ToPoliNano and *fiction*: Design Tools for Field-coupled Nanocomputing (Invited Paper)

Umberto Garlando*, Marcel Walter†, Robert Wille‡§¶, Fabrizio Riente*, Frank Sill Torres‖, Rolf Drechsler†§

*Department of Electronics and Telecommunications, Politecnico di Torino, Italy
†Group of Computer Architecture, University of Bremen, Germany
‡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 computing concept with several promising post-CMOS candidate implementations that offer tremendously low power dissipation and highest processing performance at the same time. Two of the manifold physical implementations are *Quantum-dot Cellular Automata* (QCA) and *Nanomagnet Logic* (NML). Both inherently come with domain-specific properties and design constraints that render established conventional design algorithms inapplicable. Accordingly, dedicated design tools for those technologies are required. This paper provides an overview of two leading examples of such tools, namely *fiction* and ToPoliNano. Both tools provide effective methods that cover aspects such as placement, routing, clocking, design rule checking, verification, and logical as well as physical simulation. By this, both freely available tools provide platforms for future research in the FCN domain.**

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 for the longest time followed 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 several 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], [12], [13] – even though the underlying problem is $\mathcal{NP}$-complete [14].

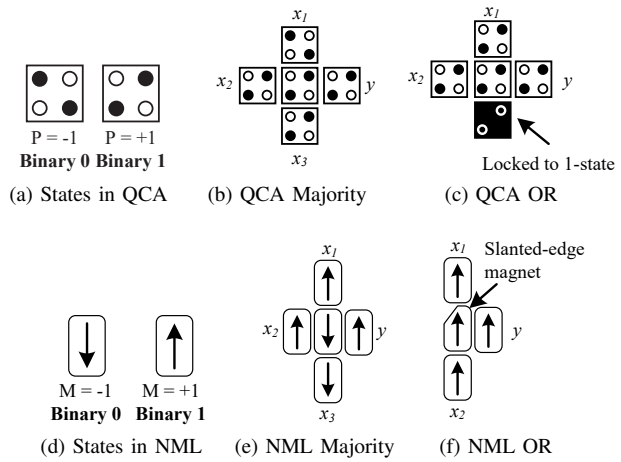
This paper presents two physical design tools for FCN circuit layouts, namely *fiction* and ToPoliNano. While *fiction* focuses on providing an open-source platform for designers, algorithm developers, and researchers alike, ToPoliNano comes with a user-centric interface to enable the study and exploration of the FCN design concepts.

The remainder of the paper is structured as follows. To keep this work self-contained, Section II provides background on the FCN concept and covers two implementations, namely *Quantum-dot Cellular Automata* (QCA) and *Nanomagnet Logic* (NML). Sections III and IV discuss the design tools *fiction* and ToPoliNano, respectively, and demonstrate their usability. Finally, the paper is concluded in Section VI.

II. Field-coupled Nanocomputing

This section provides the background on *Field-Coupled Nanocomputing* (FCN) technologies, and the basis for the remainder of this work. Instead of transistors, FCN circuits consist of elements usually called *cells* that interact via mutual repulsion of local fields. In *Quantum-dot Cellular Automata*

This paper discusses ToPoliNano version 1.2.0 and *fiction* version 0.3.1.



(a) States in QCA (b) QCA Majority (c) QCA OR

(d) States in NML (e) NML Majority (f) NML OR

Fig. 1: FCN states and basic gates

(QCA) [15], one possible implementation of the FCN concept, a cell is composed of four or six *quantum dots* which can confine an electric charge and are arranged at the corners of a square [16], [17]. Adding two free and mobile electrons into each cell, that can tunnel between adjacent dots, yields a stable state due to Coulomb interaction (note that a potential barrier prevents tunneling to the outside of the cell). 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, *Nanomagnet Logic* (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). Both concepts allow to implement Boolean functions such as AND, OR, NOT, Majority.

Fig. 1a and 1d show the two cell polarization 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. On the contrary, 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* [18] is applied that give preference to a magnetization.

As per the FCN concept, when two cells, both QCA and NML, are placed closed to each other, the field effects will interact, enabling information propagation. Unfortunately, the effect of a neighboring cell is not enough to change the magnetization of a cell. An external field, commonly referred to as clock signals, is used to force the cells in an unstable state. When the external field is removed, the cells will reach a new stable condition depending on the interaction among each other. The principle of an external clock mechanism is common to the FCN technologies. Elements are placed in different clock zones [19], [20], [13] and, in this way, the information is correctly propagated through the circuit. The need for clock zones sets completely new paradigms in the design of circuits based on FCN technologies. Furthermore, the circuits behave like a pipeline, where the data need to move across the different clock zones. The physical design task in FCN is not compatible with the placement of gates and routing of wires of CMOS technology. The new paradigms brought-in by the technological constraints require new techniques, for example, signal path balancing throughout the entire design.

III. FICTION

This section describes *fiction* [21], a framework for field-coupled technology-independent open nanocomputing which is available at https://github.com/marcelwa/fiction. The *fiction* framework is written in C++ and comes with data structures for tile-based and cell-based FCN layouts, gate libraries for technology mapping, algorithms for logic synthesis, physical design, logical simulation, and verification, as well as a *command-line interface* (CLI), benchmarking, scripting, and logging capabilities. Besides that, *fiction* also supports several input and output file formats. By this, *fiction* provides a comprehensive sandbox for designers, researchers, and developers in the FCN domain.

In the following sections, a flow from the specification level down to physical simulation is described and illustrated by demonstrating the interaction with *fiction*'s store-based CLI *alice* [22]. All commands discussed in the following can be called with the `-h` flag to print information about their further settings and arguments.

A. Specifications

The starting point of all flows is a logic network of elementary primitives that serves as a specification for the layout that is to be generated. Two input file formats for logic networks are supported: gate-level *Verilog* that exclusively uses the assign statement and logic primitives as well as *AIGER* [23] that specifies *And-inverter graphs* (AIGs). Suitable files can be generated, e. g., with *ABC* [24] by using commands

```
strash
write <outputfile>
```

on any logic network in store where `<outputfile>` uses either the `.v` or `.aig` file extension.

The parser library *lorina* [22] used in *fiction* supports MAJ operations in Verilog. Alternatively, logic networks can also be synthesized as MAJ networks from a truth table specification using *Akers' synthesis* [25] or be generated randomly with or without MAJ nodes. This enables one to quickly test algorithms on a multitude of inputs. The following snippet creates three logic networks as specifications, one read from a file, one synthesized from a truth table, and one generated at random with 4 primary inputs and 8 logic nodes (not counting inverters and fan-outs).

```
read ../benchmarks/ISCAS85/c17.v
tt 0001110010100111; akers
random -n 4 -g 8
```

Furthermore, truth tables can be generated from logic networks which can then again be used for synthesis. This is one possible way to generate a Majority network from a non-Majority one. The following commands exploit this use case.

```
read ../benchmarks/TOY/xor5R.v
simulate -n --store
akers
```

Note that truth tables represent single-output functions. Therefore, this approach generates multiple truth tables for multi-output networks.

Initially, no logic network incorporates designated fan-out nodes. However, during the physical design, fan-outs are realized as elements that occupy one tile and, by this, contribute to the critical path and the throughput. To substitute high-degree fan-out nodes in any logic network, the command `fanouts` can be used that is parameterizable with a degree, a strategy (breadth vs. depth), and a threshold. Furthermore, certain algorithms in related work propose path balancing as an important pre-processing step to physical design. The command `balance` provides this possibility. However, this command has been introduced for the sake of completeness as the state-of-the-art physical design algorithms implemented in *fiction* achieve better results on unbalanced networks.

To print statistics about a logic network, the command `ps -n` can be used. Alternatively, the command `gates` provides a detailed listing of the gate types. Additionally, `show -n` allows to inspect the network by creating a *Graphviz* `.dot` file from it and opening it with the platform's standard viewer.

B. Physical Design

The process of physical design is the transformation of a logic network into a fabricable circuit layout satisfying all technology-specific constraints so that it conducts the same functionality as the original logic network. This has been proven to be an $\mathcal{NP}$-complete problem [14]. The physical design contains the steps of placement, routing, timing (clocking), and technology mapping. In the following, *fiction*'s implementations of state-of-the-art algorithms for FCN physical design are briefly discussed.

First, an approach based on a first-order logic description of the FCN physical design problem is described. It generates optimal FCN circuit layouts in terms of area while meeting all design constraints by a sequence of incremental SMT solver calls [10]. Intuitively speaking, sets of rules are generated that encode all possible placements and routings on a fixed layout size symbolically. An SMT solver is then called to find a valid one that fulfills all design constraints. If none exists, the layout size is incremented and the process is started all over again. Various options and toggles allow for the use of, e. g., arbitrary predefined clocking schemes, crossings, primary input/output locations, unbalanced paths, and synchronization elements. Additionally, optimization targets can be set to minimize, e. g., the number of wire segments, or crossings used. Symmetry breaking and sophisticated encoding mechanisms are applied to reduce solving time as much as possible. Nevertheless, the nature of an exact approach limits its applicability to rather small logic networks with only a few dozens of nodes.

However, this approach is highly parameterizable and can produce results for a variety of settings, thereby allowing for design space exploration. The command `exact` calls this algorithm on the current logic network in store. Some important parameters are listed in the following.

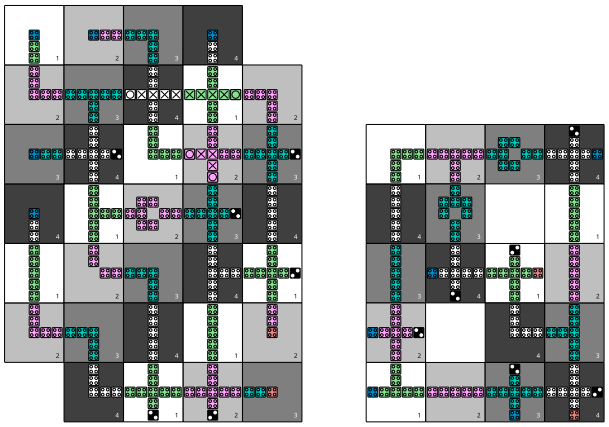
- `-s` Defines the clocking scheme to be used. Possible values are `2DDWave` [26], `USE` [27], `RES` [28], and `BANCS` [29], and `ToPoliNano` [30]. If no scheme is given, the solver takes this degree of freedom to find the most compact one for the network. Note that this has a huge negative impact on the runtime and should only be used for the smallest of networks.
- `-x` Enables wire crossings. Note that most networks require wire crossings to be layouted.
- `-i` Enables designated pins for the primary inputs and outputs of the layout.
- `-b` Routes all primary inputs and outputs to the layout's borders.
- `-l` Enables the use of clock latches (synchronization elements) to artificially balance signal paths by stalling information [31].
- `-d` Allows for de-synchronized (unbalanced) signal paths resulting in more compact layouts with reduced throughput.
- `-w` Minimizes the number of wire segments used.
- `-c` Minimizes the number of crossings used.
- `-a` Specifies a number of threads to run asynchronously. Since the threads cannot share learned clauses, this does not always increase performance. However, it turned out to be helpful especially for larger networks.

Second, a scalable method [32] is implemented which is based on an approximation for *Orthogonal Graph Drawing* (OGD) [33]. It represents the logic network as a graph that is to be embedded in the plane orthogonally and solves this problem using Biedl's algorithm [34]. It has a huge runtime advantage compared to the exact approach. Although the layouts generated by this approach are non-optimal in terms of area and restricted to the 2DDWave clocking scheme, this technique is applicable even for larger networks and provides results in reasonable runtime.

This algorithm can be called using the command `ortho` that has the same `-i` and `-b` flags as `exact`.

Generated layouts of both algorithms are gate-level abstractions. After creation, they are placed in a respective store just like logic networks. To print statistics, command `ps -g` can be used.

To apply a gate library and thereby conduct a technology mapping from the gate-level down to the cell-level, the command `cell` is available. The default library to apply is *QCA ONE* [35]. The command generates a QCA cell layout and places it in a store from where it can be inspected via



(a) `exact -ixbs 2DDWave` (b) `exact -ds USE`

Fig. 2: Two differently layouted variants of `c17.v`

`show -c` – generating a scalable vector graphic and opening it in the platform's standard viewer. Area information can be displayed via command `area` and its energy consumption can be approximated via command `energy` (using the physical model presented in [5]).

Consider the following sequence of commands that puts the entire physical design flow together by generating two differently parameterized layouts from the `c17` logic network, printing their statistics, conducting technology mapping, and generating graphics.

```
read ../benchmarks/ISCAS85/c17.v
exact -ixbs 2DDWave
ps -g
cell
show -c
exact -ds USE
ps -g
cell
show -c
```

The following statistical information about the layouts is printed.[1]

```
c17: 5 x 7, #G: 18, #W: 18, #C: 3, CP: 11, TP: 1/1
c17: 4 x 5, #G: 11, #W:  7, #C: 0, CP: 13, TP: 1/3
```

This reads as the dimension of the resulting layout in tiles, the amount of gate (`#G`) and wire tiles (`#W`), crossings (`#C`) used, the length of the critical path (`CP`) in tiles, and the throughput (`TP`), i. e., the highest delay difference of any gate in the layout, where `1/x` means that input data must be held constant for `x` full cycles to synchronize all signals while correct outputs occur only once every `x` full clock cycles [31], [36]. Note that fan-outs and I/O pins are counted as gates since they are fixed by the input. This way, the displayed amount of wires represents the net costs [37]. The resulting graphics are shown in Figure 2.

[1]The number of synchronization elements has been omitted from both lines (as this design feature was not even enabled) to shorten the output and prevent unaesthetic line breaks.

C. Validation

The correctness of designed layouts can be validated on the structural, the logical, and the physical level.

For gate-level layouts, the first step after the physical design is the design rule checking, i. e., a process that inspects the structural integrity of the generated layout. This is also a useful tool for algorithm designers that want to debug their code as it directly points to the locations of design rule violations in the layout. The command `check` executes design rule checking on the current gate-level layout and outputs a summary report. Among other features, this algorithm checks for proper wire routing, spacing, and crossing, whether data flow respects clocking, and primary input and output pin locations.

If the layout's structure is valid, its logical functionality can be checked. Therefore, the logical simulation via command `simulate -g` is available to generate a truth table for each primary output. Due to the exponential growth in the number of primary inputs, the simulation-based approach is not reasonable for larger-scale layouts. Therefore a SAT-based verification approach can be used to prove functional equivalence of a generated layout and a specification that can either be a logic network or another layout [38]. The respective command is `equiv` which compares a layout against the logic network it was created from. Adding parameter `-g <n>` allows comparison against another layout in store where `<n>` must be replaced with the respective store entry identifier.

However, this verification only validates functional correctness. To achieve a notion of the physical behavior of a circuit, QCA cell-level layouts can be written as simulation files for the *QCADesigner* [39], a standard tool for physical simulation of QCA structures, by using command `qca <filename>`. In addition, iNML cell-level layouts can be written as component files for ToPoliNano and MagCAD, which are described in Section IV, by using command `qcc <filename>`.

D. Scripting & Benchmarking

All demonstrated functionalities can be embedded into several types of scripts.

A *fiction script* is a text file that contains a sequence of commands. It can be passed to *fiction* by calling it with `./fiction -f <filename>`.

Since *fiction scripts* are rather inflexible, for the next scenario, we want to layout all files from a folder, log their statistical information, and generate simulation models for QCADesigner. To this end, we create the following bash script.

```
for filepath in ../benchmarks/TOY/*.v; do
  f="${filepath##*/}"
  ./fiction -c "read $filepath; ortho -i; ps -g;
    cell; qca ${f%.*}.qca" -l ${f%.*}.json
done
```

Using the `-c` flag, a semicolon-separated list of commands can be provided and the output is logged in a JSON file by the `-l` flag. For both, the physical models as well as the log files, the original filename is used extended by the respective file extension.

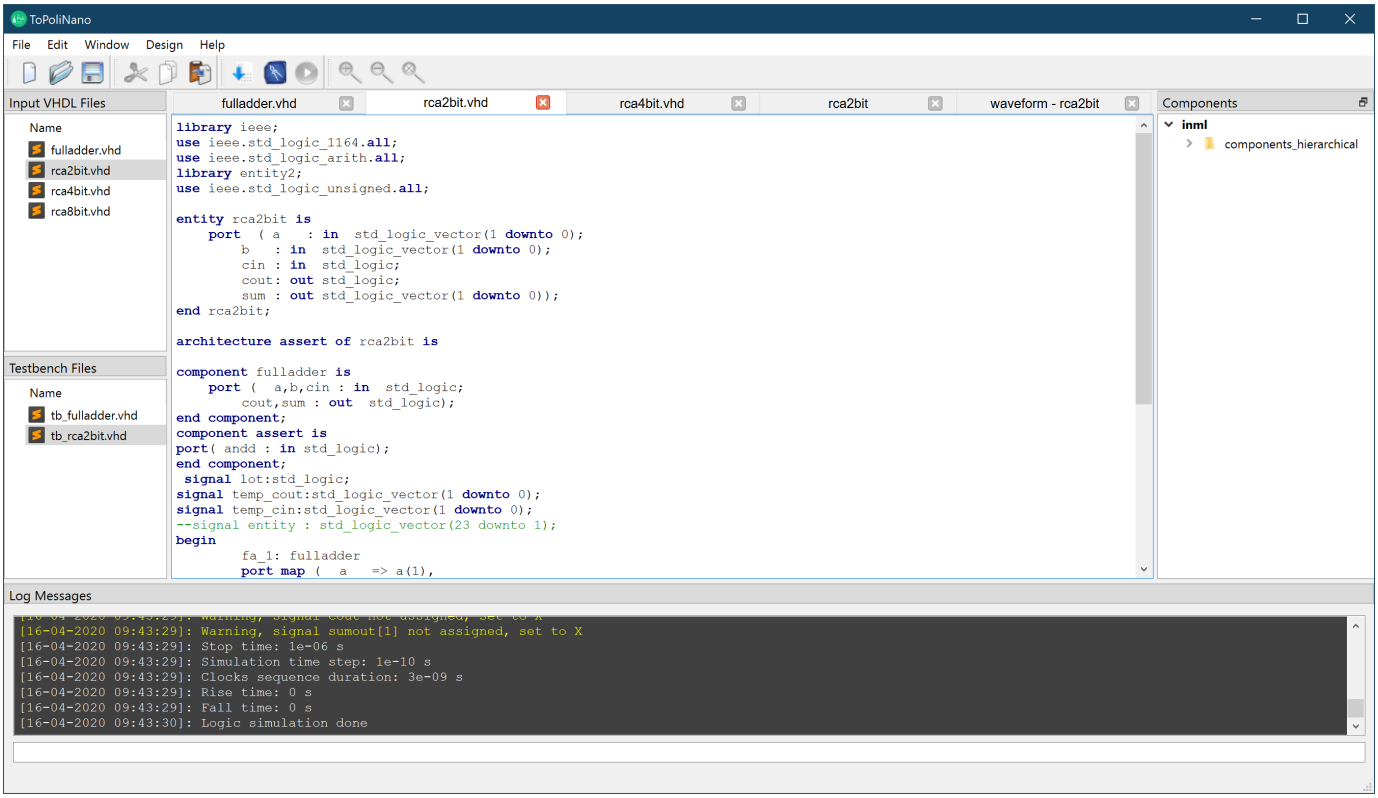


Fig. 3: Main window of ToPoliNano. In the central part a VHDL file opened in the built-in text editor. The language keywords are highlighted to improve readability.

For more sophisticated analyses and interoperability with statistical tools, *fiction* can expose a Python API. Refer to the official *alice* documentation to learn more [22].

IV. ToPoliNano

This section describes the ToPoliNano (Torino Politecnico Nanotechnology) framework, which is available at https://topolinano.polito.it. The project has grown around the idea of creating a flexible set of tools enabling the design, simulation, and test of FCN circuits. This framework aims to provide a unified approach to the exploration of emerging technologies [40]. The framework is a cross-platform EDA software developed by the VLSI group of Politecnico di Torino. It closes the gap between device engineers and system-level engineers enabling the same top-down design flow usually available with CMOS. It enables architectural exploration with a primary focus on FCN. ToPoliNano is developed in C++ and comes with a portable user interface written in *Qt* to operate across Linux, macOS, and Windows. The ToPoliNano framework is composed of two separate software: ToPoliNano [30] and MagCAD [41]. The former is an automatic place & route tool that also embeds a simulation engine. The latter enables custom circuit design using a graphical user interface (GUI).

Currently, the ToPoliNano framework supports the *in-plane Nano Magnetic Logic* (iNML) and *perpendicular Nano Magnetic Logic* (pNML) technologies. Circuits designed with MagCAD can be used as custom components in ToPoliNano. On the other hand, automatically generated layouts by ToPoliNano can be modified and embedded in larger designs by using MagCAD. This information exchange is possible thanks to a common file format defined in the framework. Layout and components files are saved in a specific format associated with the `.qll`- and `.qcc`-extensions, respectively. In the following sections, a rough description of the ToPoliNano flow is provided.

A. HDL Parsing

ToPoliNano enables the physical design and simulation of circuits in the same tool. The ToPoliNano starting point is a post-synthesis HDL description of the circuit. Logic networks synthesized by the *Synopsys Design Compiler* [42] or *ABC* [24] are applicable for instance. The synthesis needs to be performed using a particular target library. As an example, for the iNML design, the synthesis should be performed using a specific library that we provide on the ToPoliNano website as a `.db` file. The library contains the basic cells available in the target technology. Note that only combinational logic networks can be handled by ToPoliNano.

Fig. 3 presents a screenshot of ToPoliNano's main window. Four parts can be identified in the GUI. On the left, source files are listed in a tree-view. The user selects a workspace folder, and the tool handles all the files present in that folder. On the right, a similar tree-view shows the components available in

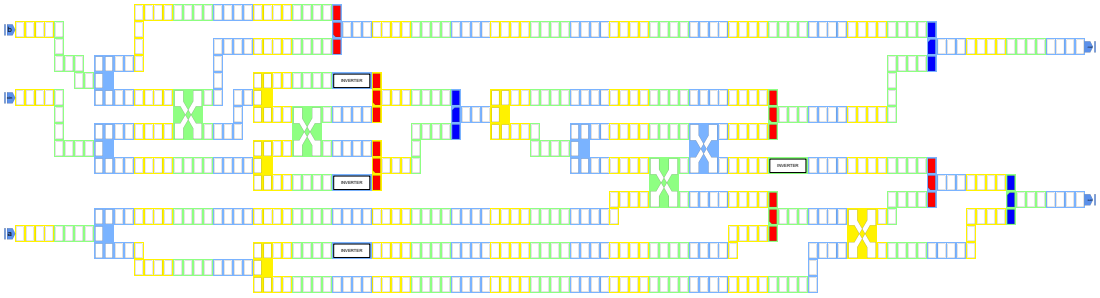


Fig. 4: Example of full adder layout obtained with ToPoliNano.

the user library. The bottom part of the GUI is reserved for a log panel. Log messages appear here, and a user can also enter commands using a *command line interface* (CLI) . All the commands can also be performed using the buttons and menus of the interactive GUI. Finally, the center part of the window holds a set of tabs. Different tabs are available: a simple text editor, a layout visualizer, and simulation waveforms.

The user can modify the descriptive code of synthesized logic networks (which can be provided in both, Verilog or VHDL) using the integrated text editor (see Fig. 3). Then, it is possible to select a top-level file, and the tool will compile all the elements in the hierarchy. ToPoliNano will automatically scan the source files and build an internal data structure that will be used during the layout phase.

B. Physical Design

The next phase is the layout generation. This phase consists of two main steps: (1) the graph elaboration and (2) the physical mapping. During the graph elaboration phase, the netlist is analyzed and optimized to verify the maximum fan-out allowed by each node. It synchronizes signals and reduces as much as possible the number of wire crossings. The user can select different optimization algorithms and technology parameters. Those algorithms are derived from the physical design of CMOS circuits, but have been adapted to meet the technology constraints of FCN. The algorithms aims at reducing the number of crossings in the layout. Given that iNML is a planar technology, the number of crossings has a significant impact on the layout dimension and, therefore, the performance. The user can select among four different algorithms: Barycenter, Kernighan-Lin, and two versions of simulating annealing [30]. The user can define a sequence of algorithms to be applied (cf. Fig. 5). It is possible to have multiple entries of the same algorithm in the sequence. Furthermore, the user can modify the parameters for the simulated annealing execution. It is also possible to change the design rules for the layout. For example, the maximum number of fan-out, the number of vertically stacked elements, and the number of magnets placed in each clock zone. Furthermore, the user can modify some element parameters, like the magnets geometry and the spacing. This possibility is extremely interesting for emerging technologies: it is possible to understand how these parameters affect the final layout.

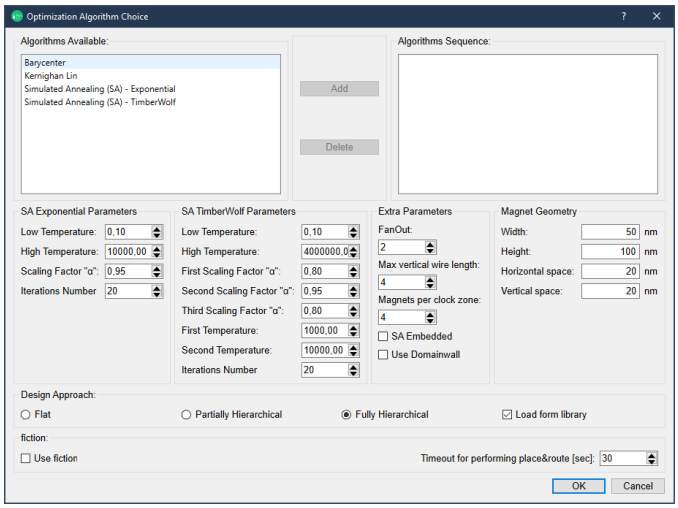


Fig. 5: Layout parameter window in ToPoliNano. In the top part the optimization algorithms, in the centre layout parameters, and in the bottom part the layout approach.

ToPoliNano can handle large logic networks exploiting hierarchy to reduce the computation time. ToPoliNano can reuse components available within the user library or generate a flat layout from a hierarchical netlist. The user can choose among different layout approaches: *flat*, *partially*, and *fully hierarchical*. In the *flat* approach, the hierarchy of the network is flattened, while it is fully exploited in the fully hierarchical mode. In the partially hierarchical approach, only the first level of the hierarchy will be used, and inner components will be flattened. In the hierarchical approach, the user has an additional possibility. It is possible to force ToPoliNano to perform the layout of every component in the hierarchy. On the contrary, enabling the "Load from library" checkbox, the tool will scan the workspace and load previously designed components used in the layout. These components could have also been modified with MagCAD [41].

After this phase, the resulting circuit layout is displayed (see Fig. 4). The user can inspect the layout up to the single element and it is possible to expand the components and examine all the hierarchical levels. During the layout execution, detailed information are reported in the log panel present in the bottom part of the ToPoliNano GUI. In particular, the execution time

Fig. 6: Simulation waveforms of the full adder inside the ToPoliNano GUI.

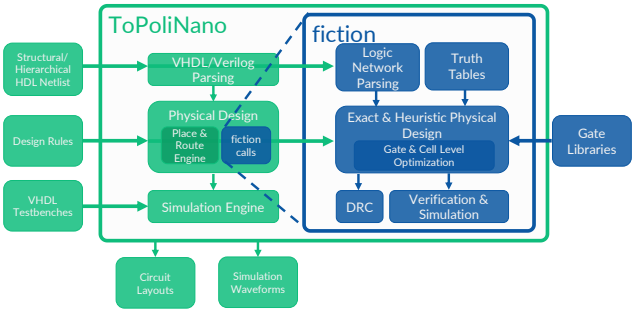


Fig. 7: Schematic view of ToPoliNano design flow and how *fiction*'s SMT-based exact physical design algorithm is available as a plug-in for ToPoliNano's place & route engine to generate optimal building blocks in the hierarchical layout process.

of all the steps is reported as well as the number of elements placed and the number of crossing in the layout before and after the optimizations.

Overall,ToPoliNano can handle large circuits. It was tested with the ISCAS85 benchmark [43], with netlists counting up to thousands of gates. The resulting circuits, consisting of up to hundred of millions of elements, are designed automatically in a reasonable time. After this phase, the layout's behavior can be simulated.

C. Simulation

At this point, the layout's behavior can be verified through simulations. ToPoliNano embeds a simulation engine that can perform a *switch level simulation* [44], [45]. It can be used to verify the proper circuit behavior according to user-defined input stimuli. ToPoliNano can parse a VHDL testbench that is used to generate the waveforms for the input pins. The tool performs the simulation, and the results are represented in a dedicated window (see Fig. 6) and saved in a textual format.

V. Common Integrated Design Flow

This section discusses a common design flow to generate iNML circuit layouts involving both ToPoliNano and *fiction*.

The ToPoliNano place & route engine is based on heuristic algorithms optimized for circuit compaction using both flat

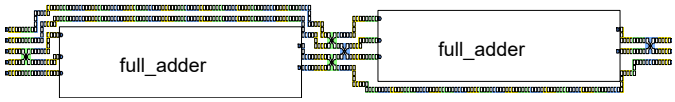


Fig. 8: Hierarchical layout of a 2-bit RCA.

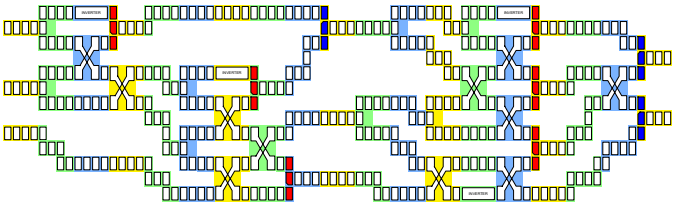


Fig. 9: Full adder layout obtained with *fiction* with the ToPoliNano clock scheme.

and hierarchical layout approaches. Therefore, *fiction*'s exact physical design approach discussed in Section III-B is a great match as it can be exploited for designing optimal building blocks that can then be re-used within ToPoliNano's hierarchical designs.

To enable this interoperability, we implemented the column-wise clocking scheme and the iNML gate library used by ToPoliNano in fiction. Additionally, ToPoliNano was modified to integrate *fiction* as a plug-in (see Fig. 5) to enable a user to perform building block design using *fiction*'s exact algorithm. If the exact placement is not available within a specified time-out, ToPoliNano's heuristic layout process is executed instead. Due to the exact task's exponential nature, larger sub-networks can exceed tolerable time limits.

The resulting block diagram after the integration is shown in Fig. 7. Fig. 8 shows an example where *fiction* was used during a layout process. In this case, a simple ripple-carry adder (RCA) circuit is used as an example. The RCA is composed of two full adders, where the carry-out of the first is fed as the carry-in to the second one. We let *fiction* generate an optimal full adder in terms of circuit area that was then handled by ToPoliNano as a black box building-block during hierarchical layout. Fig. 9 shows said full adder building block. As expected, the exact algorithm could reduce the number of elements, and therefore the area of the circuit compared to the one presented in Fig. 4. The components are then placed and interconnected by the ToPoliNano engine, enabling an integrated design flow.

VI. Conclusion

This paper discussed ToPoliNano and *fiction*, two state-of-the-art physical design tools for *Field-coupled Nanocomputing* (FCN) circuits. The tools cover aspects like placement, routing, clocking, design rule checking, verification, and logical as well as physical simulation. While *fiction* focuses on providing an open-source platform for designers, algorithm developers, and researchers alike, ToPoliNano comes with a user-centric interface to enable the study and exploration of the FCN architectural solutions and computing paradigms with the possibility to integrate new promising devices. Furthermore, integration between *fiction* and ToPoli-

Nano has been made possible by adding ToPoliNano's three-phase clocking scheme and iNML gate library in *fiction*. This way, iNML layouts generated by *fiction* can be reused in ToPoliNano. *fiction*'s SMT-based exact physical design algorithm is now available as a plug-in to ToPoliNano's hierarchical place & route engine.Both tools are available for download on their respective websites and by that enable further research in the FCN domain. ToPoliNano is available at https://topolinano.polito.it and *fiction* is available at https://github.com/marcelwa/fiction.

VII. 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] 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. S. 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] G. Causapruno, F. Riente, G. Turvani, M. Vacca, M. R. Roch, M. Zamboni, and M. Graziano, "Reconfigurable systolic array: From architecture to physical design for nml," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, no. 99, pp. 1–10, 2016.

[13] M. Vacca, F. Cairo, G. Turvani, F. Riente, M. Zamboni, and M. Graziano, "Virtual clocking for nanomagnet logic," *IEEE Transactions on Nanotechnology*, vol. 15, no. 6, pp. 962–970, Nov 2016.

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

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

[16] 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.

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

[18] 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.

[19] M. Alam, J.DeAngelis, M. Putney, X. Hu, W. Porod, M. Niemier, and G. Bernstein, "Clock Scheme for Nanomagnet QCA," in *International Conference on Nanotechnology*. Hong Kong: IEEE, 2007, pp. 403–408.

[20] M. Vacca, M. Graziano, A. Chiolerio, A. Lamberti, M. Laurenti, D. Balma, E. Enrico, F. Celegato, P. Tiberto, and M. Zamboni, "Electric clock for NanoMagnet Logic Circuits ," *In: Anderson, N.G., Bhanja, S. (eds.), Field-Coupled Nanocomputing: Paradigms, Progress, and Perspectives. LNCS, Springer, Heidelberg.*, vol. vol. 8280, 2014.

[21] 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," May 2019, arXiv:1905.02477.

[22] M. Soeken, H. Riener, W. Haaswijk, and G. De Micheli, "The EPFL Logic Synthesis Libraries," 2018, arXiv:1805.05121.

[23] A. Biere, K. Heljanko, and S. Wieringa, "AIGER 1.9 and beyond," Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, Tech. Rep. 11/2, 2011.

[24] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," in *CAV*, 2010, pp. 24–40.

[25] S. B. Akers, "Synthesis of Combinational Logic using Three-input Majority Gates," in *Annual Symposium on Switching Circuit Theory and Logical Design*, 1962, pp. 149–158.

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

[27] 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.

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

[29] R. E. Formigoni *et al.*, "BANCS: Bidirectional Alternating Nanomagnetic Clocking Scheme," in *SBCCI*, 2018, pp. 1–6.

[30] F. Riente, G. Turvani, M. Vacca, M. R. Roch, M. Zamboni, and M. Graziano, "Topolinano: A cad tool for nano magnetic logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 7, pp. 1061–1074, July 2017.

[31] F. S. Torres, M. Walter, R. Wille, D. Große, and R. Drechsler, "Synchronization of Clocked Field-Coupled Circuits," in *IEEE-NANO*, 2018, pp. 1–5.

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

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

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

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

[36] F. S. Torres *et al.*, "Exploration of the Synchronization Constraint in Quantum-dot Cellular Automata," in *DSD*, 2018, pp. 642–648.

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

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

[39] 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.

[40] U. Garlando, F. Riente, D. Vergallo, M. Graziano, and M. Zamboni, "Topolinano & magcad: A complete framework for design and simulation of digital circuits based on emerging technologies," in *2018 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2018, pp. 153–156.

[41] F. Riente, U. Garlando, G. Turvani, M. Vacca, M. R. Roch, and M. Graziano, "Magcad: A tool for the design of 3d magnetic circuits," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. PP, no. 99, pp. 1–1, 2017.

[42] "Synopsys design compiler," https://www.synopsys.com.

[43] 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.

[44] G. Turvani, F. Riente, M. Graziano, and M. Zamboni, "A quantitative approach to testing in quantum dot cellular automata: Nanomagnet logic case," in *Ph.D. Research in Microelectronics and Electronics (PRIME), 2014 10th Conference on*, June 2014, pp. 1–4.

[45] G. Turvani, F. Riente, F. Cairo, M. Vacca, U. Garlando, M. Zamboni, and M. Graziano, "Efficient and reliable fault analysis methodology for nanomagnetic circuits," *International Journal of Circuit Theory and Applications*, vol. 45, no. 5, pp. 660–680.