Verification for Field-coupled Nanocomputing Circuits

Marcel Walter[1] Robert Wille[2,3] Frank Sill Torres[4] Daniel Große[1,3] Rolf Drechsler[1,3]

[1]Group of Computer Architecture, University of Bremen, Germany
[2]Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
[3]Cyber Physical Systems, DFKI GmbH, Bremen, Germany
[4]Department for the Resilience of Maritime Systems, DLR, Bremerhaven, Germany

{m_walter, grosse, drechsler}@uni-bremen.de, robert.wille@jku.at, frank.silltorres@dlr.de

Abstract—**With the decline of Moore's Law, several post-CMOS technologies are currently under heavy consideration. Promising candidates can be found in the class of *Field-coupled Nanocomputing* (FCN) devices as they allow for highest processing performance with tremendously low energy dissipation. With upcoming design automation in this domain, the need for formal verification approaches arises. Unfortunately, FCN circuits come with certain domain-specific properties that render conventional methods for the verification non-applicable. In this paper, we investigate this issue and propose a verification approach for FCN circuits that addresses this problem. For the first time, this provides researchers and engineers with an automatic method that allows them to check whether an obtained FCN circuit design indeed implements the given/desired function. A prototype implementation demonstrates the applicability of the proposed approach.**

I. Introduction

The tremendous advancement of the capabilities of digital systems over the last decades is strongly related to the miniaturization of the transistor sizes, which enabled *cramming more components onto integrated circuits* following Moore's prediction from 1965 [1]. However, reducing the transistor size no longer yields the improvements it used to. In contrast to what one would expect, main limiting factors are not restrictions due to fabrication constraints or parasitic effects of current technologies, but the high power density of integrated circuits based on today's conventional technologies. This restraint led to a stagnation of the clock speeds in the beginning of this millennium and an increasing number of the so-called *dark silicon*, i. e. regions of a chip that must be powered off to avoid overheating [2]. This problem is worsening with the emergence of new types of applications that have to compute with massive amount of data, such as deep learning or high-resolution image processing. On the end of the scale, novel embedded systems intended for ubiquitous computing, e. g. Internet-of-Things related applications or portable biomedical devices, are strongly restricted by their energy supply, i. e. batteries or energy harvesting solutions.

Consequently, there is an increasing interest in alternative technologies that enable fast computations with considerably lower energy dissipation compared to the state of the art. Among the several candidates, *Field-coupled Nanocomputing* (FCN) [3] is a class of emerging technologies that 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 a number of magnitudes below current CMOS technologies [4], [5], [6]. This promising outlook motivated explorations on its feasibility which led to several suitable contributions to the physical implementation of FCN technology, many of them very recently (i. e. in the last 3–4 years) [7], [8].

Based on these promising physical implementations, several researchers started to consider how to efficiently design corresponding FCN circuits. While initial solutions have been obtained manually [9], also automatic solutions, e. g. for physical design, are available in the meantime [10], [11]. Even though the underlying problem is $\mathcal{NP}$-complete [12], recently, techniques have been proposed, which can handle large designs [13], [14], [15]. Together with established logic synthesis expertise from the conventional domain, this is emerging towards a design automation flow which yields a (conventional) logic network first, followed by a mapping towards an FCN circuit by a physical design step.

With this development, also the need for formal verification approaches arises. In fact, with increasing sizes of the FCN circuits that can be designed, it cannot be checked manually anymore whether e. g. the resulting realization indeed implements the initially given/desired function. But unfortunately, FCN circuits come with certain domain-specific properties, in particular synchronization issues of the signals, that render conventional methods for the verification of circuits non-applicable. In this paper, we discuss this problem and show how the actual design of an FCN circuit implies domain-specific physical restrictions e. g. signal synchronization constraints. We show how violating them can turn the circuit into conducting an unintended logic function or can impose an undefined behavior.

Based on these discussions, we eventually propose a formal verification technique that enhances conventional methods for circuit verification to make them applicable for FCN circuits as well. By this, we provide the basis to use decades of research on the verification of conventional circuitry for the verification of FCN circuits. As a representative verification problem, we thereby consider equivalence checking, i. e. the task of proving whether two different circuit representations implement the same function or not.

The rest of the paper is structured as follows. To keep this work self-contained, Section II introduces basic FCN elements and explains how to use them in order to design circuits. In Section III, we discuss, how signal synchronization makes FCN circuits so different from conventional CMOS ones and point out the implications to formal verification. In Section IV, we present a novel approach to tackle verification in FCN technologies and demonstrate its applicability in Section V. Section VI concludes the paper.

II. Background

This section provides the background on *Field-Coupled Nanotechnologies* (FCN) as well as the currently considered design approaches. By this, it provides the basis for the remainder of this work. FCN manifests in several fashions such as *atomic Quantum-dot Cellular Automata* (aQCA, [7], [16]), *molecular Quantum-dot Cellular Automata* (mQCA, [8]), or *Nanomagnet Logic* (NML, [17]). We review aspects of QCA-like and NML technologies in the following section and then focus on QCA as a running example for all further illustrations in this paper.

A. Field-coupled Nanotechnologies

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 *quantum dots* which are able to confine an electric charge and are arranged at the corners of a square [18]. Adding two free and mobile electrons into

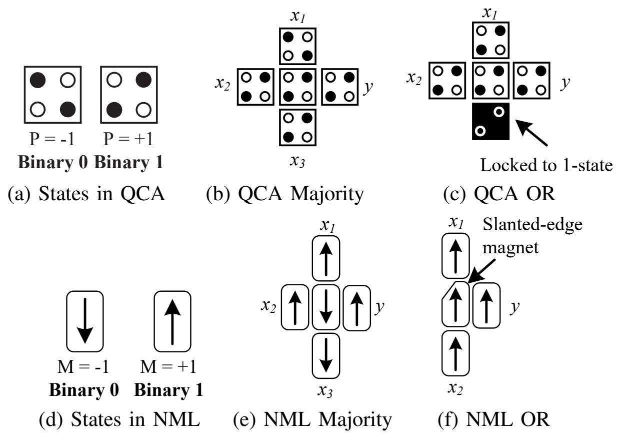


Fig. 1: FCN states and basic gates

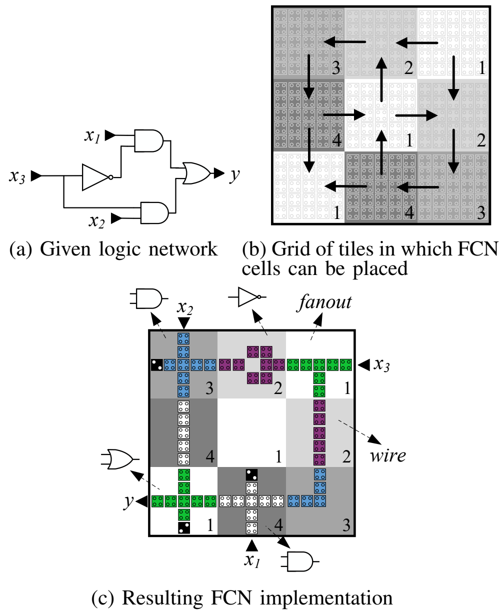
each cell, that are able to tunnel between adjacent dots, yields a stable state due to Coulomb interaction (note that tunneling to the outside of the cell is prevented by a potential barrier). Then, because of the mutual repulsion, the two electrons tend to locate themselves at opposite corners of the cell – eventually leading to two possible *cell polarizations*, namely $P = -1$ and $P = +1$ which can be defined as binary 0 and binary 1 (see Fig. 1a). In contrast, NML cells are based on single domain nanomagnets that can assume only two stable magnetization states, namely $M = -1$ and $M = +1$ which also can be used to represent the binary values 0 and 1 (see Fig. 1d). When composing several FCN cells next to each other, field interactions cause the polarization or magnetization of one cell to influence the polarization or magnetization of the others. This allows to implement Boolean functions such as AND, OR, NOT, Majority, etc.

Example 1. *Fig. 1a and 1d show the two cell polarizations and the two stable magnetization states which are used to represent the binary values 0 and 1 in QCA and NML, respectively. Furthermore, Fig. 1b shows for QCA how those cells can be combined to implement a Majority function. Here, the output y evaluates to the binary state 1, if the majority of the input values x_1, x_2, x_3 is assigned 1; otherwise, y evaluates to 0. Locking one of the three inputs to the 0-state turns this cell into an AND gate, while locking one of the inputs to the 1-state results into an OR gate as shown in Fig. 1c. In a similar fashion, those functions can be implemented in NML; as shown in Fig. 1e for the Majority function and in Fig. 1f for the OR gate. In the latter, a so-called* slanted-edge magnet *[19] is applied that give preference to a magnetization.*

B. Designing FCN Circuits

Following the concepts reviewed above, basically any functionality can be designed using FCN circuits. To this end, a so-called *tile-based design* scheme became popular [20], [21]. Here, FCN cells are grouped into *tiles* which can be used to implement certain elementary logic operations such as the Majority or the OR function covered before. A tile can be referred to by its coordinates in the grid as (x, y) assuming that the top-left tile is at position $(0, 0)$.

While a tile-based design scheme obviously restricts the flexibility in the design of FCN circuits (eventually, only logic blocks which fit into the respective tiles can be employed), it significantly simplifies the design process. In fact, tile-based design also allows for the utilization of conventional design methods. More precisely, in order to implement a desired functionality, a logic network, e. g., an *And-inverter Graph* (AIG) or a *Majority-inverter Graph* (MIG) (generated by established conventional tools and methods such as in [22], [23]) can be created first and, afterwards, each gate of this logic network



x_1
x_3
x_2
y
(a) Given logic network
3
2
1
4
1
2
1
4
3
(b) Grid of tiles in which FCN cells can be placed
x_2
fanout
x_3
wire
y
x_1
(c) Resulting FCN implementation

Fig. 2: Tile-based design steps

"only" has to be mapped and connected to a corresponding elementary FCN tile. An example illustrates the idea.[1]

Example 2. *Assume, a functionality should be implemented which is described by the logic network shown in Fig. 2a, e. g. generated using a conventional synthesis algorithm. Fig. 2b shows possible groupings of FCN cells into a grid of tiles. Each square shape represents a tile, in which FCN cells implementing a gate may be placed (possible cell positions are hinted in each tile). Using this grid, the given logic network can be mapped to a corresponding layout as shown in Fig. 2c.*

However, due to the metastability of the cells, and in order to control the data flow within a design, FCN cells and, hence, also FCN tiles cannot be connected arbitrarily. In fact, all cells, and thus also the tiles, must be associated to an external clock that controls the initialization, holding, and resetting of the states of the cells. In case of QCA, an external electric clock controls the tunneling within the cells, while in NML a magnetic clock regulates the switching ability of the nanomagnets. Depending on the technology, each cell changes during a complete clock cycle between four (QCA) or three (NML) different phases, i. e. a *switch*, a *hold*, a *reset* and a *neutral* phase (the latter only in case of QCA). In case of four phases, usually four external clocks numbered from 1 to 4 are applied, whereby each clock controls a selected adjacent set of cells (i. e. a tile). 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. Another example illustrates the issue.

Example 3. *Consider again the layout discussed in Example 2. Here, Fig. 2b does not only show the grouping of FCN cells into tiles, but also the association of each tile to one of the four clock signals (denoted by the numbers in the bottom-right corners and a corresponding gray scale). The arrows indicate the possible data flow directions. For example, a logic gate mapped to tile $(1, 0)$ (controlled by clock 2) must only receive its inputs from tiles $(2, 0)$ and $(1, 1)$ which are controlled by clock 1. Similarly, it can propagate its outputs to gates in the top-left tile that is controlled by clock 3. The FCN circuit shown before in Fig. 2c already follows this clocking scheme.*

[1]Note that even if we will consistently use the tile-based gate library proposed in [24] in this work, all concepts and also our contributions are independent of the actual physical and structural implementation.

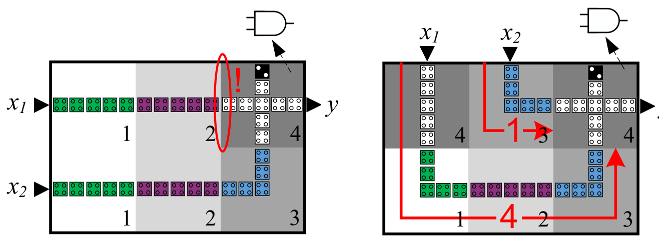


Fig. 3: Synchronization issues in FCN technologies

Overall, following the tile-based design scheme, FCN circuits may be implemented as a simple 1:1 mapping of the given logic network to a corresponding FCN layout. However, additionally considering the required clocking leads to a non-trivial design task which recently has also been proven to be $\mathcal{NP}$-complete [12]. Accordingly, researchers have considered intensely how to best tackle this problem – both, in an exact fashion but also heuristically. This resulted in several methods which have been proposed for this purpose in the last couple of years [10], [11], [13], [14].[2]

III. Verification of FCN Circuits

As in conventional CMOS design, whether an FCN circuit indeed implements the desired functionality is of utmost importance. Here, in particular, equivalence checking plays an important part. Considering the tile-based design scheme as reviewed in the previous section, this particularly becomes relevant in the "transition" from the conventional design flow (delivering the logic network) to the FCN design (yielding the FCN circuit). At a first glance, a corresponding equivalence checking is straight-forward: two functional descriptions (the conventional logic network and the FCN circuit) are provided and need to be checked for equivalence. To this end, numerous (conventional) solutions are available (e. g. [25], [26], [27], [23], [28]).

However, similar to the design of FCN circuits, the required clocking additionally hardens the problem and, in fact, yields a situation where conventional methods cannot be applied directly to the FCN domain. More precisely, while conventional (combinatorial) circuits propagate data "instantaneously" through the circuit (at least when considered functionally for equivalence checking on a logic level), FCN circuits can propagate data only in direction of an adequate sequence of clock signals (as illustrated before in Example 3). This leads to two domain-specific properties of FCN circuits that need to be considered during any verification task:

- **Local synchronization**, i. e., the fact that data is propagated in a way where tiles controlled by a clock i are followed by tiles controlled by a clock $(i+1) \bmod C$ (with C being the number of different clocks required by the technology; i. e., $C = 4$ for QCA and $C = 3$ for NML). This is a characteristic all FCN circuits must satisfy, i. e., verification methods have to check for this independently of whether two circuits are functionally equivalent [3].
- **Global synchronization**, i. e., the fact that the number of passed tiles for any two signals traveling from primary inputs to the same gate have to be equal. Otherwise, data will arrive in a desynchronized fashion – leading to different or even undefined behavior that obviously affects the function of a circuit and, hence, whether it is functionally equivalent to another one.

[2]Note that a recent work proposed in [15] also tries to circumvent this problem by exploiting additional hardware to handle the clocking constraints in a post-processing step.

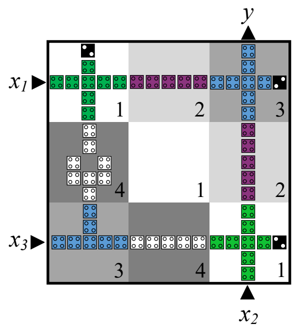


Fig. 4: FCN implementation violating global synchronization

Example 4. *Consider Fig. 3a which depicts a QCA circuit realized with 3×2 tiles. At first glance, one could assume that the circuit implements the AND function $y = x_1 \wedge x_2$, because the gate assigned to the tile at position $(2,0)$ is an AND gate while all the other tiles have wire segments assigned. But since the tile at position $(0,1)$ is controlled by clock 2 and the neighboring tile at position $(0,2)$ by clock 4, no data flow from the former to the latter is possible – the local synchronization is not satisfied.*

Consider now Fig. 3b where another QCA circuit with 3×2 tiles is depicted. Also, this one seems to compute the function $y = x_1 \wedge x_2$; and indeed local synchronization is satisfied this time. However, information from the primary input x_1 at position $(0,0)$ needs to pass 4 tiles to arrive at the joint tile $(2,0)$; while the signal from x_2 at tile position $(1,0)$ only needs to pass 1 tile. This way, x_1's signal information arrives with a delay – violating the global synchronization.

However, not only the path length differences of signals play a role but also the clock that controls the primary inputs. By definition, the processing within a FCN circuit "starts" when all tiles controlled by clock 1 are in the switch phase (cf. Section II-B), so that all signals are able to accept an incoming value e. g. from external inputs. Consequently, any primary input controlled by another clock can only accept and eventually propagate information after that many clock phases have passed.

Example 5. *Consider the QCA circuit shown in Fig. 4. At a first glance, this circuit looks like a simple permutation of the one shown in Fig. 2c and therefore functionally equivalent. However, this circuit violates global synchronization because the primary inputs x_1 and x_2 propagate their information first and are therefore ahead of primary input x_3 which only can feed in its input after 2 clock phases. However, by this time, signals from x_1 and x_2 have already reached y leading to undefined behavior at the primary output.*

We call layouts which violate global synchronization *unbalanced*. Accordingly, layouts for which global synchronization holds, we call *balanced*. Naturally, in larger layouts with numerous signals that fan-out and re-converge, checking whether a layout is balanced, can become increasingly difficult.

Unfortunately, no method for equivalence checking exists yet which considers these characteristics and issues. In this work, we propose a solution which addresses this problem and, by this, for the first time provides a base solution for tackling verification of FCN circuits.

IV. Proposed Verification Approach

In this section, we present a verification approach that is able to handle the domain-specific properties of FCN circuits which we have described in the previous section. Using this approach, the equivalence between two function and/or circuit representations is verified (following the terminology established for conventional equivalence checking, those are called *specification* and *implementation* in the following). Possible

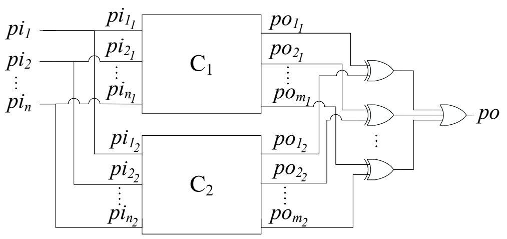


Fig. 5: Schematic representation of a miter structure

checks could, e. g., involve checking a logic network against a resulting FCN circuit (e. g. the circuit from Fig. 2a against the circuit from Fig. 2c) or checking two different FCN circuits against each other (e. g., in order to check an improved layout against an original one or the results of two different layout approaches against one another). Even comparisons across technologies become feasible, e. g. checking a QCA circuit against an NML one.

The main idea rests thereby on verification methods which got established in the domain of conventional circuits, namely the utilization of a so-called *miter* structure [26], [25]. In order to utilize this concept for FCN circuits, the miter formulation is complemented by further constraints which enforce the domain-specific properties discussed above. To this end, a dedicated analysis is conducted to gather the synchronization information needed for this task. The information is then used to accordingly complement the miter formulation. This section describes the process by first briefly reviewing the miter structure used for the verification of conventional circuits. Afterwards, the conducted analysis is described and it is shown how the information gathered here is used to create an FCN-specific formal model that eventually allows for proper equivalence checking.

A. Miter Structure

Conventional logic circuit equivalence checking is an established procedure (see e. g. [28]). Corresponding equivalence checkers either prove that a specification and an implementation are logically equivalent or provide an input assignment under which they behave differently (which, eventually, serves as a counter-example). Typically, a *miter* structure as sketched in Fig. 5 is incorporated for this purpose.

More precisely, a miter of a specification C_1 and an implementation C_2 with the same number of primary inputs n and outputs m is defined as a logic network M which contains both C_1 and C_2. Additionally, all corresponding pairs of primary inputs pi_{1_i}, pi_{2_i} are connected by fan-outs which become the new primary input pi_i; and all corresponding pairs of primary outputs po_{1_j}, po_{2_j} are connected by 2-input XOR gates whose outputs are all connected by a j-input OR gate which becomes the only primary output po, where $1 \leq i \leq n$ and $1 \leq j \leq m$.

Using this structure, basically a decision problem is formulated: Does an assignment to all (new) primary inputs pi_i exist which eventually leads to the primary output po being assigned 1? If this is the case, such an assignment constitutes a counter-example as both circuits C_1, C_2 are triggered with the equivalent input assignment but yield different output assignments (setting $po = 1$). Otherwise, if it can be proven that no such assignment exists, it has been shown that both circuits are equivalent. In order to solve this decision problem, reasoning engines such as SAT solvers [29], [30] are usually employed. To this end, the miter structure is translated into a corresponding format i. e. a *Conjunctive Normal Form* (CNF) (e. g. using *Tseitin transformation* [31]). Eventually, this results in an instance which is denoted Φ_M in the following.

However, checking the resulting miter instance Φ_M (and, by this, checking the *logical* equivalence of the given description) is a necessary but unfortunately not a sufficient condition to prove actual equivalence of FCN circuits. In fact, this formulation does not consider the synchronization issues discussed before in Section III. Therefore, we need to complement this instance to additionally enforce that also all synchronization constraints are satisfied. This is elaborated next.

B. Enforcing Proper Synchronization

The miter structure reviewed above only works if signals are *instantaneously* propagated through the respective logic networks. While this is implicitly given in conventional (combinational) networks, FCN circuits propagate their signals differently as reviewed in the previous sections. Hence, in order to make statements about correct propagation, the *delay* value for each signal in the circuit must be known. This delay of a signal is defined differently in FCN circuits as compared to conventional logic networks. As the terms are often confused in literature, we give a quick definition in accordance with our use case. A signal is a connected path on the layout starting at either a gate or a primary input inclusively, and ends at either a gate or primary output exclusively. We denote a path as a sequence of tiles $((x_1, y_1), (x_2, y_2), \dots)$. Based on this, the delay of a signal is then defined as its length. If the signal contains a primary input, its clock number is added in accordance with Example 5. We denote the delay of a signal s by $d(s)$.

Example 6. *Consider the FCN circuit shown in Fig. 3b again. There are two signals in the layout: one starting in tile* $(0, 0)$ *with* x_1 *as its input and one starting in tile* $(1, 0)$ *with* x_2 *as its input. Therefore, we will refer to the signals as* x_1 *and* x_2 *accordingly. Their path lengths are already marked in the figure as* 4 *and* 1 *respectively. To get the delay, we add their primary input tile's clock number, as it states the delay with which information is fed into the circuit, i. e.,* 4 *for* x_1 *and* 3 *for* x_2*. As a result,* $d(x_1) = 8$ *and* $d(x_2) = 4$.[3]

As this example illustrates, the delay of a signal can be viewed as the total amount of clock phases it takes a value to propagate along it. As discussed before in Examples 4 and 5, delay differences at gate inputs may lead to synchronization issues because values which were applied to the primary inputs at the same time step do not arrive at that gate simultaneously and therefore falsify the computation. In order to check whether this is the case, we propose a second instance Φ_S, which checks whether a proper synchronization is enforced and, together with Φ_M, allows to complete the equivalence checking process.

More precisely, Φ_S enforces that the sums of all delays along each path leading to any primary output must be identical. To this end, one free variable is added into every sum representing the delay of the primary input of that path (as all paths eventually lead to one) which can be assigned arbitrary to balance the signals if possible. This is described by the constraint

$$\bigwedge_{\substack{1 \leq j \leq m,\\ p \in P(po_j)}} d(po_j) = \left(\sum_{s \in p} d(s) \right) + d(pi(p)),$$

where

- m denotes the number of primary outputs,
- $P(po_j)$ denotes the set of paths leading to output po_j,
- $s \in p$ denotes a signal along path p,
- $pi(p)$ denotes the primary input of path p, and
- $d(po_j)$ and $d(pi(p))$ are new free positive integer variables (i. e. variables to be assigned by a solver) which represent the delay for each primary input and output of the FCN circuit (not the miter).

[3]Note that local synchronization is inherently checked, too, as we consider only paths emerging from locally synchronized data flows.

Informally, this constraint not only enforces computation of the resulting primary output delays but it also gives an assignment to the new primary input delays which a solver is free to assign to try to balance out the circuit if possible.

On top of that, and in order to keep the initial delay at the primary inputs as small as possible, we further apply a minimization objective over the sum of all $d(pi)$ variables. The resulting encoding can be evaluated with reasoning engines such as ILP or SMT solvers which can determine values for the $d(pi)$ variables that satisfy the constraints.

Example 7. *Consider Fig. 4 one last time. There are four signal paths leading to the primary output y in the circuit, namely*

$$\begin{aligned} p_1 &= (x_1, (0,0), (1,0), (2,0), y), \\ p_2 &= (x_2, (2,2), (2,1), (2,0), y), \\ p_3 &= (x_3, (0,2), (0,1), (0,0), (1,0), (2,0), y), \\ p_4 &= (x_3, (0,2), (1,2), (2,2), (2,1), (2,0), y). \end{aligned}$$

For each path, a delay constraint is enforced on y leading to the following formulation (considering signals in reverse order for each path as implied by the constraint formulation above):

$$\begin{aligned} d(y) &= \underbrace{\overbrace{d(((2,0)))}^{=1} + \overbrace{d(((0,0),(1,0)))}^{=2+1}}_{p_d((x_1,y))=4} + d(x_1) \\ &= \underbrace{\overbrace{d(((2,0)))}^{=1} + \overbrace{d(((2,2),(2,1)))}^{=2+1}}_{p_d((x_2,y))=4} + d(x_2) \\ &= \underbrace{\overbrace{p_d((x_1,y))}^{=4} + \overbrace{d(((0,2),(0,1)))}^{=2+3}}_{p_d((x_3,x_1,y))=9} + d(x_3) \\ &= \underbrace{\overbrace{p_d((x_2,y))}^{=4} + \overbrace{d(((0,2),(1,2)))}^{=2+3}}_{p_d((x_3,x_2,y))=9} + d(x_3) \end{aligned}$$

Here, each path signal is annotated with its delay value. Additionally, the p_d identifiers represent temporary variables which are assigned the path delay of the enclosed signals to reuse them in the following lines in a shorthand notation. Furthermore, it is enforced that $0 \leq d(x_i), d(y)$ for all free variables and that $d(x_1) + d(x_2) + d(x_3) + d(x_4)$ is to be minimized.

Finally, in order to use this constraint, the respective delay values for each signal in the FCN circuit need to be available. This, can easily be determined by a depth-first search or breadth-first search traversal.

Note that such traversals additionally identify all local synchronization issues that may exist in the circuit. Those manifest during traversal as wire segments without predecessors or gates with missing input signals (depending on the direction of traversal). If such issues can be identified, the considered FCN circuit produces undefined behavior and, therefore, by definition cannot be equivalent to any specification. Accordingly, if the traversal determines such issues, the equivalence checking process can be terminated and the user can be pin-pointed to the determined local synchronization problem. Otherwise, the needed delay values can be obtained and used for Φ_S as described above.

The number of paths that are present in the design have a direct impact on the computational effort it takes to reason about Φ_S. As previous work has shown, the number of paths can quickly grow in FCN designs [32].

In the next section, we finalize the resulting equivalence checking process and describe how to interpret the results obtained from the reasoning engines when solving Φ_M and Φ_S.

C. Resulting Equivalence Checking Process

Overall, the instances Φ_M and Φ_S as introduced above eventually allow to solve the considered equivalence checking problem. To this end, the Φ_M covers the logical equivalence while Φ_S ensures that a proper synchronization of the FCN circuit exists.

Passing Φ_M to a SAT solver may either lead to `UNSAT`, i. e. a proof that no assignment to the variables exists which satisfy the miter structure. In this case, the specification and implementation are logically equivalent since no assignment to the primary inputs $pi_1, \ldots, pi_n$ exists which lead to different values at the primary outputs $po_1, \ldots, po_m$. In contrast, if the SAT solver determines `SAT`, such an assignment exists and has been found. In this case, the FCN circuit has been shown to be non-equivalent to the given specification before even considering synchronization issues. The obtained assignment constitutes a counter-example showing which pattern to the primary inputs indeed leads to different values at the primary outputs.

In case of `UNSAT`, i. e. when logical equivalence has been shown, it remains left to check for proper synchronization using Φ_S. Passing Φ_S to an ILP or SMT solver may lead to either of the following results: (1) `UNSAT`, i. e. the circuit violates synchronization constraints so tremendously that the intended logic function will never actually be computed (an UNSAT core can point towards the location of failure), (2) `SAT` with all variables $d(pi(p)) = 0$ meaning the circuit is properly synchronized for every signal or synchronization effects eventually are balanced out in a way that no extra efforts need to be spent to accomplish equivalence, and (3) `SAT` with some variables $d(pi(p)) \neq 0$ meaning input patterns at those inputs need to be applied for a sequence of that many clock cycles to achieve a synchronization within the circuit so that all signals can stabilize [33]. We call case (2) *strong equivalence* and case (3) *weak equivalence*. In both latter cases, the FCN circuit can be considered equivalent to a specification. Besides that, as a side effect, the solver also outputs a value for the $d(po_j)$ variables in both the equivalence cases determining the minimum amount of clock phases it takes for all signals to completely propagate through the layout.

Example 8. *The formulation given in Example 7 when processed by a solver engine, yields `SAT` with the assignment*

$$d(x_1) \mapsto 4, d(x_2) \mapsto 4, d(x_3) \mapsto 0, d(y) \mapsto 7,$$

which constitutes weak equivalence to the circuit shown in Fig. 2c (as we know that they are logically equivalent) under the restriction, that a user has to apply all input patterns to x_1 and x_2 over a sequence of 4 clock phases until the output stabilizes. Furthermore, we get the information that the overall delay value of the circuit then becomes 7 as that is the delay value found for the primary output y.

V. Prototype Implementation

In order to demonstrate the applicability of the proposed solution, we implemented the approach described in the previous section in C++ on top of the publicly available FCN design framework *fiction* [34] as command `equiv`. This framework provides data structures for (tile-based) FCN layouts and has several built-in design algorithms (such as [10], [14], [15]). This additionally allowed us to automatically synthesize various FCN circuits from conventional logic networks taken from [35], [36], [37] – yielding numerous cases of equivalent circuit descriptions to be used to demonstrate the applicability

TABLE I: Obtained results

Instance				Equivalent Cases			Non-equiv. Cases	
Name	**#Gates**	**I / O**	**Dimension**	**Equiv.**	$d(pi)$	**t in s**	**Equiv.**	**t in s**
2:1 MUX	5	3 / 1	3 × 4	Strong Eq.	0	< 1	Not Eq.	< 1
XOR	6	2 / 1	3 × 6	Strong Eq.	0	< 1	Not Eq.	< 1
XNOR	8	2 / 1	3 × 6	Strong Eq.	0	< 1	Not Eq.	< 1
Half adder	10	2 / 2	4 × 5	Strong Eq.	0	< 1	Not Eq.	< 1
ParGen	14	3 / 1	5 × 7	Strong Eq.	0	< 1	Not Eq.	< 1
4:1 MUX	16	6 / 1	7 × 8	Strong Eq.	0	< 1	Not Eq.	< 1
ParCheck	21	4 / 1	7 × 8	Strong Eq.	0	< 1	Not Eq.	< 1
c17	11	5 / 2	8 × 4	Weak Eq.	2	< 1	Not Eq.	< 1
c499	1207	41 / 32	828 × 412	Weak Eq.	177	14	Not Eq.	< 1
c1355	1559	41 / 32	1140 × 452	Weak Eq.	225	22	Not Eq.	< 1
c1908	1219	33 / 25	852 × 400	Weak Eq.	133	14	Not Eq.	< 1
c3540	2997	50 / 22	2200 × 842	Weak Eq.	577	86	Not Eq.	< 1
ctrl	601	7 / 25	417 × 191	Weak Eq.	96	3	Not Eq.	< 1
int2float	801	11 / 7	574 × 238	Weak Eq.	182	6	Not Eq.	< 1
cavlc	2294	10 / 11	1666 × 638	Weak Eq.	301	50	Not Eq.	< 1
adder	3434	256 / 129	2541 × 1149	Weak Eq.	823	104	Not Eq.	< 1
i2c	4064	133 / 127	2875 × 1318	Weak Eq.	815	148	Not Eq.	< 1
bar	10001	135 / 128	7058 × 3078	Weak Eq.	2343	967	Not Eq.	< 1

#Gates	Number of gates plus fan-outs	**I / O**	Number of primary inputs / outputs
Dimension	Occupied area in tiles	$d(pi)$	Number of clock cycles to stabilize all signals

of the proposed approach. In order to also consider non-equivalent cases, errors have randomly been injected into the layouts by changing gate functions from AND to OR, wire to NOT, and, respectively, vice versa.

Having this set of benchmarks, the proposed approach has been applied to check them for equivalence. To this end, we executed the resulting implementation 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. Table I summarizes some obtained results.[4] The columns *Instance* give an overview about the used circuit specifications, i. e. their names, number of gates, number of primary inputs and outputs, as well as their dimension as an FCN circuit implemented by *fiction*. In the columns *Equivalent Cases*, we show the verification results obtained for checking the original (conventional) logic network against the synthesized FCN circuits. In the columns *Non-equiv. Cases*, we show the obtained results when an FCN circuit is considered into which we injected errors.

The results clearly demonstrate the applicability of the proposed approach. In all cases, the expected output is obtained. Even the distinction into strong equivalence (denoted by *Strong Eq.*) and weak equivalence (denoted by *Weak Eq.*) as well as the minimum amount of clock cycles it takes for all signals to stabilize (i. e. the highest $d(pi)$ value) are correctly determined.[5] In case of the FCN circuits into which we injected errors, all errors have correctly been identified by the proposed approach as well (denoted by *Not Eq.*). By this, we showed that we were able to verify FCN circuits automatically for the first time.

VI. Conclusions

This work provides researchers and engineers in the domain of *Field-coupled Nanocomputing* (FCN) with a first solution for the verification of FCN circuits. This becomes important due to the substantial improvements in FCN design which increasingly yield FCN circuits that cannot be checked manually anymore. Since conventional verification approaches are not directly applicable for FCN circuits, we proposed an alternative approach that addresses the problems. A prototype implementation demonstrated the applicability of the proposed solution which has been made reproducible by integrating it in the *fiction* framework [34] available at https://github.com/marcelwa/fiction.

Acknowledgments

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

[4]Due to space limitations, not all runs are reported in Table I, but the skipped instances can easily be executed using *fiction*.

[5]Note that, as discussed in Section IV-C, those results heavily depend on how the respectively chosen synthesis approach actually implements the FCN circuit.

References

[1] G. E. Moore *et al.*, "Cramming More Components onto Integrated Circuits," 1965.
[2] M. B. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, 2013.
[3] N. G. Anderson and S. Bhanja, *Field-coupled Nanocomputing: Paradigms, Progress, and Perspectives*, 1st ed. New York: Springer, 2014.
[4] 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.
[5] 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.
[6] F. Sill Torres, P. Niemann, R. Wille, and R. Drechsler, "Near Zero-Energy Computation Using Quantum-dot Cellular Automata," *JETC*, vol. 16, no. 1, 2020.
[7] S. Bohloul, Q. Shi, R. A. Wolkow, and H. Guo, "Quantum Transport in Gated Dangling-Bond Atomic Wires," *Nano Letters*, vol. 17, no. 1, pp. 322–327, 2017.
[8] C. S. Lent *et al.*, "Molecular Cellular Networks: A non von Neumann Architecture for Molecular Electronics," in *ICRC*, 2016, pp. 1–7.
[9] 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.
[10] 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.
[11] G. Fontes *et al.*, "Placement and Routing by Overlapping and Merging QCA Gates," in *ISCAS*, 2018, pp. 1–5.
[12] 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.
[13] F. Riente *et al.*, "ToPoliNano: A CAD Tool for Nano Magnetic Logic," *TCAD*, vol. 36, no. 7, pp. 1061–1074, 2017.
[14] 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.
[15] R. Wille, M. Walter, F. Sill Torres, D. Große, and R. Drechsler, "Ignore Clocking Constraints: An Alternative Physical Design Methodology for Field-Coupled Nanotechnologies," in *ISVLSI*, 2019, pp. 651–656.
[16] 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.
[17] X. K. Hu *et al.*, "Edge-Mode Resonance-Assisted Switching of Nanomagnet Logic Elements," *IEEE Trans. Magn.*, pp. 1–4, 2015.
[18] W. Liu, E. E. Swartzlander Jr, and M. O'Neill, *Design of Semiconductor QCA Systems*. Artech House, 2013.
[19] E. Varga *et al.*, "Experimental Realization of a Nanomagnet Full Adder using Slanted-Edge Magnets," *IEEE Trans. Magn.*, vol. 49, no. 7, pp. 4452–4455, 2013.
[20] J. Huang, M. Momenzadeh, L. Schiano, M. Ottavi, and F. Lombardi, "Tile-based QCA design using majority-like logic primitives," *JETC*, vol. 1, no. 3, pp. 163–185, 2005.
[21] 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.
[22] T. Sasao, *Logic Synthesis and Optimization*. Springer, 1993.
[23] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," in *CAV*, 2010, pp. 24–40.
[24] D. A. Reis *et al.*, "A Methodology for Standard Cell Design for QCA," in *ISCAS*, 2016, pp. 2114–2117.
[25] S. Malik, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment," in *DAC*, 1988, pp. 6–9.
[26] D. Brand, "Verification of Large Synthesized Designs," in *ICCAD*, 1993, pp. 534–537.
[27] S. Hassoun and T. Sasao, *Logic Synthesis and Verification*. Springer Science & Business Media, 2012, vol. 654.
[28] P. Molitor and J. Mohnke, *Equivalence Checking of Digital Circuits: Fundamentals, Principles, Methods*. Springer Science & Business Media, 2007.
[29] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *SAT*. Springer, 2003, pp. 502–518.
[30] R. Wille, G. Fey, D. Große, S. Eggersglüß, and R. Drechsler, "SWORD: A SAT like prover using word level information," in *VLSI-SoC*, 2007, pp. 88–93.
[31] G. S. Tseitin, "On the Complexity of Derivation in Propositional Calculus," in *Automation of Reasoning*. Springer, 1983, pp. 466–483.
[32] F. Sill Torres, R. Wille, M. Walter, P. Niemann, D. Große, and R. Drechsler, "Evaluating the impact of interconnections in Quantum-dot Cellular Automata," in *DSD*, 2018, pp. 649–656.
[33] F. Sill Torres, M. Walter, R. Wille, D. Große, and R. Drechsler, "Synchronization of Clocked Field-Coupled Circuits," in *IEEE-NANO*, 2018, pp. 1–5.
[34] 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:1905.02477.
[35] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," in *ISCAS*. IEEE Press, 1985, pp. 677–692.
[36] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL Combinational Benchmark Suite," in *Int'l Workshop on Logic Synth.*, 2015.
[37] A. Trindade *et al.*, "A placement and routing algorithm for Quantum-dot Cellular Automata," in *SBCCI*, 2016.