# Scalable Design for Field-coupled Nanocomputing Circuits

Marcel Walter[1] Robert Wille[2,3] Frank Sill Torres[1,3] Daniel Große[1,3] Rolf Drechsler[1,3]
[1]Group of Computer Architecture, University of Bremen, Germany
[2]Johannes Kepler University Linz, Austria
[3]Cyber Physical Systems, DFKI GmbH, Bremen, Germany
{m_walter, frasillt, grosse, drechsler}@uni-bremen.de, robert.wille@jku.at

***Abstract*—*Field-coupled Nanocomputing* (FCN) technologies are considered as a solution to overcome physical boundaries of conventional CMOS approaches. But despite ground breaking advances regarding their physical implementation as e. g. *Quantum-dot Cellular Automata* (QCA), *Nanomagnet Logic* (NML), and many more, there is an unsettling lack of methods for large-scale design automation of FCN circuits. In fact, design automation for this class of technologies still is in its infancy – heavily relying either on manual labor or automatic methods which are applicable for rather small functionality only. This work presents a design method which – for the first time – allows for the scalable design of FCN circuits that satisfy dedicated constraints of these technologies. The proposed scheme is capable of handling around 40 000 gates within seconds while the current state-of-the-art takes hours to handle around 20 gates. This is confirmed by experimental results on the layout level for various established benchmarks libraries.**

## I. Introduction

*Field-coupled Nanocomputing* (FCN) [1] is a class of emerging technologies that conducts computations fundamentally differently from conventional systems relying e. g. on CMOS. Here, information is represented in terms of the polarity or magnetization of nanoscale cells and can be propagated to adjacent ones using repelling forces of local fields [2], [3]. This results in devices that allow to represent and process binary information without electrical current flow. Consequently, numerous contributions on their physical realization have been made in the past and several of some them in the last three to four years, e. g. *molecular Quantum-dot Cellular Automata* (mQCA) [4], *atomic Quantum-dot Cellular Automata* (aQCA) [5], [6], or *Nanomagnet Logic* (NML) [7].

Moreover, this way of representing and processing information is doable with highest processing performance and remarkably low energy dissipation – as confirmed by several theoretical and experimental studies (see e. g. [8], [9], [10]). This makes FCN a promising alternative to conventional integrated circuit technologies.

These endeavors put pressure on the *Electronic Design Automation* (EDA) community and, in fact, more and more requires the development of design automation for FCN – in particular with respect to layout. At the same time, we can conclude that the (automatic) design of FCN is still in its infancy. In fact, the majority of FCN circuits available thus far have been derived manually – including e. g. realizations of arithmetic circuits [11], processors [12], or FPGAs [13].

First approaches that automate the design have been proposed. However, these solutions solely focus on logic synthesis [14], [15], define the actual layout problem on a quite abstract layer [16], [17], [18], or apply rather naive heuristics for the layout design which are severely limited with respect to scalability [19], [20], [21].

The major reason for this rather poor state-of-the-art is caused by inherited characteristics employed by all FCN technologies. Regardless of how information is stored in the different FCN implementations, they are all similar to a certain extend and can be represented via the same abstract model which we describe in detail in Section II.

While indeed typical functional building blocks such as AND, OR, NOT, Majority, etc. can easily be realized in terms of an FCN circuit, their composition is highly non-trivial in FCN. More precisely, *FCN design constraints* such as the arrangement of FCN structures and the appropriate consideration of an external clocking constitute serious challenges for FCN design (this is reviewed more thoroughly in Section II as well).

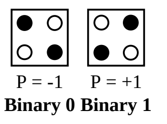
In this work, we investigate the design challenges in detail. To this end, we first conduct a conceptual discussion which unveils that the actual problem of generating a corresponding FCN layout while, at the same time, satisfying dedicated *FCN design constraints* constitutes an instance of an *Orthogonal Graph Drawing* (OGD) problem. This has intensely been considered in theoretical computer science in works such as [22], [23]. We are exploiting these findings for the purposes of FCN design and, eventually, derive an automatic design approach that explicitly addresses the problem outlined above.

Overall, this yields a method which – for the first time – allows for the scalable design of FCN circuits. Experimental results on the layout level confirm the accomplishment. While the state-of-the-art is capable of automatically realizing FCN circuits satisfying FCN design constraints e. g. for simple circuits with around 20 gates only, the proposed scheme easily handles arbitrary functionality with around 40 000 FCN gates, i. e. a 2000X improvement in terms of circuit size is achieved. These results have been obtained for established benchmarks libraries and can automatically be generated within seconds where previous state-of-the-art approaches completely fail.

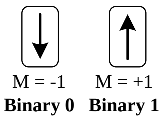
In the remainder of this paper, the made contributions are described as follows. The next section provides a review of FCN circuits and the resulting design challenge. Afterwards, Section III describes the proposed solution to the resulting problems by first investigating its relation to OGD followed by the exploitation of the corresponding findings – eventually yielding the scalable design method. The performance of the resulting scheme is then considered in Section IV which summarizes the conducted experimental evaluations and comparisons to the state-of-the-art. Finally, the paper is concluded in Section V.

## II. Design of FCN

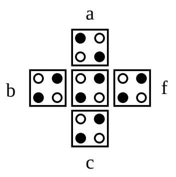
*Field-coupled Nanocomputing* (FCN) [1] circuits are realized in terms of cells that interact via local fields, and thus, enable the realization of logic functions. In case of mQCA and aQCA, in the remainder of this work referred to as QCA, these cells are based on molecules [4] or dangling bonds [5]. In contrast, NML cells utilize nanomagnets [7]. A QCA *cell* is composed of four *quantum dots* which are able to confine an electric charge and are arranged at the corners of a square [24], [25]. Adding into each cell two free and mobile electrons, that are able to tunnel between adjacent dots, yields to a stable state due to interaction (note that tunneling to the outside of the cell is prevented by a potential barrier). More precisely, because of the mutual repulsion, the two electrons tend to locate themselves at opposite corners of the cell – eventually



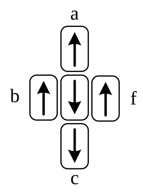
(a) States in QCA



(b) States in NML



(c) QCA Majority



(d) NML Majority

Fig. 1: FCN realizations of basic operations

leading to two possible *cell polarizations* (namely $P = -1$ and $P = +1$ which can be defined as binary 0 and binary 1, respectively).

A NML *cell* is a single domain nanomagnet that can assume only the two stable magnetization states $M = -1$ and $M = +1$, which can represent the binary values 0 and 1 [1].

**Example 1.** *Fig. 1a shows two QCA cells with their four quantum dots (denoted by circles) and two electrons (illustrated by black dots). Due to the electrostatic Coulomb interaction, those are the only stable states which the QCA cell can assume. Usually, the state shown in the left-hand side of Fig. 1a is defined as binary 0, while the state shown in the right-hand side of Fig. 1a is defined as binary 1.*

*In contrast, Fig. 1b depicts two NML cells where the arrow indicates the current magnetization of each cell. Commonly, the magnetization perpendicular to* down *direction is defined as binary 0, while magnetization to the opposite direction is is defined as binary 1.*

When composing several FCN cells next to each other, field interaction also causes the polarization or magnetization of one cell to influence the polarization or magnetization of the others. This allows to realize Boolean functions such as AND, OR, NOT, Majority, etc.

**Example 2.** *Fig. 1c shows the QCA realization of the Majority function with, where e. g. a binary 0 from input $a$ competes with two binary 1s coming from inputs $b$ and $c$. The output follows the majority of the input values, which is a binary 1 in this case. Locking one of the three inputs the to the 0-state turns this cell into an AND cell, while locking one of the inputs to the 1-state results into an OR cell. In a similar fashion, the structure in Fig. 1d depicts the NML implementation of the Majority function where the inputs $a$, $b$ and $c$ compete with each other.*

Following these principles, the design of FCN circuits might seem rather straight-forward at a first glance. In fact, the function to be realized simply needs to be decomposed into Boolean functions, using methods for conventional logic or majority logic synthesis such as [26] and [15], [14]. Afterwards, the resulting description has to be mapped to the corresponding FCN realizations (denoted FCN *gates* in the following) by using a given technology-depended gate library. However, although FCN cells allow for the realization of respective gates, their composition is highly non-trivial. This follows from the requirement that FCN cells have to enter a neutral state before assuming a new polarization or magnetization in order to avoid metastability [27], [7]. For the same reason, it has to be ensured that data is only passed from one FCN structure to the next if the source structure remains in a stable state, while the receiving structure is able to change its polarization or magnetization. To this end, external clocks are employed which regulate the ability of FCN cells to change its polarization or magnetization [28], [29]. Usually, these clocks have 3 (NML) or 4 phases (QCA), depending on the applied FCN technology. For fabrication purposes, cells are usually grouped in *clock zones* – typically having square or rectangular shape – such that all cells within a clock zone are controlled by the same external clock [30], [31], [1]. These clock zones are organized as a grid and each clock zone may contain structures that form a gate.

**Example 3.** *Fig. 2a shows possible groupings of QCA cells into clock zones. Each square shape represents a clock zone, in which QCA cells realizing a gate may be placed (possible cell positions are hinted in each clock zone). Each clock zone is related to one of the four external clocks (denoted by the numbers in the bottom-right corners and a corresponding coloring), which control all QCA cells within the respective clock zone.*

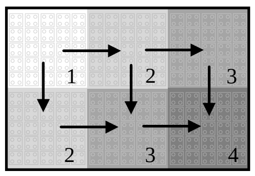
Now, in order to provide a proper data transfer between FCN gates, it has to be ensured that the output value of one gate is only applied to the input of a following gate at exactly the time when the following gate is able to accept a new value, i. e. to change its polarization or magnetization. Therefore, all inputs of an FCN gate must be fed by structures that are located in a preceding clock zone, e. g. a gate of clock zone 2 receives its data from gates of clock zone 1. Consequently, the gates need to be accordingly located onto the grid.[1] This becomes even more complex when the inputs of a gate originate from paths with different lengths. Then, additional *wires* (which can easily be realized by a simple cascade of FCN cells) have to be added so that the respective signals are extended and all arrive in the same clock zone.

**Example 4.** *Consider the multiplexer function with three variables $f(a, b, s) = a \cdot \bar{s} + b \cdot s$ which shall be realized as a QCA circuit. To this end, $f$ is decomposed into gates for which QCA realizations are available (e. g. as proposed in [32]). Next, all gates have to be located on a grid of unassigned clock zones, i. e. clock zones are still not numbered. To this end, we need to consider that all inputs of a gate must come from structures located in a preceding clock zone. Fig. 2b shows a possible solution, in which this is ensured for all gates. As discussed above, this solution requires a wire (located at the bottom center of the circuit shown in Fig. 2b) which extends a signal so that all inputs of the OR gate (located at the bottom-right of the circuit shown in Fig. 2b) arrive from the same clock zone.*

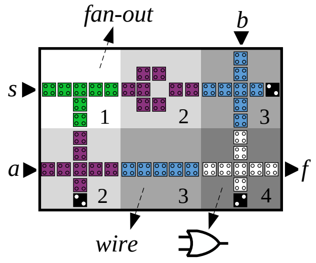
Overall and more precisely, this leads to the following main challenge in the design of FCN circuits as considered in this work: Once a function to be implemented has been decomposed into proper gates (for which QCA and NML realizations are available), how to locate these elements (as well as possibly needed wires) while, at the same time, satisfying the following *FCN design constraints*:

1) *(Unique)*: Each gate must be located in a unique clock zone (except for wires, which are allowed to cross other wires within the same clock zone, but must not cross other gates).
2) *(Adjacent)*: Gates which are connected must be either located in adjacent clock zones or be interlinked by wires.
3) *(Clock zone)*: Each clock zone must be assigned an identifier between 1 and $C$, where $C$ is the maximum

[1]Note that, at the beginning, the number of each clock zone is not fix and can actually be assigned during the design process.

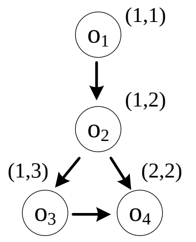


(a) Grid of clock zones with arrows indicating allowed data flows and possible QCA cell positions

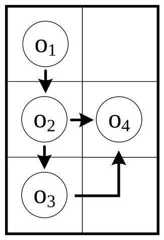


(b) QCA circuit implementing function $f(a, b, s) = a \cdot \bar{s} + b \cdot s$

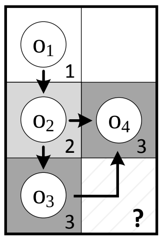
Fig. 2: QCA circuit design



(a) Graph $G$ with 4 vertices and assigned position labels



(b) Orthogonal representation of $G$ with a grid overlay



(c) Assignment of clock zone numeration and resulting conflict

Fig. 3: Naive application of OGD for FCN design leading to a conflict

clock number, i. e. 4 for QCA and 3 for NML, such that paths through consecutive clock zones (i. e. 1 is neighbored by 2, or $C$ is neighbored by 1, etc.) are possible.

Besides that, it is usually desired that the length of all paths leading to any multi-input gate (counted in the number of passed clock zones) differs by not more than 3 (QCA) or 2 (NML) clock zones. If this is the case, the FCN circuit can process new primary inputs with the frequency of the external clocks. Otherwise, primary input signals must be hold constant until all gates received the respective data. Since this only affects the performance (but not the general applicability), this is considered an *optional* FCN design constraint (named *Path length*).[2]

As already discussed in Section I, previous works simplified these constraints or ignored them at all (i. e. generated QCA designs with a corrupted data flow; see e. g. [16], [17]), addressed this challenge in a manual fashion (resulting in a time-consuming and error-prone process which only works for small designs; see e. g. [30], [33], [31], [34]), or relied on automatic solutions which are hardly scalable (i. e. they can handle functions to be synthesized composed of not more than 10 inputs; see e. g. [19], [20], [35]). Furthermore, it is assumed that respecting all FCN design constraints while layouting a circuit, is an $\mathcal{NP}$-complete problem. Because of these reasons, no scalable design of FCN circuits which assures that all required constraints are satisfied exists yet.

## III. Scalable FCN Design

In this work, we propose a solution, which addresses the main challenge reviewed above and allows for an automatic and scalable FCN design. To this end, we first conduct a conceptual discussion in which we unveil that the actual problem of generating an FCN circuit satisfying all *FCN design constraints* constitutes a dedicated instance of the *Orthogonal Graph Drawing* problem. Based on these insights, we propose a general approach which exploits findings from orthogonal graph drawing for the purposes of FCN design. The details of the correspondingly resulting implementations are finally provided in the end of this section.

### *A. Relation to Orthogonal Graph Drawing*

*Orthogonal Graph Drawing* (OGD) is a special case of general graph drawing and has intensively been studied over the years (for an overview see e. g. [22]). Due to its application in the design of information systems (e. g. UML diagram representation) and VLSI layout, the problem has gained theoretical and practical attention. In general, it is defined as follows:

[2]For this work, we decided to respect this constraint – leading to a design method gaining highest throughput.

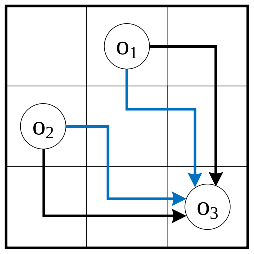
**Definition 1.** *Given a graph $G = (V, E)$, OGD searches for an assignment of unique integer tuples $(x, y)$ (with $1 \leq x < X$ and $1 \leq y < Y$ representing* positions *on a rectangular $X \times Y$-grid with $X, Y \in \mathbb{N}$) to the vertices $v \in V$ so that they can be drawn in an* orthogonal *fashion on the grid. Here, orthogonal means that the edges $e \in E$ connecting the vertices must be drawn as sequences of horizontal and vertical segments arranged in a 90° fashion. Besides that, the edges must not cross any vertex.*

**Example 5.** *Consider Fig. 3a which depicts a graph to be drawn orthogonally. A possible assignment of unique positions to the vertices via OGD is shown as annotation to each vertex in the figure. This results in an orthogonal graph drawing as depicted in Fig. 3b (including an adumbrated grid).*

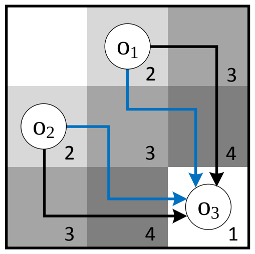
One can notice that OGD possesses several similarities to the FCN design problem reviewed in Section II. In fact, for FCN design, building blocks have to be located on a grid. This is, to some extend, equivalent to placing vertices of a graph onto a 2-dimensional grid – as done by OGD. The only additional issue is the required consideration of the *FCN design constraints* reviewed in Section II. An example shall illustrate the similarities and limitations.

**Example 6.** *Let's assume that the graph in Fig. 3a does not represent an arbitrary graph, but a netlist of a function $f$ composed of four operations $o_1$ to $o_4$. Then, an initial FCN circuit could easily be generated by simply solving the OGD problem as described above, i. e. for each vertex a unique position is determined – resulting in the arrangement as already discussed by means of Fig. 3b. Now, clock zones have to be assigned to the grid so that all the* FCN design constraints *are satisfied. Fig. 3c shows a possible attempt. However, this assignment of clock zone numeration leads to a conflict at position $(2, 3)$. Here, a wire is supposed to receive data from $o_3$ and pass it to $o_4$. This requires a consecutive clock zone to $o_3$ and a preceding clock zone to $o_4$. Since both neighboring clock zones assume clock zone 3, though, constraint 3 (*Clock zone*) is violated.*

Overall, OGD already covers a significant part of the FCN design problem considered here by satisfying constraints 1 (*Unique*) and 2 (*Adjacent*). Since OGD has heavily been investigated (see e. g. [36], [23]), several algorithms are already available which can readily be applied to determine an initial arrangement of the FCN design. The next session describes how these solutions are extended, such that additionally the clock zones are assigned correctly and paths lengths are equal, i. e. such that constraints 3 (*Clock zone*) and the optional one (*Path length*) are satisfied.



(a) Connection of localized vertices using Manhatten distance



(b) Resulting assignment of clock zone numeration

Fig. 4: Generation of valid solutions via Manhattan distance

### *B. Addressing FCN Design Constraints*

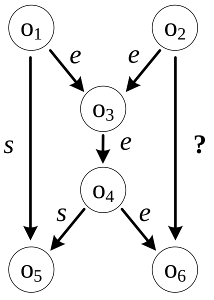
Plenty of work has been conducted in the area of OGD over the years – leading to several algorithms as well as proved bounds (see e. g. [36], [23]).

Due to its efficient runtime behavior and closeness to known optimal area bounds, we utilize the solution proposed by Biedl (i. e. [36]) in the following. Here, the OGD is considered for 3-graphs, i. e. graphs that solely consists of vertices which have at most degree 3. Since it is possible to decompose each function into a graph exclusively composed of 2-AND/2-OR/NOT operations (making it a 3-graph), this does not constitute a limitation to our solution.

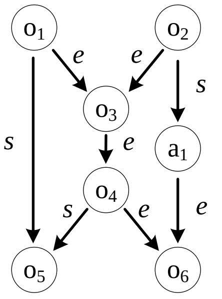
To conduct the arrangement and, at the same time, addressing the *FCN design constraints* (including the optional one), we employ a topological order onto the graph. A linear ordering is called topological if and only if for every directed edge $(u, v) \in E$ from vertex $u$ to vertex $v$, $u$ comes before $v$ in the ordering. This step is required since the existing OGD algorithm is defined on undirected graphs, while the implementation of logic functions requires to respect the data flow of its operations. Hence, we must enforce a particular direction of each edge in the graph and sort the vertices topologically. The resulting order also defines the processing sequence of the vertices, i. e. in which order the operations related to each vertex are located on the grid by the algorithm. This way, it can be guaranteed that all predecessors of the currently considered vertex have been already processed, and hence, the locations of the respective operations can be considered when placing the operation of the current vertex.

By further restricting the operations of new vertices to be located only south or east to already placed ones, we gain several advantages: (1) constraint 1 (*Unique*) is satisfied, i. e. each operation (gate) is uniquely located, (2) connections become easily realizable (supporting to satisfy constraint 2, *Adjacent*), and (3) the data flow and, therefore, the later assignment of clock zone numeration is well-defined (supporting to satisfy constraint 3, *Clock zone*). Besides that, the optional constraint (*Path length*) can trivially be satisfied by exploiting the so called Manhattan distance between an already located operation and its predecessors. This means, when new operations can only be located in south-east direction, all direct paths from the operation's output to any newly located operation have the same Manhattan distance, i. e. pass the same number of clock zones. This is illustrated by the following example.

**Example 7.** *Consider the graph shown in Fig. 4a. Operation $o_3$ in the bottom right corner has dependencies to both other operations $o1, o2$. By restricting the arrangement of $o_3$ to south and/or east of the already placed operations and by applying wires/paths along the Manhattan distance as described above, the assignment of clock zone numeration as sketched in Fig. 4b evolves naturally by the data flow.*



(a) Conflicting direction assignment



(b) Problem solved by subdividing edge $(o_2, o_6)$

Fig. 5: Direction assignment to graph edges

*Additionally, the resulting paths will always be of the same length (as respectively shown in black and blue in Fig. 4b) and, therefore, trivially meet the optional design constraint (*Path length*).*

Motivated by this, we pre-assign directions (*south*/*east*) to the graph edges such that the following two requirements are fulfilled: (R1) all incoming edges to every vertex are supposed to assume the same direction, and that (R2) all outgoing edges of every vertex are supposed to assume different directions. This is necessary in order to prevent conflicting assignments of the clock zone numbering. Unfortunately, enforcing such a scheme is not always possible for each and every graph as highlighted by the following example.

**Example 8.** *Consider Fig. 5a where a graph is given together with a partial direction assignment (*e *denoting* east *and* s *denoting* south*). For this example, it is impossible to assign directions to this graph as described above. Operation $o_3$'s incoming edges were assigned correctly with* east *each as stated by the requirement R1 and $o_1$'s as well as $o_4$'s outgoing edges were also assigned correctly with both* east *and* south *– meeting the requirement R2. The edge $(o_2, o_6)$, however, should then both be labeled* south *(because of the requirement R2 at operation $o_2$) but also* east *(to meet the requirement R1 at $o_6$) – obviously a contradiction.*

However, this problem can be resolved by subdividing the conflict edge and adding an auxiliary vertex in between. More precisely, the proposed solution is to replace the conflict edge by a new vertex and two accordingly connected edges.

**Example 9.** *A solution to the problem illustrated in Example 8 is depicted in Fig. 5b. The edge $(o_2, o_6)$ has been replaced by a new vertex $a_1$ and two additional edges $(o_2, a_1)$ and $(a_1, o_6)$. This enables the determination of a valid result, as* south *can be assigned to edge $(o_2, a_1)$ and* east *to $(a_1, o_6)$. Auxiliary vertices contain no operation and will be handled as normal wires.*

Overall, this leads to a scheme as described in Algorithm 1. We assume that the netlist synthesized in line 1 is a 3-graph. An empty FCN layout is initialized in line 2. The variables $x$ and $y$ are used as the grid's current size which increases dynamically when needed throughout the process. The aforementioned edge assignment and subsequent topological vertex ordering are performed in lines 3 and 4, respectively.

Afterwards, the operations are processed, i. e. located on the grid, consecutively. The actual arrangement depends on the predecessors of the currently considered operation. For operations that lack any predecessor, one new row and one new column are added to the grid and the operation is placed

**Algorithm 1:** Scalable FCN design

```
   Input: Boolean function $f$
   Output: FCN layout $L$
1  Synthesize netlist $N = (O, W)$ from $f$
2  $L \leftarrow$ empty FCN layout of size $(x = 0) \times (y = 0)$
3  Generate direction assignment $d : W \rightarrow \{\text{east}, \text{south}\}$
     and subdivide edges if necessary
4  Compute topological ordering $o_1, \ldots, o_n \in O$
5  for $o \in o_1, \ldots, o_n$ do
6      if $indeg(o) = 0$ then
7          Add one row and one column to $L$
8          Locate $o$ at position $(x, y)$
9      else at most two incoming edges $e_1, e_2$ to $o$
10         if $d(e_1) = d(e_2) = east$ then
11             Add one column to $L$
12             $y_p \leftarrow$ max. y-position of $o$'s predecessors
13             Locate $o$ at position $(x, y_p)$
14         else edges are labelled south
15             Add one row to $L$
16             $x_p \leftarrow$ max. x-position of $o$'s predecessors
17             Locate $o$ at position $(x_p, y)$
18         end
19         Draw wires to connect $o$ with its predecessor(s)
             accordingly
20     end
21 end
22 return $L$
```

at the bottom right corner (lines 6 to 8).[3] Operations cannot have more than two incoming edges $e_1, e_2$. Their arrangement is conducted with respect to edge annotation. Note that we have requested direction assignments to be identical for all incoming edges of any vertex by requirement R1. If those are labeled *east*, one column is added to the grid and the $y$-position for the newly arranged operation is determined by the maximum $y$-position of its predecessors while choosing the freshly inserted column as $x$-value (lines 10 to 13). This way, wire connections can be drawn with at most one bend (if the currently considered operation had two predecessors), while assuring the avoidance of any conflicts. If all incoming edges are labeled *south* instead, one row is added to the grid and the $x$-position of the current operation to be located on the grid is determined by the predecessors maximum $x$-position analogously (lines 14 to 17).

In summary, we have introduced an algorithm for scalable design of FCN circuits. Inspired by findings on the orthogonal graph drawing problem, we devised an algorithm which satisfies the three main FCN design constraints (*Unique*, *Adjacent*, *Clock zone*) when generating an FCN layout. Furthermore, we also assured that the generated designs gain highest throughput by additionally meeting the optional constraint (*Path length*). In the next section, we provide a corresponding experimental evaluation.

## IV. Evaluation

In this section, the performance of the proposed design approach is evaluated and compared to the state-of-the-art. To this end, we implemented the algorithm described in the previous section in C++ for the *fiction* framework [37] and measured its performance in terms of runtime and the resulting size of the FCN layouts when applying it to common benchmarks taken from [38], [39], [20]. Additionally, we compared the obtained results to QCA designs generated by the approach presented in [20] (considered state-of-the-art thus far)[4] as well as a method that implements an FCN layout with minimal number of clock zones (obtained by [35]). All experiments have been conducted on an Intel Xeon E5-2630 v3 machine with 2.40 GHz (up to 3.20 GHz boost) and 64 GB of main memory running Fedora 22. For verification purposes, circuits with up to 30 gates have been exported to the tool QCADesigner [40] and simulated using the coherence vector engine. Exhaustive simulation of more complex circuits was not possible due to tool limitations.

The results are summarized in Table I. The gate count (column *#Gates*) refers to the number of 2-AND, 2-OR, and NOT gates as well as fan-outs used to represent the respective functions as 3-graphs. Column *I / O* provides information about the benchmark's number of primary inputs and outputs. The following columns list, for each considered design technique, the resulting *Dimension* in terms of clock zones, which have the size of $5 \times 5$ cells, and the respectively needed *Runtime* (in CPU seconds). Note that design processes which could not have been completed within a timeout of 10 hours are dashed in Table I. The considered benchmarks are divided into three sets: First, we considered some basic logic functions which have, thus far, been considered by the state-of-the-art. Afterwards, results obtained from larger functions taken from the ISCAS85 library [38] and the EPFL library [39] are listed, which have been considered in order to evaluate the scalability of the proposed approach.

The results clearly confirm that the proposed method satisfies the objectives of this work. First, the proposed approach is capable of *automatically* implementing the desired FCN circuits, while, at the same time, satisfying all *FCN design constraints*. At the same time, it addresses the main drawback of other automatic solutions proposed thus far (such as [20]): the poor scalability. More precisely, while the state-of-the-art is capable to realize rather small functions with up to four primary inputs and not more than two dozens of gates only, the solutions proposed in this work can generate functions with thousands of gates. At the same time, the resulting quality does not decrease. In fact, the sizes of the generated FCN designs are within the same ranges as the state-of-the-art solutions. Overall, the proposed method allows to automatically realize FCN designs for large and complex functionality.

## V. Conclusion

This work focused on one of the essential challenges for the advancement of *Field-coupled Nanocomputing* (FCN) circuits – an automatic design method. The approach presented herein allows – in contrast to the state-of-the-art – for the scalable design of FCN circuits that satisfy the FCN design constraints which have mainly been ignored or led to automatic solutions which are poorly scalable thus far. Experimental results confirmed that the proposed approach is capable of handling functions to be synthesized with around 40 000 gates – without decreasing the quality of the resulting designs. This provides the basis for an automatic FCN design scheme for large and complex functionality. Future work will include optimizing the resulting designs, particularly with respect to wires which, as observed in [41] constitute a severe cost factor in FCN.

## Acknowledgments

The research reported in this paper has partially been supported by the German Research Foundation DFG, as part of Collaborative Research Center (Sonderforschungsbereich) 1320 *EASE – Everyday Activity Science and Engineering*, University of Bremen (http://www.ease-crc.org/). The research was conducted in subproject P04.

[3]Note that adding one row and one column to an empty grid results in adding a single clock zone.

[4]Thanks to the authors for providing us their source code.

TABLE I: Comparison to State-Of-The-Art

| | Benchmark | | S-o-t-a heuristic [20] | | | S-o-t-a exact [35] | | | Proposed approach | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Name** | **#Gates** | **I / O** | **Dimension** | **CP** | **t in s** | **Dimension** | **CP** | **t in s** | **Dimension** | **CP** | **t in s** |
| **2:1 MUX** | 5 | 3 / 1 | $4 \times 5$ | 5 | 9 | $3 \times 3$ | 5 | $< 1$ | $4 \times 2$ | 5 | $< 1$ |
| **XOR** | 6 | 2 / 1 | $4 \times 7$ | 7 | 11 | $3 \times 3$ | 5 | $< 1$ | $5 \times 3$ | 7 | $< 1$ |
| **XNOR** | 8 | 2 / 1 | $6 \times 6$ | 8 | 13 | $4 \times 4$ | 8 | 2 | $6 \times 4$ | 9 | $< 1$ |
| **Half adder** | 10 | 2 / 2 | $7 \times 6$ | 8 | 55 | $5 \times 5$ | 10 | 13 | $7 \times 5$ | 11 | $< 1$ |
| **ParGen** | 14 | 3 / 1 | $9 \times 10$ | 14 | 27 | $7 \times 6$ | 14 | 791 | $10 \times 7$ | 16 | $< 1$ |
| **ParCheck** | 21 | 4 / 1 | $10 \times 14$ | 14 | 3014 | $4 \times 12$ | 16 | 1140 | $15 \times 10$ | 24 | $< 1$ |
| **4:1 MUX** | 18 | 4 / 1 | $11 \times 8$ | 19 | 9612 | $7 \times 7$ | 22 | 5130 | $12 \times 7$ | 18 | $< 1$ |
| **c17** | 15 | 5 / 2 | $10 \times 6$ | 13 | 15 | $4 \times 6$ | 16 | 56 | $12 \times 4$ | 14 | $< 1$ |
| **c432** | 551 | 36 / 7 | — | — | *TO* | — | — | *TO* | $426 \times 161$ | 584 | $< 1$ |
| **c499** | 963 | 41 / 32 | — | — | *TO* | — | — | *TO* | $690 \times 306$ | 995 | $< 1$ |
| **c1355** | 1515 | 41 / 32 | — | — | *TO* | — | — | *TO* | $1243 \times 369$ | 1611 | $< 1$ |
| **c1908a** | 2043 | 33 / 25 | — | — | *TO* | — | — | *TO* | $1540 \times 536$ | 2077 | $< 1$ |
| **c2670a** | 2455 | 155 / 64 | — | — | *TO* | — | — | *TO* | $1756 \times 760$ | 2511 | $< 1$ |
| **c3540a** | 3588 | 50 / 22 | — | — | *TO* | — | — | *TO* | $2523 \times 1111$ | 3639 | 1 |
| **c5315a** | 5478 | 177 / 123 | — | — | *TO* | — | — | *TO* | $3857 \times 1751$ | 5577 | 2 |
| **c6288** | 6928 | 32 / 32 | — | — | *TO* | — | — | *TO* | $5714 \times 1246$ | 6957 | 2 |
| **ctrl** | 498 | 7 / 25 | — | — | *TO* | — | — | *TO* | $356 \times 149$ | 495 | $< 1$ |
| **router** | 658 | 60 / 3 | — | — | *TO* | — | — | *TO* | $488 \times 231$ | 717 | $< 1$ |
| **int2float** | 699 | 11 / 7 | — | — | *TO* | — | — | *TO* | $514 \times 196$ | 708 | $< 1$ |
| **i2c** | 3508 | 133 / 127 | — | — | *TO* | — | — | *TO* | $2515 \times 1123$ | 3632 | 1 |
| **bar** | 8592 | 135 / 128 | — | — | *TO* | — | — | *TO* | $6547 \times 2180$ | 8724 | 6 |
| **sin** | 14314 | 24 / 25 | — | — | *TO* | — | — | *TO* | $10549 \times 3828$ | 14374 | 14 |
| **voter** | 39476 | 1001 / 1 | — | — | *TO* | — | — | *TO* | $30542 \times 9935$ | 40476 | 10 |

| | | | | | |
|---|---|---|---|---|---|
| **#Gates** | Gate count as given by the number of 2-AND/2-OR/NOT gates plus fan-outs | **CP** | Critical path | **I / O** | Number of primary inputs / outputs |
| **Dimension** | Occupied area in clock zones (i.e. $5 \times 5$ QCA cells building blocks) | **t in s** | Time in seconds | *TO* | Timeout of 10 hours reached |

## REFERENCES

[1] N. G. Anderson and S. Bhanja, *Field-coupled Nanocomputing: Paradigms, Progress, and Perspectives*, 1st ed. New York: Springer, 2014.

[2] 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, 1997.

[3] D. Giri, G. Causapruno, and F. Riente, "Parallel and serial computation in nanomagnet logic: An overview," *TVLSI*, pp. 1–11, 2018.

[4] C. S. Lent *et al.*, "Molecular Cellular networks: A non von neumann architecture for molecular electronics," in *ICRC*, Oct 2016, pp. 1–7.

[5] S. Bohloul, Q. Shi, R. A. Wolkow, and H. Guo, "Quantum transport in gated dangling-bond atomic wires," *Nano Letters*, vol. 17, no. 1, pp. 322–327, 2017.

[6] T. R. Huff, H. Labidi *et al.*, "Atomic white-out: Enabling atomic circuitry through mechanically induced bonding of single hydrogen atoms to a silicon surface," *ACS Nano*, vol. 11, no. 9, pp. 8636–8642, 2017.

[7] X. K. Hu, H. Dey, N. Liebing, G. Csaba, A. Orlov, G. H. Bernstein, W. Porod, P. Krzysteczko, S. Sievers, and H. W. Schumacher, "Edge-mode resonance-assisted switching of nanomagnet logic elements," *IEEE Trans. Magn.*, vol. 51, no. 11, pp. 1–4, Nov 2015.

[8] J. Timler and C. S. Lent, "Power gain and dissipation in Quantum-dot Cellular Automata," *J. Appl. Phys.*, vol. 91, no. 2, pp. 823–831, 2002.

[9] J. Pitters, L. Livadaru, M. B. Haider, and R. A. Wolkow, "Tunnel coupled dangling bond structures on hydrogen terminated silicon surfaces," *JCP*, vol. 134, no. 6, 2011.

[10] F. S. Torres, R. Wille, P. Niemann, and R. Drechsler, "An energy-aware model for the logic synthesis of Quantum-dot Cellular Automata," in *TCAD*, 2018.

[11] S. Perri and P. Corsonello, "New methodology for the design of efficient binary addition circuits in QCA," *TNANO*, vol. 11, no. 6, pp. 1192–1200, 2012.

[12] E. Fazzion, O. L. Fonseca, J. A. M. Nacif, O. P. V. Neto, A. O. Fernandes, and D. S. Silva, "A Quantum-dot Cellular Automata processor design," in *SBCCI*, 2014.

[13] M. Kianpour and R. Sabbaghi-Nadooshan, "A novel Quantum-dot Cellular Automata CLB of FPGA," *J. Comput. Electron.*, vol. 13, no. 3, pp. 709–725, 2014.

[14] K. Kong, Y. Shang, and R. Lu, "An optimized majority logic synthesis methodology for Quantum-dot Cellular Automata," *TNANO*, vol. 9, no. 2, pp. 170–183, 2010.

[15] M. G. A. Martins, V. Callegaro, F. S. Marranghello, R. P. Ribas, and A. I. Reis, "Majority-based logic synthesis for nanometric technologies," in *IEEE-NANO*, 2014, pp. 256–261.

[16] M. Bubna, S. Roy, N. Shenoy, and S. Mazumdar, "A layout-aware physical design method for constructing feasible QCA circuits," in *GLSVLSI*, 2008, pp. 243–248.

[17] R. Ravichandran, S. K. Lim, and M. Niemier, "Automatic cell placement for Quantum-dot Cellular Automata," *Integration*, vol. 38, no. 3, pp. 541–548, 2005.

[18] W. J. Chung, B. Smith, and S. K. Lim, "Node duplication and routing algorithms for Quantum-dot Cellular Automata circuits," *IEE Proceedings – Circuits, Devices and Systems*, vol. 153, no. 5, pp. 497–505, 2006.

[20] A. Trindade, R. S. Ferreira, J. A. M. Nacif, D. Sales, and O. P. V. Neto, "A placement and routing algorithm for Quantum-dot Cellular Automata," in *SBCCI*, 2016.

[19] R. K. Nath, B. Sen, and B. K. Sikdar, "Optimal synthesis of qca logic circuit eliminating wire-crossings," *IET-CDS*, vol. 11, no. 3, pp. 201–208, 2017.

[21] F. S. Torres *et al.*, "Exploration of the synchronization constraint in quantum-dot cellular automata," in *DSD*, pp. 642–648.

[22] M. Eiglsperger, S. P. Fekete, and G. W. Klau, "Orthogonal graph drawing," in *Drawing Graphs*. Springer, 2001, pp. 121–171.

[23] T. C. Biedl and G. Kant, "A better heuristic for orthogonal graph drawings," *Computational Geometry*, vol. 9, no. 3, pp. 159–180, 1998.

[24] W. Liu, E. E. Swartzlander Jr, and M. O'Neill, *Design of semiconductor QCA systems*. Artech House, 2013.

[25] 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.

[26] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, 1st ed. McGraw-Hill Higher Education, 1994.

[27] M. Taucer, F. Karim, K. Walus, and R. A. Wolkow, "Consequences of many-cell correlations in clocked Quantum-dot Cellular Automata," *TNANO*, vol. 14, no. 4, pp. 638–647, 2015.

[28] 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.

[29] D. Giri, M. Vacca, G. Causapruno, W. Rao, M. Graziano, and M. Zamboni, "A standard cell approach for magnetoelastic nml circuits," in *2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, July 2014, pp. 65–70.

[30] C. A. T. Campos, A. L. P. Marciano, O. P. V. Neto, and F. S. Torres, "USE: A universal, scalable, and efficient clocking scheme for QCA," *TCAD*, vol. 35, no. 3, pp. 513–517, 2016.

[31] 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.

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

[33] M. Janez, P. Pecar, and M. Mraz, "Layout design of manufacturable Quantum-dot Cellular Automata," *Microelectroics Journal*, vol. 43, no. 7, pp. 501–513, 2012.

[34] V. Vankamamidi, M. Ottavi, and F. Lombardi, "Two-dimensional schemes for clocking/timing of QCA circuits," *TCAD*, vol. 27, no. 1, pp. 34–44.

[35] 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.

[36] T. C. Biedl, "Improved orthogonal drawings of 3-graphs," in *CCCG*, 1996, pp. 295–299.

[37] M. Walter, "fiction – field-coupled technology-independent open nanocomputing," https://github.com/marcelwa/fiction, 2018.

[38] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," in *ISCAS*. IEEE Press, 1985, pp. 677–692.

[39] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Int'l Workshop on Logic Synth.*, 2015.

[40] K. Walus and G. A. Jullien, "Design tools for an emerging SoC technology: Quantum-dot Cellular Automata," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1225–1244, 2006.

[41] F. S. Torres, R. Wille, M. Walter, P. Niemann, D. Große, and R. Drechsler, "Evaluating the impact of interconnections in Quantum-dot Cellular Automata," in *DSD*, 2018, pp. 649–656.