SAT-based Exact Physical Design for Field-coupled Nanocomputing Technologies

Marcel Walter*, Winston Haaswijk†, Robert Wille‡§¶, Frank Sill Torres‖, Rolf Drechsler*§

*Group of Computer Architecture, University of Bremen, Germany
†Cadence Design Systems, Inc., San Jose, CA, USA
‡Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
§Cyber Physical Systems, DFKI GmbH, Bremen, Germany
¶Software Competence Center Hagenberg GmbH (SCCH), Austria
‖Department for the Resilience of Maritime Systems, DLR, Bremerhaven, Germany

***Abstract*—*Field-coupled Nanocomputing* (FCN) is a class of emerging post-CMOS technologies. It includes devices such as *Quantum-dot Cellular Automata* (QCA), *Nanomagnet Logic* (NML) devices, and *Silicon Dangling Bonds* (SiDB). Since they do not rely on the flow of electric current, the promise of these technologies is to overcome the physical limitations of conventional solutions such as CMOS, by allowing for high computational throughput with low power dissipation. Despite their promise, FCN design automation is still in its infancy. State-of-the-art solutions map logic networks obtained from conventional synthesis approaches to FCN circuit layouts, even though, those networks have not been generated with routability for the tight FCN design constraints in mind. In this paper, we take advantage of recent developments in SAT-based exact logic synthesis by adapting methods from topology-based exact synthesis yielding a one-pass flow that starts with a logic description and combines synthesis with physical design of FCN circuit layouts using arbitrary topologies. We implemented this algorithm as an open-source library which we then integrated into an existing FCN design tool. Our experimental evaluation shows how our algorithm can be used to generate the most area efficient circuit layouts for given functions compared to the state-of-the-art.**

I. Introduction

Worldwide energy consumption allotted to information and telecommunication systems is growing. Some scenarios predict that the sector could reach as much as 51 % of global electricity usage by 2030 and thereby contribute up to 23 % of the globally released greenhouse gases [1].

Consequently, there is an increasing interest in alternative technologies that enable fast computations with considerably lower energy dissipation compared to state-of-the-art CMOS transistors. *Field-coupled Nanocomputing* (FCN) [2] is a class of emerging technologies and is constantly gaining more attention. In contrast to conventional technologies, FCN conducts computations without any electric current flow—allowing operations with a remarkable low energy dissipation that is several magnitudes below current CMOS technologies [3], [4]. This promising outlook motivated explorations into its feasibility which led to several suitable contributions to the physical implementation of FCN technologies in the last couple of years [5]–[8].

Motivated by these promising implementations, research on physical design algorithms began. Unfortunately, this task in FCN is not compatible with a classical placement of gates and routing of wires because much tighter domain-specific constraints apply. In FCN, *clocking* is a critical factor of combinational and sequential circuits alike because it directs the data flow and, at the same time, controls information synchronization. Among other obstacles, these *clocking constraints* are one limiting factor of the already $\mathcal{NP}$-hard design automation for FCN circuitry—as they prevent the applicability of conventional VLSI approaches—despite significant efforts in the development of corresponding methods [9].

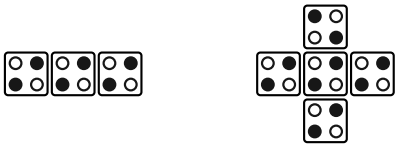
While many existing FCN circuit layouts have been obtained (partially) by human labor, e. g. [10], [11], also some fully-automatic exact and heuristic approaches for physical design exist, e. g. [12]–[15]. Those overcome the aforementioned clocking constraints by relying on regular clock topologies, so-called *clocking schemes*, onto which AIGs and MIGs as Boolean function representations are then mapped. Given a specific clocking scheme, unfavorably structured logic networks or suboptimal technology mapping can lead to tremendous overhead in both circuit area and delay.

Nevertheless, all existing techniques do utilize conventional logic synthesis algorithms for generating their graph-based logic descriptions even though those have not been developed with an understanding of FCN clocking schemes in mind. Consequently, inevitably this will lead to area overhead even when using the most sophisticated layout algorithms possible as long as they perform a direct mapping of the given network. However, so far, to the best of our knowledge, this is what all existing FCN layout algorithms do due to the absence of specialized logic synthesis.

In this paper, we take advantage of recent developments in SAT-based exact logic synthesis by adapting methods from topology-based exact synthesis yielding a one-pass flow that starts with a logic description and combines synthesis with physical design of FCN circuit layouts using arbitrary topologies. Thereby, relevant logic can be directly synthesized onto a clocking scheme avoiding overhead which grants the smallest possible circuit layouts in terms of area.

The rest of this work is structured as follows. To keep the paper self-contained, Section II reviews FCN concepts in detail and provides background on logic synthesis. Section III presents our novel SAT-based synthesis approach for FCN that we evaluate experimentally in Section IV. Finally, Section V concludes the paper.

(a) Binary 0 and binary 1 state



(b) Wire segment (c) Majority gate

Fig. 1: Elementary QCA cell devices

II. Preliminaries

This section reviews background on Field-coupled Nanocomputing and SAT-based exact logic synthesis which are the main aspects of this paper and are therefore necessary for understanding the remainder of this work.

A. Field-coupled Nanocomputing

This section provides background on *Field-coupled Nanocomputing* (FCN) and, by this, a basis for the remainder of this work. FCN can be considered as a concept that is being used as an umbrella term for a class of physical implementations that conduct computations based on the same principles.

Some prominent representatives of the FCN class are *Quantum-dot Cellular Automata* (QCA, [16], [17]), *Nanomagnet Logic* (NML, [18]), and *Silicon Dangling Bonds* (SiDB, [5], [7], [19]). Even though their physical properties differ and most of them are again divided into sub-categories, their abstract models are nearly identical which makes most algorithmic considerations applicable to the entire FCN class. For the sake of brevity, we therefore only review aspects of QCA-like technologies in this section and will use them as a running example for all further illustrations in this paper because we abstract from physical properties later on anyways. We refer the inclined reader to the cited original works for further information on the different technologies.

Generally, FCN circuits are implemented using elements that interact via local fields that are usually called *cells*. In QCA, a cell is composed of four (or six) *quantum dots* which are able to confine an electric charge each and that are arranged at the corners (and the center) of a square [20]. Adding two free and mobile electrons, that are able to tunnel between adjacent dots, into each cell, yields two stable states due to Coulomb interaction, i. e. the two electrons tend to locate themselves at opposite corner quantum dots. Tunneling to the outside of the cell is prevented by a potential barrier.

Each of the two states is called a *cell polarization*, namely $P = -1$ and $P = +1$ which can be defined as binary 0 and binary 1. Fig. 1a depicts these two states of a conceptualized QCA cell. The square denotes the potential barrier to the outside world electrons cannot overcome, quantum dots are illustrated by the four circles, and the two black bullets represent electrons occupying a quantum dot.

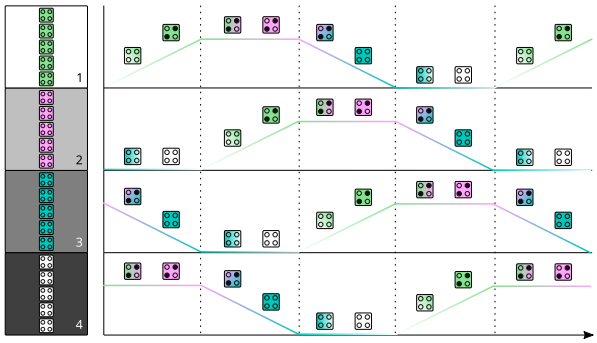


Fig. 2: A sequence of five consecutive clock phases as seen from the perspective of some wire segments

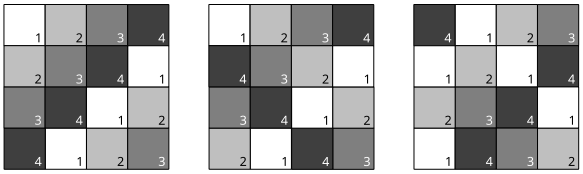
When composing a structure of several FCN cells adjacent to each other, field interactions cause the polarization of one cell to influence the polarization of the others.

Example 1. *Fig. 1b shows how to arrange multiple cells in a row to build a wire segment. It transmits binary information from left to right or vice versa as the same field interactions happen across the cell boundaries and thereby affect the polarization of adjacent cells. This formation is extended in Fig. 1c to construct a Majority gate where three input cells (e. g. top, left, and bottom) compete for the polarization of the center cell that eventually transmits its value to the output cell (e. g. the right one). By setting one input to a constant value, AND and OR gates can easily be constructed from Majority gates, too.*

While single gates and wire segments can be built this way, signals in larger designs get increasingly meta-stable. Furthermore, FCN structures as reviewed so far do not employ an information flow direction. Both issues are circumvented by *clocking* which is a fundamental aspect of combinational and sequential FCN circuits alike. In fact, all cells must be associated to an external clock that controls the initialization, holding, and resetting of their states. In case of QCA, an external electric clock controls the tunneling within the cells. Depending on the technology, each cell changes during a complete clock cycle between up to four different phases, i. e. a *switch*, a *hold*, a *reset*, and a *neutral* phase. Usually, four external clocks numbered from 1 to 4 are applied, whereby each clock controls a selected adjacent set of cells and is shifted by 90° compared to its predecessor. Furthermore, information flows from cells controlled by clock 1 to cells controlled by clock 2 etc. and eventually back to cells controlled by clock 1 again. An example illustrates the concepts.

Example 2. *Fig. 2 depicts on the left a wire consisting of four segments, each composed of five QCA cells grouped together in boxes of different gray shades. Additionally, the cells are tinted in a color depending on their group. Both, the gray shade of the boxes and the tinting visualizes the assigned clock whose number can also be found in each box' bottom right corner.*

Right next to each group is a curve that symbolizes its clock phases with time being plotted on the x-axis and dashed lines indicating time steps. Additionally, the annotated QCA



(a) 2DDWave [22] (b) USE [23] (c) RES [24]

Fig. 3: Clocking schemes for FCN circuit layouts

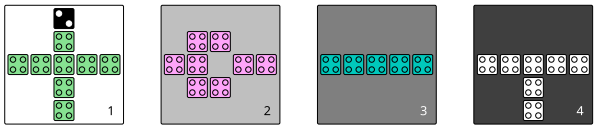
cells indicate the stabilization and destabilization of their states by vanishing and reappearing electron configurations. For instance, group 1 starts in the switch phase, in which the inter-dot barriers are being raised and so does the clock curve. In this phase, the cells can accept information from their predecessors. In the next time step, the clock has reached the hold phase and the cells thereby cannot change their polarization anymore. Hence, they influence the cells in clock zone 2 right below which are in the switch phase at the same time to assume their value, and so on. Thereby, information propagates from the top to the bottom, i. e., from clock zone 1 to 2, from 2 to 3, and from 3 to 4.

For the longest time, it was assumed by designers that these *clock zones* could be of arbitrary size and contain varying amounts of cells. Creating fully clocked circuit layouts in this so-called *cell-based* paradigm was comparably easy as the clocking could be added after laying out the gates and wires. However, the cell-based paradigm was proven to yield unfabricable or incorrect circuits in the recent past [8], [21]. Hence, nowadays state-of-the-art in FCN design follows the so-called *tile-based* paradigm in which all clock zones have a uniform (square) shape and are arranged in a repetitive *clocking scheme*. Each tile in a clocking scheme can hold up to one elementary device from an associated gate library.

Example 3. *Fig. 3 depicts cutouts of size* 4×4 *tiles of three common clocking schemes mostly used for QCA-like technologies. They can all be extrapolated seamlessly in all directions but provide different assets and drawbacks.*

Consider the 2DDWave *clocking scheme as sketched in Fig. 3a. It forms one of the simplest floor plans where each counter diagonal is assigned the same clock number. This way, the incoming information flow to a tile is solely possible from the northern and western directions while the outgoing information flow from a tile always has to utilize the eastern or southern directions. That inherently restricts the scheme in multiple ways, since e. g. (1) sequential circuits cannot be realized due to the lack of feed-back loops and (2) neither Majority gates nor 3-output fan-outs are possible due to the maximum input and output degree of* 2 *for each tile.*

This last issue is a problem with the USE *clocking scheme sketched in Fig. 3b as well. While USE indeed allows feed-back, its tiles' maximum input and output degree is also* 2*. The* RES *scheme sketched in Fig. 3c overcomes this restriction and allows for feedback, Majority gates, and 3-output fan-outs in certain tiles. However, due to the increased degree in some tiles, the degree in other tiles must naturally be lower as we are still facing a 2-dimensional grid structure. Therefore, circuit layouts tend to become more widespread in the RES scheme and consequently have higher area costs and longer critical paths in placement & routing case studies [24].*



(a) AND (b) INV (c) Wire (d) Fan-out

Fig. 4: Tiles in QCA ONE implementation

Several gate libraries have been proposed for various FCN technologies. In this paper, we utilize the *QCA ONE* library [25] for visualizations that proposes tiles of size 5×5 QCA cells. Implementations of an AND gate, an inverter, a wire segment and a fan-out are shown in Fig. 4.

B. Exact Synthesis

Exact synthesis refers to a class of logic synthesis methods that are capable of yielding results to logic synthesis problems in a way that is somehow exact with respect to their specification. Given a Boolean function $\mathbb{B}^n \to \mathbb{B}^m$ and some $r \in \mathbb{N}$, consider the following examples:

1) Synthesize multi-output *exclusive-sum-of-products* (ESOP) expression with exactly r cubes that represents f.
2) Synthesize a logic network with exactly r gates that computes f.

These are two problems in two different logic representations for which we have to exactly match the specification function f and size parameter r.

Various exact synthesis methods have been developed. The Quine-McCluskey algorithm and Petrick's method are well-known for the minimization of SOPs [26], [27]. Similar methods have been developed for so-called *exclusive* SOPs (ESOPs) as well [28]. In multi-level logic synthesis, there are different exact minimization algorithms such as the decomposition techniques of Ashenhurst, Curtis, Davidson, and Roth and Karp [29]–[32]. More recently, enumeration-based techniques were developed by Knuth and Amaru [33], [34].

The advantage of exact logic synthesis is that it can be used to synthesize optimum representations with respect to different cost functions. For example, the r parameter can be used to specify depth as well as size. By varying this parameter we can then use an exact synthesis algorithm to determine representations of different depths or sizes. The term *exact synthesis* is commonly used to refer to this approach that obtains optimum logic representations.

The disadvantage of exact synthesis is its high computational complexity. For example, determining minimum SOP representations with the algorithm is $\mathcal{NP}$-hard since it is equivalent to the *Set cover* problem [35]. In multi-level exact synthesis, determining the minimum-size logic network for a function is equivalent to the *Minimum circuit size problem* (MCSP). The multi-output version of the MCSP was recently proved to be $\mathcal{NP}$-hard [36]. It is currently not known if this result also holds for single-output functions, although strong evidence in that direction exists [37].

C. Boolean Satisfiability

The *Boolean satisfiability problem* (SAT) is the canonical $\mathcal{NP}$-complete problem in which one is given a set of Boolean

variables X and a Boolean formula $\phi(x_1, \ldots, x_n), x_i \in X$ and asked to determine whether there exists a variable assignment $\pi : X \to \mathbb{B}$ such that $\phi(\pi(x_1), \ldots, \pi(x_n)) = 1$.

In the past two decades, there has been a lot of progress in the development of advanced algorithms and data structures for SAT solvers; compute engines dedicated to solving the SAT problem. Techniques such as DPLL search, watch lists, clause learning, and non-chronological backtracking have greatly improved the efficiency of such solvers [38]. Current state-of-the-art solvers regularly deal with formulas consisting of millions of variables and clauses [39].

Since SAT is $\mathcal{NP}$-complete, all problems in $\mathcal{P}$ and $\mathcal{NP}$ can be reduced to it, allowing us to use SAT solvers to determine their solutions. The convenience of this approach is that we do not have to develop separate algorithms for each task. Instead, we can use SAT solvers as generalized problem solving engines. Many problems in logic synthesis have a natural SAT formulation, as they tend to deal with Boolean variables natively. Combined with the progress of SAT solvers, this has led to the development of many SAT-based algorithms. They range from combinational equivalence checking and SAT sweeping to automatic test pattern generation, resubstitution, and logic synthesis [40]–[44]. Conversely, logic synthesis methods have also been used to improve the efficiency of SAT solvers [45].

D. SAT-based Exact Synthesis

The progress made in SAT solving coupled with increases in compute power have led to a resurgence of exact synthesis algorithms based on SAT solver backends. In the two-level domain, SAT-based algorithms for exact ESOP synthesis have been developed for both classical and quantum logic synthesis [46], [47]. In multi-level synthesis, we can find exact synthesis methods for specific data structures such as *majority inverter graphs* (MIGs) and *XOR-majority graphs* (XMGs) as well as for arbitrary k-LUT logic networks [48], [49]. The intractability of SAT means that such methods can only be used to synthesize small multi-level representations. However, they have been successfully integrated into large-scale synthesis algorithms such as DAG-aware logic rewriting using network partitioning techniques like cut generation and windowing [50].

To mitigate the potentially exponential runtime of SAT, various techniques have been applied. These include as the development of alternative CNF encodings, the addition of symmetry breaking clauses, and the use of counterexample-guided abstraction refinement (CEGAR) [51]–[53]. State-of-the-art algorithms have started to incorporate DAG topology information in the synthesis loop, which leads to the creation of a hybrid algorithm in which the SAT search space is reduced. This has been shown to unlock significant speed-ups [54]. Further, it has been shown that graph topology families can be used to parallelize SAT-based exact synthesis [55].

In this paper, we take advantage of recent developments in topology-constrained multi-level exact logic synthesis algorithms. We find that, with some small changes, the techniques used in [54] and [55] can be adapted to the design of FCN circuits, yielding the first one-pass solution in this domain.

III. Topology-based Exact Synthesis for FCN

In this section, we describe our FCN exact one-pass synthesis method. Section III-A provides a high-level overview in which we describe how the physical synthesis problem is specified and executed. We also highlight the differences with conventional exact logic synthesis. Next, in Section III-B, we give a detailed description of the CNF encoding. The method has been implemented as a Python package named *Mugen*. This implementation is discussed in Section IV where we also demonstrate its use with sample code.

A. Method Overview

We present a SAT-based algorithm for one-pass synthesis of FCN circuit layouts. Its input is a logic specification in terms of a multi-output truth table together with a clocking scheme that serves as the topology and a gate library that defines the operations to use. Its output is an FCN circuit layout of optimal area in terms of tiles that computes the given function. The algorithm is parameterizable to incorporate various design elements that can be found in the literature. These include but are not limited to the use of different clocking schemes, gate libraries, wire-crossings, and I/O configuration (e. g. the use of designated I/O pins). Moreover, it is able to generate multiple different circuits from the same specification.

We like to highlight that conventional SAT-based logic synthesis differs from the approach we are proposing here in many ways due to the domain-specific constraints the technology imposes. We briefly discuss some of these differences in the following:

1) Different gate types and arities: Different clocking schemes allow different gate types to be used. For example, the USE clocking scheme does not allow the 3-input Majority gate. Moreover, not all gate types correspond to operations with the same arity. Finally, clocking schemes allow for some parts of the layout to remain unused and empty. In logic synthesis, this corresponds to gates with no fan-in and no fan-out. Typically, this is not allowed in SAT-based exact synthesis.

2) Fan-out restrictions: Different configurations of clocking schemes place various types of restrictions on primary input and gate fan-outs. For example, each primary input may be read only once. The same holds for most gate types. This is different from the conventional logic synthesis paradigm, in which logic sharing is encouraged to compress representations. In FCN, a designated fan-out element is required to copy a signal to two different paths.

3) I/O restrictions: In some clocking schemes, internal gates cannot refer to I/O signals directly. Rather, I/O signals are provided by dedicated I/O pins. Again, this is different from most logic synthesis representations.

4) Cycles: The potential (virtual) connections specified by clocking schemes may contain cycles (e. g. USE and RES). This is very different from conventional multi-level logic synthesis in which the directed *acyclic* graph (DAG) is a central data structure. The connections are not meant to be part of the final design which must still be acyclic. However, they are an important part of the FCN problem specification.

5) Geometry: The geometry specified by a clocking scheme is of significant importance. First, the clocking scheme determines a grid graph and fixes the maximum number of components that can be used as well as their location on the graph. As mentioned earlier, we refer to nodes on the grid graph as tiles. Second, the virtual connections specified by the scheme determine the legal directions in which data

flows through the graph. We refer to these edges as virtual because they may or may not be used in the final design. Rather, they specify which potential internal connections the scheme allows. We can think of these connections as specifying a partial topology, similar to the *fences* and *partial DAGs* defined in [55]. Depending on the scheme, a node on the graph may have fan-in and fan-out coming from any of the cardinal directions: north, east, south, and west. Which direction are fan-in and which are fan-out is determined by the clocking scheme. Moreover, some directions may be used for both fan-in and fan-out. In logic synthesis, there is typically no such a notion of an input direction. The order of inputs to a gate may matter, but only if the gate realizes an asymmetrical function. However, in a clocking scheme, it is important that we take physical direction into account: every tile has definite input and output directions. Moreover, if it has multiple virtual fan-outs, it must enable and disable certain directions depending on how the circuit is routed. Finally, geometry also determines the gate types supported at each tile. For example, if a tile has only two possible fan-in directions it cannot possibly support a Majority gate, even if it such gates are generally allowed by the clocking scheme.

Given a clocking scheme, a multi-output truth table, and further configurations, these specifications are compiled to a CNF encoding which is described in detail in the next section. A SAT solver then synthesizes and enumerates all circuits that satisfy the clocking scheme and functions.

B. CNF Encoding

We base our CNF formulation on the SSV encoding which is described in [55]. SSV is used for the synthesis of homogeneous *normal* k-input logic networks. Hence, there are several substantial differences in our encoding (see Section III-A). The correctness of our encoding follows from SSV. A more formal justification can be found in [55].

In this synthesis problem, we are given a clocking scheme of size $W \times H$ as well as a multi-output Boolean function $f = (f_1, \ldots, f_m) : \mathbb{B}^n \to \mathbb{B}^m$ over n variables $x_1, \ldots, x_n$. We identify each tile in the layout by its coordinates (x, y). The top-left corner of the clocking scheme is identified by $(0, 0)$, so we have $0 \leq x < W$ and $0 \leq y < H$. The gate types supported by tile (x, y) depend on the user specification (i. e. which types they enabled) as well as the local geometry. The same holds for its potential fan-in/fan-out connections. Therefore, we construct the following sets for each tile:

$$
\begin{aligned}
\mathcal{I}_{(x,y)} &: \text{ fan-in directions for } (x, y) \\
\Omega_{(x,y)} &: \text{ fan-out directions for } (x, y) \\
\Delta_{(x,y)} &: \text{ primary I/O directions for } (x, y) \\
\widetilde{\mathcal{I}}_{(x,y)} &: \text{ potential fan-ins for } (x, y) \\
\widetilde{\Omega}_{(x,y)} &: \text{ potential fan-outs for } (x, y) \\
\Theta_{(x,y)} &: \text{ enabled gate types for } (x, y)
\end{aligned}
$$

These sets can be constructed by a simple procedure which checks the local connectivity for each tile and refers to the clocking scheme specifications. Note that we have $\mathcal{I}_{(x,y)} \cup \Omega_{(x,y)} = \{north, east, south, west\}$.

Every tile must choose some gate type, so $\Theta_{(x,y)} \neq \emptyset$. We always enable the special type ϵ, which corresponds to the empty tile. We write $\phi_{\tilde{\iota}}(\tilde{\iota}) = k$ to indicate that fan-in option $\tilde{\iota}$ has k fan-ins. Similarly, we write $\phi_\theta(\theta) = k$ to indicate gate operator arity.

For tile (x, y), we then create the following variables, for $1 \leq h \leq m$, $\omega \in \Omega_{(x,y)}$, $\delta \in \Delta_{(x,y)}$, $\tilde{\iota} \in \widetilde{\mathcal{I}}_{(x,y)}$, $\theta \in \Theta_{(x,y)}$, $\tilde{\omega} \in \widetilde{\Omega}_{(x,y)}$, and $0 \leq t < 2^n$:

$$
\begin{aligned}
x_{(x,y)\omega t} &: \text{t}^{\text{th}} \text{ bit of } (x, y)\text{'s truth table in direction } \omega \\
g_{h(x,y)\delta} &: f_h(x_1, \ldots, x_n) \text{ points to } (x, y)\text{'s output port } \delta \\
s_{(x,y)\tilde{\iota}} &: (x, y) \text{ selects fan-in } \tilde{\iota} \\
t_{(x,y)\theta} &: (x, y) \text{ has gate type } \theta \\
c_{(x,y)\tilde{\omega}} &: (x, y) \text{ is connected to } \tilde{\omega}
\end{aligned}
$$

The $g_{h(x,y)\delta}$ variables are generated only if $\Delta_{(x,y)} \neq \emptyset$.

We constrain these variables by a set of clauses which ensures that (1) the circuit both computes the correct functions and (2) the circuit satisfies all requirements of the clocking scheme.

For each (x, y), $\omega \in \Omega_{(x,y)}$, and $0 \leq a, b, c \leq 1$, the following constraints ensure that the circuit simulates the correct function at each coordinate:

$$
\begin{cases}
(\bar{s}_{(x,y)\tilde{\iota}} \vee \bar{t}_{(x,y)\theta} \vee (x_{\tilde{\iota}(1)t} \oplus a) \vee (x_{(x,y)\omega t} \oplus \overline{\theta_\omega(a)})) & \text{if } \phi(\theta) = 1 \\
(\bar{s}_{(x,y)\tilde{\iota}} \vee \bar{t}_{(x,y)\theta} \vee (x_{\tilde{\iota}(1)t} \oplus a) \vee (x_{\tilde{\iota}(2)t} \oplus b) \vee (x_{(x,y)\omega t} \oplus \overline{\theta_\omega(a,b)})) & \text{if } \phi(\theta) = 2 \\
(\bar{s}_{(x,y)\tilde{\iota}} \vee \bar{t}_{(x,y)\theta} \vee f(x_{\tilde{\iota}(1)t} \oplus a) \vee (x_{\tilde{\iota}(2)t} \oplus b) \vee (x_{\tilde{\iota}(3)t} \oplus c) \vee (x_{(x,y)\omega t} \oplus \overline{\theta_\omega(a,b,c)})) & \text{if } \phi(\theta) = 3
\end{cases}
$$

With some abuse of notation, we use $\tilde{\iota}(k)$ here to refer to the k-th fan-in for fan-in option $\tilde{\iota}$, and $\theta_\omega(a)$ to refer to the result of applying the Boolean function corresponding to gate type θ in output direction ω. For all clauses, we ensure that $\phi_{\tilde{\iota}}(\tilde{\iota}) = \phi_\theta(\theta)$.

We describe the intuition behind these clauses for the case $\phi_\theta(\theta) = 2$. The other cases are analogous. If (x, y) has inputs $i_1 = \tilde{\iota}(1)$ and $i_2 = \tilde{\iota}(2)$ *and* (x, y) is of type θ *and* the t$^{\text{th}}$ bit of i_1 is a *and* the t$^{\text{th}}$ bit of i_2 is b *then* it must be the case that $x_{(x,y)\omega t} = \theta_\omega(a, b)$. This can be more easily understood by rewriting the constraint as follows:

$$
((\bar{s}_{(x,y)\tilde{\iota}} \wedge \bar{t}_{(x,y)\theta} \wedge (x_{\tilde{\iota}(1)t} \oplus \bar{a}) \wedge (x_{\tilde{\iota}(2)t} \oplus \bar{b})) \rightarrow (x_{(x,y)\omega t} \oplus \overline{\theta_\omega(a,b)}))
$$

Note that a, b, and c are constants which are used to set the proper variable polarities.

Let $(b_1, \ldots, b_n)_2$ be the binary encoding of truth table index t. In order to fix the proper output values, we add the clauses $(\bar{g}_{h(x,y)\delta} \vee \bar{x}_{(x,y)\delta t})$ or $(\bar{g}_{h(x,y)\delta} \vee x_{(x,y)\delta t})$ depending on the value $f_h(b_1, \ldots, b_n)$. Next, for each output, we add $\bigvee_{x=0}^{W} \bigvee_{y=0}^{H} g_{h(x,y)\delta}$. This ensures that every primary output points to the output port of some tile. Each tile must select some gate type, so we add $\bigvee_{\theta \in \Theta_{(x,y)}} t_{(x,y)\theta}$.

Clocking schemes can contain cycles, so we must add clauses to ensure that the final design does not. We achieve this by first detecting all cycles in the graph and then using the *connection variables* $c_{(x,y)\tilde{\omega}}$ to prevent them. Let $((x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n), (x_0, y_0))$ be a cycle. We then add the clause $\bigvee_{i=0}^{n} \bar{c}_{x_i y_i (x_{(i+1 \bmod n)}, y_{(i+1 \bmod n)})}$. We must further ensure that a $c_{(x,y)\tilde{\omega}}$ is set to true whenever fan-out $\tilde{\omega}$ selects (x, y) as fan-in. To that end, for all $\tilde{\omega} \in \widetilde{\Omega}_{(x,y)}$ and $\tilde{\iota} \in \widetilde{\Omega}_{\tilde{\omega}}$ we add $(\bar{s}_{\tilde{\omega}\tilde{\iota}} \vee c_{(x,y)\tilde{\omega}})$ if $(x, y) \in \tilde{\iota}$.

We have now described the main clauses. We use some additional clauses to satisfy various cardinality constraints.

```
# Create a new 3x3 clocking scheme.
g = scheme_graph(shape=(3,3))
# We want to specify the USE scheme,
# so we disable majority gates.
g.enable_maj = False
# Next, we specify the virtual
# potential connections between
# nodes on the grid.
g.add_virtual_edge((0, 0), (1, 0))
g.add_virtual_edge((1, 0), (2, 0))
# Some connections omitted for
# brevity.
...
g.add_virtual_edge((2, 2), (2, 1))
# The list of functions the circuit
# must compute. In this case we
# specify a 2:1 MUX.
functions = [[0,0,1,1,0,1,0,1]]
# Enumerate the different circuits
# which satisfy the specification.
for net in g.synthesize(functions):
g.satisfies_spec(net, functions)
yield net
```

Listing 1: An example of FCN physical design with *Mugen*

These include constraints to ensure that PIs have at most single fan-out, tile output ports may be used only once, and making sure that every tile selects at least some fan-in option (unless it is an empty tile).

IV. Experimental Evaluation

In this section, we discuss our setup for an experimental evaluation and present respective results. First, in Section IV-A we discuss integration of Mugen with an existing design tool for FCN and go over the specifications of the system used in the following evaluation. In Section IV-B, we compare Mugen against a state-of-the-art exact placement & routing algorithm for FCN [14]. In Section IV-C, we utilize Mugen to generate FCN circuit layouts for NPN classes on different clocking schemes, and, finally, in Section IV-D, we discuss notable findings we made in the process that we hope provide valuable knowledge to both the logic synthesis and the FCN community.

A. Experimental Setup

Our proposed package Mugen is open source and available to the public at https://github.com/whaaswijk/mugen. We have integrated it into the open-source FCN design framework *fiction* [56] using *pybind11* [57]. Listing 1 contains a code example in which the user specifies a 3×3 USE topology to synthesize a 2:1 MUX.

All evaluations in the following sections were run on a Fedora 28 machine with an Intel Xeon E3-1270 v3 CPU with 3.50 GHz (up to 3.90 GHz boost) and 32 GB of main memory. The underlying SAT solver used by Mugen was Glucose 3.0 [58].

All resulting designs and simulation files for *QCADesigner* [59] were made publicly available at https://github.com/marcelwa/IWLS2020FCN.

TABLE I: Comparison against exact placement & routing [14]

Function	Exact P&R [14] A	CP	t in s	Proposed approach A	CP	t in s
2:1 MUX	9	5	< 1	9	5	1
XOR	9	5	< 1	9	5	19
XNOR	16	8	2	16	8	19
Half adder	25	10	13	16	8	42
c17	30	16	56	18	10	331
ParGen	42	14	791	—	—	TO
ParCheck	48	16	1140	—	—	TO
4:1 MUX	49	22	5131	—	—	TO

A Area in tiles given by the layout's bounding box
CP Critical path in tiles
TO Timeout reached

B. Comparison Against Exact Placement & Routing

As mentioned in Section II, the default approach to FCN physical design is placement & routing of readily synthesized logic networks like AIGs or MIGs onto a clocking scheme. The biggest drawback with this approach is that those networks have been obtained from conventional synthesis algorithms and were not optimized to be routable on FCN topologies. A common assumption in the FCN community is that overhead due to wire routing could be reduced by tailoring the logic network to the clocking scheme. Consequently, when directly comparing a layout obtained by optimal placement & routing of a non-optimized logic network and a layout obtained by our proposed topology-based synthesis approach, the latter must never be worse in terms of area.

Walter et al. proposed an exact method for placement & routing of QCA circuit layouts [14], i. e. a mapping of an existing logic network to a clocking scheme. Table I compares their obtained results against the ones we could generate using our proposed one-pass synthesis on the same functions that were taken from their work using the same configuration.

The column *Function* lists the function names that were used as inputs to both approaches. The following columns *A*, *CP*, and *t in s* repeat for both approaches and list the area of the resulting circuit layout in tiles, its critical path in tiles, and the time in seconds it took to obtain the results. In [14], the authors listed the circuit area in terms of cells where each tile would be composed of 5×5 QCA cells. We converted the area values accordingly for Table I. Note that critical path was not an optimization target in either algorithm but is listed for the sake of completeness and because the authors listed it in [14] as well. Note further that both approaches have been evaluated on different hardware systems. Therefore, the runtimes are not directly comparable but give a good approximation.

The first thing to notice is that our proposed approach has a non-negligible runtime overhead compared to the placement & routing approach and even timed out on three of the functions. However, it was able to synthesize substantially smaller circuit layouts for both the half adder and the c17 function while yielding the same circuit area for the remaining functions. This observation coincides with our initial assumption.

C. Synthesizing Circuit Layouts for NPN Classes

NPN classification determines if some single-output Boolean functions are identical under permutation and negation of their inputs and negation of their output. NPN classes are of great interest in logic design because they tremendously reduce the number of representatives that are to be considered

TABLE II: Area results for all 3-input NPN classes

NPN	2DDWave A	2DDWave #G	2DDWave #W	USE A	USE #G	USE #W	RES A	RES #G	RES #W
0x00	8	5	1	8	5	3	8	8	0
0x01	10	8	1	10	7	2	8	7	1
0x03	8	6	1	6	5	0	8	5	2
0x06	18	13	3	20	12	8	20	13	5
0x07	10	8	0	10	7	2	8	7	1
0x0f	4	3	0	4	3	0	4	3	0
0x16	27	18	8	32	18	14	32	21	10
0x17	20	15	3	24	13	12	9	6	1
0x18	24	16	7	28	15	14	30	14	11
0x19	18	14	3	20	14	6	20	13	7
0x1b	15	11	2	16	11	6	15	10	5
0x1e	18	12	3	24	15	10	24	13	9
0x3c	15	10	4	20	12	8	16	9	6
0x69	32	18	6	32	18	16	32	19	13
total	227	157	42	254	155	101	234	148	71

A Area in tiles given by the layout's bounding box
#G Number of gates including I/O pins
#W Number of wire segments (counting crossings as 2)

when exhaustively enumerating function spaces without losing expressive power. Permuting and negating primary pins shifts complexity away from the designer and towards the integrator on whose side these tasks are considered to be trivial in most cases.

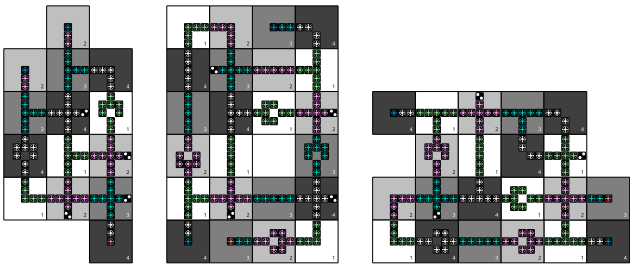
We considered all 3-input NPN classes and synthesized their canonized representatives on the three clocking schemes 2DDWave, USE, and RES that are shown in Fig. 3. This (1) provides us with a design library able to compute any Boolean function in 3 variables on any of the three clocking schemes with optimal area usage and (2) allows us to reason about appropriateness of the clocking schemes that, to the best of our knowledge, no method was able to do with an optimality guarantee so far. While such a design library is crucial for the development of hierarchical or cut-based physical design approaches, the sense of appropriateness can guide designers when setting their parameters. Furthermore, our method can be used to rate future clocking schemes.

The results of our experiments are summarized in Table II. The column *NPN* lists the truth tables of the canonized NPN representatives in hexadecimal notation. These served as inputs to the synthesis runs. The next columns *A*, *#G*, and *#W* repeat for each of the three clocking schemes and list the necessary minimal area in tiles, number of gates, and number of wire segments respectively needed for an FCN circuit layout that implements the given truth table. The final row *total* sums up the respective columns.

It can be seen that the 2DDWave clocking scheme needed the least amount of area and wire overhead to implement all given functions. However, the RES scheme needed the least amount of actual logic which is likely due to the fact that only RES supports Majority gates. The USE clocking scheme had the highest area and wire overhead.

D. Discussion

Since the FCN concept is still in its infancy, several conjectures about its properties in the physical design process could not be proven yet. For instance, it was unknown whether a crossing-free QCA ONE layout implementation of the 2-input XOR function exists that has all primary input and output pins placed exactly once and in a position at the borders where they are accessible. Since XOR is not an elementary gate in the QCA ONE library, typically the composition $a \oplus b = \neg(ab) \cdot (a + b)$ is used whose Boolean chain is non-planar when including the layout borders as fix-points.



(a) 2DDWave (b) USE (c) RES

Fig. 5: Crossing-free realizations of the 2-input XOR function

Mugen enabled us to settle this question. Fig. 5 depicts crossing-free XOR implementations in the QCA ONE gate library for all three clocking schemes investigated in this paper.[1]

Across this paper, we mentioned several times that it is assumed that placement & routing generates an overhead in terms of circuit area that is used for wire routing because the logic networks that serve as inputs were not synthesized with FCN routability in mind. While this work provides weak evidence that even in *exact* placement & routing techniques there indeed still is an overhead that could be eliminated using our proposed approach, we found that the layouts we synthesized need significantly more operations due to wire routing than pure logic networks. We certainly expected some overhead but were surprised by how large it actually is.

V. Conclusion

In this paper, we presented a SAT-based algorithm for exact topology-guided one-pass synthesis for the physical design of *Field-coupled Nanocomputing* (FCN) Technologies. This algorithm is able to overcome the drawbacks of placement & routing-based approaches that have to rely on pre-synthesized logic networks that have not been generated with routability in mind and need to be adjusted for every new clocking scheme. We integrated our approach that we call *Mugen* with the publicly available FCN design framework *fiction* and utilized it in an experimental evaluation to support our claims. Furthermore, we synthesized FCN layouts for all canonized 3-input NPN representatives that can be used as building blocks in future layout approaches. Additionally, with more research in the area, our approach could even enable rewriting techniques on the layout level to optimize existing designs by replacing sub-layouts with their optimum counterparts. Finally, we were able to answer the open question whether it is possible to generate crossing-free QCA ONE layouts for the 2-input XOR function by providing witnesses for three different clocking schemes.

Acknowledgments

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria as well as by BMK, BMDW, and the State of Upper Austria in the frame of the COMET Programme managed by FFG.

[1] Examples of XOR realizations that require less area can be found in the literature, e. g. [60]. However, these are not synthesized from existing well-proven gates, but are cell-based and hand-crafted. They do not guarantee physical correctness and fabricability out-of-the-box and need to be further verified in lab tests or quantum simulations [8].

References

[1] A. S. G. Andrae and T. Edler, "On Global Electricity Usage of Communication Technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.

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

[3] 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.

[4] F. Sill Torres, R. Wille, P. Niemann, and R. Drechsler, "An Energy-Aware Model for the Logic Synthesis of Quantum-Dot Cellular Automata," *TCAD*, vol. 37, no. 12, pp. 3031–3041, 2018.

[5] S. Bohloul, Q. Shi, R. A. Wolkow, and H. Guo, "Quantum Transport in Gated Dangling-Bond Atomic Wires," *Nano Letters*, pp. 322–327, 2017.

[6] C. S. Lent *et al.*, "Molecular Cellular Networks: A non von Neumann Architecture for Molecular Electronics," in *ICRC*, 2016, pp. 1–7.

[7] 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*, pp. 8636–8642, 2017.

[8] H. N. Chiu, S. S. H. Ng, J. Retallick, and K. Walus, "PoisSolver: a Tool for Modelling Silicon Dangling Bond Clocking Networks," 2020, arXiv:2002.10541.

[9] M. Walter, R. Wille, D. Große, F. Sill Torres, and R. Drechsler, "Placement & Routing for Tile-based Field-coupled Nanocomputing Circuits is NP-complete," in *JETC*, 2019.

[10] 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.

[11] M. Kianpour and R. Sabbaghi-Nadooshan, "A novel Quantum-dot Cellular Automata CLB of FPGA," *J. Comput. Electron.*, pp. 709–725, 2014.

[12] F. Riente *et al.*, "ToPoliNano: A CAD Tool for Nano Magnetic Logic," *TCAD*, vol. 36, no. 7, pp. 1061–1074, 2017.

[13] G. Fontes *et al.*, "Placement and Routing by Overlapping and Merging QCA Gates," in *ISCAS*, 2018, pp. 1–5.

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

[15] M. Walter, R. Wille, F. Sill Torres, D. Große, and R. Drechsler, "Scalable Design for Field-coupled Nanocomputing Circuits," in *ASP-DAC*, 2019, pp. 197–202.

[16] 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.

[17] 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.

[18] X. K. Hu *et al.*, "Edge-Mode Resonance-Assisted Switching of Nanomagnet Logic Elements," *IEEE Trans. Magn.*, vol. 51, no. 11, pp. 1–4, 2015.

[19] R. A. Wolkow, L. Livadaru *et al.*, *Silicon Atomic Quantum Dots Enable Beyond-CMOS Electronics*. Springer-Verlag, p. 33–58.

[20] W. Liu, E. E. Swartzlander Jr, and M. O'Neill, *Design of Semiconductor QCA Systems*. Artech House, 2013.

[21] 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.

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

[23] C. A. T. Campos *et al.*, "USE: A Universal, Scalable, and Efficient Clocking Scheme for QCA," *TCAD*, vol. 35, no. 3, pp. 513–517, 2016.

[24] M. Goswami *et al.*, "An efficient clocking scheme for quantum-dot cellular automata," *Electron. Lett.*, pp. 1–14, 2019.

[25] D. A. Reis *et al.*, "A Methodology for Standard Cell Design for QCA," in *ISCAS*, 2016, pp. 2114–2117.

[26] W. V. Quine, "The Problem of Simplifying Truth Functions," *The American Mathematical Monthly*, vol. 59, no. 8, pp. 521–531, 1952.

[27] E. J. McCluskey, "Minimization of Boolean Functions," *Bell System Technical Journal*, vol. 35, no. 6, pp. 1417–1444, 1956.

[28] T. Sasao, "EXMIN2: A Simplification Algorithm for Exclusive-OR-Sum-of Products Expressions for Multiple-Valued-Input Two-Valued-Output Functions," *IEEE Trans. on CAD*, vol. 12, no. 5, pp. 621–632, 1993.

[29] R. Ashenhurst, "The Decomposition of Switching Functions," 1957, pp. 74–116.

[30] A. Curtis, *New Approach to the Design of Switching Circuits*. Van Nostrand, 1962.

[31] E. S. Davidson, "An Algorithm for NAND Decomposition Under Network Constraints," *IEEE Trans. Computers*, vol. 18, no. 12, pp. 1098–1109, 1969.

[32] J. P. Roth and R. M. Karp, "Minimization Over Boolean Graphs," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 227–238, 1962.

[33] D. E. Knuth, *The Art of Computer Programming*. Upper Saddle River, New Jersey: Addison-Wesley, 2011, vol. 4A.

[34] L. Amaru, M. Soeken, P. Vuillod, J. Luo, A. Mishchenko, P. E. Gaillardon, J. Olson, R. Brayton, and G. De Micheli, "Enabling exact delay synthesis," in *Int'l Conf. on Computer-Aided Design*, 2017, pp. 352–359.

[35] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[36] R. Ilango, B. Loff, and I. C. Oliveira, "NP-Hardness of Circuit Minimization for Multi-Output Functions," Electronic Colloquium on Computational Complexity, Tech. Rep. 21, 2020.

[37] C. D. Murray and R. R. Williams, "On the (non) NP-hardness of computing circuit complexity," in *Conference on Computational Complexity*, 2015, pp. 365–380.

[38] A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfability*. IOS Press, 2009.

[39] "Proceedings of SAT Race 2019 : Solver and Benchmark Descriptions," M. Heule, M. J.H. Järvisal, and M. Suda, Eds. Department of Computer Science, University of Helsinki, 2019.

[40] E. I. Goldberg, M. R. Prasad, and R. K. Brayton, "Using sat for combinational equivalence checking," in *Design, Automation and Test in Europe*, 2001, pp. 114–121.

[41] Qi Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "Sat sweeping with local observability don't-cares," in *Design Automation Conference*, 2006, pp. 229–234.

[42] A. Mishchenko and R. K. Brayton, "Sat-based complete don't-care computation for network optimization," in *Design, Automation and Test in Europe*, 2005, pp. 412–417 Vol. 1.

[43] Chih-Chun Lee, J. R. Jiang, Chung-Yang Huang, and A. Mishchenko, "Scalable exploration of functional dependency by interpolation and incremental sat solving," in *2007 IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 227–233.

[44] A. Petkovska, "Exploiting Satisfiability Solvers for Efficient Logic Synthesis," Ph.D. dissertation, EPFL, Lausanne, Switzerland, 2017.

[45] N. Een, A. Mishchenko, and N. Sörensson, "Applying logic synthesis for speeding up sat," in *Theory and Applications of Satisfiability Testing – SAT 2007*, J. Marques-Silva and K. A. Sakallah, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 272–286.

[46] G. Meuli, B. Schmitt, R. Ehlers, H. Riener, and G. De Micheli, "Evaluating esop optimization methods in quantum compilation flows," in *Reversible Computation*, M. K. Thomsen and M. Soeken, Eds. Cham: Springer International Publishing, 2019, pp. 191–206.

[47] H. Riener, R. Ehlers, B. d. O. Schmitt, and G. D. Micheli, *Exact Synthesis of ESOP Forms*. Cham: Springer International Publishing, 2020, pp. 177–194.

[48] M. Soeken, L. G. Amarù, P. Gaillardon, and G. De Micheli, "Exact Synthesis of Majority-Inverter Graphs and Its Applications," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 36, no. 11, pp. 1842–1855, 2017.

[49] W. Haaswijk, M. Soeken, L. Amarù, P. Gaillardon, and G. De Micheli, "A novel basis for logic rewriting," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 151–156.

[50] 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 *Design, Automation and Test in Europe*, 2019, pp. 1649–1654.

[51] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, *Counterexample-Guided Abstraction Refinement*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 154–169.

[52] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Reading, Massachusetts: Addison-Wesley, 2015.

[53] M. Soeken, W. Haaswijk, E. Testa, A. Mishchenko, L. G. Amarù, R. K. Brayton, and G. De Micheli, "Practical exact synthesis," in *Design, Automation and Test in Europe*, 2018, pp. 309–314.

[54] W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli, "SAT Based Exact Synthesis using DAG Topology Families," in *Design Automation Conference*, 2018, pp. 1–6.

[55] ——, "Sat-based exact synthesis: Encodings, topology families, and parallelism," *TCAD*, vol. 39, no. 4, pp. 871–884, 2020.

[56] M. Walter, R. Wille, F. Sill Torres, D. Große, and R. Drechsler, "fiction: An Open Source Framework for the Design of Field-coupled Nanocomputing Circuits," 2019, arXiv:2002.10541.

[57] W. Jakob, "pybind11," https://github.com/pybind/pybind11.

[58] G. Audemard and L. Simon, "Glucose: a solver that predicts learnt clauses quality," *SAT Competition*, pp. 7–8, 2009.

[59] K. Walus, T. J. Dysart, G. A. Jullien, and R. A. Budiman, "QCADesigner: A Rapid Design and Simulation Tool for Quantum-dot Cellular Automata," *TNANO*, vol. 3, no. 1, pp. 26–31, 2004.

[60] M. R. Beigh, M. Mustafa, and F. Ahmad, "Performance Evaluation of Efficient XOR Structures in Quantum-dot Cellular Automata (QCA)," *Circuits and Systems*, pp. 147–156, 2013.