcal{NP}$-completeness of the task [16], restricting their application to relatively small logic networks with less than ≈ 40 gates on layouts with less than ≈ 100 tiles.

B. Heuristic Approaches

The heuristic algorithm *ortho* can design large-scale layouts for QCA circuits, handling hundreds of millions of tiles by imposing drastic restrictions on the search space [10]. It leverages knowledge from theoretical computer science, specifically *orthogonal graph drawing*, to create graphs with only horizontal and vertical edges to represent wire segments in QCA layouts. It accomplishes this impressive scale by initially coloring the logic network with two colors (red and

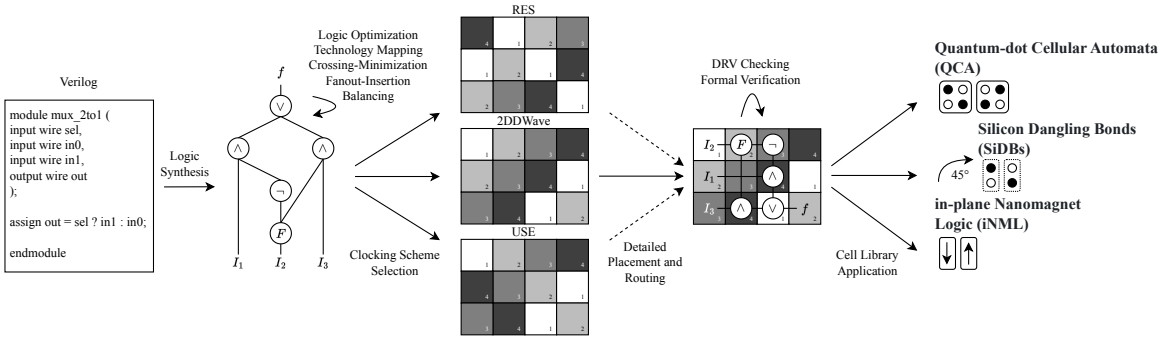
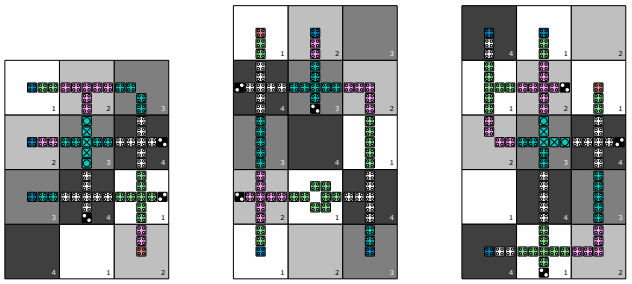


Fig. 7: The FCN physical design flow as implemented by state-of-the-art tools.

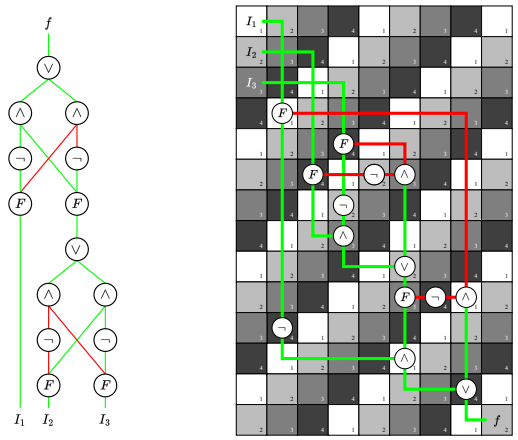


(a) 2:1-MUX layout on *2DDWave* [41].

(b) 2:1-MUX layout on *USE* [42].

(c) 2:1-MUX layout on *RES* [43].

Fig. 8: Optimal layouts for the realization of a 2-to-1 multiplexer (2:1-MUX) using different clocking schemes.



(a) Colored logic network representing a parity generator function.

(b) Resulting layout on the *2DDWave* clocking scheme.

Fig. 9: The heuristic algorithm *ortho* colors the logic network with two colors, where green lines indicate placing a node to the south of its predecessors and red lines to the east. During the placement of each gate, the color of its incoming edges determines the layout adjustments: green edges trigger the addition of a new row, while red edges require the insertion of a new column.

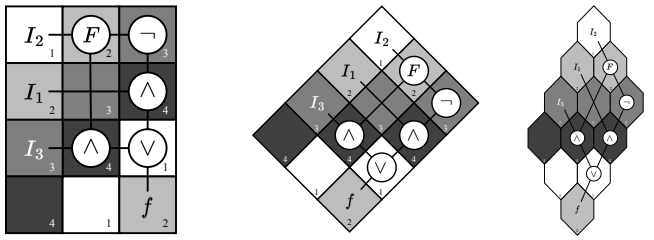
green in Fig. 9a), which provides a directive for gate placement, therefore ensuring that the two previously discussed technological constraints planarity and signal balancing are met automatically. This color-based direction assignment helps in structuring the layout such that each gate is added in a topological sequence, introducing a new row or column to the layout with each addition. On top, *ortho* not only simplifies wire routing but also effectively balances the paths across the design. It is important to recognize, however, that the *ortho* algorithm relies on approximations which can lead to layouts that are significantly larger than those derived from exact solutions. This characteristic underscores the need for subsequent optimizations to refine and potentially reduce the size of the resulting layout.

Example 2: The use of the ortho *algorithm is illustrated in Fig. 9, showcasing the design of a layout for a parity generator function. The resulting layout encompasses* 126 *tiles, while the exact algorithm is able to determine a solution for the same function on a layout with only* 32 *tiles, as the heuristic adds a new column or row for every gate, leading to multiple empty tiles as well as the necessity for many wire segments to connect the placed gates.*

To reduce the layout area and number of required wire segments introduced by the necessity to order the signals of input pins on a layout, this step can be moved to the preprocessing stage of the logic network, ensuring that the input pins are already ordered before placing them [48].

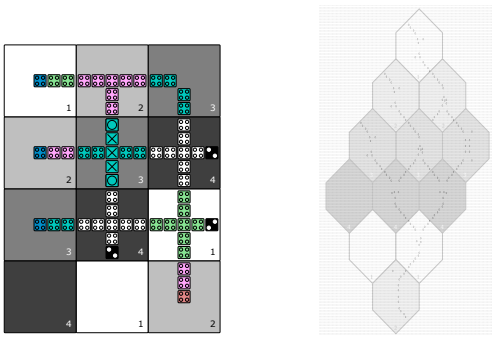
Other heuristic approaches [13], [51], [52], [53], [54] have been proposed, which can generate smaller layouts compared to *ortho*. Unfortunately, these approaches are either not scalable, can only handle a specific clocking scheme or need additional domain expert knowledge. Another approach overcomes all of these shortcoming by using reinforcement learning for gate placement [14], [15]. This method generates layouts for complex functions that are unmanageable by exact algorithms within a reasonable time frame and occupy less layout area than results generated by *ortho*. However, this method is also only applicable for logic networks with around 200 gates, but is technology- and clocking scheme-agnostic just like the exact algorithms from [11], [12].

A recent addition called *gold* [55], [56] generates gate-level layouts from logic network specifications by spanning a search space graph where each placement event can be represented as a search space vertex characterized by a partial



(a) Multiplexer realized on a Cartesian layout. (b) Layout obtained by a 45° rotation. (c) Hexagonalization.

Fig. 10: Using the hexagonalization algorithm [26], the layout created for a multiplexer using the *2DDWave* clocking scheme can be rotated to create a *row-wise*-clocked hexagonal layout suitable for Y-shaped SiDB gates.



(a) QCA layout mapped with gates from the *QCA ONE* gate library. (b) SiDB layout mapped with gates from the *Bestagon* gate library.

Fig. 11: Cartesian layout with QCA gates before the rotation and the hexagonal layout with SiDB gates after the rotation.

layout at that instance. Edges between a partial layout a and b exist iff a can be transformed into b via a single placement event. Similar to navigating through a maze, A*-search can be employed to discover a path from the starting vertex (the empty layout) to the exit of the maze (a layout with all gates placed). While this approach yields better layouts compared to the other heuristic approaches, it faces similar scalability constraints as the approach based on reinforcement learning.

In this work, layouts created by the *ortho* algorithm serve as practical examples to illustrate the effectiveness of the newly proposed post-layout optimization algorithms, as they can be generated even for logic functions with thousands of gates on layouts with millions of tiles.

C. Transforming QCA Layouts to SiDB Layouts

Since most physical design algorithms have been developed for QCA on Cartesian grids, the shift to hexagonal grids for SiDB gates raised the question of whether decades of research will again be needed to develop specialized design algorithms for SiDBs. Fortunately, with a recently discovered algorithm [26], a 45° turn is enough to transform any Cartesian, *2DDWave*-clocked [41] layout into a hexagonal configuration to accommodate Y-shaped SiDB gates to solve the topology mismatch illustrated in Fig. 6.

Example 3: This idea is illustrated in Fig. 10 using a 3×4 *Cartesian layout: Fig. 10a presents the implementation of a 2:1-multiplexer constructed using the exact algorithm described in [12], tailored for QCA gates on a Cartesian grid. Fig. 10b illustrates the adjustment of this layout through rotation to establish the* row-wise *clocking scheme, as used in [33]. To adapt this layout for use with hexagonal grid tiles, as required by the Y-shaped SiDB gates, the rectangular tiles shown in Fig. 10b are simply elongated vertically. This transformation is demonstrated in Fig. 10c. Both layouts can than be mapped with gates from the respective gate library, as illustrated in Fig. 11.*

Using this 45° turn, decades of research in the physical design for QCA can be reused for the physical design of SiDBs. More precisely: existing Cartesian *2DDWave*-clocked QCA layouts can be directly transformed to meet the requirements for placing SiDBs on the hexagonal *row-wise*-clocked layout.

Furthermore, any improvement achieved in the physical design for *2DDWave*-clocked Cartesian layouts for QCA directly translates to an improvement in the physical design for *row-wise*-clocked hexagonal layouts for SiDBs. Therefore, the development of an optimization algorithm specifically tailored to the *2DDWave* clocking scheme leads to more efficient layouts for multiple FCN technologies.

The superiority of the *2DDWave* clocking scheme compared to other clocking schemes, its usage in *state-of-the-art* heuristics like *ortho* and its connection to the hexagonal grid prompts the question: how can the characteristics of this clocking scheme be used to further minimize the layout area after the placement and routing step?

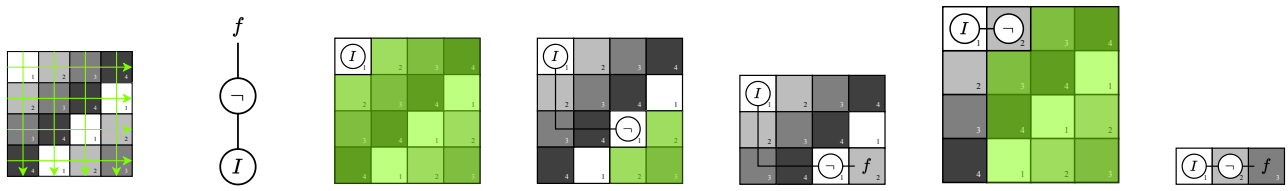
This fundamental question is tackled in the following section, which poses the main contribution of this work.

IV. Proposed Optimization Strategies

This section first outlines the special characteristics of *2DDWave*-clocked layouts created by heuristic approaches that allow for further reduction in layout area and critical path length in Section IV-A. The gate relocation algorithm proposed in Section IV-B is highly effective in optimizing area utilization, but lacks scalability. Conversely, the wiring reduction strategy presented in Section IV-C is highly scalable across various circuit sizes, although it does not achieve the same level of result quality as the gate relocation algorithm. The post-layout optimization outlined in Section IV-D combines the best aspects of the gate relocation and the wiring reduction algorithm, therefore achieving scalability *and* effectiveness.

A. Motivation

A crucial characteristic of layouts clocked by *2DDWave* unlocking optimization possibilities are the allowed information flow directions: In *2DDWave*-clocked layouts, information can only flow horizontally from left to right and vertically from top to bottom. Consequently, to effectively reduce the layout area, gates should be positioned as close as possible to the



(a) *2DDWave*-clocked layouts allow information flow from left to right and top to bottom.

(b) Simple network that negates the input.

(c) The input pin is placed in the top left corner first.

(d) Due to the flow directions, only 3 positions are available for the output.

(e) The inverter placement leads to a final layout size of 12 tiles.

(f) For a more strategic placement, 11 positions become available.

(g) The inverter placement leads to a final layout size of 3 tiles.

Fig. 12: Characteristics of the *2DDWave* clocking scheme determining area utilization.

top-left corner, from where layout construction usually starts. The strategic positioning of a gate is essential as it impacts the placement of all subsequent gates in the design due to the acyclic flow of information. In other words, the closer a gate is positioned to the top-left corner, the fewer clock zones have to be traversed to reach it. Thus, when aiming for this optimization criterion for all gates as much as possible, overall layout area and delay are reduced.

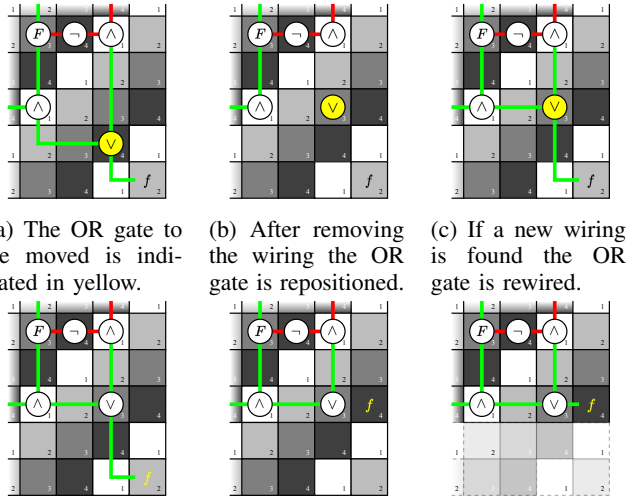
Example 4: In Fig. 12a, the permissible directions for signal flow on the 2DDWave *clocking scheme are indicated with green arrows, illustrating that in each tile information can only traverse from left to right or from top to bottom. This directional constraint means that a gate cannot be positioned above or to the left of its predecessor. The network depicted in Fig. 12b consists of a simple sequence involving an input, an inverter gate, and an output intended to be implemented as an FCN layout as a toy example. The physical design algorithm then places the input pin, inverter and output sequentially. First, the input pin is placed in the top left corner, as illustrated in Fig. 12c, so it can be accessed easily. Following this placement, if the inverter is positioned in the third column of the third row, as shown in Fig. 12d, this location restricts the possible placements for the output pin, resulting in a minimum layout area of* 12 *tiles, as illustrated Fig. 12e. Alternatively, a more strategic placement of the inverter in the second column of the first row opens up additional possibilities for positioning the output pin, as shown in Fig. 12f. This arrangement allows the output to be placed in the third column of the first row, significantly reducing the necessary layout area to just* 3 *tiles in Fig. 12g.*

This example underscores the importance of strategic gate placement within the constraints of the *2DDWave* clocking scheme to minimize layout area effectively.

B. Gate Relocation

The fundamental principle of the proposed optimization algorithm centers on relocating gates to more favorable positions within the layout in an effort to reduce area and delay. This procedure encompasses several keys steps, which will be elaborated upon subsequently. These steps include:

1) **Removal of existing wiring:** Initially, all existing connections to and from the specified gate are detached. This step is necessary because wire segments occupy the same area as standard gates, potentially obstructing the tile to which the gate is to be relocated.
2) **Identification of better placements:** Following the disconnection of the gate, the algorithm searches for more strategic locations that are closer to the top left corner of the layout, following a strategy similar to that used by *NanoPlaceR* [14], to enable more possible positions for all succeeding gates, as shown in Example 4. The algorithm determines these feasible locations by searching for better placements to the right and bottom of the location of the preceding gates.
3) **Determining a new valid location:** For each potential new position, an A*-search algorithm is employed to verify the feasibility of connections to and from the gate. If valid connections cannot be established, the algorithm continues to evaluate other possible locations.



(a) The OR gate to be moved is indicated in yellow.

(b) After removing the wiring the OR gate is repositioned.

(c) If a new wiring is found the OR gate is rewired.

(d) The output to be moved is indicated in yellow.

(e) After removing the wiring the output is repositioned.

(f) If a new wiring is found the output is rewired.

Fig. 13: Snapshot of the layout from Fig. 9b illustrating the optimization idea.

Example 5: To demonstrate the optimization process, consider again the layout depicted in Fig. 9b that was generated by the ortho *algorithm. Passing it to the proposed gate relocation algorithm, new tile positions will be considered*

Algorithm 1: Gate Relocation

```
   Input: FCN gate-level layout L
   Input: Maximum number of gate relocations m
   Output: Optimized layout
1  do
2      moved_at_least_one_gate ← false
3      foreach gate ∈ L do
4          remove present wiring of gate
5          coords ← m potentially better coordinates for gate
6          found_better_location ← false
7          foreach c ∈ coords do
8              relocate gate to c          // Fig. 13b and 13e
9              wiring ← A*-SEARCH
10             if wiring ≠ ∅ then
11                 route g using wiring // Fig. 13c and 13f
12                 found_better_location ← true
13             end if
14         end foreach
15         if found_better_location then
16             moved_at_least_one_gate ← true
17         else
18             move gate back to its initial position
19             restore original wiring of gate
20         end if
21     end foreach
22 while moved_at_least_one_gate
23 return L
```

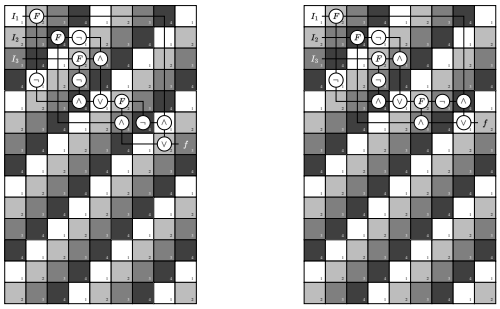
for each gate. In Fig. 13a, the bottom-rightmost gate from the aforementioned layout is designated for relocation. Here, it is highlighted in yellow. Initially, all existing connections between this gate and its predecessors and successor are removed, clearing the way for repositioning. Subsequently, potential new coordinates for the gate are calculated, ranked by their distance to the top left corner. After the gate is moved to the first new coordinate in the list, as exemplified in Fig. 13b, the A algorithm is employed to assess the feasibility of routing new connections from the predecessors to the relocated gate and from the gate to its successor. If viable new wiring routes are identified, these are implemented into the layout, as seen in Fig. 13c. If not, the original wiring is reinstated, ensuring the functionality of the circuit remains intact. Next, the output f, highlighted in yellow in Fig. 13d, is detached from its predecessor and moved to a more favorable position that became available due to the preceding move, as seen in Fig. 13e. As a new wiring is found, the output is reconnected in Fig. 13f, effectively reducing the layout area via the removal of two empty rows at the bottom.*

Algorithm 1 presents an overview of the proposed approach.

After disconnecting a gate from its preceding and succeeding gates (Line 4), the algorithm determines potential new positions. This is based on the location of its preceding gates within the layout (Line 5). The gate is then relocated to a position deemed more optimal, typically closer to the top left corner of the layout (Line 8).

Once relocated, the A* search algorithm is employed to evaluate the feasibility of reconnecting the gate with its predecessors and successors from this new position (Line 9). If a viable wiring configuration is found, it is implemented, thereby re-establishing the necessary connections within the layout (Line 10 and 11).

If, however, no suitable wiring solution can be established from any of the potential new positions (Line 17), the gate is moved back to its original position. The previous wiring connections are then restored to ensure that the functional integrity of the circuit is maintained (Line 18 and 19). This approach ensures that each gate placement optimizes the overall layout without compromising the circuit's operational capabilities.



(a) In the first iteration, every gate can be moved to a position closer to the top left corner.

(b) In the second iteration, only three gates can be relocated to even more favorable positions.

Fig. 14: Gate relocation (Algorithm 1) applied to the layout from Fig. 9b that was obtained by the *ortho* algorithm.

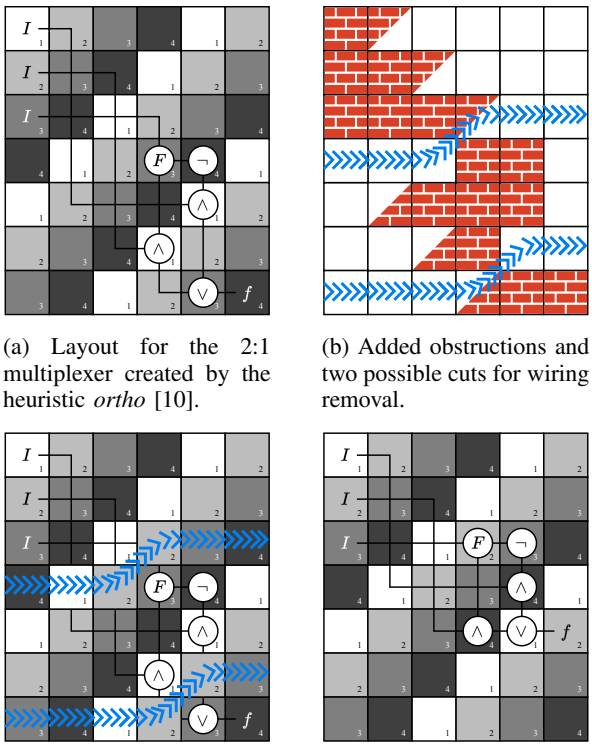
Example 6: One iteration of Algorithm 1 applied to the layout from Fig. 9b yields the optimized layout illustrated in Fig. 14a. During this process, all gates are relocated to more advantageous positions, which results in a considerable amount of layout area being freed at the bottom. In the subsequent second iteration, further adjustments are possible for only three gates, effectively reaching the limits of optimization since no further gate relocations are feasible in the third iteration. The resulting optimized layout, shown in Fig. 14b, demonstrates a more compact arrangement with gates occupying only six of the original fourteen rows resulting in more than $57.14\,\%$ *total area reduction.*

C. Wiring Reduction

In most heuristic physical design algorithms, particularly those that utilize approximations, a common issue is the inefficient use of wire segments, leading to redundant wiring. These redundancies not only increase the layout area but also introduce unnecessary delays.

A possible solution is to strategically remove certain wire segments while ensuring that the remaining layout sections can still be reassembled and reconnected without compromising the layout's functional integrity. However, detecting which wire segments can be safely deleted is a complex task. To address this challenge, we introduce obstructions within the layout that act as protective barriers. These obstructions ensure that essential components, such as gates, are preserved during the removal process and are not deleted by accident.

Once the layout is prepared with these obstructions, we utilize the *A* Search* [22] algorithm to identify feasible segments of connected tiles, also called cuts, in the layout. These cuts can extend either horizontally or vertically, depending on a



(a) Layout for the 2:1 multiplexer created by the heuristic *ortho* [10].

(b) Added obstructions and two possible cuts for wiring removal.

(c) Wires on the cut paths are deleted, leaving behind gaps.

(d) Resulting gaps are closed by pushing the layout together.

Fig. 15: One iteration of the proposed wiring reduction algorithm.

Algorithm 2: Wiring Reduction

```
   Input: FCN gate-level layout L                          // Fig. 15a
   Output: Optimized layout
1  do
2      optimize ← false
3      foreach search_direction ∈ {horizontally, vertically} do
4          add obstructions based on search_direction
             // Fig. 15b
5          cuts ← A*-SEARCH(search_direction)               // Fig. 15b
6          if cuts ≠ ∅ then
7              delete wires on cuts                         // Fig. 15c
8              move and connect layout fragments            // Fig. 15d
9              resize layout
10             optimize ← true
11         end if
12     end foreach
13 while optimize
14 return L
```

search direction, which alternates between orientations during optimization.

After identifying theses cuts, the corresponding wire segments are deleted. However, this step alone is not sufficient, as the layout fragments resulting from the cuts must be realigned to restore connections.

Algorithm 2 presents an overview of the proposed approach.

In the following, the four main steps of the approach are explained using a suboptimal layout of the 2:1 multiplexer obtained with *ortho* [10] as shown in Fig. 15a as a running example.

1) Adding Obstructions: Initially, obstructions are strategically placed within the layout (Line 4) to ensure that A^* focuses solely on identifying valid cuts. This step is crucial in preventing the algorithm from inadvertently proposing the removal of essential components. As illustrated in Fig. 15b, standard gates are entirely blocked from consideration for deletion because they are integral to the circuit's functionality and cannot be removed. Similarly, bent wire segments are partially obstructed, as these can only be cut in specific directions. These obstructions effectively guide the A^* algorithm in its search, ensuring that any cuts it suggests will preserve the necessary connectivity and functionality of the layout.

2) Determining Cuts: To the obstructed layout, the *A^*-search* algorithm is applied to identify feasible cuts through the layout (Line 5). In Fig. 15b, two potential cuts are illustrated in blue.

3) Deleting Wires: Following the identification of feasible cuts, all wire segments that fall within these designated cuts are removed from the original layout (Line 7). This action is visualized in Fig. 15c, where the segments located along the two blue cuts are removed.

4) Repositioning Gates: After the successful removal of wire segments as outlined previously, the layout is fragmented and its operational integrity has to be restored (Line 8). This is done by shifting all tiles located below the areas where wires were removed upward to close the created gaps. As demonstrated in Fig. 15d, this repositioning results in empty rows at the bottom of the layout, which are no longer necessary and can be removed completely.

The four aforementioned steps then get repeated iteratively, adjusting the layout each time to close gaps and reduce space until no more feasible cuts can be identified, indicating that the layout has been optimized as much as possible under the given constraints.

D. Post-Layout Optimization

The proposed post-layout optimization method seeks to leverage the strengths of the two presented complementary approaches: wiring reduction and gate relocation. Each of these methods offers distinct advantages and, when combined, can lead to significant improvements in layout quality.

Wiring reduction is particularly effective due to its fast runtime and ability to minimize the overall layout size quickly. By reducing the number of wire segments early in the process, the layout is simplified, which subsequently makes the gate relocation process more manageable. This initial reduction step provides a smaller and more compact starting point for further optimization.

On the other hand, gate relocation can yield more precise and targeted improvements by strategically repositioning gates to further optimize the layout. However, evaluating every possible gate location on large layouts can lead to exploding runtime behavior. To address this, the number of gate relocations is limited by a user-defined parameter, denoted as m in Algorithm 3, which summarizes the resulting methodology. Utilizing m, users can control the balance between runtime

Algorithm 3: Post-Layout Optimization

Input: FCN gate-level layout to optimize L
Input: Maximum number of gate relocations m
Output: Optimized layout

```
WiringReduction(L)                          // Algorithm 2
improvement ← true
while improvement do
    WiringReduction(L)                      // Algorithm 2
    GateRelocation(L, m)                    // Algorithm 1
    if no gate moved and no wiring reduced then
        improvement ← false
    end if
end while
return L
```

and optimization depth, preventing excessive computation time while still achieving meaningful improvements.

The optimization process alternates between wiring reduction and gate relocation, allowing each method to complement the other. The wiring reduction improves the layout quickly, and gate relocation fine-tunes it by moving gates to better locations. This iterative process continues until no further improvements can be made.

V. Experimental Evaluation

In an extensive experimental evaluation, we first demonstrate that gate relocation is effective, but not scalable, while wiring reduction is scalable, but does not achieve the same improvements as gate relocation. Second, however, it is shown that the proposed approach of combining gate relocation with wiring reduction combines the best of both worlds. To this end, the experimental setup is described in Section V-A. Subsequently, the results obtained for gate relocation, wiring reduction and the combined post-layout optimization algorithm are presented in Section V-B , Section V-C, and Section V-D, respectively. Finally, the effect of different values for the maximum number of gate relocations are discussed and analyzed in Section V-E.

A. Experimental Setup

Using the optimization methods proposed in this work, results from *any* physical design algorithm for Cartesian layouts using the *2DDWave* [41] clocking scheme can be optimized in terms of area, number of wire segments, and critical path length. For the experimental evaluation, a variety of different benchmark circuits [13], [57], [58], [59] are created using multiple physical design algorithms [10], [15], [48] and then optimized using the proposed algorithms.

The optimization methods proposed in this work have been implemented in C++17 on top of the *fiction* framework [27] as part of the *Munich Nanotech Toolkit* (MNT).[4] Additionally, the optimization algorithm has been made accessible via *fiction*'s CLI as command `optimize`. By toggling the flag `-w`, the wiring reduction algorithm is applied exclusively, and via flag `-m`, the desired maximum number of gate relocations can be specified. For the experiments, the code was compiled with AppleClang 14.0.0 and the optimization then carried out on a macOS 13.0 machine with an Apple Silicon M1 Pro SoC with $32\,\mathrm{GB}$ of integrated main memory.

[4]Code is available at https://github.com/cda-tum/fiction.

B. Gate Relocation

For gate relocation, we utilized two heuristic approaches, namely *ortho* [10] and *NanoPlaceR* [14] as representatives for existing algorithms for the design of FCN circuits, optimized the generated layouts using the proposed methodology, and verified the equivalence of the obtained layouts using the formal verification technique proposed in [60].

All methods have been evaluated using a broad variety of well-established benchmarks [13], [57]. The resulting data is summarized in Table I, which lists the benchmark configurations as well as layout characteristics of the two heuristic approaches before and after the optimization.

With the proposed post-layout optimization method, the quality of the designs generated using the *ortho* method increased significantly, with an average layout area reduction of approximately $52.44\,\%$. For layouts produced with the reinforcement learning-based *NanoPlaceR*, an average area reduction of $20.04\,\%$ could be achieved, since layouts created with that method exhibit considerably lower area costs to begin with. For small benchmarks, the layouts obtained by *NanoPlaceR* were already optimal, leaving no room for further improvement through the optimization algorithm (first five rows of Table I). Additionally, for the benchmark function *Parity Gen.* no improvement was possible by gate relocation only, even though the layout is not optimal. This can happen as the A* search algorithm is used to find a new wiring, which is not always the same as in the optimal layout found by an exact algorithm.

Overall, the application of the optimization algorithm to layouts generated by *NanoPlaceR* yielded the smallest layouts across all benchmark functions except for *t_5*, *1bitAdderAOIG*, and *cm82a*, for which a combination of *ortho* and the optimization algorithm yielded the best outcome, but only differing by one or two additional rows and/or columns. This is mainly due to the small layouts to begin with, where randomness in the generation with *NanoPlaceR* can lead to gates being placed such that fewer relocations are possible.

C. Wiring Reduction

For wiring reduction, we took layouts created by the heuristic physical design approaches *ortho* [10] and *Input Ordering SDN* [48] for a broad variety of well-established benchmark sets with large circuits [58], [59], as wiring reduction is more scalable than gate relocation, applied the proposed wiring reduction algorithm, and verified the correctness of the optimized layouts via formal verification [60]. The obtained data is summarized in Table II and Table III.

For layouts created by *ortho*, the number of wire segments was reduced by $17.90\,\%$ on average, resulting in an average area reduction and critical path shortening of $32.94\,\%$ and $18.93\,\%$, respectively, while being highly scalable, optimizing layouts with up to 1 million tiles in less than $1\,\mathrm{min}$. Even for very large layouts with more than 20 million tiles, the convergence time is still less than $6\,\mathrm{h}$. For layouts created

Table I: Comparative experimental evaluation of the proposed gate relocation algorithm.

Benchmark Circuit [13], [57]			Ortho [10]	Gate Relocation		Difference	NanoPlaceR [14]	Gate Relocation		Difference
Name	I / O	$\lvert N\rvert$	$w \times h = A$	$w \times h = A$	$t[s]$	ΔA	$w \times h = A$	$w \times h = A$	$t[s]$	ΔA
2:1 MUX	3 / 1	4	6 × 7 = 42	6 × 4 = 24	< 0.01	−42.86 %	3 × 4 = **12**	3 × 4 = **12**	< 0.01	±0.00
XOR	2 / 1	4	5 × 7 = 35	4 × 7 = 28	< 0.01	−20.00 %	3 × 6 = **18**	3 × 6 = **18**	< 0.01	±0.00
Full Adder	3 / 2	5	8 × 10 = 80	6 × 9 = 54	< 0.01	−32.50 %	4 × 7 = **28**	4 × 7 = **28**	< 0.01	±0.00
XNOR	2 / 1	6	6 × 8 = 48	4 × 6 = 24	< 0.01	−50.00 %	3 × 6 = **18**	3 × 6 = **18**	< 0.01	±0.00
Half Adder	2 / 2	6	9 × 8 = 72	4 × 6 = 24	< 0.01	−66.67 %	4 × 6 = **24**	4 × 6 = **24**	< 0.01	±0.00
Parity Gen.	3 / 1	10	9 × 13 = 117	8 × 8 = 64	< 0.01	−45.30 %	7 × 9 = **63**	7 × 9 = **63**	< 0.01	±0.00
clpl	11 / 5	10	17 × 25 = 425	9 × 13 = 117	< 0.01	−72.47 %	6 × 18 = 108	6 × 17 = **102**	< 0.01	−5.56 %
t	5 / 2	11	10 × 16 = 160	7 × 8 = 56	< 0.01	−65.00 %	8 × 8 = 64	7 × 6 = **42**	< 0.01	−34.38 %
t_5	5 / 2	11	10 × 16 = 160	7 × 9 = 63	< 0.01	−60.62 %	7 × 8 = 56	6 × 8 = **48**	< 0.01	−14.29 %
b1_r2	3 / 4	12	13 × 17 = 221	7 × 9 = **63**	< 0.01	−71.49 %	10 × 10 = 100	8 × 10 = 80	< 0.01	−20.00 %
Parity Check.	4 / 1	15	12 × 19 = 228	8 × 11 = 88	< 0.01	−61.40 %	9 × 9 = 81	7 × 10 = **70**	< 0.01	−13.58 %
1bitAdderAOIG	3 / 2	15	12 × 18 = 216	9 × 8 = **72**	< 0.01	−66.67 %	10 × 10 = 100	9 × 9 = 81	< 0.01	−19.00 %
majority	5 / 1	17	9 × 24 = 216	8 × 16 = 128	< 0.01	−40.74 %	11 × 11 = 121	10 × 11 = **110**	< 0.01	−9.09 %
majority_5_r1	5 / 1	17	10 × 23 = 230	9 × 16 = 144	< 0.01	−37.39 %	10 × 11 = **110**	9 × 12 = **108**	< 0.01	−1.82 %
newtag	8 / 1	17	12 × 25 = 300	10 × 12 = 120	< 0.01	−60.00 %	11 × 11 = 121	7 × 11 = **77**	< 0.01	−36.36 %
XOR5_R1	5 / 1	26	14 × 32 = 448	10 × 19 = 190	0.01	−57.59 %	14 × 14 = 196	13 × 10 = **130**	< 0.01	−33.67 %
1bitAdderMaj	3 / 1	29	14 × 35 = 490	13 × 30 = 390	0.01	−20.41 %	18 × 18 = 324	18 × 15 = **270**	< 0.01	−16.67 %
cm82a	5 / 3	42	26 × 48 = 1248	16 × 21 = **336**	0.08	−73.08 %	25 × 25 = 625	16 × 23 = 368	0.01	−41.12 %
2bitAdderMaj	5 / 2	54	27 × 62 = 1674	22 × 36 = 792	0.08	−52.69 %	29 × 28 = 812	19 × 29 = **551**	0.02	−32.14 %
xor5Maj	5 / 1	70	31 × 78 = 2418	26 × 52 = 1352	0.26	−44.09 %	30 × 43 = 1290	29 × 39 = **1131**	0.06	−12.33 %
parity	16 / 1	103	48 × 119 = 5712	35 × 65 = 2275	5.08	−60.17 %	48 × 48 = 2304	39 × 41 = **1599**	0.09	−30.60 %
Average Difference						−52.44 %				−20.04 %

Runtime values are in seconds; w, h and A are the width, height and resulting area of the layout respectively (lower is better); the area difference ΔA compares the layout before and after optimization. For the first five benchmark functions, *NanoPlaceR* found the optimal layout already, therefore, no further optimization is possible. The average difference is calculated based on all sub-optimal layouts.

Table II: Comparative experimental evaluation of the proposed wiring reduction algorithm on layouts generated by the heuristic physical design algorithm *ortho* [10].

Benchmark Circuit [58], [59]			Ortho [10]				Wiring Reduction					Difference		
Name	I / O	$\lvert N\rvert$	$w \times h = A$	$\lvert W\rvert$	CP		$w \times h = A$	$\lvert W\rvert$	CP	$t[s]$		ΔA	$\Delta\lvert W\rvert$	ΔCP
c17	5 / 2	8	10 × 13 = 130	63	21		8 × 11 = **88**	**51**	**17**	< 0.01		−32.31 %	−19.05 %	−19.05 %
c432	36 / 7	414	208 × 466 = 96928	35754	673		193 × 389 = **75077**	**31369**	**581**	0.97		−22.54 %	−12.26 %	−13.67 %
c499	41 / 32	816	454 × 864 = 392256	88594	1317		309 × 638 = **197142**	**65089**	**946**	21.81		−49.74 %	−26.53 %	−28.17 %
c880	60 / 26	639	328 × 748 = 245344	67890	1075		274 × 624 = **170976**	**58363**	**897**	6.61		−30.31 %	−14.03 %	−16.56 %
c1355	41 / 32	1064	494 × 1176 = 580944	110557	1669		383 × 935 = **358105**	**90494**	**1317**	41.04		−38.36 %	−18.15 %	−21.09 %
c1908	33 / 25	813	435 × 876 = 381060	98201	1310		352 × 678 = **238656**	**80163**	**1029**	13.34		−37.37 %	−18.73 %	−21.45 %
c2670	157 / 63	1463	772 × 1672 = 1290784	309743	2434		649 × 1356 = **880044**	**261660**	**1995**	83.11		−31.82 %	−15.52 %	−18.04 %
c3540	50 / 22	1987	931 × 2188 = 2037028	445193	3118		857 × 1828 = **1566596**	**396523**	**2684**	152.05		−23.09 %	−10.93 %	−13.92 %
c5315	178 / 123	3628	1884 × 3940 = 7422960	1628867	5787		1572 × 3206 = **5039832**	**1377142**	**4741**	1681.55		−32.10 %	−15.45 %	−18.07 %
c6288	32 / 32	6467	2273 × 6628 = 15065444	844173	8900		2215 × 5385 = **11927775**	**752370**	**7599**	3441.68		−20.83 %	−10.87 %	−14.62 %
c7552	206 / 107	4501	2139 × 4837 = 10346343	2243213	6970		1751 × 3722 = **6517222**	**1808281**	**5467**	4829.12		−37.01 %	−19.39 %	−21.56 %
dec	8 / 256	320	673 × 472 = 317656	168258	1144		256 × 465 = **119040**	**66307**	**720**	106.95		−62.53 %	−60.59 %	−37.06 %
ctrl	7 / 25	409	218 × 423 = 92214	26396	640		160 × 366 = **58560**	**22098**	**525**	2.18		−36.50 %	−16.25 %	−17.97 %
router	60 / 3	490	257 × 557 = 143149	53292	813		245 × 391 = **95795**	**42511**	**635**	4.52		−33.08 %	−20.23 %	−21.89 %
int2float	11 / 7	545	251 × 580 = 145580	47139	828		230 × 514 = **118220**	**42975**	**741**	1.60		−18.79 %	−8.83 %	−10.51 %
cavlc	10 / 11	1600	658 × 1668 = 1097544	282450	2325		617 × 1453 = **896501**	**257646**	**2069**	49.76		−18.32 %	−8.78 %	−11.01 %
priority	128 / 8	2349	988 × 2484 = 2454192	664415	3471		961 × 1892 = **1818212**	**575032**	**2852**	272.85		−25.91 %	−13.45 %	−17.83 %
adder	256 / 129	2541	1279 × 2797 = 3577363	765201	4075		769 × 2038 = **1567222**	**526696**	**2806**	1521.25		−56.19 %	−31.17 %	−31.14 %
i2c	136 / 127	2728	1480 × 2978 = 4407440	1061867	4451		1155 × 2602 = **3005310**	**917563**	**3752**	721.20		−31.81 %	−13.59 %	−15.70 %
max	512 / 130	6110	3110 × 6638 = 20644180	5309831	9747		2443 × 5780 = **14120540**	**4618748**	**8222**	18307.73		−31.60 %	−13.02 %	−15.65 %
bar	135 / 128	6672	3306 × 7094 = 23452764	3959962	10399		3039 × 6059 = **18413301**	**3601381**	**9097**	10240.56		−21.49 %	−9.06 %	−12.52 %
Average Difference												−32.94 %	−17.90 %	−18.93 %

Runtime values are in seconds; w, h and A are the width, height and resulting area (in tiles) of the layout, respectively; $|W|$ and CP indicate the number of wire segments and the length of the critical path, respectively; the area, number of wire segments and critical path length difference ΔA, $\Delta|W|$ and ΔCP, compare the layout before and after optimization, lower is better.

by *Input Ordering SDN*, which produces even smaller layouts than *ortho*, the number of wire segments was reduced by 13.89 % on average, resulting in an average area reduction and critical path shortening of 22.35 % and 12.85 %, respectively.

D. Post-Layout Optimization

For the proposed post-layout optimization algorithm presented in Section IV-D, which combines gate relocation and wiring reduction, we again took layouts created by the heuristic physical design approach *ortho* [10] for the four sets of benchmark functions, ranging from logic networks with 4 nodes to 6672 [13], [57], [58], [59]. The obtained data is summarized in Table IV.

The number of maximum gate relocations was set as follows, to ensure the optimization does not exceed a time limit of 6 h:

$$m(A) = \begin{cases} \max & \text{if } A < 100\,000, \\ 1 & \text{if } 100\,000 \leq A < 20\,000\,000, \\ 0 & \text{if } A \geq 20\,000\,000. \end{cases}$$

Here, $m(A)$ represents the number of gate relocations based on A, the area in tiles of the layout to optimize. The boundaries depending on A have been determined in experimental evaluations, but can still differ depending on the layout to be optimized, but act as a rough estimate. Note that when setting m to $\max$, not every tile is tested, as only valid and

Table III: Comparative experimental evaluation of the proposed wiring reduction algorithm on layouts generated by the heuristic physical design algorithm *Input Ordering SDN* [48].

Benchmark Circuit [58], [59]			Input Ordering SDN [48]				Wiring Reduction					Difference		
Name	I / O	$\|N\|$	$w \times h$ =	A	$\|W\|$	CP	$w \times h$ =	A	$\|W\|$	CP	$t[s]$	ΔA	$\Delta\|W\|$	ΔCP
c17	5 / 2	8	7 × 10 =	70	37	16	7 × 10 =	**70**	**37**	**16**	< 0.01	±0.00	±0.00	±0.00
c432	36 / 7	414	198 × 429 =	84942	36096	626	186 × 339 =	**63054**	**29923**	**524**	1.87	−25.77 %	−17.10 %	−16.29 %
c499	41 / 32	816	405 × 716 =	289980	79030	1120	310 × 608 =	**188480**	**63214**	**917**	11.74	−35.00 %	−20.01 %	−18.12 %
c880	60 / 26	639	278 × 666 =	185148	65996	937	259 × 540 =	**139860**	**54641**	**792**	5.18	−24.46 %	−17.21 %	−15.47 %
c1355	41 / 32	1064	406 × 1119 =	454314	103685	1524	357 × 1082 =	**386274**	**97105**	**1438**	15.43	−14.98 %	−6.35 %	−5.64 %
c1908	33 / 25	813	389 × 739 =	287471	89513	1127	344 × 630 =	**216720**	**77653**	**973**	8.51	−24.61 %	−13.25 %	−13.66 %
c2670	157 / 63	1463	675 × 1466 =	989550	296443	2128	623 × 1254 =	**781242**	**260282**	**1864**	36.74	−21.05 %	−12.20 %	−12.41 %
c3540	50 / 22	1987	874 × 2012 =	1758488	436723	2885	823 × 1738 =	**1430374**	**392045**	**2560**	113.42	−18.66 %	−10.23 %	−11.27 %
c5315	178 / 123	3628	1687 × 3498 =	5901126	1608953	5141	1533 × 3035 =	**4652655**	**1428874**	**4524**	703.17	−21.16 %	−11.19 %	−12.00 %
c6288	32 / 32	6467	1330 × 5714 =	7599620	705119	7043	1310 × 5125 =	**6713750**	**651671**	**6434**	714.17	−11.66 %	−7.58 %	−8.65 %
c7552	206 / 107	4501	1890 × 4200 =	7938000	2031870	6075	1699 × 3660 =	**6218340**	**1812236**	**5344**	1961.71	−21.66 %	−10.81 %	−12.03 %
dec	8 / 256	320	418 × 466 =	194788	102746	883	252 × 453 =	**114156**	**65624**	**704**	32.53	−41.39 %	−36.13 %	−20.27 %
ctrl	7 / 25	409	167 × 369 =	61623	22835	535	135 × 349 =	**47115**	**20147**	**483**	0.85	−23.54 %	−11.77 %	−9.72 %
router	60 / 3	490	251 × 449 =	112699	51040	699	241 × 319 =	**76879**	**39405**	**559**	4.00	−31.78 %	−22.80 %	−20.03 %
int2float	11 / 7	545	233 × 546 =	127218	44731	776	214 × 499 =	**106786**	**40899**	**710**	1.13	−16.06 %	−8.57 %	−8.51 %
cavlc	10 / 11	1600	604 × 1544 =	932576	263276	2147	573 × 1413 =	**809649**	**245752**	**1985**	22.60	−13.18 %	−6.66 %	−7.55 %
priority	128 / 8	2349	826 × 2094 =	1729644	592669	2919	801 × 1694 =	**1356894**	**503829**	**2494**	185.56	−21.55 %	−14.99 %	−14.56 %
adder	256 / 129	2541	899 × 2668 =	2398532	858682	3566	898 × 1907 =	**1712486**	**662717**	**2804**	122.10	−28.60 %	−22.82 %	−21.37 %
i2c	136 / 127	2728	1288 × 2655 =	3419640	1028038	3940	1123 × 2347 =	**2635681**	**907159**	**3467**	335.98	−22.93 %	−11.76 %	−12.01 %
max	512 / 130	6110	2855 × 6422 =	18334810	4850513	9276	2412 × 5026 =	**12122712**	**3830618**	**7437**	15656.79	−33.88 %	−21.03 %	−19.83 %
bar	135 / 128	6672	2540 × 6441 =	16360140	3514605	8980	2396 × 5640 =	**13513440**	**3195255**	**8035**	7398.71	−17.40 %	−9.09 %	−10.52 %
Average Difference												−22.35 %	−13.89 %	−12.85 %

Runtime values are in seconds; w, h and A are the width, height and resulting area (in tiles) of the layout, respectively; $|W|$ and CP indicate the number of wire segments and the length of the critical path, respectively; the area, number of wire segments and critical path length difference ΔA, $\Delta|W|$ and ΔCP, compare the layout before and after optimization, lower is better.

Table IV: Comparative experimental evaluation of the proposed post-layout optimization algorithm.

Benchmark Circuit [13], [57]			Ortho [10]		Gate Relocation			Difference	Wiring Reduction			Difference	Post-Layout Optimization				Difference
Name	I / O	$\|N\|$	$w \times h$ =	A	$w \times h$ =	A	$t[s]$	ΔA	$w \times h$ =	A	$t[s]$	ΔA	$w \times h$ =	A	$t[s]$	m	ΔA
2:1 MUX	3 / 1	4	6 × 7 =	42	6 × 4 =	**24**	< 0.01	−42.86 %	6 × 5 =	30	< 0.01	−28.57 %	6 × 4 =	**24**	< 0.01	max	−42.86 %
XOR	2 / 1	4	5 × 7 =	35	4 × 7 =	28	< 0.01	−20.00 %	5 × 6 =	30	< 0.01	−14.29 %	5 × 4 =	**20**	< 0.01	max	−42.86 %
Full Adder	3 / 2	5	8 × 10 =	80	6 × 9 =	54	< 0.01	−32.50 %	7 × 8 =	56	< 0.01	−30.00 %	6 × 7 =	**42**	< 0.01	max	−47.50 %
XNOR	2 / 1	6	6 × 8 =	48	4 × 6 =	**24**	< 0.01	−5.00 %	6 × 6 =	36	< 0.01	−25.00 %	5 × 7 =	35	< 0.01	max	−27.08 %
Half Adder	2 / 2	6	9 × 8 =	72	4 × 6 =	**24**	< 0.01	−66.67 %	7 × 8 =	56	< 0.01	−22.22 %	4 × 6 =	**24**	< 0.01	max	−66.67 %
Parity Gen.	3 / 1	10	9 × 13 =	117	7 × 9 =	**63**	< 0.01	−46.15 %	9 × 10 =	90	< 0.01	−23.08 %	7 × 9 =	**63**	< 0.01	max	−46.15 %
clpl	11 / 5	10	17 × 25 =	425	9 × 13 =	**117**	< 0.01	−72.47 %	13 × 24 =	312	< 0.01	−26.59 %	10 × 12 =	120	0.01	max	−71.76 %
t	5 / 2	11	10 × 16 =	160	7 × 8 =	**56**	< 0.01	−65.00 %	9 × 12 =	108	< 0.01	−32.50 %	7 × 10 =	70	< 0.01	max	−56.25 %
t_5	5 / 2	11	10 × 16 =	160	7 × 9 =	63	< 0.01	−60.62 %	9 × 13 =	117	< 0.01	−26.88 %	6 × 7 =	**42**	< 0.01	max	−73.75 %
b1_r2	3 / 4	12	13 × 17 =	221	7 × 9 =	**63**	< 0.01	−71.49 %	9 × 13 =	117	< 0.01	−47.06 %	8 × 12 =	96	< 0.01	max	−56.56 %
Parity Check.	4 / 1	15	12 × 19 =	228	8 × 11 =	**88**	< 0.01	−61.40 %	12 × 15 =	180	< 0.01	−21.05 %	9 × 10 =	90	< 0.01	max	−60.53 %
1bitAdderAOIG	3 / 2	15	12 × 18 =	216	9 × 8 =	**72**	< 0.01	−66.67 %	11 × 16 =	176	< 0.01	−18.52 %	10 × 13 =	130	< 0.01	max	−39.81 %
majority	5 / 1	17	9 × 24 =	216	8 × 16 =	128	< 0.01	−40.74 %	9 × 18 =	162	< 0.01	−25.00 %	8 × 14 =	**112**	< 0.01	max	−48.15 %
majority_5_r1	5 / 1	17	10 × 23 =	230	9 × 16 =	144	< 0.01	−37.39 %	10 × 15 =	150	< 0.01	−34.78 %	9 × 14 =	**126**	< 0.01	max	−45.22 %
newtag	8 / 1	17	12 × 25 =	300	10 × 12 =	**120**	< 0.01	−60.00 %	12 × 20 =	240	< 0.01	−20.00 %	10 × 12 =	**120**	< 0.01	max	−60.00 %
XOR5_R1	5 / 1	26	14 × 32 =	448	10 × 19 =	**190**	0.01	−57.59 %	12 × 23 =	276	< 0.01	−38.39 %	10 × 19 =	**190**	0.01	max	−57.59 %
1bitAdderMaj	3 / 1	29	14 × 35 =	490	13 × 30 =	390	0.01	−20.41 %	14 × 28 =	392	< 0.01	−20.00 %	13 × 28 =	**364**	0.01	max	−25.71 %
cm82a	5 / 3	42	26 × 48 =	1248	16 × 21 =	**336**	0.08	−73.08 %	22 × 35 =	770	< 0.01	−38.30 %	16 × 23 =	368	0.06	max	−70.51 %
2bitAdderMaj	5 / 2	54	27 × 62 =	1674	22 × 36 =	792	0.08	−52.69 %	24 × 47 =	1128	0.01	−32.62 %	20 × 39 =	**780**	0.09	max	−53.41 %
xor5Maj	5 / 1	70	31 × 78 =	2418	26 × 52 =	1352	0.26	−44.09 %	30 × 64 =	1920	0.01	−20.60 %	26 × 48 =	**1248**	0.24	max	−48.39 %
parity	16 / 1	103	48 × 119 =	5712	35 × 65 =	2275	5.08	−60.17 %	48 × 75 =	3600	0.03	−36.97 %	39 × 52 =	**2028**	1.71	max	−64.50 %
c17	5 / 2	8	10 × 13 =	130	8 × 11 =	**88**	< 0.01	−32.21 %	8 × 11 =	**88**	< 0.01	−32.31 %	8 × 11 =	**88**	< 0.01	max	−32.31 %
c432	36 / 7	414	208 × 466 =	96928	timeout limit reached				193 × 389 =	75077	0.75	−22.54 %	192 × 324 =	**62208**	7004.73	max	−35.82 %
c499	41 / 32	816	454 × 864 =	392256	timeout limit reached				309 × 638 =	197142	21.81	−49.74 %	295 × 614 =	**181130**	49.21	1	−53.82 %
c880	60 / 26	639	328 × 748 =	245344	timeout limit reached				274 × 624 =	170976	6.61	−30.31 %	264 × 580 =	**153120**	33.57	1	−37.59 %
c1355	41 / 32	1064	494 × 1176 =	580944	timeout limit reached				383 × 935 =	358105	41.04	−38.36 %	378 × 876 =	**331128**	105.19	1	−43.00 %
c1908	33 / 25	813	435 × 876 =	381060	timeout limit reached				352 × 678 =	238656	13.34	−37.37 %	399 × 641 =	**217299**	41.33	1	−42.98 %
c2670	157 / 63	1463	772 × 1672 =	1290784	timeout limit reached				649 × 1356 =	880044	83.11	−31.82 %	642 × 1268 =	**814056**	394.51	1	−36.93 %
c3540	50 / 22	1987	931 × 2188 =	2037028	timeout limit reached				857 × 1828 =	1566596	152.02	−23.09 %	853 × 1726 =	**1472278**	905.54	1	−27.72 %
c5315	178 / 123	3628	1884 × 3940 =	7422960	timeout limit reached				1572 × 3206 =	5039832	1681.55	−32.10 %	1528 × 3093 =	**4726104**	12358.60	1	−36.33 %
c6288	32 / 32	6467	2273 × 6628 =	15065444	timeout limit reached				2215 × 5385 =	11927775	3441.68	−20.83 %	2192 × 4839 =	**10607088**	16250.49	1	−29.59 %
c7552	206 / 107	4501	2139 × 4837 =	10346343	timeout limit reached				1751 × 3722 =	6517222	4829.13	−37.01 %	1726 × 3557 =	**6139382**	19173.93	1	−40.66 %
dec	8 / 256	320	673 × 472 =	317656	timeout limit reached				256 × 465 =	119040	106.95	−62.53 %	246 × 459 =	**112914**	165.50	1	−64.45 %
ctrl	7 / 25	409	218 × 423 =	92214	timeout limit reached				160 × 366 =	58560	2.18	−36.50 %	145 × 261 =	**37845**	9777.33	max	−58.96 %
router	60 / 3	490	257 × 557 =	143149	timeout limit reached				245 × 391 =	95795	3.22	−33.08 %	243 × 358 =	**86994**	12.19	1	−39.23 %
int2float	11 / 7	545	251 × 580 =	145580	timeout limit reached				230 × 514 =	118220	1.60	−18.79 %	217 × 490 =	**106330**	11.86	1	−26.96 %
cavlc	10 / 11	1600	658 × 1668 =	1097544	timeout limit reached				617 × 1453 =	896501	49.76	−18.32 %	607 × 1336 =	**810952**	378.81	1	−26.11 %
priority	128 / 8	2349	988 × 2484 =	2454192	timeout limit reached				961 × 1892 =	1818212	272.85	−25.91 %	921 × 1758 =	**1619118**	1652.23	1	−34.03 %
adder	256 / 129	2541	1279 × 2797 =	3577363	timeout limit reached				769 × 2038 =	1567222	1521.25	−56.19 %	764 × 1841 =	**1406524**	4220.36	1	−60.68 %
i2c	136 / 127	2728	1480 × 2978 =	4407440	timeout limit reached				1155 × 2602 =	3005310	721.20	−31.81 %	1108 × 2447 =	**2711276**	5027.09	1	−38.48 %
max	512 / 130	6110	3110 × 6638 =	20644180	timeout limit reached				2443 × 5780 =	**14120540**	18307.73	−31.60 %	2443 × 5780 =	**14120540**	18307.73	0	−31.60 %
bar	135 / 128	6672	3306 × 7094 =	23452764	timeout limit reached				3039 × 6059 =	**18413301**	10240.56	−21.49 %	3039 × 6059 =	**18413301**	10240.56	0	−21.49 %
Average Difference								−26.98 %				−30.34 %					−45.58 %

Runtime values are in seconds; w, h and A are the width, height and resulting area of the layout respectively; timeout limit was set to 6 h; the area difference ΔA compares the layout before and after optimization.

empty positions are determined first, which also have to be better than the previous position. For large layouts, only trying one valid position usually yields a good trade-off between scalability and quality, which will also be discussed based on further experiments in Section V-E

While gate relocation achieved a significant average area reduction of $52.44\,\%$ for smaller benchmark functions as shown in Table I, it failed to converge for larger benchmarks [58], [59] within a 6-hour timeout limit. When considering all benchmark circuits, gate relocation yielded an average area reduction of only $26.98\,\%$ and proved infeasible for layouts exceeding approximately $50\,000$ tiles.

In comparison, the wiring reduction approach achieved an average area reduction of $30.34\,\%$. However, it resulted in smaller reductions for smaller benchmarks when compared to gate relocation.

Integrating both approaches into a unified post-layout optimization algorithm resulted in a substantial average reduction in layout area of $45.58\,\%$, effectively almost halving the layout size. This hybrid method surpassed gate relocation

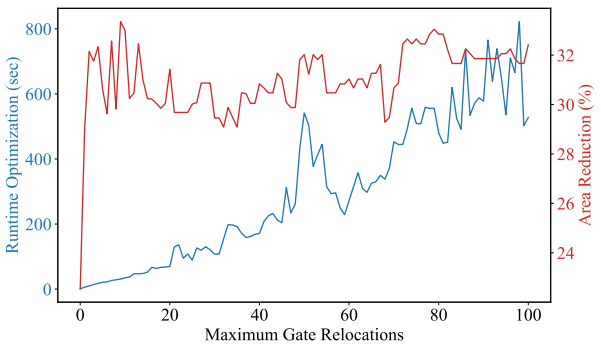


Fig. 16: Area reduction compared to the total runtime for different settings of the maximum number of gate relocations for the benchmark functions *c432* using the proposed post-layout optimization algorithm.

in efficiency for nearly all smaller benchmark functions and matched the scalability of the wiring reduction approach. For the two largest benchmark functions, namely *max* and *bar*, the algorithm defaulted to solely using wiring reduction due to their extensive size, each comprising over 20 million tiles.

E. Maximum Gate Relocations

To elucidate the optimal setting for the maximum number of gate relocations, Fig. 16 presents the trade-off between area reduction and runtime across various values of m. For the benchmark function *c432*, Fig. 16 plots the area reductions for different values for m, the number of maximum gate relocations. Notably, a significant increase in layout area reduction is observed when m is increased from 0 (indicating wiring reduction only) to 1 (where only the most favorable position is tested for gate relocation). Additionally, the increment in runtime is minimal and tends to increase almost linearly with further relocations.

However, the area reduction does not consistently improve with an increase in the number of maximal gate relocations. In some cases, it may result in the repositioning of gates to less optimal locations. Additionally, for large layouts, trying every possible location leads to increased runtime and might be impractical. Consequently, limiting the number of gate relocations to only try a single position offers the most effective balance between efficiency and scalability.

The chart in Fig. 16 also reveals that even with $m = 100$ maximum gate relocations, the improvement remain sub-optimal when compared to the improvement possible with $m = \max$ as shown in Table IV. For the benchmark function *c432*, post-layout optimization yields a reduction of $32.41\,\%$ for $m = 100$, compared to a potential reduction of $35.82\,\%$ when m is set to the maximum.

For scalability reasons, m should be set to 1 when optimizing large layouts, as most of the improvement can already be achieved with that setting.

VI. Conclusion

As *Field-coupled Nanocomputing* (FCN) transitions from theoretical exploration to practical implementation, there is an increasing need for optimization methods that enhance physical designs post-placement and routing. This study introduced novel optimization methods tailored to FCN layouts utilizing the *2DDWave* clocking scheme. These algorithms are publicly accessible and have been incorporated into the *fiction* framework, as part of the *Munich Nanotech Toolkit* (MNT).

Our proposed methods synergize the efficiency of gate relocation with the scalability of wiring reduction, significantly refining the performance of existing heuristic algorithms such as *ortho*. This integration achieves an average reduction of $45.58\,\%$ in layout area, as validated by applications to well-established benchmark sets, while being highly scalable. By minimizing the layout area during the early stages of the physical design phase, this approach offers substantial benefits. It not only reduces the computational load required for simulations but also leads to cost savings in the manufacturing process. Furthermore, it contributes to enhanced device performance by shortening critical path lengths, thereby decreasing delay and increasing throughput.

Optimizing layouts is a critical milestone in advancing *Field-coupled Nanocomputing*, as it significantly bridges the gap between the capabilities of conventional CMOS and this class of emerging technologies. This progress is driven not only by advancements in manufacturing but also by the development of more area-efficient layouts, which are essential for realizing complex functions at a scale previously achievable only with conventional technologies.

References

[1] K. M. U. Ahmed *et al.*, "A Review of Data Centers Energy Consumption And Reliability Modeling," *IEEE Access*, 2021.
[2] S. Samsi *et al.*, "From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference," 2023.
[3] A. Andrae and T. Edler, "On Global Electricity Usage of Communication Technology: Trends to 2030," *Challenges*, vol. 6, pp. 117–157, 2015.
[4] N. Anderson and S. Bhanja, Eds., *Field-Coupled Nanocomputing - Paradigms, Progress, and Perspectives*. Springer, 2014.
[5] R. Achal *et al.*, "Lithography for robust and editable atomic-scale silicon devices and memories," *Nat. Commun.*, vol. 9, no. 1, 2018.
[6] N. Pavliček *et al.*, "Tip-induced passivation of dangling bonds on hydrogenated Si(100)-2×1," *APL*, vol. 111, no. 5, p. 053104, 2017.
[7] F. Altincicek, "Atomically Defined Wires on P-Type Silicon," *Bull. Am. Phys. Soc.*, 2022.
[8] T. Huff *et al.*, "Binary atomic silicon logic," *Nat. Electron.*, vol. 1, no. 12, pp. 636–643, 2018.
[9] M. Haider *et al.*, "Controlled Coupling and Occupation of Silicon Atomic Quantum Dots at Room Temperature," *PRL*, vol. 102, p. 046805, 2009.
[10] M. Walter *et al.*, "Scalable Design for Field-Coupled Nanocomputing Circuits," in *ASP-DAC*, 2019, pp. 197–202.
[11] ——, "One-pass Synthesis for Field-coupled Nanocomputing Technologies," in *ASP-DAC*, 2021, pp. 574–580.
[12] ——, "An Exact Method for Design Exploration of Quantum-dot Cellular Automata," in *DATE*, 2018, pp. 503–508.
[13] G. Fontes *et al.*, "Placement and Routing by Overlapping and Merging QCA Gates," in *ISCAS*, 2018.
[14] S. Hofmann *et al.*, "Late Breaking Results From Hybrid Design Automation for Field-coupled Nanotechnologies," in *DAC*, 2023.
[15] ——, "Thinking Outside the Clock: Physical Design for Field-coupled Nanocomputing with Deep Reinforcement Learning," in *ISQED*, 2024.
[16] M. Walter *et al.*, "Placement and Routing for Tile-based Field-coupled Nanocomputing Circuits is $\mathcal{NP}$-complete," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 3, 2019.
[17] Y. T. Chang *et al.*, "Post-Placement Power Optimization with Multi-Bit Flip-Flops," in *ICCAD*, 2010, pp. 218–223.
[18] V. Bertacco *et al.*, "Post-Placement Rewiring and Rebuffering by Exhaustive Search for Functional Symmetries," in *ICCAD*, 2005, pp. 56–63.

[19] S. S. Kiran Pentapati *et al.*, "Pin-3D: A Physical Synthesis and Post-Layout Optimization Flow for Heterogeneous Monolithic 3D ICs," in *ICCAD*, 2020.

[20] X. Gao *et al.*, "Post-Layout Simulation Driven Analog Circuit Sizing," *Science China Information Sciences*, vol. 67, no. 4, 2024.

[21] S. Hofmann *et al.*, "Post-Layout Optimization for Field-coupled Nanotechnologies," in *NANOARCH*, 2023.

[22] P. Hart *et al.*, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[23] S. Hofmann *et al.*, "Late Breaking Results: Wiring Reduction for Field-coupled Nanotechnologies," in *DAC*, 2024.

[24] J. Drewniok *et al.*, "*QuickSim:* Efficient *and* Accurate Physical Simulation of Silicon Dangling Bond Logic," in *IEEE-NANO*, 2023, pp. 817–822.

[25] ——, "The Need for Speed: Efficient Exact Simulation of Silicon Dangling Bond Logic," in *ASP-DAC*, 2024.

[26] S. Hofmann *et al.*, "Scalable Physical Design for Silicon Dangling Bond Logic: How a 45 ° Turn Prevents the Reinvention of the Wheel," in *IEEE-NANO*, 2023, pp. 872–877.

[27] M. Walter *et al.*, "*fiction*: An Open Source Framework for the Design of Field-coupled Nanocomputing Circuits," 2019.

[28] ——, "The Munich Nanotech Toolkit (MNT)," in *IEEE-NANO*, 2024, pp. 454–459.

[29] S. Hofmann *et al.*, "MNT Bench: Benchmarking Software and Layout Libraries for Field-coupled Nanocomputing," in *DATE*, 2024.

[30] C. Lent *et al.*, "Quantum Cellular Automata: The Physics of Computing with Arrays of Quantum Dot Molecules," in *PhysComp*, 1994, pp. 5–13.

[31] J. Pitters *et al.*, "Atomically Precise Manufacturing of Silicon Electronics," *ACS Nano*, 2024.

[32] D. Reis *et al.*, "A Methodology for Standard Cell Design for QCA," in *ISCAS*, 2016, pp. 2114–2117.

[33] M. Walter *et al.*, "Hexagons Are the Bestagons: Design Automation for Silicon Dangling Bond Logic," in *DAC*, 2022, pp. 739–744.

[34] T. Huff *et al.*, "Atomic White-Out: Enabling Atomic Circuitry through Mechanically Induced Bonding of Single Hydrogen Atoms to a Silicon Surface," *ACS nano*, vol. 11 9, pp. 8636–8642, 2017.

[35] J. Pitters *et al.*, "Charge Control of Surface Dangling Bonds Using Nanoscale Schottky Contacts," *ACS nano*, vol. 5, pp. 1984–9, 2011.

[36] R. Wolkow *et al.*, "Silicon Atomic Quantum Dots Enable Beyond-CMOS Electronics," in *Field-Coupled Nanocomputing*, 2013.

[37] M. Rashidi *et al.*, "Initiating and Monitoring the Evolution of Single Electrons Within Atom-Defined Structures," *PRL*, vol. 121, p. 166801, 2018.

[38] R. Lupoiu *et al.*, "Automated Atomic Silicon Quantum Dot Circuit Design via Deep Reinforcement Learning," *ArXiv*, vol. abs/2204.06288, 2022.

[39] S. Ng *et al.*, "SiQAD: A Design and Simulation Tool for Atomic Silicon Quantum Dot Circuits," *IEEE TNANO*, vol. 19, pp. 137–146, 2020.

[40] J. Drewniok *et al.*, "Minimal Design of SiDB Gates: An Optimal Basis for Circuits Based on Silicon Dangling Bonds," in *NANOARCH*, 2023.

[41] V. Vankamamidi *et al.*, "Clocking and Cell Placement for QCA," in *IEEE-NANO*, vol. 1, 2006, pp. 343–346.

[42] C. Campos *et al.*, "USE: A Universal, Scalable and Efficient clocking scheme for QCA," *IEEE TCAD*, vol. 35, pp. 513–517, 2016.

[43] M. Goswami *et al.*, "An Efficient Clocking Scheme for Quantum-dot Cellular Automata," *Int. J. Electron. Lett.*, vol. 8, no. 1, pp. 83–96, 2020.

[44] K. Hennessy and C. S. Lent, "Clocking of Molecular Quantum-dot Cellular Automata," *J. Vac. Sci. Technol. B*, vol. 19, no. 5, pp. 1752–1755, 2001.

[45] C. Lent and P. Tougaw, "A Device Architecture for Computing with Quantum Dots," *Proc. IEEE*, vol. 85, no. 4, pp. 541–557, 1997.

[46] F. Sill Torres *et al.*, "Synchronization of Clocked Field-Coupled Circuits," in *IEEE-NANO*, 2018.

[47] F. Sill Torres *et al.*, "On the Impact of the Synchronization Constraint and Interconnections in Quantum-dot Cellular Automata," *MICPRO*, vol. 76, pp. 103–109, 2020.

[48] M. Walter *et al.*, "Versatile Signal Distribution Networks for Scalable Placement and Routing of Field-coupled Nanocomputing Technologies," in *ISVLSI*, 2023.

[49] W. Porod *et al.*, *Nanomagnet Logic (NML)*. Springer Berlin Heidelberg, 2014, pp. 21–32.

[50] F. Riente *et al.*, "ToPoliNano: A CAD Tool for Nano Magnetic Logic," *IEEE TCAD*, vol. 36, no. 7, pp. 1061–1074, 2017.

[51] R. E. Formigoni *et al.*, "Ropper: A Placement and Routing Framework for Field-Coupled Nanotechnologies," in *SBCCI*. ACM, 2019.

[52] Y. Li *et al.*, "Field-Coupled Nanocomputing Placement and Routing with Genetic and A* Algorithms," *IEEE TCAS-I*, vol. 69, no. 11, pp. 4619 – 4631, 2022.

[53] G. Li *et al.*, "A QCA placement and routing algorithm based on the SA algorithm," *Int. J. Electron*, 2023.

[54] B. Zhang *et al.*, "Quantum-dot Cellular Automata Placement and Routing with Hierarchical Algorithm," *Nano Commun. Netw.*, vol. 39, p. 100495, 2024.

[55] S. Hofmann *et al.*, "A* is Born: Efficient and Scalable Physical Design for Field-coupled Nanocomputing," in *IEEE-NANO*, 2024, pp. 80–85.

[56] ——, "Physical Design for Field-coupled Nanocomputing with Discretionary Cost Objectives," in *LASCAS*, 2025.

[57] A. Trindade *et al.*, "A Placement and Routing Algorithm for Quantum-dot Cellular Automata," in *SBCCI*, 2016.

[58] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a targeted translator in FORTRAN," in *ISCAS*, 1985.

[59] L. G. Amarù, P.-E. Gaillardon, and G. D. Micheli, "The EPFL Combinational Benchmark Suite," in *IWLS*, 2015.

[60] M. Walter *et al.*, "Verification for Field-coupled Nanocomputing Circuits," in *DAC*, 2020.

Simon Hofmann (S'23) received his Master's degree in Electrical Engineering from the Technical University of Munich (TUM), Germany, in 2022. He is currently pursuing a Ph.D. at the Chair for Design Automation at the same university. His primary research focus is on design automation for Field-coupled Nanotechnologies (FCN).

Marcel Walter (S'18–M'22) received his Ph.D. degree in Computer Science from the University of Bremen, Germany, in 2021 for his work on algorithms for the physical design of emerging post-CMOS nanotechnologies. He is currently a Postdoc at the Chair for Design Automation at the Technical University of Munich (TUM) in Germany. He has also been working as a Visiting Professor for the University of Bremen in 2024. Furthermore, he is the initiator and maintainer of the "fiction" framework for the logic synthesis, physical design, verification, and simulation of Field-coupled Nanotechnologies.

Robert Wille (M'06–SM'15) is a Full and Distinguished Professor at the Technical University of Munich, Germany, and Chief Scientific Officer at the Software Competence Center Hagenberg, Austria. He received the Diploma and Dr.-Ing. degrees in Computer Science from the University of Bremen, Germany, in 2006 and 2009, respectively. Since then, he worked at the University of Bremen, the German Research Center for Artificial Intelligence (DFKI), the University of Applied Science of Bremen, the University of Potsdam, and the Technical University Dresden. From 2015 until 2022, he was Full Professor at the Johannes Kepler University Linz, Austria, until he moved to Munich. His research interests are in the design of circuits and systems for both conventional and emerging technologies. In these areas, he published more than 350 papers and served in editorial boards as well as program committees of numerous journals/conferences such as TCAD, ASP-DAC, DAC, DATE, and ICCAD. For his research, he was awarded, e.g., with Best Paper Awards, e.g., at TCAD and ICCAD, an ERC Consolidator Grant, a Distinguished and a Lighthouse Professor appointment, a Google Research Award, and more.