Technology Mapping for Beyond-CMOS Circuitry with Unconventional Cost Functions

Dewmini Sudara Marakkalage
Integrated Systems Laboratory
EPFL
Lausanne, Switzerland
dewmini.marakkalage@epfl.ch

Marcel Walter
Chair for Design Automation
Technical University of Munich, Germany
marcel.walter@tum.de

Siang-Yun Lee
Integrated Systems Laboratory
EPFL
Lausanne, Switzerland
siang-yun.lee@epfl.ch

Robert Wille
Chair for Design Automation
Technical University of Munich, Germany
Software Competence Center Hagenberg GmbH, Austria
robert.wille@tum.de

Giovanni De Micheli
Integrated Systems Laboratory
EPFL
Lausanne, Switzerland
giovanni.demicheli@epfl.ch

***Abstract*—With beyond-CMOS circuit technologies emerging from scientific endeavors in an effort to outperform transistor-based logic in feature size, operation speed, and energy dissipation, it has become apparent that besides their differences in physical implementations, their design automation techniques also have to evolve past established norms. While conventional logic synthesis aggressively optimizes the number of nodes in logic networks (as a proxy criterion for area, delay, and power improvements), this trope does not incorporate the additional costs caused by inverters and interconnects in the form of wire segments, signal splitters, and cross-over cells as imposed onto novel circuit implementations such as photonic crystals and field-coupled nanotechnologies. In this paper, we propose a scalable technology mapping algorithm that captures these unconventional costs by utilizing subcircuit databases that are obtained by applying technology-aware exact physical design techniques. Hereby, we overcome the substantial quality loss that previously inevitably occurred when generating beyond-CMOS circuit layouts from conventionally optimized logic networks.**

***Index Terms*—Logic Synthesis, Technology Mapping, Beyond-CMOS, Physical Design Constraints**

I. Introduction

As Moore's Law has lost momentum, alternative circuit technologies that transcend past conventional transistor-based logic are arising from studies into material science and physics. These beyond-CMOS devices promise enhancements over conventional silicon circuits in diverse aspects. While *Photonic Crystals* perform logic operations via wave interference of photons at the speed of light for instance [1], [2], *Silicon Dangling Bonds* (SiDBs) conduct logic-in-memory computations via the repulsion of electric fields with ultra-low power dissipation [3], [4]. Similar *Field-coupled Nanotechnologies* (FCN) [5] are, e. g., *Quantum-dot Cellular Automata* (QCA) [6], [7] and *Nanomagnet Logic* (NML) [8], [9].[1] While each of the above technologies—and other unlisted ones—come with their own set of design constraints, certain similarities continue to reappear that are not captured by conventional optimization criteria.

To this end, an abundance of design flows for emerging circuit technologies rely on conventional logic synthesis that aggressively optimizes the number of nodes of any given logic network, e. g., represented as an *And-Inverter Graph* (AIG), before incorporating technology-specific constraints on the physical design level. If dedicated placement and routing tools are able to legalize such sub-par logic networks, it tends to come at an increased overall layout cost due to the prior negligence of, e. g., inverter costs or interconnects in the form of wire segments, signal splitters, and cross-over cells for planarization, each of which adding to the total area, delay, and power dissipation metrics [10], [11].

[1]In fact, new circuit technologies are constantly being proposed and their physical details are not of importance for the motivation and comprehension of this work.

It has been shown that it is not uncommon for interconnection costs to dominate gate costs by several orders of magnitude in certain beyond-CMOS implementations [10]. Therefore, optimizing logic networks primarily for their number of nodes can be counterproductive as more important cost factors are completely ignored.

In this work, we propose to avoid this overhead by properly incorporating physical design constraints into the logic synthesis step to optimize for unconventional but realistic cost functions in emerging domains. To this end, we present a technology mapping algorithm that utilizes databases of exact subcircuits implemented in specific technologies to overcome the substantial quality loss that inevitably occurs when generating beyond-CMOS circuit layouts from conventionally optimized logic networks. The proposed algorithm is not limited to a certain target technology but is applicable to a wide range of non-conventional circuit implementations.

The remainder of this paper is structured as follows: Section II discusses preliminaries and related work necessary for the comprehension of this manuscript. Section III proposes the novel technology mapping algorithm that constitutes the main contribution of this work. In Section IV, a comparative experimental evaluation is conducted against the state of the art and its results are discussed in detail. Finally, Section V concludes the article.

II. Background

In the following, we review the technology constraints of selected beyond-CMOS implementations in Section II-A, discuss general circuit-level modeling in Section II-B, and give an overview of related work on technology mapping in Section II-C.

A. Beyond-CMOS Technologies with Unconventional Cost Functions

In conventional technology-independent logic synthesis, *AND-Inverter Graphs* (AIGs) are often used as circuit representation to optimize for area and/or delay. For NAND-based CMOS technologies, the AIG size (or depth) measured in the number of AND gates in the network (or on the critical path) serves as a good estimation of the post-mapping area (or delay). However, in some

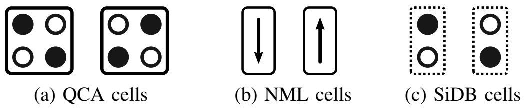
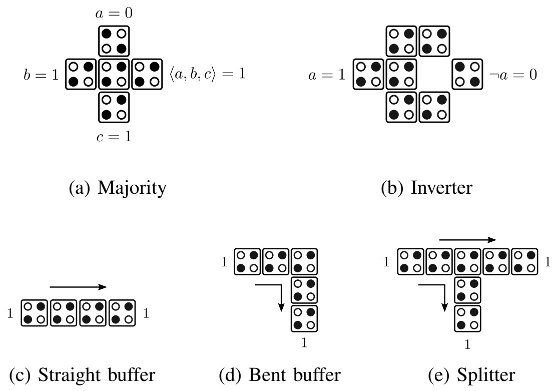

(a) QCA cells (b) NML cells (c) SiDB cells

Fig. 1: Elementary FCN building blocks



(a) Majority (b) Inverter

(c) Straight buffer (d) Bent buffer (e) Splitter

Fig. 2: QCA gates and wire segments

emerging technologies, additional design constraints are imposed, and special cells need to be inserted to fulfill the constraints, which largely increases the discrepancy between technology-independent cost metrics and post-mapping layout cost metrics.

A plethora of beyond-CMOS computational technologies have been proposed over the years. In the following, we discuss some emerging implementations that share similar properties in the aspect of design automation challenges and cost metrics. Due to this paper's brevity, the technologies cannot be discussed in-depth. Instead, a general overview with a focus on cost functions is provided.

Field-coupled Nanocomputing (FCN): FCN is an umbrella term for a variety of nanotechnologies that can be summarized in similar high-level models [5]. In particular, the class of FCN contains *Quantum-dot Cellular Automata* (QCA) [6], [7], *Nanomagnet Logic* (NML) [8], [9], and *Silicon Dangling Bonds* (SiDBs) [3], [4]. Although their possible physical implementations differ greatly, their concepts are nearly identical. In all cases, information is represented by the polarization (or magnetization in the case of NML) of elementary nanometer-scale building blocks called *cells*. When manufactured in close proximity, multiple cells influence each other's polarization (or magnetization) through *Coulomb interaction*. Thus, it is possible to transmit information via the coupling of electric (or magnetic) fields without the flow of current [5]. These effects greatly reduce power dissipation and consequently require less cooling than MOSFETs [11], [12]. Reversible FCN have been shown to be able to operate below the *Landauer limit* [13], [14], a theoretical bound for the energy dissipation of non-reversible computation.

Figure 1 displays elementary cells of the three aforementioned FCN technologies in the two binary states `0` (left) and `1` (right). The topological arrangements of cells lead to the formation of wire segments and gates that conduct Boolean operations, as shown exemplarily in Figure 2 for the QCA implementation [15]. Although NML and SiDB gates and wire segments look slightly different, their concepts do not fundamentally differ.

During physical design, gates and wire segments are arranged in uniform *standard tiles* [16], [17], i. e., higher-level building blocks of precisely specified Boolean functions that shift the abstraction from physical effects to the logic design layer. Placement and routing of standard tiles attempt to create a layout from these building blocks that is functionally equivalent to a given logic network. Here, each tile has the same unit cost values in both the area and the delay. In other words, realizing a single buffer, splitter, or inverter consumes the same area and causes the same delay as placing a gate. Furthermore, FCN circuits are functionally sensitive to timing delays. Signal paths of different lengths (in terms of the number of tiles) desynchronize and cause faulty computation [18].

Photonic Crystals: As a means of realizing completely optical logic circuits with information transmission at the speed of light, research has focused on photonic crystals, that is, optical nanostructures with periodically changing refractive indices [1]. This property allows or prohibits electromagnetic radiation to propagate through a photonic crystal based on its wavelength. Within the crystal lattice, *waveguides* [19] can be fabricated that restrict incoming light to propagate along certain channels; effectively creating wires for photons. At intersections, light originating from two different waveguides interferes to either cancel out or amplify, based on its phase shift. This property has been used to envision optical Boolean gates [2]. However, interacting waveguides of different lengths can cause signals to desynchronize and thereby distort or break gate functionality.

Unconventional design constraints and their associated costs: In this work, we identify and consider the following four unconventional costs that are traditionally not considered in AIG-based logic synthesis but are inherent to the technologies mentioned above.

1) *Path-balancing buffers*: Due to the intrinsic property of these technologies to be sensitive to delay differences in their signal paths, the path-balancing constraint is imposed, requiring that all paths from a primary input leading to the fanins of the same gate must be of the same length. When shortening longer paths is not possible, buffer cells must be inserted into shorter paths to equalize the delay.
2) *Fanout-branching splitters*: Because the design of logic gates in these technologies does not naturally support driving multiple fanout signals, additional splitter cells need to be inserted at the output of multi-fanout gates to fulfill the fanout-branching constraint. Moreover, splitters are also counted in the path lengths of path balancing, thus, the fanout-branching and path-balancing constraints are strongly interwoven.
3) *Planarizing crossings*: The physical design of these technologies often requires special crossing cells to realize wire crossings on a 2-dimensional layout. The placement of logic gates may be altered to minimize such cases, but it is usually not possible to completely planarize the input network. Similar to splitters, crossings also contribute to the path lengths and have to be considered together with path balancing. In some technologies, crossing cells are hard to fabricate or lead to less robust circuits due to their weaker signal strengths. Hence, in these cases, it is of particular interest to minimize the number of crossings, and unavoidable crossing cells are associated with a higher cost.
4) *Non-cost-free inverters*: Unlike traditional logic synthesis, where inverters (complemented edges) in AIGs do not contribute towards their size, inversion is not for free in these technologies. It is not always possible to embed an inversion as a free negated input to a gate. Mitigating the effects of such inverters has been studied in [20]. However, these dedicated inverters not only

increase the circuit size but also need to be considered together with the path-balancing constraint.

Through this consideration, it becomes apparent that cost metrics of conventional logic synthesis algorithms are not suited for the beyond-CMOS due to their negligence of the important aspects enumerated above.

B. Circuit Model

In this work, we propose a technology mapping algorithm that incorporates beyond-CMOS cost metrics. We locate this work at the intersection of logic synthesis (concerned with logic networks) and physical design (concerned with circuit layouts). The output of the proposed algorithm is a mixed logic network consisting of logic gates that are supported by a given technology library and that we represent as *k-input look-up tables* (k-LUTs), and special cells including path-balancing buffers, fanout-branching splitters, and planarizing crossings. Furthermore, to insert crossings in a meaningful way and to ensure the network's planarity, our mapping algorithm entails a coarse-grained placement with relative node positions. That is, gates and cells in the mapped network are sorted into path-balanced *ranks* and are ordered within each rank.

Thereby, our algorithm outputs a partially placed, mapped network that 1) is functionally equivalent to the input network; 2) consists only of gates and cells supported by the given technology library; and 3) satisfies all four constraints described in the previous section (path balancing, fanout branching, planarization, and dedicated inverters). In addition, our mapping algorithm aims at minimizing the size and depth of this network model, which considers both logic gates and special cells. The area cost of each cell type can be parameterized to reflect the target technology as precisely as possible.

By considering such a circuit model, our size and depth evaluation is closer to the actual area and delay of the resulting layout after physical design. As design constraints are already satisfied and cells are ranked and ordered, the remaining placement and routing tasks become trivial for some technologies. However, for some other technologies, the resulting network still needs to be placed and routed according to the target layout topology, e. g., for QCA circuit layouts with non-linear clocking schemes [21], [22].

C. Conventional Technology Mapping

In typical logic synthesis flows, technology-aware optimizations are performed in the technology mapping stage, which happens after heavily optimizing the technology-independent network (e. g., AIG) with methods such as rewriting (cf. e. g., [23], [24]). Technology mapping is the process of transforming a technology-independent logic representation into a technology-dependent one, where mapped circuits are obtained by consecutively substituting small sections with standard cells that represent the elements of the target technology. This methodology has been well studied and numerous mapping algorithms have been proposed over the years [25], [26]. However, most technology mapping approaches are specific to CMOS-based synthesis flows and produce sub-par results when synthesizing for emerging technologies.

III. Proposed Methodology

In this section, we describe the proposed novel technology mapping approach for beyond-CMOS technologies. We start with an introduction of the general idea in Section III-A and go into more detail of the two individual steps, which are one-time optimal subcircuit generation in Section III-B and rewriting in Section III-C.

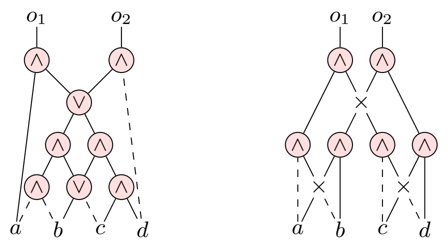


Fig. 3: Two realizations of a network computing $o_1 = a \wedge b \wedge c \wedge d$ and $o_2 = \neg a \wedge \neg b \wedge \neg c \wedge \neg d$. Inverters are denoted by dashed edges and crossing cells are denoted by a $\times$ symbol.

A. General Idea

As our area of application, we particularly target unconventional cost functions that arise from technology constraints of beyond-CMOS circuits as described in Section II-A for which classical logic synthesis algorithms were not developed. As outlined above, certain costs are hard to model or predict at the logic stage. While it is certainly possible to, e. g., levelize a logic network in order to estimate the overhead created by additional buffers, final wire costs will depend on layout topology, placement, and routing. The proposed algorithm is applicable to a wide range of beyond-CMOS technologies that require, e. g., path-balancing, branching, or planarization.

We propose generating a design database prior to technology mapping of optimal subcircuits up to a certain number of inputs that can be used as potential replacements within the context of larger networks. Although this idea has already been successfully applied in logic rewriting [23], it was only ever able to capture abstract technology-independent costs such as the size or depth of subcircuits. Instead, we propose to generate the database with optimal physical design algorithms that are tuned to the desired target technology, thus incorporating all elements of potential final circuit costs.

Consequently, networks rewritten with our proposed technology mapper inherently respect (configurable) inverter, buffer, splitter, and crossing costs.[2] As such, they represent the final circuit layout much more closely and prevent overhead at the physical design stage.

To visualize the need to consider technology-dependent optimizations in the logic synthesis stage, consider a simple circuit with the input variables a, b, c, d that computes $o_1 = a \wedge b \wedge c \wedge d$ and $o_2 = \neg a \wedge \neg b \wedge \neg c \wedge \neg d$. Suppose that we are optimizing for a simplified technology with AND2 and OR2 gates with no additional cost of inversion, where no path-balancing or branching is needed, but that requires planarization using crossings. The gate configuration shown in Fig. 3 (left) uses 8 gates. However, a naive technology-independent logic synthesis algorithm might give a more compact representation with only 6 gates, but due to the planarization constraint, it would require three crossings, thus entailing a total cost of 9 as shown in Fig. 3 (right).

B. Generation of Optimal Subcircuits

Conventional Boolean rewriting algorithms have successfully provided the groundwork for the idea of utilizing databases of optimal subcircuits. Usually, NPN canonization is utilized to significantly

[2]Although we focus on these four cost functions in this paper because they represent important roadblocks to overcome in contemporary emerging technologies, our general approach is applicable to arbitrary cost functions as long as there exists a physical design algorithm for generating the optimal design database.

reduce the number of database entries to generate [23]. Two single-output Boolean functions are considered to belong to the same NPN class if they can be translated into one another by optionally negating (N) the primary inputs, permuting (P) the primary inputs, and optionally negating (N) the primary output. The canonical NPN representative of each class is its lexicographically smallest member. Since in AIGs inverters are considered to be available for free and input permutations can be neglected because no sense of fixed topology is employed, NPN canonization is a strong tool for complexity reduction and optimization.

However, when considering beyond-CMOS cost functions, inverters matter, and input permutations can only be altered via costly crossing networks. Therefore, only considering NPN representatives does not suffice. In fact, the beyond-CMOS costs of members belonging to the same NPN class might substantially differ in the final layouts.

Therefore, we propose exploring a middle ground between the exhaustive enumeration of all 2^{2^n} Boolean functions in n variables and their n-input NPN representatives. To this end, we decided to rely on a class that we call NN that respects input/output permutations but not primary inversions. The number of NN classes is greater by a factor of $n!$ compared to NPN. For example, while the number of NPN classes for 4-input functions is 222, the number of NN classes is $222 \cdot 4! = 5328$, which is still a significant reduction compared to the total number of 4-input functions that is $2^{2^4} = 65536$.

For each canonized (lexicographically smallest) NN representative, we generate an optimal subcircuit layout in the target technology that respects all cost functions imposed by it.[3] To this end, we modified an open-source physical design algorithm [27], [28] for the obtainment of optimal FCN circuit layouts to respect fixed primary input permutations. Since that algorithm is based on SMT solving, we can enforce any input permutation $\pi : x_1 \succ x_2 \succ \cdots \succ x_n$ by adding an additional constraint. Let p_{xt} be the Boolean variable that, when set to `true`, represents that the logic network node x is placed on the layout tile t. One can then enforce that if a primary input is placed on some tile, no other primary input that follows in the permutation order may be placed on any tile that appears at a prior location in the layout. The constraint is defined via:

$$\bigwedge_{\substack{t \in T \\ x \in \pi}} p_{xt} \implies \bigwedge_{\substack{t' \succ t \\ x \succ x'}} \neg p_{x't'}$$

Due to the brevity of this article, the inclined reader is referred to [27] for an in-depth explanation of preexisting constraints for valid node placement, wire routing, crossing insertion, path balancing, etc. Finally, incremental SMT solver calls that iteratively increase the available layout area for the physical design process ensure optimality of the eventually found result. Symmetry breaking allows for effective search space pruning, and highly specialized cardinality constraint engines in the utilized *Z3* solver [29] enable critical runtime reductions that keep the approach scalable up to ≈ 100 layout tiles, which is more than enough to realize all 4-input NN representatives.

C. Rewriting

This step decomposes the input network into small logic blocks and then substitutes them using the appropriate optimum structures described in the previous section. The high-level pseudocode of the algorithm is given in Algorithm 1.

[3]For the sake of this work's brevity, we focus on one technology as a running example. We picked FCN, because their latest fabrication breakthroughs make them a promising competitor in the beyond-CMOS domain [3].

Algorithm 1: Technology mapping with path-balancing, branching, planarization, and inverter constraints

```
   Input: Input network $N$ and database DB
   Output: A technology-mapped version of $N$
1  $N_{\text{lut}} \leftarrow N$ mapped to a 4-LUT network with ABC
2  Assign levels to nodes in $N_{\text{lut}}$ and fix the ordering of nodes
3  $N_{\text{buf}} \leftarrow$ buffer inserted version of $N_{\text{lut}}$
4  $N_{\text{xing}} \leftarrow$ crossing minimized version of $N_{\text{buf}}$
5  $L \leftarrow$ number of logic levels in $N_{\text{xing}}$
6  foreach level $\ell \in \{1, \dots, L\}$ do
7      $DesiredOrder \leftarrow [\,]$
8      foreach node $n$ of level $\ell$ do
9          Reorder fanins of $n$ to avoid self-crossings
10         Update node function of $n$
11         Append reordered fanins to $DesiredOrder$
12     Construct buffer/splitter/crossing layers in $N_{\text{xing}}$ to achieve
         the permutation dictated by $DesiredOrder$ at the outputs of
         level $\ell - 1$
13     foreach node $n$ of level $\ell$ do
14         $(S, InvConfig) \leftarrow$ find best network structure and I/O
             inversion configuration for $n$ from DB
15         Replace $n$ in $N_{\text{xing}}$ with $S$ after applying $InvConfig$
16     Add buffers to $N_{\text{xing}}$ to balance the outputs at level $\ell + 1$
17 return $N_{\text{xing}}$
```

Decomposing into small logic blocks: Typical technology-mapping algorithms consider small cuts rooted at different nodes in the network and replace them with optimized versions. When replacing a particular cut, conventional algorithms usually give only a minor importance to other fanouts of the cut leaves. However, in order to consider branching and planarization constraints of emerging technologies, when replacing a cut, it is important to know the relative positions of the other fanouts of the cut leaves with respect to the part that is being replaced to preserve already instantiated (partial-)planarization. To this end, we 1) fix the decomposition of the input network into small logic blocks by mapping it to 4-LUTs using the `if -K 4` command of the logic synthesis tool ABC [30] (Line 1), and 2) fix the relative positions of those logic blocks by assigning levels (i. e., ranks) to LUTs and imposing an ordering of the LUTs in each level (Line 2).

Initial path-balancing and crossing optimization: Before substituting LUTs with optimum substructure, the algorithm must know the locations through which the lower-level signals will be propagated to higher levels. For example, suppose there is a LUT node a in level 1 which is a fanin of a LUT node d in level 3, and level 2 has two LUTs b and c in that order. Then, the algorithm needs to know whether the path connecting a to d should go through the space left of b, between b and c, or right of c, in order to properly planarize the output logic network. In the initial path-balancing, we thus insert buffers to denote such path propagation locations and ensure all input-to-output paths are of the same length (Line 3). When substituting LUTs, these buffers will be extended to buffer chains to meet the path-balancing constraint for the output circuit.

To minimize crossings, we insert these initial buffers in a locally optimal way, subject to the fixed ordering of LUTs. Namely, for a LUT a in level ℓ, if a buffer needs to be inserted in level $\ell + 1$, it will be placed such that the number of level-ℓ-to-level-$\ell + 1$ connections that cross the path from the LUT to the buffer is minimized. After placing the buffers, we perform another round of crossing optimization for each level (Line 4): namely, for each adjacent pair of nodes in each level, we swap them if it leads to a smaller number of crossings. We repeat this step until no more swaps are possible.

Substituting LUTs from the database entries: In the final step, the path-balanced network is reconstructed in a level-by-level fashion (Lines 6-16). Reconstructing level $\ell + 1$ consists of three main steps: 1) for each LUT in $\ell + 1$, their fanins are reordered to avoid self-crossings (i. e., crossings between pairs of their own fanins), and the node functions are rewritten accordingly (Lines 9-10); 2) zero or more layers, each consisting of buffers/splitters/crossings, are inserted between level ℓ and $\ell + 1$ in order to obtain the fanins of level $\ell + 1$ in the correct order (Line 12); and 3) the LUTs in level $\ell + 1$ are replaced with the respective optimal structures from the database (Lines 14-15). Note that the database includes entries for all 4-input NN classes, but does not include all input/output inverted versions as that would entail pre-computing the entire domain of 4-input functions. Hence, the algorithm considers all possible input/output inversions for LUT node functions and finds a match in the database. Then, the LUT is replaced with the found entry, after applying appropriate input/output inversions.

Run-time and space complexity: The run-time and space complexity of our algorithm is dominated by the planarization step where crossing are inserted between two consecutive levels in Line 12. Namely, if the number of gates in layer ℓ is n_ℓ and $m_\ell = \max(n_\ell, n_{\ell-1})$, then the worst-case for this step would need $O(m_\ell)$-many new layers each consisting of $O(m_\ell)$ crossings/buffers (consider the case where outputs of level $\ell - 1$ are connected to the inputs of level ℓ in the completely opposite order). Thus the total run-time and space needed for this step would be $O\left(\sum_{\ell=1}^{L} m_\ell^2\right)$, which can be $O(n^2)$ in the worst-case where n is the size of the network. As the database of optimum substructures is computed for constant-sized functions, each entry in the database has constant size, and hence the algorithm step that replaces the LUTs with optimum structures would increase the running time by at most a constant factor.

The culmination of techniques outlined here is evaluated experimentally in the following section.

IV. Experimental Evaluation

This section constitutes a quantitative evaluation of the proposed technology mapping algorithm. We demonstrate its applicability and compare it against a state-of-the-art technique. First, we describe the experimental setup in Section IV-A before presenting and discussing the results in Section IV-B.

A. Setup

The proposed technology mapping approach was implemented in C++ on top of the open-source tools *mockturtle* [34] and *fiction* [28] and evaluated using the *ISCAS85* benchmarks [31] and *EPFL Benchmark Suite* [32]. We generated a database of optimal FCN layouts—reyling on state-of-the-art technology constraints [35]—implementing all 5328 canonical NN representatives. Each entry in our database is encoded as a Verilog module, and the complete database takes 12MB of space in the uncompressed format. We then applied the proposed technology mapping algorithm to all circuits available in the aforementioned benchmark suites. We measured their resulting gate-level costs with respect to number of buffers (including splitters), number of crossings, and critical path (CP) length, and compared them to the—to the best of the authors' knowledge—best available large-scale FCN physical design algorithm that can handle layouts with more than 100 million tiles [33]. The code of that state-of-the-art algorithm is publicly available [28], which enabled us to run all experiments with the exact same set of configurations. All evaluations were run on a MacBook Pro M1 with 10 CPU cores, 16 GPU cores, and 32 GB of RAM. The compiler was clang++ 14.0.

B. Results

We applied the proposed technology mapping to all benchmarks in AIGER format without conducting any pre-processing. That is, we did not alter the benchmark networks with other logic optimization algorithms prior to technology mapping. In the same way, we also applied the state-of-the-art FCN algorithm [33] to the unaltered benchmarks to enable a fair comparison.

The obtained results are shown in Table I. It lists the initial properties of the benchmarks in the part labeled *Benchmark Circuit*; the columns with the caption *State of the Art* indicate the statistics of the FCN layouts generated by [33]; and the last part titled *Proposed Approach* provides the data obtained from our technology mapping algorithm when applied to the FCN domain. For both algorithms, we list the required number of total nodes, the number of buffers (including splitters), the number of crossings, and the critical path (CP) length. Additionally, we provide relative improvements of buffer, crossing, and CP costs in the last three columns. The final row states a weighted average for these three columns, which constitutes the total reductions in costs across all benchmarks (excluding the ones for which the state of the art could not generate a solution).

The proposed approach consistently achieves over $50\,\%$ improvement in the critical path length for all benchmarks except for three. A similar level of improvement is also evident in the number of crossings and in the number of buffers for most of the benchmarks within a similar runtime. The average improvement for the number of buffers, number of crossings, and critical path length is $84.5\,\%$, $74.5\,\%$, and $65.2\,\%$, respectively, which establishes a major advantage over the state of the art. Moreover, our approach demonstrates to be more scalable as it yields results for the seven EPFL benchmarks on which the state of the art ran out of memory.

On the downside, a degradation of the critical path length for benchmark 'adder', and a similar order of magnitude degradation of the number of crossings for benchmark 'dec' can be noticed, which appear to be outliers. However, it is to be noted that the benchmark 'adder' exhibits an improvement in the number of crossings, and the benchmark 'dec' shows an improvement in the critical path length.

Generally, an increase in the number of crossings results in an increase in the critical path length, but, as the results for the above two benchmarks suggest, this is not always the case. The critical path length is more related to the maximum number of crossings a single wire has than to the total number of crossings. That is, if there is a single wire that crosses m other wires and no other pairs of wires cross, the planarization needs at least m crossing layers and thus increases the critical path delay by m levels. On the other hand, even if there are m crossings in total between two layers, but each wire only crosses a handful of other wires, that structure can be planarized with much fewer crossing layers, so the critical path length will be small. Thus, to minimize the critical path length, a better objective might be to minimize the maximum number of crossings for any wire, rather than minimizing the total number of crossings.

Additionally, when our crossing optimization and planarization steps are applied directly to the 'adder' AIG without any LUT-mapping, it yields a much better depth. This implies that LUT mapping results in an increased amount of crossings among the resulting LUT nodes, which, in turn, increases the number of logic levels

TABLE I: Results of the proposed technology mapping approach on the ISCAS [31] and EPFL [32] benchmark suites

	Benchmark Circuit					State of the Art [33]				Proposed Approach							
	Name	PI	PO	Gates	Depth	Total Nodes	Buffers	Crossings	CP	Total Nodes	Buffers	Crossings	CP	Runtime in sec.	Buffers impr. %	Crossings impr. %	CP impr. %
	c17	5	2	6	3	99	69	16	26	63	31	6	13	0.04	55.1	62.5	50.0
	c432	36	7	208	26	35 776	31 131	4201	701	14 910	12 170	1973	299	0.06	60.9	53.0	57.3
	c499	41	32	398	19	94 621	88 261	5456	1402	11 842	8754	1948	227	0.06	90.1	64.3	83.8
	c880	60	26	325	25	70 108	61 040	8438	1062	28 920	23 055	4593	457	0.07	62.2	45.6	57.0
	c1355	41	32	502	25	117 056	109 738	6198	1722	11 565	8521	1936	219	0.06	92.2	68.8	87.3
[31]	c1908	33	25	341	27	77 950	71 177	5995	1201	17 201	14 026	2187	364	0.06	80.3	63.5	69.7
	c2670	157	64	716	20	323 824	281 067	41 267	2464	88 295	72 777	13 613	774	0.11	74.1	67.0	68.6
	c3540	50	22	1024	41	587 468	531 807	53 498	3280	97 627	65 925	28 312	977	0.11	87.6	47.1	70.2
	c5315	178	123	1776	37	1 864 282	1 710 369	150 222	5869	301 880	246 660	50 522	1504	0.27	85.6	66.4	74.4
	c6288	32	32	2337	120	988 542	939 626	42 447	8901	97 313	73 052	15 542	1280	0.15	92.2	63.4	85.6
	c7552	207	108	1469	26	1 481 318	1 351 266	126 753	5169	411 383	333 298	73 019	1797	0.36	75.3	42.4	65.2
	adder	256	129	1020	255	794 313	708 447	83 316	4083	2 181 853	2 146 359	31 625	9350	1.65	−203.0	62.0	−129.0
	arbiter	256	129	11 839	87	61 392 432	56 342 578	5 025 982	36 160	7 432 960	6 661 959	731 089	11 646	6.10	88.2	85.5	67.8
	bar	135	128	3336	12	4 050 823	3 680 537	363 230	10 782	1 023 446	448 804	562 234	2369	0.63	87.8	−54.8	78.0
	cavlc	10	11	693	16	286 509	259 800	25 100	2333	86 999	54 839	29 403	760	0.09	78.9	−17.1	67.4
	ctrl	7	26	174	10	28 188	25 097	2667	654	4480	2689	1273	128	0.05	89.3	52.3	80.4
	dec	8	256	304	3	161 857	154 666	6871	1143	25 321	3859	19 894	215	0.06	97.5	−189.5	81.2
	i2c	147	142	1342	20	1 129 553	1 030 768	95 968	4568	294 762	238 114	52 198	1163	0.30	76.9	45.6	74.5
	int2float	11	7	260	16	48 219	42 793	4875	833	12 161	8175	3124	293	0.06	80.9	35.9	64.8
	max	512	130	2865	287	5 378 865	4 720 729	651 642	10 130	6 182 860	5 879 043	294 321	11 589	8.53	−24.5	54.8	−14.4
[32]	priority	128	8	978	250	668 097	607 825	57 916	3477	290 810	273 901	13 407	2487	0.25	54.9	76.9	28.5
	router	60	30	257	54	54 074	45 627	7955	814	20 033	18 439	648	348	0.06	59.6	91.9	57.2
	sin	24	25	5416	225	8 237 614	7 711 879	514 234	16 990	1 145 537	934 129	192 740	6340	0.64	87.9	62.5	62.7
	voter	1001	1	13 758	70	53 955 839	50 500 625	3 421 110	48 864	3 375 273	2 734 141	601 139	5442	7.60	94.6	82.4	88.9
	div	128	128	57 247	4372		*out of memory*			104 918 980	101 310 752	3 409 196	249 354	56.49	—	—	—
	hyp	256	128	214 335	24 801		*out of memory*			1 128 917 685	1 029 593 313	98 519 782	1 226 868	1172.27	—	—	—
	log2	32	32	32 060	444		*out of memory*			24 816 049	20 493 947	4 215 965	47 005	10.22	—	—	—
	mem_ctrl	1204	1231	46 836	114		*out of memory*			371 071 747	341 555 500	29 369 279	108 019	577.26	—	—	—
	multiplier	128	128	27 062	274		*out of memory*			33 370 845	28 610 811	4 668 200	28 842	16.83	—	—	—
	sqrt	128	64	24 618	5058		*out of memory*			41 768 215	41 226 185	470 491	169 800	26.22	—	—	—
	square	64	128	18 484	250		*out of memory*			17 588 918	14 252 309	3 266 047	14 747	7.36	—	—	—
Weighted average															**84.5**	**74.5**	**65.2**

due to crossings that occur in series. In other words, LUT mapping on 'adder' seems to over-optimize for LUT depth, inadvertently making it harder to planarize, because the LUT mapping stage of our current approach is unaware of the technology constraints. Thus, it seems promising to conduct further research on technology-aware decomposition techniques, which will help in mitigating such type of outlier situations.

V. Conclusion

Many technological implementations in the beyond-CMOS domain come with unconventional cost functions that are not respected by classical logic synthesis and, hence, cause significant overhead in the physical design stage. In this paper, we proposed an algorithm for technology mapping of beyond-CMOS circuitry that respects these unconventional cost functions via the application of a physical design database of optimal circuit layouts that is employed for logic rewriting, thus capturing cost factors that would otherwise remain transparent to the logic synthesis. Via an experimental evaluation, it could be shown that the proposed algorithm delivers average improvements of $84.5\,\%$, $74.5\,\%$, and $65.2\,\%$ for the number of buffers, the number of crossings, and the critical path length, respectively, compared to a state-of-the-art physical design algorithm for FCN circuits. Furthermore, results could be obtained for the seven largest EPFL benchmark circuits on which the state of the art ran out of memory, proving our approach to be more scalable. Thereby, this work constitutes a major improvement for the design automation of several emerging beyond-CMOS technology classes, which enables the cost-effective realization and integration of large-scale circuits in this domain.

References

[1] J. D. Joannopoulos, P. R. Villeneuve, and S. Fan, "Photonic crystals," *Solid State Communications*, vol. 102, no. 2-3, pp. 165–173, 1997.

[2] Y. Zhang, Y. Zhang, and B. Li, "Optical switches and logic gates based on self-collimated beams in two-dimensional photonic crystals," *Optics Express*, vol. 15, no. 15, pp. 9287–9292, 2007.

[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, pp. 636–643, 2018.

[4] 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.

[5] N. G. Anderson and S. Bhanja, Eds., *Field-Coupled Nanocomputing - Paradigms, Progress, and Perspectives*, ser. Lecture Notes in Computer Science. Springer, 2014, vol. 8280.

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

[7] 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.

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

[9] 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.

[10] F. Sill Torres, P. A. Silva, G. Fontes, M. Walter, J. A. M. Nacif, R. Santos Ferreira, O. P. Vilela Neto, J. F. Chaves, R. Wille, P. Niemann, D. Große, and R. Drechsler, "On the Impact of the Synchronization Constraint and Interconnections in Quantum-dot Cellular Automata," *Microprocessors and Microsystems*, vol. 76, pp. 103–109, 2020.

[11] J. Timler and C. S. Lent, "Power Gain and Dissipation in Quantum-dot Cellular Automata," *Journal of Applied Physics*, vol. 91, no. 2, pp. 823–831, 2002.

[12] E. Blair and C. Lent, "Clock Topologies for Molecular Quantum-Dot Cellular Automata," *Journal of Low Power Electronics and Applications*, vol. 8, no. 3, 2018.

[13] R. W. Keyes and R. Landauer, "Minimal Energy Dissipation in Logic," *IBM Journal of Research and Development*, vol. 14, no. 2, pp. 152–157, 1970.

[14] R. Landauer, "Irreversibility and Heat Generation in the Computing Process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, 1961.

[15] C. S. Lent and P. D. Tougaw, “A device architecture for computing with quantum dots,” *Proceedings of the IEEE*, vol. 85, no. 4, pp. 541–557, April 1997.

[16] J. Huang, M. Momenzadeh, L. Schiano, M. Ottavi, and F. Lombardi, “Tile-based QCA Design Using Majority-like Logic Primitives,” *JETC*, vol. 1, no. 3, pp. 163–185, 2005.

[17] D. A. Reis, C. A. T. Campos, T. R. B. S. Soares, O. P. V. Neto, and F. Sill Torres, “A methodology for standard cell design for QCA,” in *ISCAS*, 2016, pp. 2114–2117.

[18] F. Sill Torres, M. Walter, R. Wille, D. Große, and R. Drechsler, “Synchronization of Clocked Field-Coupled Circuits,” in *IEEE-NANO*, 2018.

[19] S. G. Johnson, P. R. Villeneuve, S. Fan, and J. D. Joannopoulos, “Linear waveguides in photonic-crystal slabs,” *Physical Review B*, vol. 62, no. 12, p. 8212, 2000.

[20] E. Testa, M. Soeken, O. Zografos, L. Amaru, P. Raghavan, R. Lauwereins, P.-E. Gaillardon, and G. De Micheli, “Inversion optimization in majority-inverter graphs,” in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. Ieee, 2016, pp. 15–20.

[21] C. A. T. Campos, A. L. Marciano, O. P. V. Neto, and F. S. Torres, “USE: A Universal, Scalable, and Efficient Clocking Scheme for QCA,” *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 35, no. 3, pp. 513–517, 2015.

[22] M. Goswami, A. Mondal, M. H. Mahalat, B. Sen, and B. K. Sikdar, “An Efficient Clocking Scheme for Quantum-dot Cellular Automata,” *International Journal of Electronics Letters*, vol. 8, no. 1, pp. 83–96, 2020.

[23] A. Mishchenko, S. Chatterjee, and R. K. Brayton, “DAG-aware AIG rewriting: A fresh look at combinational logic synthesis,” in *DAC*, E. Sentovich, Ed., 2006, pp. 532–535.

[24] H. Riener, W. Haaswijk, A. Mishchenko, G. De Micheli, and M. Soeken, “On-the-fly and DAG-aware: Rewriting Boolean networks with exact synthesis,” in *DATE*, J. Teich and F. Fummi, Eds., 2019, pp. 1649–1654.

[25] J. Cong and Y. Ding, “Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs,” *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 13, no. 1, pp. 1–12, 1994.

[26] Y. Kukimoto, R. K. Brayton, and P. Sawkar, “Delay-optimal technology mapping by DAG covering,” in *DAC*, 1998, pp. 348–351.

[27] 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.

[28] M. Walter, R. Wille, F. S. Torres, D. Große, and R. Drechsler, “fiction: An open source framework for the design of field-coupled nanocomputing circuits,” 2019.

[29] L. d. Moura and N. Bjørner, “Z3: An efficient SMT solver,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008, pp. 337–340.

[30] R. Brayton and A. Mishchenko, “ABC: An academic industrial-strength verification tool,” in *CAV*, 2010, pp. 24–40.

[31] F. Brgles and H. Fujiwara, “A neutral netlist of 10 combinational circuits and a target translator in fortran,” in *ISCAS*, 1985.

[32] L. Amarú, P.-E. Gaillardon, and G. De Micheli, “The EPFL combinational benchmark suite,” in *IWLS*, 2015.

[33] M. Walter, R. Wille, F. S. Torres, D. Große, and R. Drechsler, “Scalable design for field-coupled nanocomputing circuits,” in *ASP-DAC*, 2019, pp. 197–202.

[34] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. Tempia Calvino, D. S. Marakkalage, and G. De Micheli, “The EPFL logic synthesis libraries,” 2022. [Online]. Available: http://arxiv.org/abs/1805.05121

[35] 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, pp. 739–744.