A Fully Planar Approach to Field-coupled Nanocomputing: Scalable Placement and Routing Without Wire Crossings

Benjamin Hien, Marcel Walter, Simon Hofmann, and Robert Wille

`https://www.cda.cit.tum.de/research/nanotech/`

Abstract—*Field-coupled Nanocomputing* (FCN) is a class of promising post-CMOS technologies that transmit information through electric or magnetic fields instead of current flow. They utilize basic building blocks called *cells*, which can form gates that implement Boolean functions. However, the design constraints for FCN circuits differ significantly from those for CMOS. One major challenge is that wires in FCN have to be realized as gates, i. e., they are constructed from cells and incur the same costs as gates. Additionally, all FCN technologies are fabricated on a single layer, e. g., a silicon surface, requiring all elements—gates and wires—to be placed within that same layer. Consequently, FCN employs special gates, called wire crossings, to enable signals to cross. While existing wire-crossing implementations are complex and were previously considered costly, initial efforts have aimed at minimizing their use. However, recent physical simulations and experiments on a quantum annealing platform have shown that currently used wire crossings in FCN significantly compromise signal stability, to the extent that circuits cannot function reliably. This work addresses that issue by introducing the first placement and routing algorithm that produces fully planar FCN circuits, eliminating the need for all wire crossings. For a comparative evaluation, a state-of-the-art placement and routing algorithm was also modified to enforce planarity. However, our proposed algorithm is more scalable and can handle inputs with up to $149k$ gates, enabling it to process circuits that are $182\times$ more complex than those handled by the modified state-of-the-art algorithm.

I. INTRODUCTION & MOTIVATION

Field-coupled Nanocomputing (FCN, [1]) encompasses emerging nanotechnologies that depart from CMOS by utilizing physical fields instead of transistors and current flow for computation. This signal transfer mechanism allows FCN to operate near the *Landauer Limit* [2], offering a highly energy-efficient technology for modern computing. Recent breakthroughs in FCN fabrication, particularly via *Silicon Dangling Bonds* (SiDBs, [3]), enable logic devices and interconnects smaller than $30\,\text{nm}^2$ [4], driving progress in the field. SiDBs also facilitate *Quantum-dot Cellular Automata* (QCA, [5]), a foundational technology for numerous studies [6]–[8].

Despite differing physical implementations, FCN technologies share similar design constraints in logic synthesis and physical design, allowing layouts for QCA and SiDB, at an abstract level, to be translated into one another [9]. However, these constraints differ significantly from CMOS. A key distinction is the high cost of wires in FCN, which,

unlike in CMOS, are as expensive as gates since both are built from elementary cells [10]. Hence, while the design process in CMOS commonly focuses on reducing delay, area, and power consumption [11], in FCN, design constraints like minimizing wiring effort are much more emphasized.

Additionally, FCN is limited to a single layout layer for both gates and wiring, whereas CMOS utilizes multiple metal layers for routing without signal integrity issues. Consequently, FCN relies on wire crossings—specialized gate types that enable signals to cross on the same layer—which are standard components in modern *Placement and Routing* (P&R) algorithms.

Since wire crossing implementations are physically complex, they were initially regarded as costly. Consequently, research focused on crossing-aware design for FCN technologies to reduce this cost factor, while balancing the significant area overhead associated with layouts that minimize crossing counts [12], [13]. These studies primarily targeted wire crossings during the logic synthesis process, although they did not fully eliminate them. Moreover, these methods have not been validated at the physical design stage due to the absence of a comprehensive design process, particularly lacking a crossing-aware P&R algorithm.

However, recent research on SiDB crossings [14], along with earlier studies on QCA crossings [15], has shown that wire crossings are ultimately impractical to implement in FCN. Thus, state-of-the-art P&R algorithms are categorically inapplicable. This emphasizes the need for P&R algorithms that avoid the use of wire crossings.

This work proposes a fully planar approach that eliminates wire crossings, addressing key challenges in FCN. To achieve this, we introduce a novel design flow for planar FCN circuits, featuring our key contribution: a scalable planar P&R algorithm. First, we show that while a state-of-the-art P&R method [8] can be adapted to generate planar layouts for small-scale networks, it lacks scalability and is insufficient for real-world applications. To overcome this, we present a new scalable P&R algorithm that eliminates wire crossings and processes circuits with up to $149k$ gates— $182\times$ larger than previous methods—paving the way for FCN to become a viable technology.

This paper is structured as follows: Section II overviews FCN technologies, while Section III reviews recent work on wire crossings. Section IV introduces our main contribution—a scalable, planar P&R algorithm. Section V presents experimental evaluations, comparing its performance to a modified P&R algorithm. Finally, Section VI concludes the paper.

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

(a) Basic FCN cells.
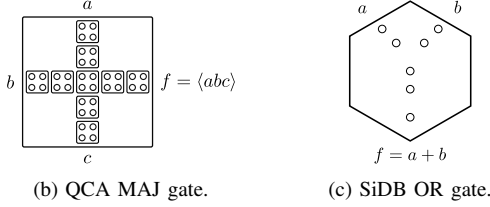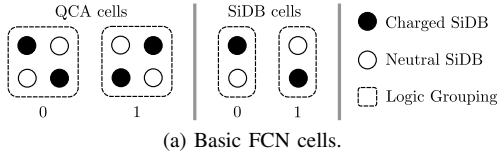

(b) QCA MAJ gate.   (c) SiDB OR gate.

Fig. 1: FCN technology implementations.
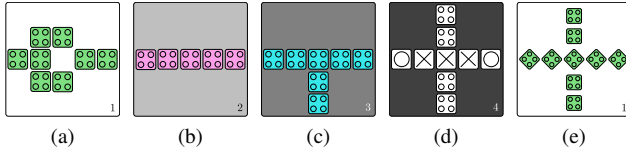

Fig. 2: QCA ONE [10] gate implementations of an (a) Inverter, (b) wire, (c) fan-out, (d) 3D- and (e) co-planar wire crossing.

(a) 2DDWave [20].   (b) USE [21].   (c) RES [22].

Fig. 3: Common clocking schemes for FCN circuit layouts.

II. BACKGROUND

This section reviews the foundational concepts essential for FCN technologies. Section II-A details the primary components of FCN, while Section II-B explores clocking mechanisms within FCN frameworks.

A. Cells and Gates

All FCN technologies share a fundamental building block called a *cell*, which can represent binary encoding [1]. The logic state is determined by the cell's electrostatic polarization. When cells are placed in close proximity, they can polarize each other, transmitting signals without the need for electric current [5].

A basic QCA cell has four quantum dots and two electrons, which, due to *Coulomb* interactions, occupy opposite corners, resulting in two stable states for binary 0 and 1. SiDB cells follow a similar principle but use only two quantum dots. Both cell types and their states are illustrated in Fig. 1a.

By exploiting the field interactions between adjacent cells, logic gates can be constructed using specific spatial arrangements of cells. QCA gates are typically arranged in squares, as illustrated in Fig. 1b, while SiDB gates form hexagons, as shown in Fig. 1c. Numerous gates have been proposed for FCN technologies [10], [16], [17]. In addition to the QCA majority gate depicted in Fig. 1b, which can implement an AND or OR gate by fixing one input to 0 or 1, respectively, exemplary implementations of the QCA ONE [10] standard cell library are shown in Fig. 2. An inverter can be achieved through diagonal coupling, as shown in Fig. 2a. The gates shown in Fig. 2b and Fig. 2c do not perform logic operations but instead serve as wire and fan-out implementations, respectively. This highlights that, in FCN, wires and gates
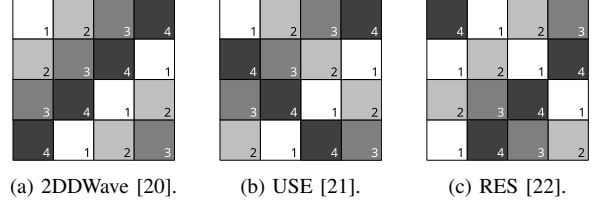
have similar implications for fabrication costs, area usage, and implementation delays.

B. Clocking

FCN circuits utilize clocking to synchronize signal propagation and control information flow. To efficiently manage clocking in large circuits, FCN layouts are subdivided into uniform sections called *tiles*, which are activated by an external field known as the *clock*. This clock is distributed via buried electrodes in the substrate [18], [19]. The clock works in four phases, numbered 1 through 4, enabling a pipeline-like signal flow through the circuit. Each tile holds either a gate or a wire segment, as seen in Fig. 2. When constructing a layout, signal synchronization is essential. To achieve this, adjacent tiles must possess consecutive clock phases, referred to as *local* synchronization, and signals that meet at the same tile must have traveled the same number of tiles, known as *global* synchronization. The layout design process can be simplified by using predefined extensible clocking schemes and placing gates and wires accordingly. Popular clocking schemes from the literature are visualized in Fig. 3.

III. RELATED WORK

While state-of-the-art research has considered wire crossings feasible for FCN layouts [7], [8], recent studies show that their physical implementations for both QCA and SiDBs are not viable for functional circuits [14], [15].

Two QCA wire crossing implementations have been proposed [17]. The first, 3D wire crossings (Fig. 2d), use a second QCA layer, allowing one signal to move up while the other remains on the original plane before returning. However, current fabrication technologies only support quantum-dot generation on a single silicon surface [23], and multi-layer implementations are not physically feasible at present, rendering this approach impractical.

The second approach, co-planar crossings (Fig. 2e), rotates one wire by $45°$ to avoid signal interference between the two orientations, enabling signals to cross on the same plane. However, quantum annealer simulations show a severe reliability drop, with logic correctness falling to 60% [15], making it unsuitable for dependable logic circuits. Furthermore, additional wiring is required to convert between rotated and standard cells, further reducing reliability and increasing area overhead, making this method equally impractical.

For SiDB crossings, simulations of various designs reveal extreme temperature sensitivity [14]. The most robust implementation operates reliably only up to $21.78\,\mathrm{K}$, far below liquid nitrogen levels ($77\,\mathrm{K}$), contradicting FCN's energy efficiency goals and making such crossings infeasible.
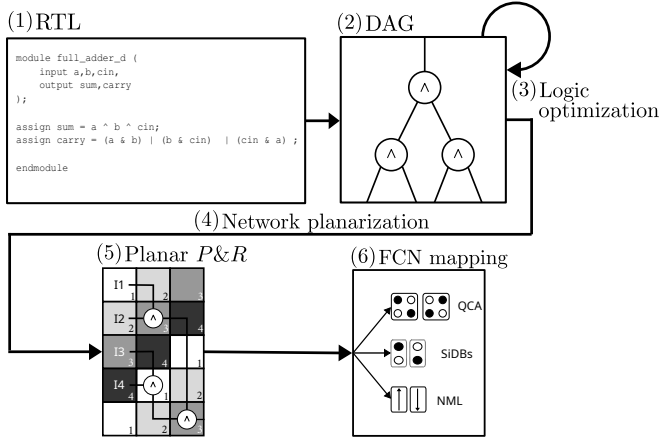
Fig. 4: Proposed design flow for fully planar FCN circuit layouts, starting from a high-level description and eventually yielding a cell-level layout of a specific technology.

These combined findings show that wire crossings do not function properly in FCN, necessitating their exclusion from the physical design of FCN. To the authors' knowledge, state-of-the-art algorithms rely on wire crossings and are therefore considered impractical in this work. To address this issue, this work presents a novel P&R algorithm that eliminates wire crossings. FCN layouts without wire crossings are subsequently referred to as *planar*.

IV. Proposed Approach

This section presents the first scalable P&R algorithm developed for fully planar FCN circuit layouts. To this end, first a high-level overview of a novel planar design flow for FCN is introduced in Section IV-A. From this high-level view, we explore the modifications needed at each step to design planar FCN layouts. Achieving planarity at the logic synthesis level is detailed in Section IV-B. Section IV-C discusses how planarity is preserved throughout the transition from logic synthesis to the physical design. Finally, Section IV-D presents the scalable P&R algorithm, taking all these considerations into account.

A. A Novel Design Flow for Fully Planar FCN Layouts

The proposed design flow transforms an abstract circuit description into a planar FCN layout, systematically eliminating wire crossings at each stage. Fig. 4 illustrates this process.

It begins at the Register-Transfer Level (RTL) (1), where the circuit's logic is defined programmatically at a high level. This RTL description is converted into a logic network (2), typically represented as a Directed Acyclic Graph (DAG) using structures like And-Inverter Graphs (AIGs) for efficient optimization (3). Traditionally, area and delay are minimized by reducing node count and network depth [11].

While area and delay remain important in FCN, minimizing wire crossings is crucial. At the logic network stage, some approaches reduce edge crossings as a proxy, even at the cost of area and delay [12], [13], while others eliminate crossings entirely to create planar logic networks [24]. Planarizing the logic network (4) is a key preprocessing step in this work.

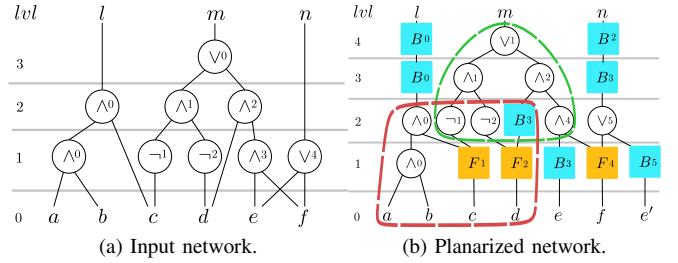(a) Input network. (b) Planarized network.

Fig. 5: Logic network planarization through balancing and node duplication.

However, conventional P&R algorithms do not preserve planarity, meaning that even if the input logic network is fully planar, wire crossings may still be introduced in the final layout. This fundamental limitation severely restricts the feasibility of FCN layouts.

To overcome this, we propose a P&R algorithm that guarantees planarity throughout the process (5), ensuring a fully planar layout without reintroducing wire crossings. This marks a significant departure from prior methods, making large-scale FCN designs feasible. Finally, the FCN layout can be adapted across different implementations [9], and the cell representations of the gates are mapped onto the layout in the respective technology (6).

The next section details the first step—planarizing the logic network—as the foundation for a fully planar FCN layout.

B. Network Planarization

The planarization process for a logic network [24] consists of three steps: fan-out substitution, network balancing, and node duplication. After fan-out substitution, only fan-out nodes have multiple outputs, typically limited to two signals, in accordance with FCN standard gate requirements [17].

The remaining steps prepare the network for a planar embedding—a drawing on a plane where no edges intersect. First, each node is assigned a level (the longest path from primary inputs (PIs) to the node) and a rank (its order within a level, numbered left to right). DAGs in this work are drawn bottom-up, with increasing levels. Fig. 5a illustrates an example network, showing levels on the left and ranks as small numbers beside each node's function.

To achieve a fully planar embedding, edges spanning multiple levels must be segmented with buffers, ensuring each edge spans only one level—this step is called network balancing. Next, the node duplication algorithm iteratively processes levels from primary outputs (POs) to PIs, reordering and duplicating nodes to eliminate crossings between adjacent levels, ultimately producing a fully planar network [24].

Example 1. *Consider the logic network in Fig. 5a and its planar embedding in Fig. 5b. First, fan-out substitution introduces fan-out nodes (orange boxes). Next, network balancing inserts buffers (blue boxes) to divide multi-level edges. Finally, node duplication eliminates the crossing between level 1 and the PIs by duplicating PI e. The assigned levels and ranks form a fully planar embedding of the network.*

(a) Exemplary valid P&R of green sub-network from Fig. 5b.

(b) Routing issue due to invalid node placement of red sub-network from Fig. 5b.
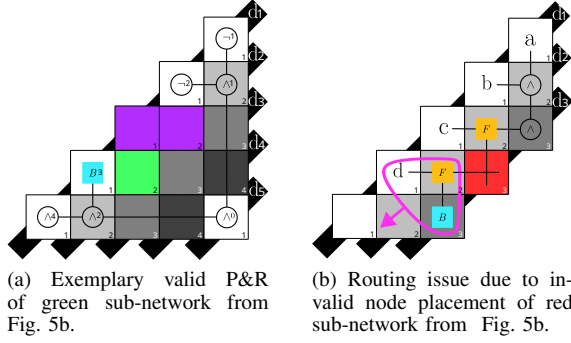
Fig. 6: P&R on 2DDWave clocking scheme.

C. Planarity Preservation

The planar embedding generated in the previous step serves as input for P&R. However, standard algorithms disregard this embedding, leading to wire crossings in the layout. To prevent this, the proposed algorithm must satisfy two key requirements: (1) the planar embedding must be preserved throughout P&R, maintaining the nodes' levels and ranks in the final layout, and (2) signal transfer must follow the level-by-level structure ensured by the balancing step.

To achieve this in a tile-based layout, all nodes from a given network level are placed along a diagonal of tiles in rank order, ensuring the first requirement. The second requirement is met by assigning the same clock number to all tiles on a diagonal, enabling synchronized activation of nodes at that level. Adjacent levels are placed on consecutive diagonals, with ascending clock numbers matching uniform FCN clocking. This results in the 2DDWave scheme (Fig. 3a), as illustrated by the following example:

Example 2. *A possible arrangement for the sub-network highlighted in green in Fig. 5b is shown on the layout grid in Fig. 6a. Level 2 nodes are placed along diagonal d_1, ordered by rank from top right to bottom left. All share clock number 1, represented by white tiles. The next level follows on diagonal d_2, assigned clock number 2, shown in light gray, and so forth.*

Although placement alone provides a large solution space, the placement from the previous example is not straightforward. The nodes must be arranged to allow routing between adjacent levels. The combined P&R problem is addressed in the following section.

D. Placement & Routing (P&R)

The P&R problem in FCN, inherently tied to clocking, is $\mathcal{N}!\mathcal{P}$-complete [25]. However, our approach reduces the search space by predefining clocking and fixing relative node positions through the planar embedding. Additionally, network balancing restricts routing to adjacent levels, enabling iterative routing. This leads to the P&R problem, described in the following.

With predefined clocking and fixed relative positions, a valid planar layout depends on two degrees of freedom. (1) Node placement: Nodes within a level are arranged diagonally by rank, but for valid routing to adjacent diagonals, they cannot always be placed directly next to each other. Empty

tiles, referred to as *gaps*, must be introduced, as shown in Fig. 6a (green and purple tiles). (2) Diagonal usage: Not all diagonals can be occupied, as seen in Fig. 6a, where diagonals d_3 and d_4 are left empty to enable proper routing.

We first define and compute gaps, which exist at each network level, corresponding to a diagonal in the layout. A gap occurs between adjacent nodes (i.e., nodes with consecutive ranks), with a gap value of 0 indicating direct adjacency. For a level l with N_l nodes, there are $N_l - 1$ gaps, stored in a gap vector v_l. Each level has its own gap vector, and all are combined into a gap array A_G for the entire network.

Gap computation depends solely on the network structure, including levels, ranks, and fan-in/out relationships. It follows an iterative process, where gaps at each level are determined by local network structure and previously computed gaps in adjacent levels. The formation of gaps follows three distinct cases:

1) *2-ary gaps*: When two consecutive pairs of nodes feed into separate 2-ary nodes (e.g., AND gates) on the next diagonal, gaps arise due to placement constraints, as shown in Fig. 6a. Each 2-ary node's position is determined by its predecessors: the northern predecessor sets the x-coordinate, and the western predecessor the y-coordinate ($\{x, y\} = \{x_{\text{north}}, y_{\text{west}}\}$). Since predecessors controlling the same coordinate (x or y) are spaced one tile apart, a gap (green tile) forms. The purple gap is addressed in the third case.

2) *Fan-out gaps*: Consider two adjacent fan-out nodes on the same diagonal, each with two distinct outputs. This requires placing four nodes on the next diagonal, but only three tiles are available for routing. As shown in Fig. 6b, both fan-out nodes attempt to use the same red-marked tile, causing a conflict. Resolving this requires introducing a gap between the fan-out nodes.

3) *Propagated gaps*: Gaps can propagate *forward* (PIs to POs) if an unconnected gap exists between two nodes at one level, as seen in Fig. 6a, where a purple-highlighted gap on diagonal d_1 moves forward. Similarly, *fan-out gaps* can propagate *backward* (POs to PIs) if nodes remain unconnected in the previous level. In Fig. 6b, pink-circled nodes shift, requiring both the fan-out node and its predecessor to adjust, affecting input spacing.

Gap computation for individual levels, based on three cases, is integrated into an iterative algorithm that first computes gaps and then propagates them forward and backward to resolve routing conflicts. It begins at the PI level with forward iteration, computing gaps for each level. If a fan-out gap causes a conflict, the algorithm switches to backward iteration, adjusting earlier levels to resolve it. Forward iteration then resumes until the POs are reached, completing the gap array. In practice, conflicts are more frequent in lower levels, while higher levels typically have larger gaps, reducing conflicts.

As mentioned, the second degree of freedom in placement involves leaving empty diagonals. This is necessary when fan-ins of 2-ary nodes are separated by a gap, as shown in Fig. 6a, where two AND nodes on diagonal d_2 with a gap of size 2 both feed into a node on diagonal d_5. For 2-ary nodes, where $\{x, y\} = \{x_{\text{north}}, y_{\text{west}}\}$, the number of
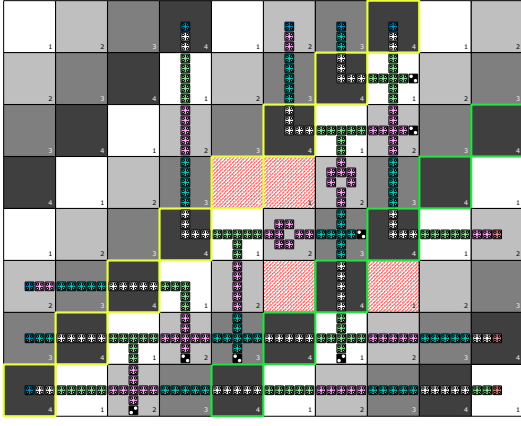
Fig. 7: Planar layout of the network shown in Fig. 5b. Yellow frames indicate the initial placement of PIs, red striped tiles represent gaps, and green-framed tiles mark an empty diagonal, following the algorithm used for scalable placement and routing.

empty diagonals N_d matches the gap size between fan-ins. This computation is included in the algorithm, with the value inserted into the gap vector of the preceding level.

Example 3. *In the planarized network (Fig. 5b), the algorithm starts with all gaps set to zero and propagates forward. At level 0, no gaps form. At level 1, two fan-out nodes lack a common fan-in at the next level, creating a fan-out gap. This gap propagates backward, adjusting the gap between PIs c and d in the preceding level. As a result, the updated gap vector for level 0 is $v_0 = [0, 0, 1, 0, 0, 0, 0]$, indicating a gap of size 1 between PIs c and d, with no gaps elsewhere. Forward propagation then resumes, iterating through all levels to compute the complete gap array: [[0,0,1,0,0,0,0], [0,1,0,0,0,0], [0,0,0,0,0,0], [0,1,0,0], [1,0,1]]*

After gap computation, the layout can be directly mapped, as illustrated in the following example:

Example 4. *Fig. 7 shows the final layout for the network and gap array from Example 3. The PIs are placed along a diagonal that accounts for their count and total gaps in the first level, highlighted by yellow frames, and are then routed to the layout's borders for accessibility.*

The gap between the second and third PI, represented in the first vector as [0,0,1,0,0,0,0], is marked with red stripes. Since the last entry of the next level's vector [0,1,0,0,0,0] is zero, the next level is placed on the following diagonal. The same scheme applies to subsequent levels.

A special case arises in the last level, where the final vector [1,0,1] ends with a 1, necessitating an empty diagonal, shown in green frames. This gap results from an AND node with predecessors separated by a gap of size 1. After all nodes are placed, the POs are routed to the layout's borders.

V. Experiments

The experiments were conducted on an AMD Ryzen 7 PRO 6850U with $32\,\mathrm{GB}$ of DDR5 RAM. To promote open research and data sharing, both the planarization and scalable P&R algorithms are publicly available as part of the

Munich Nanotech Toolkit (MNT) at `https://github.com/cda-tum/fiction`.

The state-of-the-art *graph-oriented layout design* (*gold*, [8]) was modified to generate planar layouts and executed with a $100\,\mathrm{s}$ timeout, serving as the reference. Both *gold* and the proposed algorithm were tested on benchmarks from *Trindade* [26], *Fontes* [27], and *IWLS93* [28]. A post-layout optimization (PLO) algorithm [6], [7] was adapted to maintain planarity and applied to layouts from both methods, minimizing area by rearranging gates. Logical correctness was verified using formal methods [29].

Table I presents the results. The left-most column lists each benchmark with its number of primary inputs (I) and outputs (O). The initial node count in the input logic network is provided for both approaches. Execution times for P&R and PLO, optimized layout area, and area improvement are also reported. PLO was executed with a $100\,\mathrm{s}$ soft timeout. The DIFFERENCE column tracks node count differences due to buffer addition/removal, area after P&R, and area after PLO. The final row averages results over benchmarks where *gold* produced a valid layout.

For the *Trindade* and *Fontes* suites, input networks for the proposed algorithm contain $77.59\,\%$ more nodes on average, leading to a $188.03\,\%$ area overhead. PLO reduces this to $75.55\,\%$ but finds no improvements for *gold* layouts, which are already near-optimal [8]. Despite this, the proposed algorithm completes all benchmarks in under $0.01\,\mathrm{s}$, whereas *gold* times out on *clpl*, *2bitAdderMaj*, and *xor5Maj*.

Scalability is further demonstrated by select *IWLS* benchmarks [28], where node counts range from 20k to 150k. The proposed approach successfully processes all benchmarks, while *gold* times out on each. The largest benchmark handled, *cordic*, is $182\times$ larger than *parity*, the largest benchmark *gold* could process. Due to PLO's scalability limitations, optimization improvements decline with increasing benchmark size, from $6.4\,\%$ for *x4* to $0.0\,\%$ for *cordic*.

These results demonstrate that the proposed algorithm scales efficiently for large benchmarks, while the adapted *gold* algorithm performs better on smaller circuits. Real-world circuits often contain thousands of nodes, making scalability crucial, especially for planar FCN layouts, where ensuring planarity increases the node count due to duplication. Thus, a scalable approach is essential for practical planar FCN design.

VI. Conclusion

This work introduced a scalable placement and routing (P&R) algorithm for Field-Coupled Nanocomputing (FCN), eliminating wire crossings through a fully planar design flow. By addressing the impracticality of wire crossings, we removed a key barrier in FCN's path to scalability and established a framework that aligns with FCN's distinctive layout constraints. Our approach extends prior methods, scaling beyond small networks to handle circuits up to $182\times$ larger, marking a crucial step toward large-scale, real-world FCN applications.

Future work will focus on refining optimization techniques across the design flow, further improving area efficiency and computational performance. These enhancements will enable FCN to better address the demands of modern computing and support even larger, more complex circuits.

TABLE I: Comparative experimental evaluation of the proposed fully planar physical design approach against the state of the art.

	Name	I	O	$|N|$	$W \times H$	$t_{gold}[s]$	$W_{PLO} \times H_{PLO}$	$\Delta A_{PLO}[\%]$	$t_{PLO}[s]$	$|N|$	$W \times H$	$t_{prop}[s]$	$W_{PLO} \times H_{PLO}$	$\Delta A_{PLO}[\%]$	$t_{PLO}[s]$	$\Delta|N|[\%]$	$\Delta A[\%]$	$\Delta A_{PLO}[\%]$
					BENCHMARK CIRCUIT		SoTA [8]					PROPOSED					DIFFERENCE	
Trindade [26]	mux21	3	1	9	5 × 3	0.10	5 × 3	0.00	0.00	13	6 × 3	0.00	5 × 3	16.67	0.00	44.44	20.00	±0.00
	xor2	3	1	9	3 × 5	0.09	3 × 5	0.00	0.00	12	6 × 3	0.00	5 × 3	16.67	0.00	33.33	20.00	±0.00
	xnor2	4	1	11	3 × 6	2.26	3 × 6	0.00	0.00	17	8 × 4	0.00	6 × 4	25.00	0.00	54.55	77.78	33.33
	par_gen	5	1	18	8 × 4	33.22	8 × 4	0.00	0.00	33	11 × 6	0.00	9 × 5	31.82	0.00	83.33	106.25	40.63
	HA	5	2	14	4 × 6	19.78	4 × 6	0.00	0.00	26	8 × 5	0.00	6 × 5	25.00	0.00	85.71	66.67	25.00
	FA	7	2	16	6 × 4	19.19	6 × 4	0.00	0.00	31	13 × 7	0.00	5 × 7	61.54	0.00	93.75	279.17	45.83
	par_check	16	1	42	5 × 17	100.00	5 × 17	0.00	0.00	84	22 × 16	0.00	10 × 17	51.70	0.00	100.00	314.12	100.00
Fontes [27]	xor	4	1	10	5 × 3	0.80	5 × 3	0.00	0.00	16	7 × 4	0.00	5 × 4	28.57	0.00	60.00	86.67	33.33
	1bitAdderAOIG	7	2	28	6 × 9	100.00	6 × 9	0.00	0.00	52	15 × 7	0.00	11 × 7	26.67	0.00	85.71	94.44	42.59
	t_5	7	2	21	6 × 8	100.00	6 × 8	0.00	0.00	35	11 × 8	0.00	8 × 8	27.27	0.00	66.67	83.33	33.33
	t	8	2	23	10 × 4	100.00	10 × 4	0.00	0.00	39	11 × 8	0.00	10 × 8	9.09	0.00	69.57	120.00	100.00
	c17	8	2	20	6 × 6	100.00	6 × 6	0.00	0.00	41	15 × 8	0.00	8 × 8	46.67	0.00	105.00	233.33	77.78
	b1_r1	8	4	28	5 × 10	100.00	5 × 10	0.00	0.00	50	11 × 9	0.00	8 × 9	27.27	0.00	78.57	98.00	44.00
	majority	8	1	27	8 × 12	100.00	8 × 12	0.00	0.00	54	11 × 9	0.00	10 × 9	9.09	0.00	100.00	3.13	−6.25
	newtag	9	1	28	6 × 10	100.00	6 × 10	0.00	0.00	49	14 × 9	0.00	11 × 8	30.16	0.00	75.00	110.00	46.67
	majority_5_r1	10	1	29	5 × 11	100.00	5 × 11	0.00	0.00	52	14 × 10	0.00	9 × 11	29.29	0.00	79.31	154.55	80.00
	1bitAdderMaj	14	1	72	43 × 8	100.00	43 × 8	0.00	0.00	151	44 × 22	0.00	21 × 16	65.29	0.00	109.72	181.40	−2.33
	xor5_r1	16	1	70	9 × 24	100.00	9 × 24	0.00	0.00	127	35 × 20	0.00	19 × 17	53.86	0.00	81.43	224.07	49.54
	clpl	18	5	42	*timeout limit reached*					158	50 × 19	0.00	18 × 17	67.79	0.01	276.19		
	cm82a_5	22	3	91	48 × 6	100.00	48 × 6	0.00	0.00	186	44 × 27	0.00	28 × 22	48.15	0.01	104.40	312.50	113.89
	2bitAdderMaj	57	2	243	*timeout limit reached*					573	129 × 61	0.00	61 × 58	55.04	0.13	135.80		
	xor5Maj	108	1	591	*timeout limit reached*					1215	400 × 188	0.00	132 × 156	72.62	1.80	105.58		
	parity	121	1	580	295 × 9	100.00	295 × 9	0.00	0.07	820	209 × 162	0.00	164 × 122	40.91	0.45	41.38	1175.25	653.60
IWLS93 [28]	x4	1749	71	3413	*timeout limit reached*					20 960	2470 × 1888	0.27	2362 × 1848	6.40	104.42			
	duke2	2109	29	3737	*timeout limit reached*					20 970	2775 × 2131	0.33	2739 × 2116	1.99	102.50			
	rd84	3239	4	5196	*timeout limit reached*					24 899	4031 × 3241	0.78	4016 × 3237	0.50	111.46			
	t481	4208	1	7144	*timeout limit reached*					44 244	8145 × 4213	1.52	8134 × 4211	0.18	123.92			
	c880	4737	26	8296	*timeout limit reached*					44 266	13365 × 4765	2.23	13348 × 4754	0.36	132.85			
	vda	5235	39	8679	*timeout limit reached*					66 960	6171 × 5303	3.99	6162 × 5294	0.32	141.97			
	table5	5235	39	11 255	*timeout limit reached*					62 693	6171 × 5303	3.99	6162 × 5294	0.32	141.97			
	table3	6988	14	11 261	*timeout limit reached*					68 878	11030 × 6993	5.47	11030 × 6992	0.01	185.50			
	apex3	8071	50	13 633	*timeout limit reached*					91 970	11422 × 8116	8.09	11422 × 8109	0.09	238.79			
	cordic	15 148	2	25 104	*timeout limit reached*					149 182	41884 × 15157	31.85	41884 × 15157	0.00	758.13			
Average																77.59	188.03	75.55

For each benchmark, I and O represent the logic network's number of primary inputs (PIs) and outputs (POs), respectively. The number of nodes (including PIs and POs) is given as $|N|$. For the gold algorithm, the buffers introduced during planarization are removed, resulting in a smaller node count. The layout dimensions, represented as width (W) and height (H), are provided both before and after post-layout optimization (PLO). The area improvement from PLO for each algorithm is denoted as ΔA_{PLO}. Run times for the $P\&R$ algorithms are given as t_{prop}, t_{gold}, and for PLO as t_{PLO}. The differences between the two algorithms are tracked in terms of the number of nodes in the logic network (due to removed buffers), denoted as $\Delta|N|$, as well as the area difference before PLO (ΔA) and after PLO (ΔA_{PLO}).

REFERENCES

[1] N. G. Anderson *et al.*, *Field-coupled Nanocomputing: Paradigms, Progress, and Perspectives*. New York: Springer, 2014.

[2] R. Landauer, "Irreversibility and Heat Generation in the Computing Process," *IBM J. Res. Dev.*, vol. 5, no. 3, pp. 183–191, 1961.

[3] J. Pitters, J. Croshaw, R. Achal, L. Livadaru, S. Ng, R. Lupoiu, T. Chutora, T. Huff, K. Walus, and R. Wolkow, "Atomically Precise Manufacturing of Silicon Electronics," *ACS Nano*, 2024.

[4] T. Huff, H. Labidi, M. Rashidi, L. Livadaru, T. Dienel, R. Achal, W. Vine, J. Pitters, and R. A. Wolkow, "Binary Atomic Silicon Logic," *Nat. Electron.*, vol. 1, pp. 636–643, 2018.

[5] C. S. Lent, B. Isaksen, and M. Lieberman, "Molecular quantum-dot cellular automata," *J. Am. Chem. Soc.*, vol. 125, no. 4, pp. 1056–1063, 2003.

[6] S. Hofmann, M. Walter, and R. Wille, "Late Breaking Results: Wiring Reduction for Field-coupled Nanotechnologies," in *DAC*, 2024.

[7] ——, "Post-Layout Optimization for Field-coupled Nanotechnologies," in *NANOARCH*, 2023.

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

[9] ——, "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.

[10] D. A. Reis, C. A. T. Campos, T. R. B. S. Soares, O. P. V. Neto, and F. S. Torres, "A Methodology for Standard Cell Design for QCA," in *ISCAS*, 2016, pp. 2114–2117.

[11] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG Rewriting a Fresh Look at Combinational Logic Synthesis," in *DAC*, 2006.

[12] D. S. Marakkalage, M. Walter, S.-Y. Lee, R. Wille, and G. De Micheli, "Technology Mapping for Beyond-CMOS Circuitry with Unconventional Cost Functions," in *IEEE-NANO*, 2024, pp. 51–56.

[13] B. Hien, M. Walter, and R. Wille, "Reducing Wire Crossings in Field-Coupled Nanotechnologies," in *IEEE-NANO*, 2024, pp. 155–160.

[14] J. Drewniok, M. Walter, and R. Wille, "Temperature Behavior of Silicon Dangling Bond Logic," in *IEEE-NANO*, 2023, pp. 925–930.

[15] J. Retallick, M. Babcock, M. Aroca-Ouellette, S. McNamara, S. Wilton, A. Roy, M. Johnson, and K. Walus, "Algorithms for Embedding Quantum-Dot Cellular Automata Networks onto a Quantum Annealing Processor," 2017.

[16] M. Walter, S. S. H. Ng, K. Walus, and R. Wille, "Hexagons are the Bestagons: Design Automation for Silicon Dangling Bond Logic," in *DAC*, 2022.

[17] P. D. Tougaw and C. S. Lent, "Logical devices implemented using Quantum Cellular Automata," *J. Appl. Phys.*, vol. 75, no. 3, pp. 1818–1825, 1994.

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

[19] 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.

[20] V. Vankamamidi, M. Ottavi, and F. Lombardi, "Clocking and Cell Placement for QCA," in *IEEE-NANO*, vol. 1, 2006, pp. 343–346.

[21] C. A. T. Campos, A. L. P. Marciano, O. P. V. Neto, and F. Sill Torres, "USE: A Universal, Scalable, and Efficient Clocking Scheme for QCA," *TCAD*, vol. 35, no. 3, pp. 513–517, 2016.

[22] M. Goswami, A. Mondal, M. H. Mahalat, B. Sen, and B. K. Sikdar, "An Efficient Clocking Scheme for Quantum-dot Cellular Automata," *Int. J. Electron. Lett.*, vol. 8, no. 1, pp. 83–96, 2020.

[23] R. A. Wolkow, L. Livadaru, J. Pitters, M. Taucer, P. Piva, M. Salomons, M. Cloutier, and B. V. C. Martins, *Silicon Atomic Quantum Dots Enable Beyond-CMOS Electronics*. Springer, 2014, pp. 33–58.

[24] A. Chaudhary, D. Z. Chen, X. S. Hu, M. T. Niemier, R. Ravichandran, and K. Whitton, "Fabricatable Interconnect and Molecular QCA Circuits," *TCAD*, vol. 26, no. 11, pp. 1978–1991, 2007.

[25] M. Walter, R. Wille, D. Große, F. Sill Torres, and R. Drechsler, "Placement and Routing for Tile-based Field-coupled Nanocomputing Circuits is $\mathcal{NP}$-complete," *JETC*, vol. 15, no. 3, 2019.

[26] A. Trindade, R. Ferreira, J. A. M. Nacif, D. Sales, and O. P. V. Neto, "A Placement and Routing Algorithm for Quantum-dot Cellular Automata," in *SBCCI*, 2016.

[27] G. Fontes, P. A. R. L. Silva, J. A. M. Nacif, O. P. V. Neto, and R. Ferreira, "Placement and Routing by Overlapping and Merging QCA Gates," in *ISCAS*, 2018.

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

[29] M. Walter, R. Wille, F. Sill Torres, D. Große, and R. Drechsler, "Verification for Field-coupled Nanocomputing Circuits," in *DAC*, 2020.