Minimizing Area and Delay in Planar Physical Design for Field-coupled Nanocomputing

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

***Abstract*— Field-coupled Nanocomputing (FCN) is a promising post-CMOS technology that transmits information via electric or magnetic fields rather than current flow. It uses nanoscale cells to implement logic gates and interconnects, enabling minimal circuit footprints and high computational density. These properties make FCN well-suited for future high-performance applications such as artificial intelligence and scientific computing. Unlike CMOS, FCN circuits are constrained to a single physical layer, requiring logic and interconnects to coexist in the same plane. In FCN layouts, wire crossings are commonly used to route intersecting signals. However, experimental and simulation studies have shown that such crossings degrade signal stability and reliability, making them unsuitable for practical FCN designs. As a result, planar layouts that completely avoid wire crossings are a strict requirement for reliable circuit implementation. This work presents a placement and routing algorithm that generates fully planar FCN layouts while optimizing both area and delay. In contrast to the current state-of-the-art planar physical design flow, which relies on greedy heuristics, our method performs placement and routing in a level-by-level manner, effectively minimizing wiring overhead within each level. Experimental results demonstrate that our approach achieves an average reduction of $21.09\,\%$ in layout area and $15.24\,\%$ in delay, providing a scalable and efficient solution for practical FCN circuit implementation.**

I. Introduction & Motivation

As demands from artificial intelligence and other compute-intensive applications continue to grow, conventional CMOS technology approaches its physical scaling limits. Sustaining further performance improvements therefore requires the exploration of alternative computing paradigms.

Field-Coupled Nanocomputing (FCN, [1]) is a promising candidate in which information is transmitted through electric or magnetic field interactions rather than current flow. Its nanoscale cells enable extremely dense circuits and potentially terahertz operating frequencies [2].

Recent fabrication [3], [4] and simulation [5] advances in Silicon Dangling Bonds (SiDBs) have significantly increased the practical viability of FCN. SiDB technology further enabled Quantum-dot Cellular Automata (QCA), which form the foundation of much of the field's research [6]. Although SiDB and QCA differ in their physical realization, their abstract design principles are sufficiently similar to allow layout translation between the technologies [7].

Benjamin Hien, Marcel Walter, and Robert Wille are with the Chair for Design Automation, Technical University of Munich, Germany. Marcel Walter, and Robert Wille are also with the Munich Quantum Software Company (MQSC) GmbH, Garching near Munich, Germany. E-mail: {benjamin.hien, marcel.walter, robert.wille}@tum.de

Despite these advances, efficient design automation for FCN remains a major challenge. Established CMOS design flows have been refined over decades to optimize area and delay [8], [9]. However, these techniques cannot be directly applied to FCN due to its fundamentally different physical constraints.

In FCN, logic gates and interconnects are realized using the same physical structures and therefore incur identical cost [10]. This eliminates the traditional separation between logic and routing and fundamentally alters design trade-offs. Furthermore, FCN circuits are typically restricted to a single physical layer for both logic and interconnects, unlike CMOS technologies that rely on multiple metal layers to resolve routing conflicts.

To enable signal crossings on a single layer, FCN layouts often employ dedicated wire-crossing structures. However, both experimental and simulation studies for SiDB [5] and QCA [11] indicate that such crossings significantly reduce signal stability. Consequently, reliable FCN layouts must avoid wire crossings entirely. We refer to layouts satisfying this constraint as *planar*.

A recent design flow enforces planarity during logic synthesis by constructing planar logic networks that can subsequently be placed and routed without introducing crossings [12]. While this guarantees valid layouts, the resulting placement and routing stage relies on greedy wiring heuristics that introduce substantial area and delay overheads and limit the integration of more advanced optimization techniques.

In this work, we present an improved placement and routing algorithm for FCN that guarantees planar layouts while explicitly optimizing physical design quality. By targeting wiring overhead—a major contributor to both layout area and signal delay—our method overcomes key limitations of existing approaches. Experimental results demonstrate average reductions of $21.09\,\%$ in layout area and $15.24\,\%$ in delay compared to the state-of-the-art, while maintaining comparable scalability.

The remainder of this paper is organized as follows. Section II introduces background on FCN and logic networks. Section III reviews related work. Section IV presents the proposed planar placement and routing algorithm. Section V reports experimental results, and Section VI concludes the paper.

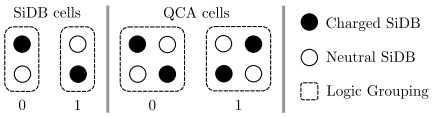
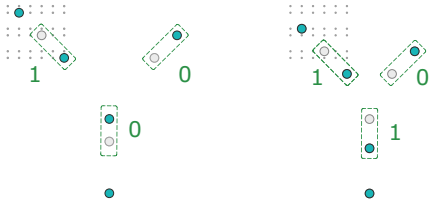


Fig. 1: FCN cell implementations using SiDBs and QCA.



(a) SiDB AND gate with inputs set to `10` [14].

(b) SiDB OR gate with inputs set to `10` [14].

Fig. 2: SiDB AND and OR gate implementations.

II. Background

This section provides an overview of FCN technologies and their application in logic circuit design and logic networks.

A. Cells and Gates

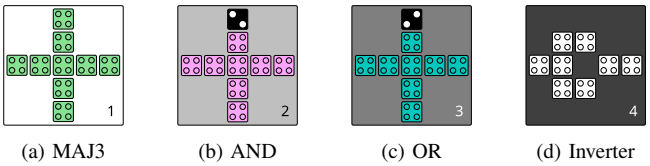
One of the most promising FCN implementations is based on Silicon Dangling Bonds (SiDBs [3]). SiDBs are created on a hydrogen-passivated silicon surface (H-Si(100)-2×1) by removing hydrogen atoms using a *Scanning Tunneling Microscope* (STM) [3], [4]. Each SiDB forms a quantum dot that can hold 0, 1, or 2 electrons depending on its electrostatic environment [13].

Logic is implemented using SiDB pairs that share a single excess electron whose position—left or right—encodes binary states. This model, shown in Figure 1, is known as Binary Dot Logic (BDL) [3]. Nearby SiDBs or external electric fields influence the charge configuration, enabling signal propagation.

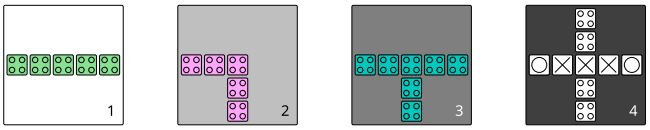
Specific SiDB arrangements realize logic gates [3]. For example, AND and OR gates (Figure 2) use additional SiDBs called *perturbers* as inputs that shift the shared electron via electrostatic interaction. This enables complete logic circuits at atomic scale with gate areas below $30\,\mathrm{nm}^2$.

Although SiDBs offer significant advantages, physical design research often uses Quantum-dot Cellular Automata (QCA [15]) due to their easier visualization. We therefore illustrate examples using QCA layouts. However, the proposed algorithms are technology-independent since QCA layouts can be translated into SiDB layouts [7].

A QCA cell consists of four quantum dots and two mobile charges whose Coulomb repulsion forces them into opposite corners, representing binary 0 and 1 (Figure 1, right). Logic gates are formed by arranging these cells on uniform 5×5 *tiles*, as in the QCA ONE library [10] (Figure 3). In both QCA and SiDB, wires use the same structures as logic gates, so interconnects incur identical area and delay cost.



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



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

Fig. 3: The QCA ONE gate library [10].

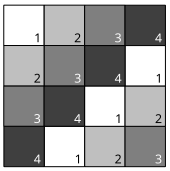


Fig. 4: The 2DD wave clocking scheme [19].

B. Clocking

FCN circuits use a clocking mechanism to synchronize signal propagation and control information flow. Layouts are divided into tiles, each containing a logic gate or wire segment, and are controlled by an external clock, typically distributed via buried electrodes [16].

The FCN clock operates in four distinct phases (1 through 4): hold, release, relax, and acquire, enabling inherently pipelined signal flow. Correct synchronization [17], [18] is essential and occurs on two levels. *Local synchronization* requires adjacent tiles to use consecutive clock phases for smooth transitions. *Global synchronization* ensures all signals arriving at a gate pass through the same number of phases, enabling correct logic evaluation.

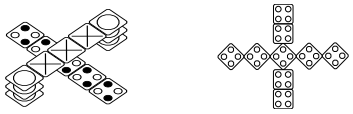
To support scalable design, predefined clocking schemes assign phases uniformly to guide placement. The *2DDWave* scheme [19] is particularly effective, ensuring both local and global synchronization. With its consistent top-left to bottom-right signal flow, delay scales with layout area, as signals traversing larger layouts must pass through more tiles, each corresponding to an additional clock zone and thus increasing latency. Figure 4 illustrates the 2DDWave clocking scheme [19].

C. Wire Crossings

Due to the single-layer fabrication of QCA and SiDB technologies, implementing reliable wire crossings is highly challenging.

Two crossing strategies have been proposed for QCA [20]. The first, *three-dimensional crossings* (Figure 5a), lifts one signal onto a second layer. While conceptually valid, it is impractical since current fabrication supports quantum dots only on a single surface [21].

The second, *coplanar crossings* (Figure 5b), intersects wires on the same layer by rotating one wire by $45°$ to re-



(a) 3D QCA wire crossing. (b) Co-Planar wire crossing.

Fig. 5: Different QCA wire crossing implementations.

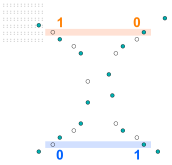


Fig. 6: SiDB crossing implementation.

duce interference. However, quantum annealing simulations report severe signal degradation, with correctness dropping to about 60 % [11]. Moreover, rotated signals must be restored, introducing additional complexity.

SiDB crossings (Figure 6) exhibit similar limitations. Simulations show strong temperature sensitivity, with stable operation only up to 21.78 K [5], far below practical cooling methods such as liquid nitrogen (77 K).

Consequently, reliable wire crossings are not feasible in QCA or SiDB, requiring *planar physical design* in which circuits are constructed without crossings.

D. Logic Networks

A *logic network* is a directed acyclic graph in which nodes represent Boolean functions, typically implemented as logic gates. Nodes without incoming edges are *primary inputs* (PIs), while nodes without outgoing edges are *primary outputs* (POs). Edges represent signal flow: each node receives inputs from its fanins and may drive other nodes as fanouts.

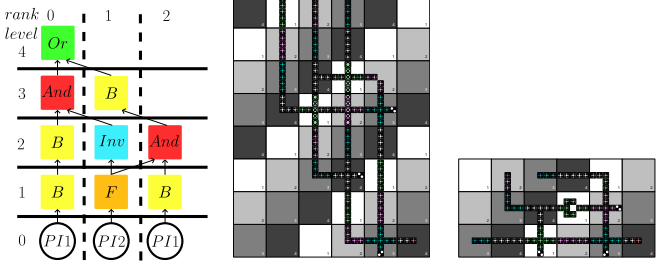
The *level* of a node is the length of the longest path from any PI to that node, and the network *depth* is the maximum level, i.e., the longest PI–PO path. The *rank* defines the ordering of nodes within the same level. Together, level and rank provide a two-dimensional structure for organizing nodes in layout and visualization.

A graph is *planar* if it can be drawn without edge crossings; such a drawing is called a *planar embedding*. In logic networks, this corresponds to assigning nodes positions based on their level and rank, enabling crossing-free layouts during placement and routing.

III. Related Work

Several placement and routing (P&R) approaches have been proposed, including scalable deterministic algorithms [6]. However, most rely on wire crossings, which are infeasible in FCN due to its single-layer constraint. While some works attempt to reduce crossings [22], [23], this work—motivated by their limitations—aims to eliminate them entirely.

Prior work [12], referred to as the SotA in this paper, introduced a complete flow for planar FCN circuits. The logic network is first path balanced with buffers on multi-level



(a) Planar MUX embedding.

(b) Planar MUX embedding placed and routed using `ortho` [6], invalidating planarity.

(c) Planar MUX embedding placed and routed with planar placement and routing, preserving planarity.

Fig. 7: Planarized MUX graph placed and routed once with `ortho` [6] and once with planar placement and routing.

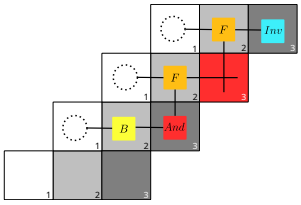
edges. Fanout substitution limits each fanout node to two outputs, forming fanout trees. Structural transformations [24] eliminate edge crossings, producing a planar logic network and embedding. The P&R stage then starts from this network, with the main challenge being to preserve its planarity in the physical layout.

Example 1. *Figure 7a shows a 2:1 multiplexer (MUX) after path balancing, fanout substitution, and planarization, together with a valid planar embedding. Preserving this embedding during P&R is non-trivial. As shown in Figure 7b, applying the scalable* `ortho` *algorithm [6] introduces wire crossings, indicating that the embedding is not preserved.*

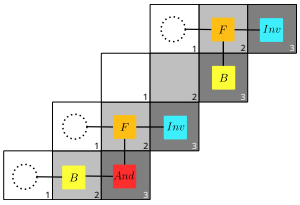
To maintain planarity, the planar embedding of the logic network is transferred to the layout, defining a structure of *depth* and *rank*. Nodes with the same depth lie on the same diagonal, with signal flow from the top left to the bottom right. Diagonals run from the upper right to the lower left, and within each the *rank* determines node order. This structure aligns with the *2DDWave* clocking scheme [19] (Figure 4) and enables a crossing-free layout.

Example 2. *Consider again the MUX from Figure 7a. Figure 7c shows the result of applying the SotA planar P&R algorithm [12]. Nodes from the same logic level are placed on the same diagonal while preserving their rank order.*

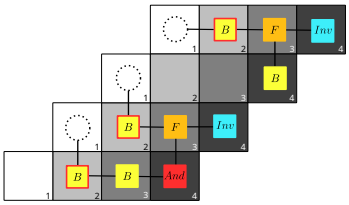
Although trivial for Example 2, transferring the planar embedding to a layout is challenging. It requires determining distances between nodes on the same diagonal and the spacing between diagonals while routing connections between diagonals without introducing crossings. The SotA [12] addresses this through a preprocessing step that computes spacing directly from the logic network. It iterates through levels to detect and resolve layout conflicts by inserting *gaps*. A conflict occurs when the fanouts of two gates cannot be placed on the next diagonal without overlap.



(a) Conflict: Two nodes are to be placed onto the same tile.



(b) SotA solution: Conflict resolved by pushing nodes to earlier levels.



(c) Proposed solution: Conflict resolved using excess wiring.

Fig. 8: Conflict management in SotA and proposed algorithm.

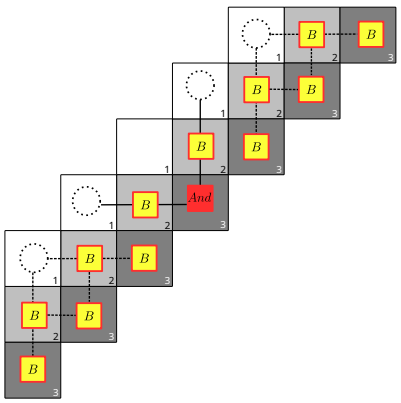


Fig. 9: 2-input nodes can cause excess wiring. Other nodes within the same level as the 2-input node contributing to the excess wiring need to be routed onto the same diagonal and still retain some placement flexibility.

Example 3. *Figure 8a shows such a conflict, marked by a red tile. Both fanout nodes attempt to place a node at the same location. Since the target is not a shared fanin, two different nodes would occupy the same tile, which is infeasible.*

When a conflict is detected, additional space must be introduced on the diagonal and the required gap propagated to earlier levels. This requires precise distance calculations along each diagonal, significantly increasing the algorithm's complexity.

Example 4. *Figure 8b shows how the SotA resolves the conflict by shifting previous diagonals to create space. In the worst case, spacing changes may propagate back to the PIs. As this is derived purely from network structure, the algorithm becomes complex and often area-inefficient, with gaps potentially spanning the entire layout.*

Another challenge arises when placing 2-input gates with widely separated fanins. Such gates cannot be placed on the next diagonal but must align with the spatial distribution of their fanins. Due to the planar embedding constraint, all nodes in the same level must then be routed onto this new diagonal. This additional routing effort, called *excess wiring*, is handled greedily by the SotA due to its already high complexity.

Example 5. *Figure 9 shows a case where the fanins of an AND node are separated by a gap on the previous diagonal. Due to the planar embedding, all nodes from the same level must be routed onto the new diagonal. The remaining nodes have three placement options; the SotA approach simply selects the leftmost buffer.*

Consequently, while the algorithm successfully produces planar layouts, it neglects significant area optimizations. This shortcoming motivates the new algorithm introduced in the next section.

IV. Proposed Placement and Routing Approach

We propose a placement and routing algorithm for planar FCN layouts that builds on the SotA design flow [12] and improves it through targeted optimizations. The goal is to maintain planarity while reducing wiring overhead, thereby lowering area and delay. Our method addresses excess wiring and tile conflicts using structured, non-greedy strategies that are particularly effective for complex networks, where naive routing often incurs large area and delay penalties. Owing to its modular design, the algorithm integrates seamlessly into existing physical design flows.

A. Proposed P&R Algorithm

The algorithm places and routes a planar network level by level while preserving the planar embedding. Each logic level maps to a diagonal, with nodes placed in rank order.

Unlike the SotA, which repositions nodes in earlier levels, the proposed algorithm resolves placement conflicts through rewiring. This introduces additional wiring from two sources: (i) *excess wiring*, when a two-input gate has non-adjacent fanins, and (ii) *conflict wiring*, added to avoid tile overlaps.

To reduce routing overhead, both cases are handled jointly. As shown in Algorithm 1, the algorithm begins by placing all PIs adjacently on the first diagonal and proceeds level by level in topological order. For each level, it computes the required wiring (Section IV-B) and places nodes in rank order. This continues until all levels are placed, yielding a planar layout with minimal area overhead.

Algorithm 1: High-Level Planar Layout Algorithm

Input: Logic network N
Output: Planar layout L with minimized area overhead
1 Place all PIs adjacently on the first diagonal of L;
2 **foreach** *logic level l in topological order* **do**
3 Compute wiring for l using ResolveWiring(l);
4 Place all nodes in l on the assigned diagonal of L in rank order;
5 **return** L

Algorithm 2: RESOLVEWIRING(l): Combined Wiring Minimization

Input: Logic level l with nodes and gate connections
Output: Updated routing coordinates for all nodes in l

Identify all clusters $\mathcal{C}$ in l (conflicts not involving 2-input gates);
foreach *cluster* $C \in \mathcal{C}$ **do**
    Compute conflict wiring for C;
foreach *cluster or 2-input gate in* l *in order of increasing rank* **do**
    Propagate y-values to resolve conflicts and align placement with excess wiring;
foreach *cluster or 2-input gate in* l *in order of decreasing rank* **do**
    Propagate x-values to resolve conflicts and align placement with excess wiring;
return *Updated coordinates for nodes in* l

B. Conflict and Excess Wiring Management

The following describes how wiring is computed for each case and how both are combined into the final routing plan for a level.

1) Conflict Wiring: Placement conflicts are resolved by rerouting nearby nodes to create space. Nodes with lower ranks (above and to the right of the conflict tile) are shifted rightward (x-direction), while higher-ranked nodes (below and to the left) are shifted downward (y-direction), ensuring sufficient separation.

Example 6. *To resolve the conflict in Figure 8a, an empty tile must be inserted between the two fanouts. As shown in Figure 8c, the lower-ranked node is routed rightward and the higher-ranked node downward.*

Since each new diagonal provides one additional tile, every conflict extends wiring to a later diagonal. The required shift for node i is $x_i = \sum_{j>i} \text{conflict}[j]$ and $y_i = \sum_{j \leq i} \text{conflict}[j]$.

Here, $\text{conflict}[j] = 1$ if a conflict occurs at rank j, and 0 otherwise. x_i accumulates horizontal offsets from higher ranks, while y_i accumulates vertical offsets from the current and lower ranks. This preserves spacing, signal order, and prevents overlaps.

2) 2-Input Gate Excess Wiring: Two-input gates require additional spacing when their fanins originate from non-adjacent ranks, as illustrated in Figure 9. This spacing must be respected during placement of the next level and is referred to as *2-input excess wiring*. Unlike wiring used to resolve placement conflicts, this routing is flexible and can be adjusted locally.

3) Combined Wiring Computation: When processing a level, the algorithm must coordinate both 2-input excess wiring and conflict wiring. If handled independently, the wiring overhead sums up for every occurrence of excess wiring and conflict. Since excess wiring already provides local spacing without a fixed routing direction, it can be reused to resolve conflicts, avoiding redundant routing.

Rather than introducing additional routing solely for conflict resolution, the algorithm reuses available excess wiring whenever possible. This reduces overall wiring overhead and helps maintain compact layouts.

Algorithm 2 formalizes this strategy. It begins by identifying clusters within a level, defined as groups of adjacently ranked nodes not separated by any 2-input gate. For each cluster, the required conflict wiring is computed.

During routing, the wiring requirements of clusters and adjacent 2-input gates are compared. If the gate requires more spacing, the cluster is shifted toward it using its available flexibility. Conversely, if the cluster requires more spacing, the gate is shifted—rightward in the x-direction for lower ranks or downward in the y-direction for higher ranks.

Finally, a two-phase propagation is applied: y-values propagate from lower to higher ranks, followed by x-values from higher to lower ranks. This ensures that conflicts are resolved, nodes are placed compactly, and the nearest valid diagonal is selected.

V. EXPERIMENTS

In this section, we present the results of an experimental evaluation of the proposed planar physical design algorithms on logic networks derived from established benchmark circuits [25]–[27] reaching up to 150k gates. The experiments were conducted on an AMD Ryzen 7 PRO 6850U system with 32 GB of DDR5 RAM. To support reproducibility and open research, the proposed algorithm is publicly available as part of the *Munich Nanotech Toolkit* (MNT) at `https://github.com/cda-tum/fiction`.

We compare the proposed approach against the SotA algorithm, with a particular focus on layout area, which correlates with delay through geometric scaling laws. Additionally, all generated layouts were formally verified to ensure functional correctness.

All results are shown in Table I. The proposed approach outperforms the SotA in terms of layout area on all benchmarks, except for a few small circuits where no optimization is possible due to the simplicity of the networks. The most significant relative area improvement was achieved for *xor5Maj* with a reduction of $64.48\,\%$. In absolute terms, *cordic* stands out, as a large amount of layout area was saved with $46.46\,\%$, corresponding to over 200 million tiles. This also demonstrates the scalability of the proposed algorithm, which places and routes a benchmark with nearly 150k gates in only $36.87\,\text{s}$.

A clear trend can be observed where larger benchmarks with more nodes benefit more from the proposed algorithm regarding area savings. This is likely because such networks introduce more placement conflicts and 2-input gate connections that require excess wiring. The proposed method reduces these effectively, resulting in significant area savings. Since real-world applications are likely to be much larger, the impact of these savings becomes even more critical.

Concerning runtime, both algorithms complete the small benchmarks from the Trindade and Fontes suites in less than $0.01\,\text{s}$. For the larger IWLS circuits the runtime ranges from a $22.19\,\%$ decrease to a $17.60\,\%$ increase. Interestingly, the most significant runtime increase occurs for the smallest

TABLE I: Comparison between SotA and proposed layout in terms of area and runtime.

	Benchmark Circuit				SotA [12]				Proposed				Difference		
	Benchmark	I	O	$\lvert N\rvert$	$W_{\text{sota}} \times H_{\text{sota}}$	A_{sota}	D_{sota}	t_{sota}[s]	$W_{\text{prop}} \times H_{\text{prop}}$	A_{prop}	D_{prop}	t_{prop}[s]	ΔA[%]	ΔD[%]	Δt[%]
Trindade [25]	xor2	3	1	11	6 × 3	18	7	0.00	6 × 3	18	7	0.00	±0.00	±0.00	±0.00
	mux21	3	1	12	6 × 3	18	7	0.00	6 × 3	18	7	0.00	±0.00	±0.00	±0.00
	xnor2	4	1	16	8 × 4	32	10	0.00	8 × 4	32	10	0.00	±0.00	±0.00	±0.00
	HA	5	2	24	8 × 5	40	9	0.00	8 × 5	40	9	0.00	±0.00	±0.00	±0.00
	FA	6	2	25	10 × 6	60	13	0.00	9 × 6	54	12	0.00	10.00	7.69	±0.00
	par_gen	5	1	32	11 × 6	66	14	0.00	10 × 6	60	13	0.00	9.09	7.14	±0.00
	par_check	16	1	83	22 × 16	352	30	0.00	21 × 16	336	29	0.00	4.55	3.33	±0.00
Fontes [26]	xor	4	1	15	7 × 4	28	9	0.00	7 × 4	28	9	0.00	±0.00	±0.00	±0.00
	t_5	7	2	33	11 × 8	88	13	0.00	11 × 7	77	12	0.00	12.50	7.69	±0.00
	t	8	2	37	11 × 8	88	15	0.00	11 × 8	88	15	0.00	±0.00	±0.00	±0.00
	c17	8	2	39	15 × 8	120	17	0.00	13 × 8	104	16	0.00	13.33	5.88	±0.00
	b1_r2	7	4	43	11 × 9	99	14	0.00	11 × 9	99	14	0.00	±0.00	±0.00	±0.00
	newtag	9	1	48	14 × 9	126	18	0.00	13 × 9	117	17	0.00	7.14	5.56	±0.00
	1bitAdderAOIG	7	2	50	15 × 7	105	16	0.00	14 × 7	98	16	0.00	6.67	±0.00	±0.00
	majority_5_r1	10	1	51	14 × 10	140	19	0.00	12 × 10	120	17	0.00	14.29	10.53	±0.00
	majority	8	1	53	11 × 9	99	16	0.00	12 × 8	96	16	0.00	3.03	±0.00	±0.00
	xor5_r1	16	1	126	35 × 20	700	45	0.00	26 × 20	520	38	0.00	25.71	15.56	±0.00
	1bitAdderMaj	15	1	151	45 × 22	990	57	0.00	24 × 17	408	33	0.00	58.79	42.11	±0.00
	clpl	20	5	177	44 × 20	880	52	0.00	25 × 20	500	33	0.00	43.18	36.54	±0.00
	cm82a_5	27	3	213	50 × 30	1500	65	0.00	35 × 30	1050	51	0.00	30.00	21.54	±0.00
	2bitAdderMaj	53	2	546	105 × 56	5880	124	0.00	66 × 55	3630	89	0.00	38.27	28.23	±0.00
	parity	121	1	819	209 × 162	33 858	291	0.00	170 × 160	27 200	269	0.00	19.66	7.56	±0.00
	xor5Maj	108	1	1214	400 × 188	75 200	495	0.00	168 × 159	26 712	273	0.00	64.48	44.85	±0.00
IWLS93 [27]	x4	1749	71	20 890	2462 × 1887	4 645 794	2611	0.45	1766 × 1765	3 116 990	2039	0.35	32.91	21.91	22.19
	duke2	2109	29	20 941	2706 × 2131	5 766 486	2874	0.47	2121 × 2119	4 494 399	2394	0.41	22.06	16.70	11.49
	rd84	3241	4	24 903	4033 × 3243	13 079 019	5324	0.97	3248 × 3241	10 526 768	4537	1.01	19.51	14.78	−3.39
	C880	4737	26	44 240	12641 × 4766	60 247 006	13 269	2.34	4774 × 4764	22 743 336	6696	2.58	62.25	49.54	−10.40
	t481	4208	1	44 243	8145 × 4213	34 314 885	10 251	1.82	4214 × 4208	17 732 512	6318	1.90	48.32	38.37	−4.49
	table5	6527	15	62 678	9426 × 6573	61 957 098	10 241	4.66	6546 × 6545	42 843 570	7775	5.17	30.85	24.08	−11.03
	vda	5234	39	66 919	6007 × 5308	31 885 156	6259	3.92	5255 × 5253	27 604 515	5513	3.97	13.43	11.92	−1.27
	table3	6988	14	68 864	10289 × 6993	71 950 977	10 966	5.32	6995 × 6993	48 916 035	8074	5.91	32.01	26.37	−11.21
	apex3	8071	50	91 920	11122 × 8116	90 266 152	12 158	8.53	8094 × 8094	65 512 836	9828	10.03	27.42	19.16	−17.60
	cordic	15148	2	149 180	28305 × 15156	428 990 580	35 105	34.34	15157 × 15153	229 674 021	22 542	36.87	46.46	35.79	−7.37
Average													**21.09**	**15.24**	**-1.00**

For each benchmark, I and O represent the number of primary inputs (PIs) and primary outputs (POs) of the logic network, respectively. The total number of nodes, including PIs, is denoted as $|N|$. For both the SotA and the proposed algorithm, the layout dimensions—width (W) and height (H)—as well as the area A, delay D and the runtime are reported. Additionally, the relative area difference $\Delta A[\%]$ and runtime difference $\Delta t[\%]$ are provided. Here, positive values correspond to improvements over the SotA, while negative values indicate inferior results.

IWLS benchmark *x4* while for the largest benchmark *cordic*, the increase is only $7.37\,\%$. This indicates that runtime depends strongly on the structure of the network. On average, the runtime changed by only $-1.00\,\%$, which is an excellent trade-off considering the average $21.09\,\%$ area improvement achieved across all benchmarks.

VI. Conclusion

In this work, we presented a placement and routing algorithm for planar FCN layouts that optimizes area and delay. By utilizing localized conflict and excess wiring management, our approach significantly reduces structural overhead while maintaining full planarity. Experimental results demonstrate average reductions of $21.09\,\%$ in area and $15.24\,\%$ in delay compared to the state-of-the-art. These improvements demonstrate that high layout quality and computational scalability are not mutually exclusive in planar FCN design. Ultimately, our method provides a robust and scalable solution for the physical realization of reliable, high-performance atomic-scale computing systems.

References

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

[2] L. Livadaru, P. Xue, Z. Shaterzadeh-Yazdi *et al.*, "Dangling-bond Charge Qubit on a Silicon Surface," *New J. Phys.*, vol. 12, no. 8, p. 083018, 2010.

[3] R. Achal, M. Rashidi, J. Croshaw *et al.*, "Lithography for Robust and Editable Atomic-Scale Silicon Devices and Memories," *Nat. Commun.*, vol. 9, no. 1, p. 2778, 2018.

[4] M. Rashidi, J. Croshaw, and R. Wolkow, "Automated Atomic Scale Fabrication," US Patent 20 220 130 033, 2022.

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

[6] M. Walter, R. Wille, F. Sill Torres *et al.*, "Scalable Design for Field-coupled Nanocomputing Circuits," in *ASP-DAC*. ACM New York, NY, USA, 2019, pp. 197–202.

[7] S. Hofmann, M. Walter, and R. Wille, "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.

[8] A. Costamagna, A. T. Calvino, A. Mishchenko *et al.*, "Area-oriented resubstitution for networks of look-up tables," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

[9] A. T. Calvino, G. De Micheli, A. Mishchenko *et al.*, "Enhancing delay-driven lut mapping with boolean decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[10] D. A. Reis, C. A. T. Campos, T. R. B. S. Soares *et al.*, "A Methodology for Standard Cell Design for QCA," in *ISCAS*, 2016, pp. 2114–2117.

[11] J. Retallick, M. Babcock, M. Aroca-Ouellette *et al.*, "Algorithms for Embedding Quantum-Dot Cellular Automata Networks onto a Quantum Annealing Processor," 2017.

[12] B. Hien, M. Walter, S. Hofmann *et al.*, "A fully planar approach to field-coupled nanocomputing: Scalable placement and routing without wire crossings," in *IEEE-NANO*, 2025.

[13] M. Taucer, "Silicon Dangling Bonds Non-equilibrium Dynamics and Applications," 2015.

[14] S. S. H. Ng, J. Retallick, H. N. Chiu *et al.*, "SiQAD: A Design and Simulation Tool for Atomic Silicon Quantum Dot Circuits," *TNANO*, vol. 19, pp. 137–146, 2020.

[15] 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.

[16] 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.

[17] F. Sill Torres, M. Walter, R. Wille *et al.*, "Synchronization of Clocked Field-Coupled Circuits," in *IEEE-NANO*, 2018.

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

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

[20] 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.

[21] R. A. Wolkow, L. Livadaru, J. Pitters *et al.*, *Silicon Atomic Quantum Dots Enable Beyond-CMOS Electronics*. Springer, 2014, pp. 33–58.

[22] D. S. Marakkalage, M. Walter, S.-Y. Lee *et al.*, "Technology Mapping for Beyond-CMOS Circuitry with Unconventional Cost Functions," in *IEEE-NANO*, 2024, pp. 51–56.

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

[24] A. Chaudhary, D. Z. Chen, X. S. Hu *et al.*, "Fabricatable Interconnect and Molecular QCA Circuits," *TCAD*, vol. 26, no. 11, pp. 1978–1991, 2007.

[25] A. Trindade, R. Ferreira, J. A. M. Nacif *et al.*, "A Placement and Routing Algorithm for Quantum-dot Cellular Automata," in *SBCCI*, 2016.

[26] G. Fontes, P. A. R. L. Silva, J. A. M. Nacif *et al.*, "Placement and Routing by Overlapping and Merging QCA Gates," in *ISCAS*, 2018.

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