One-pass Synthesis for Field-coupled Nanocomputing Technologies

Marcel Walter
University of Bremen
Group of Computer Architecture
Bremen, Germany
m_walter@uni-bremen.de

Winston Haaswijk
Cadence Design Systems, Inc.
San Jose, CA, USA
haaswijk@cadence.com

Robert Wille[*] [†]
Johannes Kepler University
Institute for Integrated Circuits
Linz, Austria
robert.wille@jku.at

Frank Sill Torres
German Aerospace Center (DLR)
Department for the Resilience of Maritime Systems
Bremerhaven, Germany
frank.silltorres@dlr.de

Rolf Drechsler[‡]
University of Bremen
Group of Computer Architecture
Bremen, Germany
drechsler@uni-bremen.de

ABSTRACT

Field-coupled Nanocomputing (FCN) is a class of post-CMOS emerging technologies, which promises to overcome certain physical limitations of conventional solutions such as CMOS by allowing for high computational throughput with low power dissipation. Despite their promises, the design of corresponding FCN circuits is still in its infancy. In fact, state-of-the-art solutions still heavily rely on conventional synthesis approaches that do not take the tight physical constraints of FCN circuits (particularly with respect to routability and clocking) into account. Instead, physical design is conducted in a second step in which a classical logic network is mapped onto an FCN layout. Using this two-stage approach with a classical and FCN-oblivious logic network as an intermediate result, frequently leads to substantial quality loss or completely impractical results. In this work, we propose a one-pass synthesis scheme for FCN circuits, which conducts *both* steps, synthesis *and* physical design, in a *single* run. For the first time, this allows to generate exact, i. e., minimal FCN circuits for a given functionality.

CCS CONCEPTS

• **Hardware** → **Quantum dots and cellular automata**; **Combinational synthesis**; **Technology-mapping**; **Physical synthesis**; **Placement**; **Wire routing**.

1 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, by this, contribute up to 23 % of the globally released greenhouse gases [2]. 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) [1] is a class of emerging post-CMOS technologies that comprises implementations such as *Quantum-dot Cellular Automata* (QCA), *Nanomagnet Logic* (NML), and *Silicon Dangling Bonds* (SiDB) amongst others that all share similar properties. 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 [25, 27, 28]. This promising outlook motivated explorations into its feasibility, which led to several suitable contributions to the physical implementation of FCN technologies—in particular in the last couple of years [7, 9, 15, 18].

Accordingly, there is an increasing interest in automatic methods for the design and synthesis of corresponding FCN circuits. Unfortunately, realizing a given functionality in an FCN technology and particularly the placement of gates and routing of wires in an FCN circuit differs significantly from the same task in conventional CMOS. In fact, because of the much tighter domain-specific physical constraints in FCN, *clocking* is a critical factor in sequential and combinational circuits alike because it directs the data flow and, at the same time, controls information synchronization. Among other obstacles, corresponding *clocking constraints* are one limiting factor of the $\mathcal{NP}$-hard design automation for FCN circuitry [31].

Thus far, all available methods, such as [11, 24, 30, 34], address this problem by an application of two consecutive stages, namely

(1) **Logic Synthesis**, i. e., realizing the desired functionality in terms of a classical logic network using conventional synthesis approaches, and
(2) **Physical Design**, i. e., mapping the resulting network onto an FCN layout satisfying the respective physical constraints (e. g., clocking, timing, etc.).

This state-of-the-art flow comes with severe drawbacks. Existing conventional logic synthesis approaches are, by no means, suited for subsequent FCN physical design. They optimize with respect to abstract or conventional cost metrics such as number of gates, area, depth, etc. which, however, do not apply for FCN circuits in the same way. Moreover, adjustments to cost metrics of existing logic

[*]Also with Software Competence Center Hagenberg GmbH (SCCH).
[†]Also with DFKI GmbH, Cyber Physical Systems.
[‡]Also with DFKI GmbH, Cyber Physical Systems.

synthesis solutions, which may be able to "factor in" FCN design constraints, are not possible, since the final costs of an FCN circuit almost exclusively rely on the resulting layout. Consequently, all existing solutions for automatic FCN design, which work with a classical and FCN-oblivious logic network as an intermediate result, frequently lead to substantial quality loss or completely impractical results (cf. Section 2.2).

In this paper, we address this problem by introducing a one-pass synthesis scheme for FCN circuits, which conducts logic synthesis *and* physical design in a *single* run. This makes the problem substantially harder because, suddenly, the search spaces of both, the synthesis step and the physical design step, need to be considered. To cope with the combined complexity of two $\mathcal{NP}$-complete problems [31], we utilize the deductive power of satisfiability solvers. At the same time, our approach also offers such a high degree of flexibility that it can be parameterized with various design features such as wire-crossings, gate libraries, and clocking schemes to restrict or loosen certain constraints and, thereby, leverage the required runtime overhead.

For the first time, this allows for the automatic realization of FCN circuits that do not depend on classical logic networks (generated without any FCN-context), but are optimized towards FCN constraints from the beginning. Experimental evaluations confirm the resulting benefits: although the required effort for covering the search space is significant, the first truly minimal FCN circuit realizations in terms of area could be generated for given Boolean functions. While the exponential nature of this problem makes the proposed approach applicable to rather small circuits only, it can be used to assemble a design library of recurring functions, e. g., by exploiting NPN classes, which can then be utilized as building-blocks in hierarchical design processes.

To keep this paper self-contained, Section 2 provides background on the FCN concept and introduces the considered design problem in detail. Section 3 presents our novel SAT-based one-pass approach for FCN. Section 4 provides a detailed experimental evaluation. Finally, Section 5 concludes the paper.

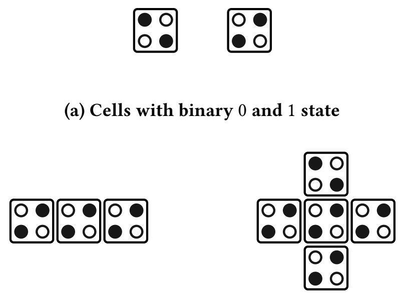
2 PRELIMINARIES

In this section, we briefly review the FCN concept and discuss its (physical) design problem.

2.1 Field-coupled Nanocomputing (FCN)

Some prominent representatives of the FCN class are *Quantum-dot Cellular Automata* (QCA, [19, 20]), *Nanomagnet Logic* (NML, [14]), and *Silicon Dangling Bonds* (SiDB, [7, 15, 37]). Even though their physical properties differ and most of them are further 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. 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


(a) Cells with binary 0 and 1 state

(b) Wire segment

(c) Majority gate

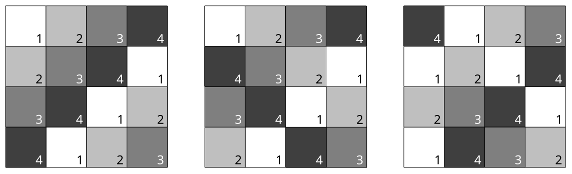
Figure 1: Elementary QCA cell devices

an electric charge each and that are arranged at the corners (and the center) of a square [21]. Adding two free and mobile electrons into each cell (that are able to tunnel between adjacent dots) 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. Figure 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. 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.

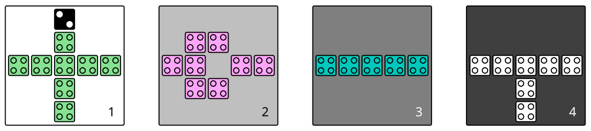
Example 1. *Figure 1b shows how to arrange multiple cells in a row to build a wire segment. It transmits binary information from left to right or vice versa as the same field interactions happen across the cell boundaries and thereby affect the polarization of adjacent cells. This formation is extended in Figure 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. Also, inverters and fan-outs can be build easily with only a few cells and, therefore, are mostly also considered as elementary operations.*

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, stalling, and resetting of their states. In case of QCA, an external electrical field-generator acts as the clock and 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



(a) 2DDWave [29] (b) USE [8] (c) RES [12]

Figure 2: Clocking schemes for FCN circuit layouts



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

Figure 3: Tiles in QCA ONE implementation

to 4 are applied, whereby each clock controls a selected adjacent set of cells and is shifted by 90° compared to its predecessor. Thereby, 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.

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 [6, 9]. 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 2. *Figure 2 depicts cutouts of size 4×4 tiles of three common clocking schemes that are mostly used for QCA-like technologies. They can be extrapolated seamlessly in all directions, but provide different assets and drawbacks, e. g., whether they support 3-ary gates or feedback loops.*

Several gate libraries have been proposed for various FCN technologies. In this paper, we utilize the *QCA ONE* library [22] 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 Figure 3.

2.2 FCN Design and Resulting Problems

With the components we reviewed, it is possible to design FCN circuit layouts by positioning the elementary gates of a predefined gate library on a clocking scheme and connecting them by wire segments. The default approach to this physical design of FCN is placement & routing of readily synthesized logic networks, e. g., *And-inverter graphs* (AIGs) or *Majority-inverter graphs* (MIGs). 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.

More precisely, there are manifold difficulties: In contrast to classical CMOS, each wire segment in FCN causes the same costs as a gate and takes up the same amount of space on the layout. Since FCN is a semi-planar technology class, wires may block the routing of other wires, which can only be compensated with long detours or expensive wire crossings. Finally, FCN clocking determines not only the direction of information flow, but also data synchronization. In each phase, each signal is propagated by exactly one tile. To prevent signals from arriving desynchronized at multi-input gates, it must be ensured during the physical design process that all paths in the layout pass through the same number of tiles.

We illustrate these shortcomings with a simplified example.

Example 3. *Consider the logic network given in Figure 4a. It consists of four operations $o_1, \ldots, o_4$ and four connections. This network has been obtained by a conventional logic synthesis procedure, but shall be placed and routed onto an FCN layout in the following.*

In Figure 4b, a possible placement of the operations onto layout tiles is given. It allows to realize the connections (o_1, o_2), (o_2, o_3), and (o_2, o_4) by direct adjacencies without the need for extra wire segments. For the connection (o_3, o_4), however, one wire segment is needed (drawn in white). This is a problem, because there is no legal clock number that could be assigned to the respective tile so that the information could be passed from o_3 to o_4. Moreover, assuming that such a clock number was available, the two paths leading to operation o_4 would be of different length, which would desynchronize the circuit. This can be viewed as either a clocking or a routing failure.

To resolve the conflict, the operation o_4 has to be relocated and its connections rerouted. The resulting conflict-free layout can be seen in Figure 4c. Thereby, the area costs increased from 6 to 9 tiles. The connections (o_3, o_4) and (o_2, o_4) caused extra 2 and 3 wires respectively, which led to a final critical path of 6 tiles. Neither of these costs could have been predicted from considering the logic network alone.

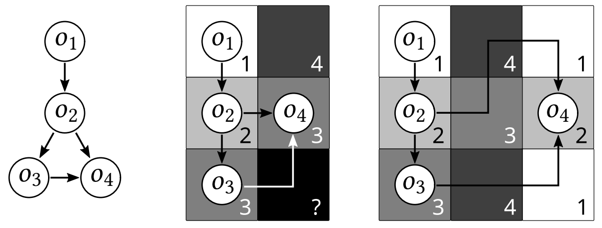
As we have shown, the established two-stage process for FCN design has major shortcomings that can result in extremely large layouts. In the worst case, certain logic networks cannot be implemented as FCN circuitry at all. In the following, we present a novel one-pass synthesis approach that bypasses the weaknesses of previous algorithms by synthesizing required operations directly onto a layout, while taking all technological constraints into account.

3 ONE-PASS DESIGN OF FCN CIRCUITS

In this section, we propose a solution to the issues arising from the two-step FCN physical design process reviewed above.

3.1 General Idea

Given an empty layout of fixed size, an arbitrary clocking scheme, and a multi-output truth table as a functional specification, we propose a one-pass synthesis algorithm that represents the *entire* design problem (i. e., the logic synthesis and the physical design combined) as a *satisfiability problem* (SAT) [5]. By this, all possible FCN circuits on the given clocking scheme are symbolically described. A SAT solver then enumerates all such designs and determines the one that satisfies the specification and all technology constraints.



(a) Synthesized logic network **(b) Placement has led to a routing or clocking failure** **(c) Rerouting had to be applied to resolve the conflict**

Figure 4: Shortcomings of FCN placement & routing

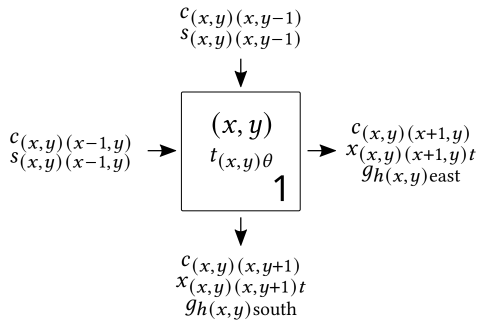


Figure 5: Variables for a tile at position (x, y)

In the literature, many important research results have already been achieved with such schemes (e. g., cf. [13, 16, 23, 26, 36, 38]). However, none of them investigated the potential for FCN and the corresponding physical placement & routing constraints.

In order to describe the resulting solution, in the following section, we first introduce the sets of Boolean variables utilized to symbolically describe all possible FCN circuits (including those containing invalid solutions), and then introduce constraints formulated in propositional logic to restrict the possible solutions to the ones that are valid and which satisfy the logical specification and all technological constraints.

3.2 Formulation as a SAT Problem

In the considered one-pass synthesis problem, we are given an empty layout of size $W \times H$, an arbitrary clocking scheme, and a multi-output Boolean function $f = (f_1, \ldots, f_m) : \mathbb{B}^n \to \mathbb{B}^m$ over n variables $x_1, \ldots, x_n$ to be synthesized. We identify each tile in the layout by its coordinates (x, y). The coordinate $(0, 0)$ defines the top-left corner of the layout, i. e., we have $0 \leq x < W$ and $0 \leq y < H$. The gate types supported by tile (x, y) depend on the local geometry. The same holds for its potential fan-in/fan-out connections. Each tile can be expressed by the following entities that can be considered as sets:

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

Note that we have $\mathcal{I}_{(x,y)} \cup \Omega_{(x,y)} \subseteq \{north, east, south, west\}$. In the following, we utilize these entities to introduce the necessary Boolean variables for our SAT formulation, which is inspired by the *SSV encoding* described in [13]. SSV itself is based on earlier formulations by Eén and Knuth [10, 17]. Since it is originally used for the synthesis of homogeneous normal k-input logic networks, our encoding has to differ substantially in various ways. However, the correctness of our encoding follows directly from SSV. A more formal justification can be found in [13].

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

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

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

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

Example 4. *Consider the tile given in Figure 5 that has been cut out from a layout that utilizes the 2DDWave clocking scheme. Let its clock number be* 1 *and its position be* (x, y)*. The arrows indicate possible incoming and outgoing data flow directions. Assigned to the tile and each cardinal direction, the respectively introduced Boolean variables are shown. Those are the variables defining the gate type of the tile, connection variables for each direction, fan-in variables for the incoming directions, and truth table variables as well as primary output pin variables for the outgoing directions.*

Passing these variables to a SAT solver, arbitrary assignments representing different FCN circuits are obtained. For example, assigning $t_{(x,y)\theta} = 1$ *would represent that tile* (x, y) *is occupied by a gate of type* θ*, while assigning* $g_{h(x,y)east} = 1$ *would represent that tile* (x, y) *computes the primary output* f_h *in eastern direction.*

Having these variables, it is left to add constraints in propositional logic to ensure that (1) the synthesized FCN circuit layout computes the given function specification, and (2) it additionally satisfies all physical design constraints imposed by the technology and the given clocking scheme.

For each (x, y), $\tilde{\imath} \in \widetilde{\mathcal{I}}_{(x,y)}$, $\omega \in \Omega_{(x,y)}$, and $a, b, c \in \mathbb{B}$, the following constraints ensure that the FCN circuit layout simulates the correct sub-function at each tile:

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

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

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

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

Note that a, b, and c are constants, which are used to set the proper variable polarities. Furthermore, note that we have to support non-symmetric functions such as wire crossings, so we may have $\theta_\omega(a, b) \neq \theta_\omega(b, a)$. Hence, it is important that different variable orderings are encoded as separate fan-in options $\tilde{\imath}$.

Let $(b_1, \ldots, b_n)_2$ be the binary encoding of a truth table at 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 permit cycles in the information flow on the layout (cf. Figure 2). Since we are considering combinational functions only, we must add clauses to ensure that the final design is acyclic. We achieve this by first detecting all possible cycles permitted by the clocking scheme in the layout and then using the *connection variables* $c_{(x,y)\tilde{\omega}}$ to prevent them from being part of valid solutions. Let the path $((x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n), (x_0, y_0))$ be a cycle. We add the clause $\bigvee_{i=0}^{n} \bar{c}_{(x_i,y_i)(x_j,y_j)}$, where $j = (i{+}1 \bmod n)$. 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 $\tilde{\imath} \in \widetilde{\Omega}_{\tilde{\omega}}$ we add $(\bar{s}_{\tilde{\omega}\tilde{\imath}} \vee c_{(x,y)\tilde{\omega}})$ if $(x, y) \in \tilde{\imath}$.

We have now described the main clauses. We use some additional clauses to satisfy various cardinality constraints. These include constraints to ensure that primary inputs have at most one 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).

4 EXPERIMENTAL EVALUATION

In this section, we summarize the experiments we conducted in order to evaluate the proposed one-pass synthesis scheme. To this end, we first discuss our implementation and go over the specifications of the system used in the following evaluations. In Section 4.2, we then investigate the impact that the proposed one-pass approach has on performance by comparing it against a state-of-the-art exact placement & routing algorithm for QCA that works on conventionally synthesized logic networks [30]. In Section 4.3, we showcase the benefits of the proposed approach by utilizing it to generate a comprehensive design library of optimal building blocks on different clocking schemes by using NPN classes. Finally, in Section 4.4, we discuss further benefits we see in the proposed one-pass synthesis scheme.

Table 1: Comparison against exact placement & routing [30]

	Exact P&R [30]			Proposed approach		
Function	A	CP	t in s	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	7	42
c17	30	16	56	18	13	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 of 2 hours reached

4.1 Experimental Setup

We implemented the proposed one-pass synthesis algorithm as a Python library called *Mugen*[1] and integrated it into the open-source FCN design framework *fiction* [33].[2] 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 was Glucose 3.0 [3]. The correctness of the results has been checked using [32].[3]

4.2 Impact on Performance

The proposed one-pass synthesis scheme for FCN circuits conducts logic synthesis and physical design together in a single run. This makes the problem substantially harder because, suddenly, the search spaces of both, the synthesis step and the physical design step, need to be considered. To evaluate the resulting impact on the performance, we conducted an experimental comparison against an existing placement & routing algorithm for QCA circuit layouts [30]. That approach is also exact, but only performs the physical design step of a given conventionally pre-synthesized logic network onto a given clocking scheme. Consequently, when directly comparing a layout obtained by exact placement & routing of a non-optimized logic network to a layout obtained by the proposed one-pass synthesis scheme, the latter must never be worse in terms of area.

Table 1 compares their obtained results against the ones we could generate using the 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, respectively. In [30], 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 1. Note that the critical path length was not an optimization target in either algorithm, but is listed for the sake of completeness

[1]The source code is publicly available at https://github.com/whaaswijk/mugen.
[2]The source code is publicly available at https://github.com/marcelwa/fiction.
[3]The obtained designs and physical simulation files for QCADesigner [35] are publicly available at https://github.com/marcelwa/FCN-Design-Library.

and because the authors listed it in [30] 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.

As postulated, the one-pass synthesis indeed has a severe effect on the run-time. It is substantially slower than placement & routing and times out on three functions. However, this was expected as one-pass synthesis eventually has to cope with the combined complexity of two $\mathcal{NP}$-complete problems. 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 a common assumption in the FCN community, namely, that overhead due to wire routing could be reduced by tailoring the logic network to the clocking scheme. Moreover, for the first time, truly optimal FCN designs for a given function to be synthesized could be generated due to the joint consideration of logical synthesis *and* physical design. Thus far, previously proposed approaches (such as [30]) could only guarantee optimality of a physical design with respect to a logic network synthesized before (but not with respect to the actual function).

4.3 Generating a Design Library

As revealed in the previous section, the runtime overhead of the proposed one-pass approach is substantial. However, there are still applications where this exact one-pass synthesis can be of great benefit. Since it enables absolute minimality of circuit layouts, it can be used to assemble a comprehensive design library of recurring functions, which can be used as optimal building-blocks in hierarchical design processes.

Exemplary, we generated such a design library covering the entire Boolean function space in 3 variables by utilizing *NPN classification*, which determines if some single-output Boolean functions are identical under negation and permutation 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 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 exemplary clocking schemes 2DDWave, USE, and RES that are shown in Figure 2. 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,[4] 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 thus 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.

[4]The obtained design files and circuit images are publicly available at https://github.com/marcelwa/FCN-Design-Library.

Table 2: Area results for all 3-input NPN classes

	2DDWave			USE			RES		
NPN	A	#G	#W	A	#G	#W	A	#G	#W
`0x00`	8	5	1	8	5	3	8	8	0
`0x01`	10	7	1	10	7	1	8	7	1
`0x03`	8	6	1	6	5	0	8	5	3
`0x06`	18	13	4	20	13	8	20	13	5
`0x07`	10	8	0	10	8	0	8	7	1
`0x0f`	4	3	0	4	3	0	4	3	0
`0x16`	27	20	6	32	18	14	32	20	10
`0x17`	20	15	2	24	15	10	9	6	1
`0x18`	24	16	7	28	15	14	30	18	8
`0x19`	18	15	1	20	15	6	20	14	7
`0x1b`	15	12	2	16	11	5	15	10	6
`0x1e`	18	12	3	24	16	9	24	12	10
`0x3c`	15	11	2	20	11	9	16	9	8
`0x69`	32	18	6	32	18	16	32	21	12
total	227	161	36	254	160	95	234	153	72

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)

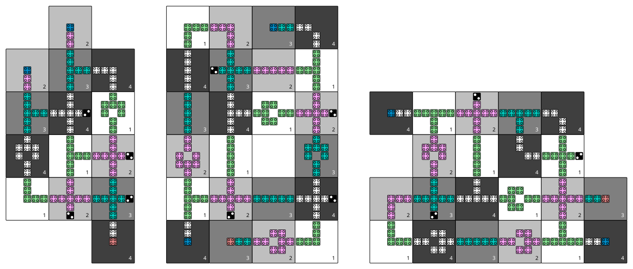

(a) 2DDWave (b) USE (c) RES

Figure 6: Crossing-free layouts of the 2-input XOR function

The results of our experiments are summarized in Table 2. 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 minimum 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.

4.4 Further Benefits of the One-Pass Scheme

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. The proposed approach enabled us to settle this question. Figure 6 depicts crossing-free XOR implementations in the QCA ONE gate library for all three clocking schemes investigated in this paper.[5]

Furthermore, as we already mentioned, placement & routing generate an overhead in terms of circuit area, which is used for wire routing. This overhead occurs because logic networks, that serve as inputs, were not synthesized with FCN routability in mind. This work, henceforth, provides weak evidence that even in *exact* placement & routing techniques there exists an overhead that can be eliminated using the proposed approach. These propositions, thus, further confirm the benefits of the scheme demonstrated in this paper.

5 CONCLUSIONS

In this paper, we presented a SAT-based approach for exact one-pass synthesis of *Field-coupled Nanocomputing* (FCN) technologies, which we made publicly available. By this, we combined synthesis and physical design to a single run—allowing, for the first time, the automatic realization of FCN circuits that do not depend on classical logic networks (generated by conventional synthesis approaches without any FCN-context). Since this obviously has a substantial effect on the run-time performance (caused by the fact that one-pass synthesis eventually has to cope with the combined complexity of two $\mathcal{NP}$-complete problems), the proposed scheme is only applicable to small circuits thus far. However, we demonstrated that this already allows to generate truly optimal results, the generation of optimal design libraries, the exact evaluation of arbitrary clocking schemes, and the consideration of problems and properties of FCN designs that could not be addressed yet. Motivated by these results, we believe this work provides the basis for a further consideration of one-pass synthesis for FCN in the future.

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.

REFERENCES

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

[2] A. S. G. Andrae and T. Edler. 2015. On Global Electricity Usage of Communication Technology: Trends to 2030. *Challenges* 6, 1 (2015), 117–157.

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

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

[5]Several examples of XOR realizations that require less area can be found in the literature, e. g., [4]. However, these were not synthesized from existing well-proven gates, but are cell-based and hand-crafted. Such solutions do not guarantee physical correctness and fabricatibility out-of-the-box and need to be further verified in lab tests or quantum simulations [9].

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

[6] E. Blair and C. Lent. 2018. Clock Topologies for Molecular Quantum-Dot Cellular Automata. *Journal of Low Power Electronics and Applications* 8, 3 (2018).

[7] S. Bohloul, Q. Shi, R. A. Wolkow, and H. Guo. 2017. Quantum Transport in Gated Dangling-Bond Atomic Wires. *Nano Letters* 17, 1 (2017), 322–327.

[8] C. A. T. Campos et al. 2016. USE: A Universal, Scalable, and Efficient Clocking Scheme for QCA. *TCAD* 35, 3 (2016), 513–517.

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

[10] N. Eén. 2007. Practical SAT: a tutorial on applied satisfiability solving. (2007). http://minisat.se/Papers.html FMCAD.

[11] G. Fontes et al. 2018. Placement and Routing by Overlapping and Merging QCA Gates. In *ISCAS.* 1–5.

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

[13] W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli. 2020. SAT-Based Exact Synthesis: Encodings, Topology Families, and Parallelism. *IEEE Trans. on CAD of Integrated Circuits and Systems* 39, 4 (2020), 871–884.

[14] X. K. Hu et al. 2015. Edge-Mode Resonance-Assisted Switching of Nanomagnet Logic Elements. *IEEE Trans. Magn.* 51, 11 (2015), 1–4.

[15] T. R. Huff, H. Labidi, et al. 2017. Atomic White-Out: Enabling Atomic Circuitry through Mechanically Induced Bonding of Single Hydrogen Atoms to a Silicon Surface. *ACS Nano* 11, 9 (2017), 8636–8642.

[16] O. Keszöcze, R. Wille, and R. Drechsler. 2014. Exact Routing for Digital Microfluidic Biochips with Temporary Blockages. In *ICCAD.* 599–606.

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

[18] C. S. Lent et al. 2016. Molecular Cellular Networks: A non von Neumann Architecture for Molecular Electronics. In *ICRC.* 1–7.

[19] C. S. Lent, B. Isaksen, and M. Lieberman. 2003. Molecular Quantum-dot Cellular Automata. *Journal of the American Chemical Society* 125, 4 (2003), 1056–1063.

[20] C. S. Lent and P. D. Tougaw. 1997. A Device Architecture for Computing with Quantum Dots. *Proc. IEEE* 85, 4 (1997), 541–557.

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

[22] D. A. Reis et al. 2016. A Methodology for Standard Cell Design for QCA. In *ISCAS.* 2114–2117.

[23] H. Riener, W. Haaswijk, A. Mishchenko, G. De Micheli, and M. Soeken. 2019. On-the-fly and DAG-aware: Rewriting Boolean Networks with Exact Synthesis. In *Design, Automation and Test in Europe.* 1649–1654.

[24] F. Riente et al. 2017. ToPoliNano: A CAD Tool for Nano Magnetic Logic. *TCAD* 36, 7 (2017), 1061–1074.

[25] F. Sill Torres, R. Wille, P. Niemann, and R. Drechsler. 2018. An Energy-Aware Model for the Logic Synthesis of Quantum-Dot Cellular Automata. *TCAD* 37, 12 (2018), 3031–3041.

[26] M. Soeken, W. Haaswijk, E. Testa, A. Mishchenko, L. Amarù, R. K. Brayton, and G. De Micheli. 2018. Practical Exact Synthesis. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE).* 309–314. https://doi.org/10.23919/DATE.2018.8342027

[27] J. Timler and C. S. Lent. 2002. Power Gain and Dissipation in Quantum-dot Cellular Automata. *J. Appl. Phys.* 91, 2 (2002), 823–831.

[28] F. Sill Torres, P. Niemann, R. Wille, and R. Drechsler. 2020. Near Zero-Energy Computation Using Quantum-dot Cellular Automata. In *JETC.*

[29] V. Vankamamidi, M. Ottavi, and F. Lombardi. 2006. Clocking and Cell Placement for QCA. In *IEEE-NANO*, Vol. 1. 343–346.

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

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

[32] M. Walter, R. Wille, F. Sill Torres, and R. Drechsler. 2020. Verification for Field-coupled Nanocomputing Circuits. In *DAC.*

[33] M. Walter, R. Wille, F. Sill Torres, D. Große, and R. Drechsler. 2019. fiction: An Open Source Framework for the Design of Field-coupled Nanocomputing Circuits. arXiv:1905.02477

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

[35] K. Walus, T. J. Dysart, G. A. Jullien, and R. A. Budiman. 2004. QCADesigner: A Rapid Design and Simulation Tool for Quantum-dot Cellular Automata. *TNANO* 3, 1 (2004), 26–31.

[36] R. Wille and D. Große. 2007. Fast Exact Toffoli Network Synthesis of Reversible Logic. In *ICCAD.* 60–64.

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

[38] A. Zulehner and R. Wille. 2018. One-pass Design of Reversible Circuits: Combining Embedding and Synthesis for Reversible Logic. *TCAD* 37, 5 (2018), 996–1008.