# $\text{A}^*$ is Born: Efficient and Scalable Physical Design for Field-coupled Nanocomputing

Simon Hofmann, Marcel Walter, and Robert Wille
https://www.cda.cit.tum.de/research/nanotech/

***Abstract*— *Field-coupled Nanocomputing* (FCN) has emerged as a promising alternative to traditional CMOS technology, driven by recent advancements in atomic-scale logic gate fabrication and simulation. However, the efficient placement and routing of logic functions remain significant challenges, with existing algorithms lacking scalability or quality. In this paper, we present a novel method aimed at addressing these challenges by focusing on the generation of layouts with outstanding quality in a fraction of the time compared to existing approaches. Through extensive experimentation, we demonstrate that our method significantly reduces area overhead, outperforming two state-of-the-art heuristics by more than $70\,\%$ and $24\,\%$, respectively, while achieving these results at a remarkable $460$ times faster pace compared to the latter. Furthermore, we contribute to open science by releasing our algorithm as an open-source implementation, fostering collaboration and further advancements in the field of FCN.**

## I. Introduction

*Field-coupled Nanocomputing* (FCN, [1]) is a class of post-CMOS technologies operating at the nanoscale without the flow of electricity, targeting the ever increasing need for computational power while addressing environmental concerns. Recently, FCN started to gain momentum by breakthroughs in the manufacturing [2] and simulation [3]–[5] of logic gates using *Silicon Dangling Bonds* (SiDBs, [6]).

To keep up with this rapid progress, physical design methods for layout mapping of logic functions are needed. For small functions, *exact* approaches with exponential runtime behavior [7], [8] are able to determine optimal solutions, but for larger functions, *scalable* algorithms [9]–[11] either generate solutions of sub-par quality in negligible time or trade-off scalability for improved quality. This work proposes a scalable algorithm that combines the best of both worlds, generating near-optimal solutions in a short amount of time.

The remainder of this paper is structured as follows: Section II reviews technical background on selected FCN technologies and search algorithms. Section III discusses heuristic state-of-the-art design automation methods for FCN. After a computational complexity analysis in Section IV, an $A^*$-based physical design algorithm is proposed in Section V, which constitutes the main contribution of this work. It is experimentally evaluated on a set of common benchmark functions in Section VI. Finally, after outlining limitations of the proposed approach in Section VII, Section VIII concludes the paper.

An open-source implementation on top of the *fiction* framework [12] is available as part of the *Munich Nanotech Toolkit* (MNT, [13]).[1] Furthermore, the generated layouts have been included in the benchmark suite *MNT Bench* [14].[2]

Simon Hofmann, Marcel Walter, and Robert Wille are with the Chair for Design Automation, Technical University of Munich, Germany. Marcel Walter is also with the University of Bremen, Germany. Robert Wille is also with the Software Competence Center Hagenberg GmbH (SCCH), Austria. E-mail: {simon.t.hofmann, marcel.walter, robert.wille}@tum.de

## II. Background

This section covers the preliminaries of the FCN technologies *Quantum-dot Cellular Automata* (QCA, [15]) and *Silicon Dangling Bonds* (SiDBs, [6]), as well as select search algorithms.

### A. Field-coupled Nanocomputing

*1) Quantum-dot Cellular Automata (QCA):* In QCA, elementary devices are called *cells*, which consist of four *quantum dots* arranged in a square frame, hosting two charges. Due to the shorter distance between two adjacent dots compared to the distance between dots located on the diagonal, the charges stabilize in either of the two configurations illustrated in Fig. 1a, representing the binary values of 0 and 1.

By placing multiple of these cells next to each other, the polarization of one cell influences the next due to electrical fields, creating a wire that is able to propagate information, as seen in Fig. 1b. Furthermore, standard logic gates like the majority-of-three (MAJ3) function, AND, OR, and inverter can be created by arranging multiple cells in configurations as shown in Fig. 2. Using this gate library [16], logic functions can be composed by combining multiple gates.

*2) Silicon Dangling Bonds (SiDBs):* To create SiDBs, acting as atomically-sized, chemically identical quantum dots, a scanning tunneling microscope tip [6] can be used to remove hydrogen atoms from a passivated silicon (H-Si(100)-$2\times1$) surface [17].

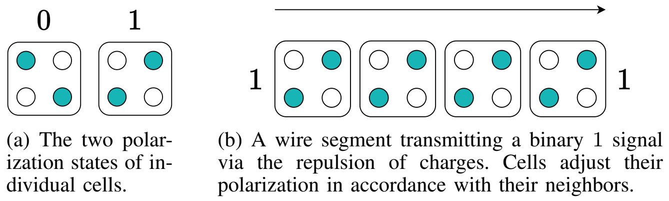
In contrast to the cells with four quantum dots used in QCA, pairs of SiDBs are used to yield a concept known as *Binary-dot Logic* (BDL, [18]), which has been applied to implement standard gate libraries as well [19].

A fully functioning SiDB OR gate with a footprint of less than $30\,\text{nm}^2$ was successfully manufactured [18] using recent breakthroughs in the domain, allowing unparalleled control over the placement of theses dots [2], [20]–[23].

*3) Technology Constraints:* The potential of FCN is often limited by inherent technological constraints. Most FCN implementations are planar with limited crossing capabilities, complicating wire routing. Additionally, ensuring signal synchronization requires meticulous management of wire segment lengths throughout the layout [24].

Clocking mechanisms are integral to FCN implementations, crucial for maintaining signal stability and controlling

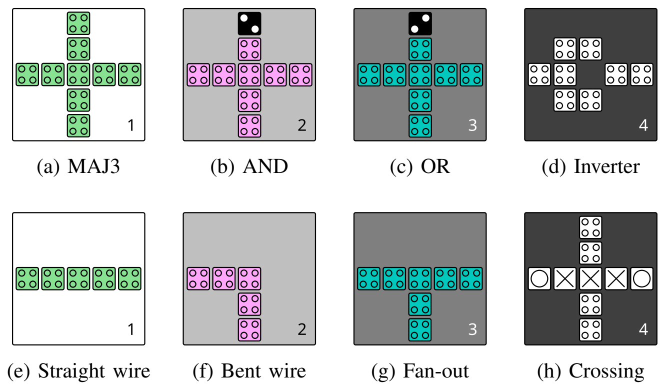
[1]Code is available at https://github.com/cda-tum/fiction.
[2]https://www.cda.cit.tum.de/mntbench



(a) The two polarization states of individual cells.

(b) A wire segment transmitting a binary 1 signal via the repulsion of charges. Cells adjust their polarization in accordance with their neighbors.

Fig. 1: Elementary QCA cells and a wire segment.



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

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

Fig. 2: The QCA ONE gate library [16].

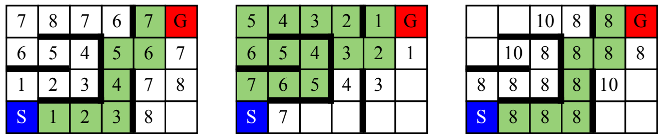
information flow. The standard QCA clocking system proposes four consecutive signals (clock 1 through 4), which are distributed to individual tiles through buried electrodes within the circuit substrate [25], to facilitate a pipeline-like transmission of information among tiles. Various clocking schemes, such as *2DDWave* [26], offer tailored arrangements of regular clock zones to streamline FCN layout design.

## B. Search Algorithms

Search algorithms play a crucial role in solving problems across various domains in computer science by finding the shortest path in, e. g., a graph or maze. Similar to physical design algorithms in FCN, they often also have to trade-off efficiency and scalability.

One approach guaranteeing optimality in finding the shortest path is *Dijkstra's Algorithm* [27], which may not always be the most efficient option, especially in large graphs. In the worst case, every vertex in a graph or tile in a maze has to be visited. *Greedy Best-First Search* sacrifices optimality for efficiency by always expanding the vertex with the least (heuristic) cost, but may fail in certain scenarios without a well-designed heuristic. In contrast, the *$A^*$-Search Algorithm* [28] strikes a balance between these approaches, offering optimality when the heuristic is admissible, while maintaining efficiency.

*Example 1:* Fig. 3 illustrates the application of the three aforementioned search algorithms in a maze, where walls are colored black and cannot be passed. In Fig. 3a, Dijkstra's algorithm is used to find the shortest path from the blue start tile $S$ to the red goal tile $G$ by expanding the tile with the lowest cost, which is the number of steps taken from the start tile. Due to the structure of the maze, almost every tile has to be visited before finding the shortest path. In contrast, the greedy best-first search in Fig. 3b can find a path more



(a) Dijkstra (b) Best-first search (c) $A^*$-search

Fig. 3: Application of three different search algorithms to find a path (green) through a maze from start tile $S$ (blue) to goal tile $G$ (red).

quickly by consistently expanding the tile with the lowest heuristic cost—the Manhattan distance from the current tile to $G$—although the resulting path may not be the shortest one. The $A^*$-search algorithm used in Fig. 3c combines the best of both worlds by expanding the tile with the least overall cost, which is the sum of taken steps and an approximation for the distance to the goal, which is the Manhattan distance in our example. This allows for finding the shortest path while exploring fewer tiles compared to Dijkstra's algorithm.

## III. Related Work: Scalable Physical Design Methods for Field-coupled Nanocomputing

As outlined earlier, FCN layouts exhibit unique traits that set them apart from conventional CMOS-based computing systems. Notably, the physical design challenges inherent in FCN technologies pose significant hurdles, characterized by constraints like planarity and signal balancing [25], [29].

Predictably, the tasks of placement and routing in FCN circuits are acknowledged to be $\mathcal{NP}$-complete [30], rendering the quest for optimal solutions impractical, even for circuits of modest scale when using exact approaches [7], [8].

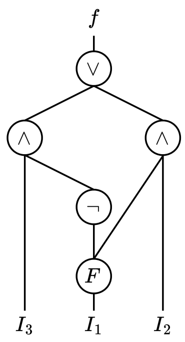
To address this issue, numerous heuristic algorithms have been proposed in the literature, many of which are specifically designed for employment with the *2DDWave* clocking scheme or demonstrate superior performance when utilized with it as opposed to other clocking schemes. In layouts clocked by *2DDWave*, data propagation is restricted to left-to-right and top-to-bottom directions, providing symmetry and the ability to integrate a $45°$ rotation [31] to convert resulting layouts into a hexagonal arrangement suitable for accommodating Y-shaped SiDB gates [19].

### A. ortho

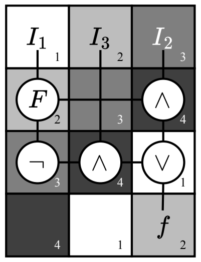
The heuristic algorithm *ortho* [9] leverages an approximation to orthogonal graph drawing and is applicable to logic networks with thousands of gates. This approach obviates the necessity of pre-defining the layout area before placement, as the required size is determined automatically.

### B. NanoPlaceR

Another heuristic strategy called *NanoPlaceR* [10], [11] employs reinforcement learning to anticipate advantageous positions for logic gates, coupled with $A^*$-search for routing. Although, on average, it produces layouts with less than half the area overhead compared to *ortho*, it exhibits longer runtimes and necessitates pre-defining the layout's dimension to initialize policy and value networks for the reinforcement learning agent.



(a) Network comprised of nine nodes (gates), including primary inputs and outputs.



(b) One out of two possible 2:1 MUX FCN layouts on a $3 \times 4$ *2DDWave*-clocked grid.

Fig. 4: Logic network and its corresponding optimal FCN layout for the 2:1 multiplexer function.

## C. Post-Layout Optimization

A novel post-layout optimization technique [32], [33] aims at reducing the area overhead after the physical design stage by relocating gates to more favorable positions, leading to an overall compaction of the given layout. However, the initial layout plays a significant role in this post-layout optimization process, as the computation time not only increases with the number of gates but also with the initial size of the layout. It has been observed that layouts generated by *NanoPlaceR* exhibit not only less area overhead compared to *ortho* before optimization, but also after, further highlighting the importance of an effective initial physical design algorithm.

The necessity of these heuristic approaches prompts the question: why is the underlying physical design problem so challenging?

## IV. Complexity

The computational complexity of the physical design problem in FCN is characterized by the disproportion between the number of possible solutions and the vast search space. In the following, the logic network for the 2:1 multiplexer illustrated in Fig. 4a will be used to illustrate this phenomenon.

Selecting the optimal position for each gate in a logic network on a layout is a combinatorial problem, where the number of possibilities is determined by

$$P_G(t,g) = \frac{t!}{(t-g)!} \tag{1}$$

possible placements for $g$ gates on $t = w \cdot h$ available tiles when not taking symmetries into account.

We assume that adjacent gates are automatically connected if they are also neighbors in the logic network, and that the underlying clocking scheme is *2DDWave*, which further limits the number of possible configurations, as there are only seven ways to fill the remaining empty tiles while taking signal flow directions into account: four different single wire segments (original and rotated versions the straight wire in Fig. 2e and the bent wire in Fig. 2f), two different double wire segments (either the crossing shown in Fig. 2h or the overlay of the bent wire in Fig. 2f and its rotated version), or no wiring at all. Therefore there are

$$P_W(t,g) = 7^{t-g} \tag{2}$$

possibilities to fill the remaining $t-g$ tiles with wires or leave them empty, which results in

$$P(t,g) = P_G(t,g) \cdot P_W(t,g) = \frac{7^{t-g} \cdot t!}{(t-g)!} \tag{3}$$

total possibilities to fill each tile with gates and wires.

*Example 2:* The 2:1 multiplexer shown in Fig. 4a consists of $g = 9$ gates that need to be placed on a layout. Assuming a layout size of $t = w \times h = 3 \times 4 = 12$, there are a total of

$$P(12,9) = \frac{7^{12-9} \cdot 12!}{(12-9)!} = 27\,382\,924\,800 \tag{4}$$

possible combinations for placing gates and wires.

Using a brute-force approach, we created all 27 382 924 800 permutations and tested them for functional equivalence with the logic network. Of all these possibilities, only 2 layouts were found valid, one of which is shown in Fig. 4b. Therefore, the probability of determining a valid layout for this network by random assignment on a $3 \times 4$ grid is only

$$\frac{2}{27382924800} \approx 7.304 \times 10^{-11}, \tag{5}$$

similar to the probability of guessing the correct outcome of a coin toss 33 times in a row, which further underscores the complexity of the physical design problem in FCN.

## V. Proposed Approach: A* Search Space Traversal

Iterative placement of gates can be viewed as a walk through a search space graph, where each placement event can be represented as a search space vertex characterized by a partial 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). This characterizes the fundamental concept of the proposed approach.

In the subsequent sections, the individual steps will be elucidated using Fig. 5.

## A. Topological Sort

First, the input logic network's gates are topologically sorted. This process establishes a sequential ordering of gates based on their interdependence, ensuring that the predecessors of a gate already exist on a layout when it is placed, making it possible to connect them with wires if a path can be found.

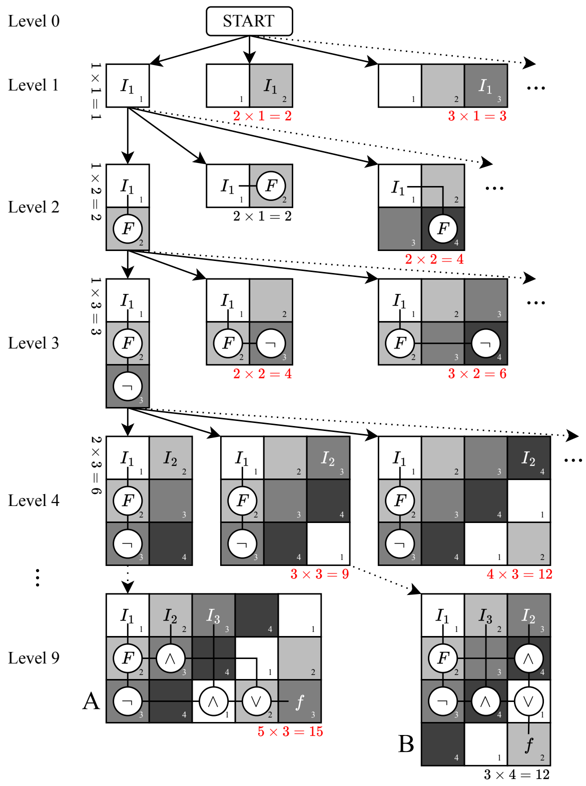


Fig. 5: The search space graph created for the 2:1 multiplexer, consisting of a total of 453 vertices with the optimal solution in the bottom right corner.

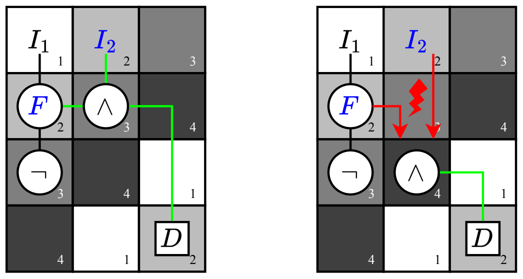


Fig. 6: Empty coordinates are checked for validity.

## B. Initialization

Initially, the explored search space contains only the empty layout, representing the start vertex of each exploration. The first element to be placed from a topological arrangement of a logic network is always a primary input pin, which has to be located at the border of a layout to enable interconnection with other parts of a circuit.

*Example 3:* Fig. 5 visualizes a part of the search space graph for the placement and routing problem outlined in Fig. 4. The initial empty layout is represented by the vertex labeled *START*. The first primary input can then be placed anywhere in the first row or column.

## C. Cost Definition

As outlined in Section II-B, the $A^*$-search algorithm employs both the actual cost and a heuristic for exploration of the underlying graph. In the proposed method, the primary cost factor is the count of gates still to place, prioritizing the establishment of *any valid* layout. To prevent the selection of suboptimal solutions as in greedy best-first search, the layout size is integrated as a heuristic cost function to break ties between various incomplete layouts with the same number of placed gates.

*Example 4:* Following initialization, the cost of the *START* vertex on *Level* 0 is equal to the number of gates in the logic network since no gate has been positioned yet, and the layout size is zero. In Fig. 5 the cost of the initial vertex is, thus, equal to $g = 9$.

Upon placing the initial primary input, the cost decreases by 1. However, the layout size can fluctuate based on the input's placement. For instance, there is only one tile in the layout when placing the input in the first column, but two when placing it in the second column of the first row, as seen on *Level* 1 of the graph in Fig. 5. Consequently, placing the input in the top left corner results in the lowest overall cost, and will be prioritized for further exploration.

## D. Search Space Vertex Expansion

To restrict the size of the search space (which extends infinitely with an ever-increasing layout size), multiple checks ensure that new partial layouts explored during the expansion of a vertex fulfill certain criteria:

1) Primary inputs can only be located at the layout's top and left, and outputs only at the right and bottom borders.
2) A path must exist between a placed gate and its predecessors to be able to connect them after placement.
3) A valid path must exist between a placed gate and a virtual drain tile $D$, which is located in the bottom right corner, as any signal has to arrive at one of the borders either directly or through other gates. This heuristic is employed to detect deadlocks early during the search to prevent revisiting lower levels in the graph too often.
4) Placed gates are not allowed to block other gates that are already placed on the layout but are still unconnected to their successors.

*Example 5:* In the leftmost layout on *Level* 4 of Fig. 5, four gates are already placed and the next gate to be placed is an AND gate with its two predecessors being the input at the top of the second column and the fanout in the second row of the first column, which are colored blue in a recreation of this partial layout in Fig. 6. By placing the AND gate right next to the fanout tile, as seen in Fig. 6a, all criteria are satisfied, as paths exist from any predecessor to the gate and from the gate to the drain tile $D$. Therefore, the partial layout is included as a valid vertex in the search space graph. If the gate was placed in the third row next to the inverter, as illustrated in Fig. 6b, not all criteria are satisfied, as no valid paths exists between its predecessors and the placed gate, because two wires are not allowed to enter or exit a tile from the same direction.

## E. Termination

After determining a first functionally-valid layout, its size is used as another criteria while backtracking, i. e., traversing

TABLE I: Comparative experimental evaluation of the state of the art against the proposed algorithm.

| Benchmark Circuit [34], [35] | | | State of The Art | | | | | | Proposed Approach | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | NanoPlaceR [10], [11] | | | ortho [9] | | | High Efficiency | | | | | | High Effort | | | | |
| Name | $I$ / $O$ | $\|G\|$ | $w \times h$ | $= A$ | $t[s]$ | $w \times h$ | $= A$ | $t[s]$ | $w \times h$ | $= A$ | $t[s]$ | $\Delta t_{\text{NP}}$ | $\Delta A_{\text{NP}}$ | $\Delta A_{\text{ortho}}$ | $w \times h$ | $= A$ | $t[s]$ | $\Delta A_{\text{NP}}$ | $\Delta A_{\text{ortho}}$ |
| 2:1 MUX | 3 / 1 | 9 | 3 × 4 | **12** | 1.58 | 6 × 8 | 48 | < 0.01 | 3 × 4 | **12** | 0.01 | −99.37 % | ±0.00% | −75.00 % | 3 × 4 | **12** | 0.05 | ±0.00% | −75.00 % |
| XOR | 2 / 1 | 9 | 3 × 6 | **18** | 1.27 | 5 × 8 | 40 | < 0.01 | 6 × 3 | **18** | < 0.01 | −99.61 % | ±0.00% | −55.00 % | 6 × 3 | **18** | 0.02 | ±0.00% | −55.00 % |
| XNOR | 2 / 1 | 11 | 3 × 6 | **18** | 2.82 | 6 × 9 | 54 | < 0.01 | 6 × 3 | **18** | < 0.01 | −99.97 % | ±0.00% | −66.67 % | 6 × 3 | **18** | < 0.01 | ±0.00% | −66.67 % |
| Half Adder | 2 / 2 | 14 | 4 × 6 | **24** | 1.93 | 9 × 10 | 90 | < 0.01 | 7 × 4 | 28 | 0.14 | −99.27 % | +16.67 % | −68.89 % | 7 × 4 | 28 | 0.50 | +16.67 % | −68.89 % |
| Parity Gen. | 3 / 1 | 18 | 7 × 9 | 63 | 18.90 | 9 × 14 | 126 | < 0.01 | 8 × 5 | 40 | < 0.01 | −99.99 % | −36.51 % | −68.25 % | 7 × 5 | **35** | 36.28 | −44.44 % | −72.22 % |
| Parity Check. | 4 / 1 | 26 | 9 × 9 | 81 | 18.37 | 12 × 20 | 240 | < 0.01 | 14 × 5 | 70 | < 0.01 | −99.98 % | −13.58 % | −70.83 % | 5 × 10 | **50** | 6.11 | −38.27 % | −79.17 % |
| XOR5_R1 | 5 / 1 | 40 | 14 × 14 | 196 | 24.69 | 14 × 33 | 462 | < 0.01 | 7 × 20 | 140 | < 0.01 | −99.97 % | −28.57 % | −69.70 % | 6 × 15 | **90** | 8.96 | −54.08 % | −80.52 % |
| cm82a | 5 / 3 | 68 | 25 × 25 | 625 | 105.14 | 26 × 51 | 1326 | < 0.01 | 26 × 11 | 286 | 0.48 | −99.54 % | −54.24 % | −78.43 % | 28 × 10 | **280** | 0.10 | −55.20 % | −78.88 % |
| 2bitAdderMaj | 5 / 2 | 82 | 29 × 29 | 841 | 175.43 | 26 × 64 | 1664 | < 0.01 | 22 × 19 | **418** | 0.05 | −99.97 % | −50.30 % | −74.88 % | 22 × 19 | **418** | 0.36 | −50.30 % | −74.88 % |
| xor5Maj | 5 / 1 | 102 | 30 × 45 | 1350 | 290.56 | 30 × 79 | 2370 | < 0.01 | 44 × 22 | 968 | 0.08 | −99.97 % | −28.30 % | −59.16 % | 18 × 41 | **738** | 362.06 | −45.33 % | −68.86 % |
| parity | 16 / 1 | 150 | 48 × 48 | 2304 | 1005.67 | 48 × 120 | 5760 | < 0.01 | 61 × 9 | **549** | 0.41 | −99.96 % | −76.17 % | −90.47 % | 61 × 9 | **549** | 2.53 | −76.17 % | −90.47 % |
| *Average Difference* | | | | | | | | | | | | −99.78 % | −24.64 % | −70.66 % | | | | −31.56 % | −73.69 % |

$I$, $O$, and $|G|$ are the number of primary inputs, primary outputs, and gates including fanouts in the logic network, respectively; $w$, $h$ and $A$ are the width, height, and resulting area of the layout, respectively; due to the probabilistic nature of *NanoPlaceR* (*NP*), its runtime indicates the average of 3 successful runs; numbers in bold indicate the best layout area across all four approaches; $\Delta t$ and $\Delta A$ compare the runtime and area of the proposed methods to the two heuristics. The high-effort mode manages multiple search space graphs at the same time to determine better solutions.

backwards and forwards through different levels of the graph, to find better solutions. Any vertex with a partial layout that already possesses a bigger layout size than the current best solution is cut from the graph with all its outgoing vertices to further speed-up the algorithm by restricting the search space to explore. The proposed algorithm terminates either when a sufficiently small layout, as initially specified, is found, when the whole search space graph has been explored, or after reaching a timeout, yielding the best solution found to that point in terms of area.

*Example 6:* In Fig. 5, the first found solution (*A*) is not optimal, as determining *any valid* placement is preferred over the smallest possible layout size while expanding vertices. This preference is chosen as the number of solutions is often disproportional to the size of the search space, as outlined in Section IV. In comparison to the $27\,382\,924\,800$ different possibilities for placement and routing determined in Section IV, the explored graph only consists of $453$ vertices. The optimal solution in the bottom right corner (*B*), as already highlighted in Fig. 4b, was found after visiting $157$ vertices, which took around $10\,\text{ms}$ on an M1 MacBook Pro.

## VI. Experiments

In this section, we present the results of an experimental evaluation performed with the proposed physical design algorithm on logic networks taken from a set of benchmark circuits commonly utilized in the domain [34], [35]. We conducted a comparative analysis on an M1 MacBook Pro, evaluating its performance against state-of-the-art heuristic methodologies [9]–[11] and validating the correctness of the created layouts using formal verification [36].

The proposed algorithm comes in two flavors, namely a high-efficiency mode, which is fast but may not produce the best achievable results, and a high-effort mode, which creates multiple search space graphs based on different fanout substitution strategies, topological sorts, etc., which is usually slower but often finds even better layouts and overcomes the limitations outlined later in Section VII. The following discussion concerns the high-efficiency mode (with results for the high-effort mode in parentheses).

The data presented in Table I exclusively pertains to QCA implementations using the *2DDWave* clocking scheme due to the underlying Cartesian grid suitable for gates from the QCA ONE library shown in Section II-A.1. However, each layout can be adapted for SiDBs by transforming the Cartesian layout into a hexagonal one utilizing a 45°-turn [31].

Our proposed approach achieves significant average area reductions in every but one benchmark of more than $24\,\%$ ($31\,\%$) and $70\,\%$ ($73\,\%$) compared to the state-of-the-art heuristic algorithms *NanoPlaceR* and *ortho*, respectively, while maintaining similar runtime performance relative to *ortho* and beating *NanoPlaceR* by more than a factor of $460$ in high-efficiency mode and still more than a factor of $2$ in high-effort mode. High runtimes for the *xor5Maj* and *Parity Gen.* function in high-effort mode are due to the necessity of revisiting lower levels after a first valid layout was found to determine even smaller solutions.

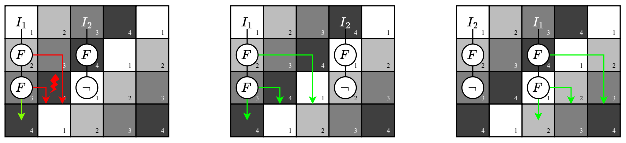
For the largest benchmark function, *parity*, *NanoPlaceR* manages to find a solution in $951\,\text{s}$ on a layout with $2304$ tiles, while *ortho* accomplishes this task in less than $0.01\,\text{s}$, albeit with twice the area, totaling $5760$ tiles. In contrast, our proposed approach achieves a layout size of $549$ tiles in only $0.41\,\text{s}$ using the high-efficiency mode, almost matching *ortho* in speed, and more than $2400$ times faster than *NanoPlaceR*, while reducing area utilization by $76\,\%$ and $90\,\%$ compared to *NanoPlaceR* and *ortho*, respectively.

The results outlined in Table I underscore the superiority of our proposed approach over *NanoPlaceR* in terms of scalability and over all heuristic techniques with regard to layout area utilization, thus, for the first time, offering a compelling synergy of scalability and efficiency beating all other competitors and enabling advanced physical design for FCN.

## VII. Limitations and Future Work

While the proposed algorithm excels at swiftly generating compact layouts, early unfavorable visited vertices in the search space graph can significantly impede its efficiency if detected too late. This can happen if the search algorithm has to backtrack through a lot of vertices and levels to get back to a state where this unfavorable placement can be reverted.

*Example 7:* Fig. 7a illustrates one such instance of sub-optimal placement: In this scenario, the two fanouts in the first column—depending on $I_1$—were placed before $I_2$ and its successive gates in the third column. Consequently, there is no feasible way to route both of $I_1$'s fanouts, as two wires cannot exit a tile in the same direction, as already seen in Fig. 6b. A comparison of two functions with and without this phenomenon are shown in Table II. For *cm42a*



(a) One of $I_1$'s fanouts is blocked (deadlock). (b) A possible solution using tile reservations. (c) Another solution using a different topological sorting.

Fig. 7: A placement configuration leading to a deadlock and two possible solutions.

TABLE II: Comparison of runtimes for a function with early unfavorable placements (*cm42a*) and without (*i3*).

| Benchmark Circuit [37] | | | ortho [9] | | Proposed Approach | | |
|---|---|---|---|---|---|---|---|
| Name | $I$ / $O$ | $\lvert G\rvert$ | $w \times h = A$ | $t[s]$ | $w \times h = A$ | $t[s]$ | $\Delta A_{\text{ortho}}$ |
| cm42a | 4 / 10 | 79 | 37 × 57 = 2109 | < 0.01 | 35 × 22 = **756** | 65 | −64.15 % |
| i3 | 132 / 6 | 456 | 139 × 456 = 63384 | < 0.01 | 22 × 138 = **3036** | 5 | −95.21 % |

with only 79 gates, the proposed approach needs 13 times longer compared to *i3* with 456 gates.

Two potential remedies of this limitation are suggested: Firstly, reserving more space next to such configurations, as depicted in Fig. 7b. Alternatively, changing the order of the topological sorting, either before the application of the search algorithm or when encountering this configuration, could lead to placing the two fanouts later, thereby avoiding blockages.

By addressing these issues, which is left for future work, the proposed approach could potentially scale to accommodate logic networks with even more gates while still exhibiting fast runtimes like for the *i3* function in Table II, for which it only took 5 s to generate a layout with an area reduction of more than 95 % compared to the state of the art. Furthermore, the high-effort mode can be accelerated by parallelizing the exploration of multiple search space graphs, thereby regaining some of its forfeited performance.

## VIII. Conclusion

With the rise of *Field-coupled Nanocomputing* (FCN) as a promising post-CMOS technology, the necessity for efficient methods in automatic physical design becomes increasingly apparent. This study presented a novel approach that constructs a search space graph for gate placement and applies a modified $A^*$-search algorithm to yield layouts with near-optimal area, outperforming the state-of-the-art heuristic physical design algorithms *NanoPlaceR* and *ortho* by more than 24 % and 70 %, respectively. Compared to *NanoPlaceR*, the proposed approach demonstrates remarkable efficiency in generating valid circuit layouts with over 99.7 % runtime reduction. This amalgamation of scalability and efficiency enables advanced physical design for FCN and acts as a first step toward the creation of layouts with thousands of gates with minimal area overhead.

## References

[1] N. G. Anderson and S. Bhanja, Eds., *Field-Coupled Nanocomputing - Paradigms, Progress, and Perspectives*. Springer, 2014.

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

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

[4] ——, "Minimal Design of SiDB Gates: An Optimal Basis for Circuits Based on Silicon Dangling Bonds," in *NANOARCH*, 2023.

[5] ——, "The Need for Speed: Efficient Exact Simulation of Silicon Dangling Bond Logic," in *ASP-DAC*, 2024, pp. 576–581.

[6] R. Achal *et al.*, "Lithography for robust and editable atomic-scale silicon devices and memories," *Nat. Commun.*, vol. 9, no. 1, 2018.

[7] M. Walter *et al.*, "An Exact Method for Design Exploration of Quantum-dot Cellular Automata," in *DATE*, 2018, pp. 503–508.

[8] ——, "One-pass Synthesis for Field-coupled Nanocomputing Technologies," in *ASP-DAC*, 2021, pp. 574–580.

[9] ——, "Scalable Design for Field-Coupled Nanocomputing Circuits," in *ASP-DAC*, 2019, pp. 197–202.

[10] S. Hofmann *et al.*, "Late Breaking Results From Hybrid Design Automation for Field-coupled Nanotechnologies," in *DAC*, 2023, pp. 1–2.

[11] ——, "Thinking Outside the Clock: Physical Design for Field-coupled Nanocomputing with Deep Reinforcement Learning," in *ISQED*, 2024, pp. 1–8.

[12] M. Walter *et al.*, "fiction: An Open Source Framework for the Design of Field-coupled Nanocomputing Circuits," 2019, arXiv:1905.02477.

[13] ——, "The Munich Nanotech Toolkit (MNT)," in *IEEE-NANO*, 2024.

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

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

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

[17] M. B. Haider *et al.*, "Controlled Coupling and Occupation of Silicon Atomic Quantum Dots at Room Temperature," *Phys. Rev. Lett.*, vol. 102, p. 046805, 2009.

[18] T. Huff *et al.*, "Binary atomic silicon logic," *Nat. Electron.*, vol. 1, no. 12, pp. 636–643, 2018.

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

[20] 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.

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

[22] N. Pavliček *et al.*, "Tip-induced passivation of dangling bonds on hydrogenated Si(100)-2×1," *APL*, vol. 111, no. 5, p. 053104, 2017.

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

[24] 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.

[25] 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.

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

[27] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[28] P. E. 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.

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

[30] M. Walter *et al.*, "Placement and Routing for Tile-Based Field-Coupled Nanocomputing Circuits Is NP-Complete (Research Note)," *JETC*, vol. 15, no. 3, 2019.

[31] 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.

[32] ——, "Post-Layout Optimization for Field-coupled Nanotechnologies," in *NANOARCH*, 2023, pp. 1–6.

[33] ——, "Late Breaking Results: Wiring Reduction for Field-coupled Nanotechnologies," in *DAC*, 2024, pp. 1–2.

[34] A. Trindade *et al.*, "A Placement and Routing Algorithm for Quantum-dot Cellular Automata," in *SBCCI*, 2016, pp. 1–6.

[35] G. Fontes *et al.*, "Placement and Routing by Overlapping and Merging QCA Gates," in *ISCAS*, 2018, pp. 1–5.

[36] M. Walter *et al.*, "Verification for Field-coupled Nanocomputing Circuits," in *DAC*, 2020, pp. 1–6.

[37] K. McElvain, "IWLS'93 Benchmark Set: Version 4.0," Tech. Rep., 1993.