Reducing Wire Crossings in Field-Coupled Nanotechnologies

Benjamin Hien[1], Marcel Walter[2], and Robert Wille[3]

***Abstract*— In the realm of circuit design, emerging technologies such as *Field-Coupled Nanotechnologies* (FCN) provide unique opportunities compared to conventional transistor-based logic. However, FCN also introduces a critical concern: the substantial impact of wire crossings on circuit robustness. These crossings are either unrealizable or can severely degrade signal integrity, posing significant obstacles to efficient circuit design. To address this challenge, we propose a novel approach focused on reducing wire crossings in FCN circuits. Our methodology introduces a combination of LUT mapping and decomposition aimed at producing advantageous network structures during logic synthesis to minimize wire crossings. This new optimization metric is prioritized over node count and critical path length to effectively tackle this challenge. Through empirical evaluations, we demonstrate the effectiveness of the proposed approach in reducing a first approximation for wire crossings by 41.69%. This research significantly contributes to advancing wire crossing optimization strategies in emerging circuit technologies, paving the way for more reliable and efficient designs in the post-CMOS logic era.**

I. Introduction & Motivation

As Moore's Law reaches its limits in transistor-based logic, emerging technologies garner scientific interest, promising significant improvements in power consumption compared to traditional CMOS circuits [1]. *Field-coupled Nanotechnologies* (FCN) employ electric and magnetic fields for logic transfer, diverging from conventional electric currents [2]. Recent breakthroughs in FCN fabrication, utilizing *Silicon Dangling Bonds* (SiDBs, [3], [4]), have propelled its momentum [3], [5]–[7]. Other FCN implementations include *Quantum-dot Cellular Automata* (QCA, [8], [9]) and *Nanomagnet Logic* (NML, [10], [11]).

Though these technologies differ in their physical domains, they share similar principles regarding constraints during logic synthesis and physical design. While these similarities persist among FCN technologies, they notably deviate from the constraints typical in conventional CMOS. In CMOS logic synthesis, primary optimization objectives involve minimizing the number of nodes and the critical path early in the process, often using abstract representations like *And-Inverter Graphs* (AIGs) as heuristics for circuit size, power consumption, and delay [12]–[14].

An important characteristic of FCN technologies is the costly nature of wire crossings, unlike CMOS where wires are inexpensive. In FCN, existing wire crossing implementations have been found to compromise signal stability for QCA [15] and SiDBs [16], with some structures yet to be fabricated [17]. In contrast, CMOS allows wire crossings to be routed using multiple metal layers without affecting signal stability. This distinct feature alters the significance of cost functions in logic synthesis for FCN.

However, in the past, conventional logic synthesis overlooked the unique challenges of this novel domain, as wire crossings were only addressed during later physical design stages. Given that the logic network structure greatly influences the final circuit [18], the current design flow significantly increases crossing costs, resulting in suboptimal solutions in the FCN domain. This also means that optimizations related to wire crossings are deferred until these late physical design stages, missing out on the significant impact of logic synthesis.

This work aims to enhance network structures beyond conventional logic synthesis, targeting more efficient FCN circuit designs through crossing minimization. We propose adapting the FCN design flow to incorporate early-stage optimizations using a novel algorithm. Our evaluation demonstrates a significant reduction of a first approximation for wire crossings at the logic synthesis level by 41.69%.

The rest of this work is structured as follows. Section II provides the necessary background on FCN, with a particular focus on wire crossings. Section III presents a literature overview of the current Design Automation flow in FCN and offers the motivation for the proposed methodology. Our approach to reduce wire crossings in logic synthesis is proposed and evaluated in Section IV and Section V, respectively. Finally, Section VI concludes the paper.

II. Field-Coupled Nanotechnologies

Field-coupled Nanotechnologies (FCN) offer promising post-CMOS solutions for increased computing power while addressing environmental concerns [2]. Recent advancements, particularly in manufacturing using *Silicon Dangling Bond* (SiDB) technology [3], [5]–[7], demonstrate the potential for nanoscale circuit implementation without electrical current flow, such as a fully functioning OR gate on less than $30nm^2$ of substrate area. Since SiDB circuits can be derived from *Quantum-dot Cellular Automata* (QCA, [8], [9]) circuits [19], QCA is considered as the representative FCN technology in this paper for its clear visual representation. This choice remains applicable to all FCN technologies examined in this work.

Each QCA cell consists of four quantum dots placed at the corners of a square shape. Binary values 0 and 1 are encoded using the stable states resulting from charges placed in diagonally opposing quantum dots, driven by the Coulomb force (Fig. 1). The electrostatic force from cell polarization can affect neighboring cells, enabling computation and information propagation. When organized in a 5×5 grid, QCA cells form various logic gates, exemplified by gates from the *QCA ONE* library [20] (Fig. 2). Unlike conventional CMOS where wires are cheaper than gates, in QCA, *wires are gates*,

[1]Benjamin Hien is with the Chair for Design Automation, Technical University of Munich, Germany. Email: benjamin.hien@tum.de

[2]Marcel Walter is with the Chair for Design Automation, Technical University of Munich, Germany. Email: marcel.walter@tum.de

[3]Robert Wille is with the Chair for Design Automation, Technical University of Munich, Germany, and Software Competence Center Hagenberg GmbH, Austria. Email: robert.wille@tum.de

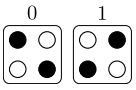


Fig. 1: Elementary QCA cells.

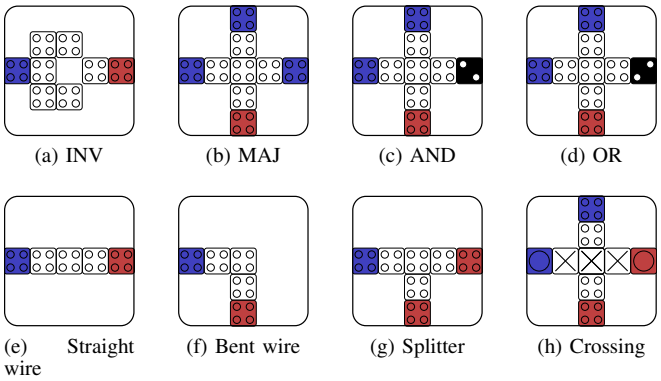


Fig. 2: The *QCA ONE* gate library [20].

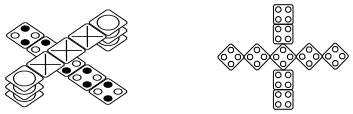


Fig. 3: Different QCA wire crossing implementations.

equating delay and area costs between wires and gates in the final circuit layout.

In Fig. 2a, an inverter is depicted, functioning via diagonal cell interaction. Inputs, as with other gates in the library, are colored blue, while outputs are red. From the 3-input MAJ gate in Fig. 2b, an AND gate (Fig. 2c) and an OR gate (Fig. 2d) can be easily derived by fixing one input to a constant 0 or 1, respectively. Additionally, wire types, also known as buffers, are shown in Figs. 2e, 2f, and 2g, while the symbol for a wire crossing is depicted in Fig. 2h.

Wire crossings, although seemingly innocuous, present significant challenges. One approach to implementing a wire crossing in QCA is through three-dimensional fabrication, as depicted in Fig. 3a. However, despite some proposals [17], it remains unclear how this fabrication method can be achieved at the nanoscale. Another idea is using co-planar wire crossings, as shown in Fig. 3b [15]. Here, rotated and non-rotated cells act as independent wires, preventing interference due to rotation. However, this method is prone to signal stability issues within the circuit, as the coupling between non-rotated cells is weak over the gap. Moreover, cell rotation introduces overhead, requiring pre- and post-crossing rotations. These challenges extend beyond QCA cells, as demonstrated by the susceptibility of SiDB crossings to thermal noise [16].

III. Contemporary FCN Design Flow

In this section, we provide a comprehensive review of the current FCN design flow, offering both a brief overview and an in-depth analysis of its key steps. We then highlight the reasons why wire crossings should be optimized earlier in the flow, thus motivating the novel approach proposed in this work.

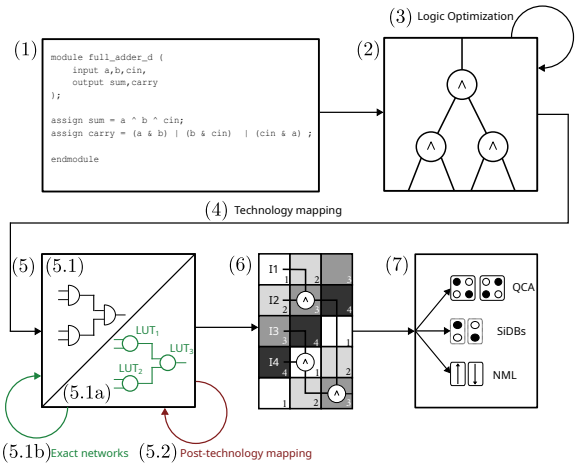


Fig. 4: The design flow for FCN. The steps marked in green are novel to the design stage. The red step already exists but is modified by including our decomposition algorithm.

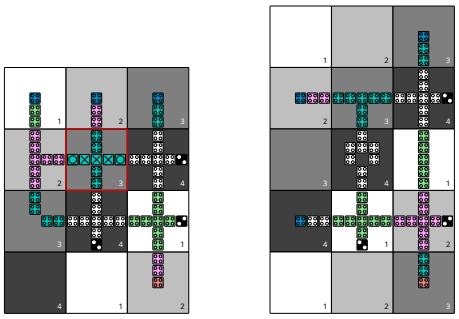
A. Overview

A design flow represents the journey from an abstract description of a circuit's functionality to its physical realization in a specific technology (Fig. 4, excluding green segments). Steps (1)–(5) fall under *Logic Synthesis*, while steps (6) and (7) are termed *Physical Design*. The process begins with an *Register-transfer level* (RTL) description of the circuit (1), translated into a technology-independent graph, such as an AIG (2). Logic optimizations like rewriting [12] can be performed (3), followed by technology mapping (4) to a technology-dependent structure (5). Post-mapping, technology-dependent optimizations can be applied (5.1). The structure undergoes placement and routing using an FCN algorithm (6), then translated into a specific technology (7).

In the following, first the steps (4) to (7) are more closely reviewed, as these steps form the basis onto which this paper is built. Secondly, the shortcomings of the contemporary design flow are discussed.

B. Technology Mapping & Physical Design

During technology mapping (4), the technology-independent representation (2) (e.g., as an AIG) is converted into a technology-dependent netlist (5.1) using a *Standard Cell Library* (SCL). This library offers standard cells with Boolean functions and metrics like area, delay, and power, derived from the target technology (e.g., QCA or NML). Subsequently, the technology mapper assigns standard cells to cover AIG nodes, ensuring each standard cell's Boolean function matches the corresponding AIG nodes it covers. This process yields a technology-dependent netlist (5.1), which can be visualized once more as a graph. This netlist represents standard cell covers instead of AND nodes, maintaining consistency with conventional CMOS electronics. However, rather than having AND nodes, it comprises standard cells, each serving as a cover of a corresponding set of AND nodes implementing the (usually



(a) Non-planar: $3 \times 4 = 12$ tiles.

(b) Planar: $3 \times 5 = 15$ tiles.

Fig. 5: Area comparison of two 2:1 MUX layouts.

more complex) standard cell function. Notably, up to this point, the FCN flow mirrors that of conventional CMOS electronics.

Post-mapping algorithms [21] optimize the technology-dependent graph further, focusing on objectives that are neglected or cannot be targeted during technology mapping. In CMOS, this involves gate sizing or fan-out optimization. In FCN, elements such as buffers and splitters [22] can be inserted.

In the subsequent physical design stage (6), designers choose a placement and routing algorithm, such as [23]–[25], to map the technology-dependent netlist onto an FCN layout. Certain algorithms can alter netlists to enhance routability, as evidenced in [18]. Post-layout optimization algorithms, exemplified in [26], further refine the layout. Finally, in the last step (7) of physical design, abstract standard cell layouts are transformed into precise physical representations in the target technology for fabrication.

C. Early Optimization of Wire Crossings

Following this design flow, wire crossings become apparent only at the physical design stage. And even at this stage, most algorithms do not optimize for wire crossings but for layout area instead [26], or minimize crossings as a secondary objective, as in [18], [24]. Optimizing for wire crossings instead might contradict current design objectives, as shown in the following example.

Example 1: Consider the FCN layouts depicted in Fig. 5, both implementing a simple 2:1 MUX circuit. The layout in Fig. 5a uses a single wire crossing (highlighted with a red frame) and occupies a total area of 12 tiles. Conversely, the layout in Fig. 5b is planar, without any wire crossings, but incurs an area overhead with a total of 15 tiles. While area optimization is typically prioritized, critical issues like manufacturing challenges and signal stability are disregarded in such decisions.

Even from this minimal example, it becomes apparent that it is possible to trade unfavorable wire crossings for less critical layout area. Consequently, optimizing for area might introduce more crossings into the final layout—as is often the case with state-of-the-art algorithms [25]—, which should generally be considered highly disadvantageous.

Furthermore, as mentioned earlier, the significance of wire crossings is typically recognized only during the physical design stage. However, structural manipulations of the netlist before passing it to placement and routing can significantly affect the resulting layout's characteristics—also in terms of the total crossing cost. The effect of logic synthesis on these final layout characteristics has been extensively demonstrated in CMOS throughout the literature [12]–[14]; yet, it is almost completely disregarded in the FCN domain to date. Thus, we aim to address the minimization of wire crossings during the logic synthesis stage rather than postponing it to the physical design phase.

IV. Proposed Methodology

In this section, we initially present our general proposed idea. Then, we propose a new design flow utilizing a precomputed database of small FCN sub-circuits with optimal crossing count, as motivated in Section III. This design flow facilitates our primary contribution, which involves approximating wire crossings at the logic synthesis level and minimizing this proxy using an algorithm during the post-mapping stage. Thus, we introduce the concept of the algorithm, followed by a demonstration of its implementation.

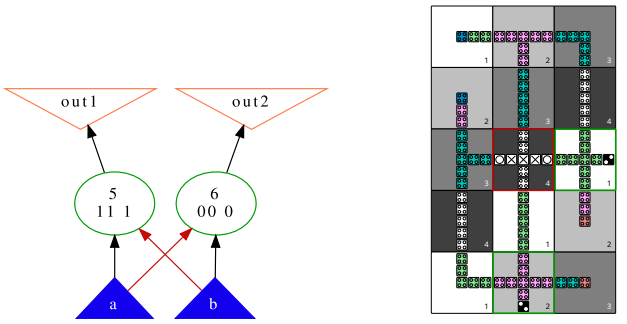
A. General Idea

In order to utilize logic synthesis for crossing cost minimization, we propose the following main ideas. 1) To draw inspiration from general logic synthesis literature, where exact solutions for small sub-AIG structures are employed to replace their corresponding counterparts within the AIG, thereby iteratively optimizing the overall graph for various metrics; an established method called *rewriting* [12]. We propose a similar approach with an application to FCN crossing minimization, where we can utilize a precomputed database of small FCN sub-circuits with favorable characteristics. 2) To approximate wire crossings at the logic synthesis stage and to utilize them as a cost function, guiding rewriting algorithms.

Our primary contribution lies in an optimization algorithm aimed at minimizing this approximative crossing cost value to achieve a favorable network structure conducive to wire crossing minimization, in alignment with the findings of [18]. In the following, the design flow is proposed to facilitate our main contribution.

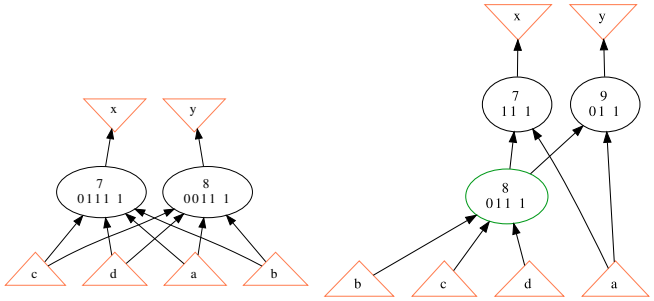
B. Novel Synthesis Flow for FCN

We introduce an innovative design flow to enhance wire crossing minimization by modifying the technology mapping stage (5) shown in Fig. 4, highlighted in green. We suggest transitioning to an alternative technology mapping strategy (5.1a) using an *Field Programmable Gate Array* (FPGA) mapper. Unlike the current technology mapper, which relies on a standard cell library, this mapper utilizes *lookup tables* (LUTs)—generic Boolean functions with a fixed number of input variables, known as its *size*—for mapping. Precomputed sub-layouts(5.1b) with minimized wire crossings are then inserted for the LUTs, enhancing performance and scalability. It has been shown that precomputing such a database is feasible for an input size of 4 [12]. Hence, this size is used as the LUT size for the rest of the paper. Most importantly, the post-mapping stage, marked in red, gains flexibility with the introduction of an extra step. Algorithms can now be



(a) LUT representation. (b) FCN representation

Fig. 6: Edge crossing to wire crossing.



(a) Original sub-network (b) New sub-network

Fig. 7: Decomposition of two four input LUTs.

executed not only after standard cells are apparent in the netlist (5.1), but also directly after LUT mapping (5.1a).

C. Post-Mapping Wire Crossing Approximation

Our goal is to approximate wire crossings at this new stage and minimize them using an algorithm. Our proposed approximation method relies on a simple concept: in a mapped netlist represented as a graph, nodes represent cells to be placed, and edges represent wires to be routed. Identifying instances where two wires cross translates to intersecting edges in the graph. This basic idea, borrowed from layered graph drawing, forms the foundation of our wire crossing approximation method, facilitating integration with subsequent optimization algorithms.

Example 2: Consider the LUT-mapped network shown in Fig. 6a. It consists of two nodes labeled 5 and 6, driving outputs out1 and out2, respectively. Each node relies on primary inputs a and b. The signal flow is illustrated by graph edges, with red edges indicating intersections. In Fig. 6b, the corresponding QCA circuit is displayed, where the LUTs are represented by their respective gates. Inputs (blue cells), gates (outlined in green), and outputs (red cells) are positioned on the layout and connected via wires. In this example, intersections of graph edges translate to wire crossings in the QCA layout, outlined red.

To compute wire crossing costs, we use a technique called layered graph drawing. Initially, all nodes in the graph are assigned levels based on their distance from the inputs. Edges spanning multiple levels are divided by buffer insertion, and each node is given a rank within its level. Crossings between adjacent levels are detected by comparing the coordinates of all edge combinations. After traversing all levels, we obtain the total graph crossings, forming the cost function. The order of nodes within a level affects crossings, leading to various techniques to reduce them by swapping nodes. However, minimizing crossings across the entire graph is NP-hard.

D. Post-Mapping Crossing Reduction

With our approximation method for wire crossings, viewed as edge crossings in a graph, we introduce our algorithm to minimize these crossings. While existing methods like layered graph drawing algorithms tackle this, we aim for a more comprehensive optimization. Our strategy focuses on reducing crossings by restructuring the graph using Boolean decomposition. An example will clarify this approach.

Example 3: In Fig. 7a, the network consists of four inputs and two outputs, efficiently represented by a single 4-input LUT for each output. This setup achieves optimal traditional costs, with area and delay both at 2, but entails 6 edge crossings. However, conventional techniques like layered graph drawing offer no improvements due to the already minimized crossings.

The network in Fig. 7b presents the same Boolean function as the original design but with a different LUT configuration. Instead of two 4-input LUTs, it employs one 3-input LUT and two 2-input LUTs. Despite structural differences, the logic remains consistent, ensuring intended functionality. Notably, this setup reduces edge crossings to 1, aligning more closely with optimization objectives, even though increasing both area and delay by 1.

The improved crossing costs on the right-hand side are due to Boolean decomposition of the network on the left-hand side, introducing shared logic nodes. In Fig. 7b, the shared node 8 is highlighted in green as it contributes to both outputs, consolidating crossings within this node.

Motivated by the efficacy of decomposition and shared nodes, we propose an algorithm to minimize edge crossings in resulting networks. This algorithm targets structures with numerous edge crossings, which can be mitigated by shared logic nodes. This algorithm operates after FPGA mapping, starting with a LUT network with 4-input LUTs (Section IV-B). It identifies pairs of LUTs sharing a common node, requiring a minimum of two shared inputs. Initially, the algorithm searches for such LUT pairs and tests if decomposition can reduce crossings. If decomposition is viable, it applies the change iteratively, continuing to optimize the network's structure and minimize edge crossings.

From the example, we infer that both the area and delay increase. However, this poses no issue for us since, as discussed, minimizing crossings takes precedence over these considerations. Furthermore, the introduction of decomposition may not necessarily increase delay, particularly in larger networks where the new node is not on the critical path. In such cases, delay remains unaffected, illustrating the negligible impact of additional nodes in certain contexts.

Algorithm 1: Shared Nodes

```
Input: 4-input LUT network N
Output: true iff shared nodes could be decomposed
foreach lut_1, lut_2 ∈ N, with lut_1 ≠ lut_2 do
    if lut_1 and lut_2 share fanins then
        d ←DECOMPOSITION(lut_1, lut_2)        // Algorithm 2
        if d ≠ ∅ then
            replace lut_1, lut_2 in N with d
            return true
        end if
    end if
end foreach
return false
```

Algorithm 2: Decomposition

```
Input: Two 4-LUTs lut_1, lut_2
Output: Decomposition of lut_1, lut_2 or ∅ if no decomposition exists
// Check for valid decompositions ranked by the
   number of crossings
// Exemplary decomposition with shared node size 3
foreach f^3 ∈ B^3 → B^1 do
    foreach f_1^2, f_2^2 ∈ B^2 → B^1 do
        if f^3 ∘ f_1^2 ⊨ lut_1 and f^3 ∘ f_2^2 ⊨ lut_2 then
            return (f^3, f_1^2, f_2^2)
        end if
    end foreach
end foreach
// Check other valid decompositions
return ∅
```

E. Crossing Reduction Implementation

The FPGA mapping facilitated by the novel design flow is executed using the ABC synthesis framework [27], employing the command `if -K 4`. The proposed algorithm is integrated into the mapper and activates decomposition when the new `-w` flag is passed to the `if` command. The algorithm consists of two primary tasks.

Initially, Algorithm 1 identifies pairs of nodes sharing inputs through a depth-first search of the network with two loops. The first loop iteratively selects one LUT after another, while the second loop traverses the LUT network until a shared-input LUT is found. The number of crossings—1, 3, and 6 respectively—increases with the number of shared inputs (from 2 to 4), preferring nodes with more shared inputs for decomposition due to greater optimization potential.

Next, the algorithm evaluates the possibility of a valid decomposition by introducing a shared node between the two LUTs. One possible decomposition with a shared node of input size 3 is as depicted in Algorithm 2 and Fig. 7b. Utilizing an exhaustive approach, the algorithm iterates through all 3-input LUTs, combining them with 2-input LUTs to reconstruct the original logic for both outputs. The search for other decompositions works accordingly. This process iterates over all valid decompositions, prioritizing the ones reducing more crossings in the sub-network until a valid decomposition is found, then applying it to the original network.

Despite its simplicity, this implementation is computationally intensive. For each LUT, the network must be traversed again to find another LUT with shared nodes, resulting in significant runtime expenses. Moreover, for nodes with shared inputs, the iteration proceeds through multiple decompositions, necessitating the iteration through numerous LUTs until a valid solution is found. This process becomes particularly burdensome for LUTs that share inputs but cannot be decomposed, as each loop must be fully executed in such cases.

V. Experimental Evaluations

In this section, we make a comparative analysis of the novel design flow utilizing FPGA mapping without and with post-mapping optimization using structural manipulation.

A. Experimental Setup

The algorithm has been implemented in C using the open-source synthesis framework ABC [27] and analyzed with the ISCAS85 [28] and EPFL [29] benchmark suites. We generated LUT netlists twice: once for basic LUT mapping with an input size of 4 and once for LUT mapping followed by LUT decomposition. Subsequently, we evaluated the netlists employing layered graph drawing and simulated annealing to optimize the NP-hard crossing minimization problem in the graphs before and after decomposition. Our analysis tracked network-dependent metrics such as area, delay, and the number of crossings, along with the time required for the proposed algorithm and the evaluation process. All evaluations were conducted on an AMD Ryzen 7 PRO 6850U with 2.7 GHz to 4.7 GHz with 16 CPU cores and 32 GB of RAM, running Ubuntu 23.10.

B. Obtained Results

The results obtained are presented in Table I. The table compares standard LUT mapping with the proposed approach, LUT mapping with Decomposition. Analysis of the edge crossing numbers reveals that the decomposition method consistently achieves fewer edge crossings across all benchmarks with an average of $41.69\,\%$. It is important to note that these results are obtained after the execution of the evaluation script, which involves 1000 sweeps of the network. Each sweep entails a simulated annealing call for every network level. Due to the increasing complexity with larger network sizes, we were only able to obtain results for small and medium-sized benchmarks, ensuring a fair comparison. This can be seen in the increasing evaluation times for bigger benchmarks. For the other metrics area and delay, which relate to the LUT count and critical path in the LUT-network, we can see that for every benchmark both increase with an average of $16.50\,\%$ for area and $43.06\,\%$ for delay. Notably, the percentage-wise increase in delay is very substantial due to the generally small delay values in the original circuits. For the smallest network, *c17*, the decomposition method eliminates crossings entirely, resulting in a crossing improvement of $100\,\%$, a $100\,\%$ worsening of delay and a $50\,\%$ worsening of area. Additionally, benchmarks *router* and *c6288* show significant improvements in wire crossings, with reductions of $83.22\,\%$ and $73.61\,\%$, respectively. However, it's noteworthy that both benchmarks also exhibit the largest increase in delay (excluding *c17*), with a deterioration of $70.00\,\%$ for *router* and $88.00\,\%$ for *c6288*. For area *c6288* also has the highest deterioration of $43.13\,\%$, while *router* has a deterioration of $19.00\,\%$.

VI. Conclusions

To ensure the robustness of fabricated *Field-coupled Nanotechnology* (FCN) circuits, the optimization target on all abstraction levels has to be shifted to the minimization of

TABLE I: Comparative experimental evaluation of FPGA mapping without and with decomposition.

	Benchmark Circuit					Standard LUT mapping					LUT mapping /w Dec (ours)						Difference		
	Name	I	O	$\|G\|$	D	A	D	C	$t_r[s]$	$t_e[s]$	N_{dec}	A	D	C	$t_r[s]$	$t_e[s]$	$A[\%]$	$D[\%]$	$C[\%]$
ISCAS85 [28]	c17	5	2	6	3	2	1	3	< 0.01	1.09	1	3	2	0	0.03	1.25	50.00	100.00	-100.00
	c432	36	7	208	26	98	10	167	< 0.01	94.35	10	108	13	109	0	75.74	5.10	30.00	-34.73
	c499	41	32	398	19	74	4	719	< 0.01	184.34	20	94	6	502	4.45	146.39	16.22	50.00	-30.18
	c880	60	26	325	25	135	8	351	< 0.01	216.27	4	139	10	190	0	168.29	2.22	25.00	-45.87
	c1355	41	32	502	25	74	4	765	< 0.01	188.35	20	94	6	395	4.48	148.17	16.22	50.00	-48.37
	c1908	33	25	341	27	127	9	823	< 0.01	213.63	8	135	11	334	0.04	146.65	6.30	22.22	-59.42
	c2670	157	64	716	20	220	7	2207	< 0.01	1044.88	13	234	8	1763	0.07	928.69	1.82	14.29	-20.12
	c3540	50	22	1024	41	351	12	4385	< 0.01	1663.08	57	409	17	2095	1.5	912.39	15.10	41.67	-52.22
	c5315	178	123	1776	37	494	9	19054	0.01	6157.62	24	523	14	11097	0.18	5048.81	3.04	55.56	-41.76
	c6288	32	32	2337	120	517	25	1853	0.01	1322.7	228	740	47	489	0.56	642	43.13	88.00	-73.61
	c7552	207	108	1469	26	618	8	18958	0.01	7210.27	48	677	11	14438	0.16	4476.01	6.15	37.50	-23.84
EPFL [29]	adder	256	129	1020	255	339	85	254	< 0.01	778.67	43	382	128	281	0.01	907.44	12.68	50.59	10.63
	cavlc	10	11	693	16	297	6	18563	< 0.01	2825.65	45	342	7	11234	2.7	1874.16	15.15	16.67	-39.48
	ctrl	7	26	174	10	55	3	632	< 0.01	123.52	12	67	5	247	0.6	69.44	21.82	66.67	-60.92
	dec	8	256	304	3	288	2	34080	< 0.01	2460.96	16	304	3	30952	0	2235.95	5.56	50.00	-9.18
	i2c	147	142	1342	20	506	6	6415	< 0.01	4231.71	43	550	8	5770	0.78	3569.14	7.51	33.33	-10.05
	int2float	11	7	260	16	88	6	1183	< 0.01	205.16	15	103	7	653	0.66	139.8	17.05	16.67	-44.80
	priority	128	8	978	250	302	62	437	< 0.01	484.8	26	328	62	328	0.04	347.07	8.61	0.00	-24.94
	router	60	30	257	54	100	10	149	< 0.01	130.76	19	119	17	25	0.22	69.02	19.00	70.00	-83.22
Average Difference																	16.50	43.06	-41.69

I, O, and $|G|$ are the number of primary inputs, primary outputs, and gates in the logic network, respectively; A, D and C are the area, delay, and resulting edge crossings of the LUT network, respectively; the runtime for the decomposition algorithm is given as t_r and the evaluation time is given as t_e; the table holds the values for LUT mapping without and with out decomposition strategy and compares the network relevant values in the difference column.

wire crossings. In this paper, we proposed a novel design flow, with the main contribution being an algorithm that optimizes wire crossings at first approximation at the logic synthesis level. The proposed method uses LUT mapping with decomposition and showed a 41.69% improvement over standard LUT mapping in the number of resulting wire crossings. Therefore, this research marks a significant advancement in the design automation of FCN, facilitating the realization and integration of large-scale circuits within this domain.

References

[1] P. M. Solomon, "Device Proposals Beyond Silicon CMOS," *Future Trends in Microelectronics: From Nanophotonics to Sensors and Energy*, pp. 127–140, 2010.

[2] N. G. Anderson and S. Bhanja, *Field-coupled Nanocomputing*. Springer, 2014.

[3] T. Huff, H. Labidi, M. Rashidi, L. Livadaru, T. Dienel, R. Achal, W. Vine, J. Pitters, and R. A. Wolkow, "Binary Atomic Silicon Logic," *Nature Electronics*, vol. 1, no. 12, pp. 636–643, 2018.

[4] R. A. Wolkow, L. Livadaru, J. Pitters, M. Taucer, P. Piva, M. Salomons, M. Cloutier, and B. V. Martins, "Silicon Atomic Quantum Dots Enable Beyond-CMOS Electronics," *Field-Coupled Nanocomputing: Paradigms, Progress, and Perspectives*, pp. 33–58, 2014.

[5] R. Achal, M. Rashidi, J. Croshaw, D. Churchill, M. Taucer, T. Huff, M. Cloutier, J. Pitters, and R. A. Wolkow, "Lithography for Robust and Editable Atomic-scale Silicon Devices and Memories," *Nature communications*, vol. 9, no. 1, p. 2778, 2018.

[6] F. Altincicek, C. Leon, T. Chutora, M. Yuan, R. Achal, J. Croshaw, L. Livadaru, J. Pitters, and R. Wolkow, "Atomically Defined Wires on P-Type Silicon," in *APS March Meeting Abstracts*, vol. 2022, 2022.

[7] N. Pavliček, Z. Majzik, G. Meyer, and L. Gross, "Tip-induced Passivation of Dangling Bonds on Hydrogenated Si (100)-2× 1," *Applied Physics Letters*, vol. 111, no. 5, 2017.

[8] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein, "Quantum Cellular Automata," *Nanotechnology*, vol. 4, no. 1, p. 49, 1993, publisher: IOP Publishing.

[9] C. S. Lent, B. Isaksen, and M. Lieberman, "Molecular Quantum-dot Cellular Automata," *Journal of the American Chemical Society*, vol. 125, no. 4, pp. 1056–1063, 2003, publisher: ACS Publications.

[10] R. Cowburn and M. Welland, "Room Temperature Magnetic Quantum Cellular Automata," *Science*, vol. 287, no. 5457, pp. 1466–1468, 2000.

[11] G. H. Bernstein, A. Imre, V. Metlushko, A. Orlov, L. Zhou, L. Ji, G. Csaba, and W. Porod, "Magnetic QCA Systems," *Microelectronics Journal*, vol. 36, no. 7, pp. 619–624, 2005.

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

[13] A. Mishchenko, R. Brayton, S. Jang, and V. Kravets, "Delay Optimization Using SOP Balancing," in *ICCAD*, 2011.

[14] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang, "Scalable Don't-Care-Based Logic Optimization and Resynthesis," *TRETS*, vol. 4, no. 4, pp. 1–23, 2011.

[15] P. D. Tougaw *et al.*, "Logical devices implemented using Quantum Cellular Automata," *Journal of Applied Physics*, vol. 75, no. 3, pp. 1818–1825, 1994.

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

[17] A. Gin *et al.*, "An Alternative Geometry for Quantum-dot Cellular Automata," *Journal of Applied Physics*, vol. 85, no. 12, pp. 8281–8286, 1999.

[18] M. Walter, B. Hien, and R. Wille, "Versatile Signal Distribution Networks for Scalable Placement and Routing of Field-coupled Nanocomputing Technologies," in *ISVLSI*, 2023.

[19] 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.

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

[21] R. Murgai, "Technology-dependent Logic Optimization," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2004–2020, 2015.

[22] A. T. Calvino and G. De Micheli, "Depth-optimal Buffer and Splitter Insertion and Optimization in AQFP Circuits," in *ASP-DAC*, 2023.

[23] M. Walter, W. Haaswijk, R. Wille, F. S. Torres, and R. Drechsler, "One-pass Synthesis for Field-coupled Nanocomputing Technologies," in *ASP-DAC*, 2021, pp. 574–580.

[24] M. Walter, R. Wille, D. Große, F. S. Torres, and R. Drechsler, "An Exact Method for Design Exploration of Quantum-dot Cellular Automata," in *DATE*, 2018, pp. 503–508.

[25] M. Walter, R. Wille, F. S. Torres, D. Große, and R. Drechsler, "Scalable Design for Field-coupled Nanocomputing Circuits," in *ASP-DAC*, 2019.

[26] S. T. Hofmann, M. Walter, and R. Wille, "Post-Layout Optimization for Field-coupled Nanotechnologies," in *NANOARCH*, 2023.

[27] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-strength Verification Tool," in *CAV*, 2010.

[28] F. Brglez *et al.*, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Targeted Translator in FORTRAN," in *ISCAS*, June 1985.

[29] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL Combinational Benchmark Suite," in *IWLS*, 2015.