Toward Nanometric Scale Integration: An Automatic Routing Approach for NML Circuits

Pedro Arthur R. L. Silva*, José Augusto M. Nacif*
E-mail:{pedro.arthur, jnacif}@ufv.br
*Universidade Federal de Viçosa, Florestal, MG, Brazil

***Abstract*—In the recent years, many technologies have been studied in order to take place as a replacement or complement to CMOS. These emerging technologies, known as Field Coupled Nanotechnologies, seek to operate at nanometric scales and overcome the problems that are hard or impossible to be addressed by CMOS technology. However, these new technologies introduce the need of developing tools to perform circuit mapping, placement, and routing. NanoMagnetic Logic Circuit (NML) is one of these emergent technologies. It relies on the polarization of nanomagnets to perform operations through majority logic. In this work we propose an approach to automatically map a gate-level circuit to a NML layout. We use the Breadth First Search to perform the placement and the A* algorithm to transverse the circuit and build the routes for each node. To evaluate the effectiveness of our approach, we use a series of ISCAS'85 benchmarks. Our results show an area reduction varying from 20% to 60%.**

I. Introduction

For many years and still now, the CMOS (Complementary Oxide Semiconductor) has been considered the standard technology in the manufacturing of digital devices. However, as the density of integrated circuits increases, problems with reliability and power dissipation are rising at an alarming pace. Therefore, new alternative technologies such as Field Coupled Nanotechnologies (FCN) have recently attracted researchers attention [1], [2]. Quantum-Dot Cellular Automata (QCA) [3] and Nanomagnetic Logic (NML) [4] are two of these emerging technologies. NML is a non-volatile FCN and operates based on field interactions between nanomagnets. NML technology operates at room temperature with ultra-low power dissipation [5]. Also, in NML circuits the computation is conducted based on majority logic, that can be used to implement any logic function [6].

The NML basic cell is a bistable nanomagnet whose polarization is likely to lie alongside its long axis, in order to minimize the shape energy. The two possible configurations for the polarization of a NML cell are *up* and *down*, as depicted in Fig. 1, respectively seen as the logic states 1 and 0. A nanomaget interacts with its neighbor through magnoestatic dipolar coupling. Relying on these interactions the information can be propagated in the circuit as ferromagnetic or antiferromagnetic coupling. The behavior of how NML transmits information is possible due to how magnetic interaction works.

As the NML circuits become more complex, the need of a clocking scheme turns out to be more relevant. A NML circuit with few magnets reproduces the coupling correctly. However,

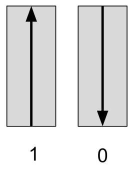


Fig. 1. Polarized NML cells.

as the size of an array of nanomagnets increases, it is most likely that the circuit will present more errors as well [7]. A clock scheme controls the flow of information in the circuit, enabling the design of more complex NML circuits. As the integration levels of NML circuits increase, the design of the circuits becomes a complex process, being unfeasable to be performed manually.

Since the CMOS has been the leading technology to build digital electronic devices, the literature presents a well established set of papers that address the problem of automatically generating a physical circuit layout with CMOS. Although NML can overcome many of the challenges that emerge at nanometric scale, the technology is not fully mature yet, and there is a lack of tools and techniques to aid the process of physical design of NML circuits. Due to the differences between CMOS and NML technologies, and the several constraints presented by the latter, the current tools and techniques are not suitable to generate an efficient NML circuit physical layout. Furthermore, the synchronization constraint added by the use of a clock scheme impacts directly in the circuit area and delay.

Synchronized circuits all have a *throughput* of 1, meaning that at every clock cycle a result arrives at the outputs. A throughput of 0.5, for example, means that the results arrive at the outputs at every 2 clock cycles.

In this paper, we propose the utilization of an alternative algorithm to route wires in NML circuits exploring the synchronization constraint. We apply the A* search [8] to interconnect the components of the circuit. We compare the total area occupied when the circuit doesn't respect the synchronization constraint and when it does.

This work is organized as follows: Section II reviews the basics of NML technology. Section III discusses previous approaches for automatic design of QCA and NML circuits. Section IV presents an algorithm for automatic layout generation of NML circuits. In Section V, we use the ISCAS'85 [9] benchmarks to perform the routing of circuits and an analysis on the performance of our algorithm and the trade between area and *throughput* of the circuit. Finally, Section VI concludes this work.

II. Background

In this Section, we present an overview on Nanomagnetic Logic (NML), explaining the basic devices of the technology, how they can be used to implement complex circuits, and how a clock scheme controls the flow of information in the circuit.

Although a nanomagnet can present several geometries, in this work we assume the rectangular-shape nanomagnet, as shown in Fig. 1. As explained before, an NML cell is basically a bi-stable nanomagnet whose magnetic polarization is likely to lie alongside the longer axis, in order to minimize the shape energy. The two possible configurations for the polarization of an NML cell are *up* and *down*, respectively. When we apply a magnetic field to the shorter axis of a nanomagnet, it changes to a meta-stable state, mapping to a *null* logic state [7].

Wires are the basic element to propagate information in circuits. In order to transmit signals through the circuit, such elements can be arranged in two configurations due to the coupling interactions between the nanomagnets. The two configurations in which the propagation may occur are anti-ferromagnetic and ferromagnetic. Antiferromagnetic, presents antiparallel direction of the magnetization vectors, while the latter, ferromagnetic, presents a parallel orientation. An inverter in NML can be formed exploiting the wire structure.

Wires with an odd number of cells just propagates the information through the wire, but if the wire is composed of an even number of particles it yields to an inverter. Another important element of NML is the majority voter gate. The 3-input majority gate replicates on the output the logic level of majority of the inputs of the gate. We can reduce the majority voter gate to an AND or OR gates by fixing one of the inputs in 0 or 1 logic levels. Therefore, by relying on the majority and inverter gates, it is possible to implement any kind of logic function.

Seeking to guarantee the correct functionality of NML, a clocking system is extremely important. Without a clocking scheme, the circuit is most likely to present meta-stable states, leading to incorrect functionality. The clocking system in NML is composed of three clock zones. Each clock zone is controlled by a periodic clock signal composed of three phases called *Hold*, *Reset*, and *Switch*. In the *Hold* phase, the magnetization of the nanomagnets remain unchanged. In the *Reset* phase, the magnetic field is applied, inducing the nanomagnets into a *null* magnetization state. In the *Switch* phase, the magnetic field is gradually removed, allowing the nanomagnets to polarize according to their neighbors. When a circuit is split into clock zones, the magnetic fields act upon each zone independently, thus eliminating errors. A clock cycle in NML is the time a clock zone needs to pass through all these three phases.

III. Related Work

Considering that QCA and NML are not the leading technologies for integrated circuits manufacturing, there are few works that investigate automatic methods to generate physical layout of circuits in these two technologies. There are three works that mainly contributed to this paper, in two of them algorithms for placement and routing of QCA circuits are proposed [10], [11], and the last one describe the design of a CAD (Computer Aided Design) tool for NML [12].

NML and QCA present some similarities, but there are differences as well. One of the main differences is in the clock scheme, while QCA has four clock zones, NML only counts with three. Also, there are some layout rules unique to NML, such as wires with an even number of nanomagnets acting as inverters. The approach presented in [10] also allows multi-layer layouts. A multi-layer solution for NML technology has not yet been proposed, all NML circuits are single-layer. Therefore, if one desires a NML physical design, the approach presented in [10] must be adjusted to meet this requirement.

Recently, another work that investigates the automatic design of QCA circuits has been proposed [11]. The work proposed an exact method to build circuits, respecting some design objectives and physical constraints. In order to generate a QCA layout for a given circuit the authors relax some of the constraints, as the maximum area of the circuit or whether wirecrossing are allowed or not. In QCA, one of the ways to implement wirecrossing is by relying on the multilayer resource. In [11], the authors compare designs of the same circuit generated with and without wirecrossing, and evaluate the overall impact on the final area of the circuit. But in NML, since we cannot rely on the multilayer resource, we must address this issue as well.

In [12], the authors present a CAD tool able of design, test and simulate NML circuits. The authors explore a series of optimizations in order to achieve an efficient design in terms of area and delay. To perform the desired optimizations, heuristics and metaheuristics such as the *BaryCenter* [13] and *Simulated Annealing* [14] were implemented. Although [12] presents an important step in the automatic design of NML circuits, the investigation of more methods to automatically perform this task is still a work in progress. Our mainly focus on this work is to investigate an alternative approach that can be used to route interconnections in order to generate an physical design of NML circuits. Another important objective of this work is to explore the synchronization constraint of NML circuits. In [15], this analysis has already been made for general QCA circuits, but since NML presents its own unique characteristics, this analysis can lead to different results.

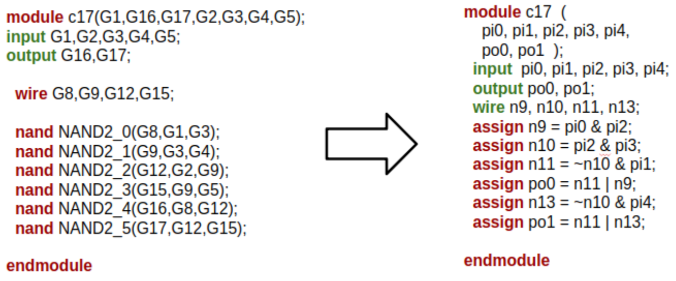
IV. Methodology

Routing is one of the many phases that compose modern integrated circuit design flow. Even though our goal is to explore an algorithm for *Routing* of NML circuits, we have to pass through all the phases that precedes the one we are focusing on. Subsections IV-A and IV-B give more details about the Graph Elaboration and the Placement phase, respectively.

Subsection IV-C explain about our approach to route NML circuits.

A. Graph Elaboration

Our first task is to read a Verilog circuit description and transform it in a graph, that is easier to manipulate. Before we take the Verilog description as input to the parser, we guarantee that all the gates in the description of the circuit have at most two inputs, as shown in Fig. 2. We have to do this because in NML, there are few proposed gates yet, and the wide range of them are 3-input, being one of the inputs a fixed cell. Some works have given attention to majority logic synthesis, such as [16], [17], but yet, most part of the circuit benchmarks work with gates that have a high *fan-in*, that is, high quantity of inputs to manipulate.



```
module c17(G1,G16,G17,G2,G3,G4,G5);
input G1,G2,G3,G4,G5;
output G16,G17;

  wire G8,G9,G12,G15;

  nand NAND2_0(G8,G1,G3);
  nand NAND2_1(G9,G3,G4);
  nand NAND2_2(G12,G2,G9);
  nand NAND2_3(G15,G9,G5);
  nand NAND2_4(G16,G8,G12);
  nand NAND2_5(G17,G12,G15);

endmodule
```

```
module c17  (
    pi0, pi1, pi2, pi3, pi4,
    po0, po1  );
  input  pi0, pi1, pi2, pi3, pi4;
  output po0, po1;
  wire n9, n10, n11, n13;
  assign n9 = pi0 & pi2;
  assign n10 = pi2 & pi3;
  assign n11 = ~n10 & pi1;
  assign po0 = n11 | n9;
  assign n13 = ~n10 & pi4;
  assign po1 = n11 | n13;

endmodule
```

Fig. 2. Fan-in management

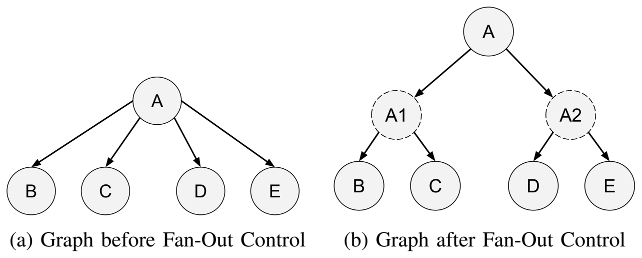
In order to do this, we use ABC [18], a well known tool for logic synthesis. Given a file with a Verilog description of the circuit we execute the script depicted in Fig. 3. First we read the *Verilog* file with the description of the circuit, then we transform it in And-Inverter Graph(AIG), a type of representation of digital circuits that use graphs in which each node is a AND gate. Then, we use the *strash* command to perform a transformation in the structure of the circuit and return a logical network composed of only two-input AND gates and inverters. The last step is to write the structure to a new *Verilog* file with the output circuit that will be given as an input to our parser.

```
read_verilog
aig
strash
write_verilog
```

Fig. 3. ABC script to generate circuits with 2-input gates.

Our implementation of the parser read the Verilog generated by following the script described in Fig.3 and then returns a Directed Acyclic Graph(DAG) in which each node is a logical element of the circuit and an edge going from a node *u* to a node *v* means that the former node is an input of the latter. Before we can go forward to the next phases, we perform the *Fan-out* management as proposed in [12], to control the number of inputs that a determined gate can fed, that is, to how many other gates its inputs are connected. We also perform the graph balancing in order to guarantee the synchronization of the circuit.

1) Fan-Out Management: In a DAG that represents a digital circuit, each edge between two nodes indicates that in the final NML circuit design there is a wire connecting those two logic elements. The number of inputs a logic gate can drive is called *fan-out*. In both CMOS and NML, a gate has a limit to how much others gates it can fed. Generally, digital circuits do not respect these limits. Therefore, we must perform a search in the graph for the nodes that violate this constraint and then map them to nodes that are in compliance with this constraint. This can be done by adding children to a node and then redistributing its old children as children of the new nodes. In Fig. 4, we depict the result of the *fan-out* control operation. In Fig. 4a we depict a graph that doesn't respect the fan-out constraint. Fig. 4b shows the same graph but with a constraint of a *fan-out* of two outputs. Before, node *A* was input to four other nodes, to solve this, we introduce the nodes *A1* e *A2* as its new children and then split the four old children among the nodes we have just created.



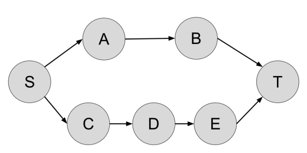
(a) Graph before Fan-Out Control (b) Graph after Fan-Out Control

Fig. 4. FanOut management of circuit.

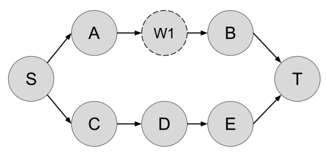
2) Reconvergent Paths - Graph Balancing: The synchronization of NML circuits are extremely related with the length of the paths in the circuit. The distance between a logic element and each one of its inputs must be the same. This phenomenon is know as the *layout=timing* problem [19], inherent to QCA and NML. All the paths leading to the same logic element must have the same delay in terms of clock cycles. If an input signal arrives at its output before any of the others, an unexpected computation is performed. These paths are named reconvergent paths.

Two paths are called reconvergents if they diverge from and then reconverge to the same logic element or block [20]. In [10], the authors take advantage of the reconvergent paths in order to elaborate their approach. In this work, as in [12], we choose to balance the reconvergent paths before generating a physical layout of the circuit. In order to balance the reconvergent paths, we introduce wire nodes in all the unbalanced paths. This step is essential to eliminate asynchronicity from the circuit. However, this leads to an inevitable increase in the area of the circuit. Fig. 5a depicts an unbalanced graph. One can see that are two paths that start in *S* and end in *T*, however, the size of these two paths aren't the same. So, we introduce a wire node *W1*, yielding the graph in Fig. 5b.

In Section V we investigate more about the impact of circuit



(a) Unbalanced Graph



(b) Balanced Graph

Fig. 5. Reconvergent Paths.

balancing in the final area.

B. Placement

After the graph elaboration is complete, the next step is generate a mapping the circuit to a grid in which the componets can be allocated and then interconnected. This task can be split into two major subtasks: *Placement* and *Routing*.

The *Placement* of a circuit is the stage in which we define the physical position of each logical element in the circuit. In this phase of the project, one of the main concerns is to guarantee that we have enough area left, so we can interconnect the elements in the *Routing* stage. As the number of elements in the circuit increases, the *Placement* becomes more complex. We have to avoid the congestion of regions, because this leads to a poor *Routing* of the circuit, since more wires are necessary to work around a congested area, what leads to a bigger solution space, therefore impacting in the execution time of the algorithm and the final area of the circuit.

In this work, we seek to find a better placement by topologically ordering the graph of the circuit. What we try to do is to put a parent right above its children, this way, we can reduce the total wire length of the circuit, since the adjacent nodes will be physically near to each other. In order to achieve our goal, we utilize the *Breadth-First Search* (BFS) algorithm [21]. This algorithm starts the search from a given initial node, discovering others nodes from the neighbors. This is done with all the nodes until there are no more nodes left in the graph. Our BFS search starts from the inputs of the circuit, assigning a position for each node as soon as they are discovered in the search. If a node is visited more than once, the first position assigned for it prevails. We do not perform further optimization in the placement phase, since in this work our focus in the routing phase.

C. Routing

Once the position of each element is defined, we can finally perform the *Routing* of the circuit. This stage is responsible for connecting the inputs and outputs of the components in the circuit, that is, finding routes between the connected logic elements. We use the A* search algorithm [22] to perform the routing. This approach is widely used in problems in which one needs to find a path while avoiding obstacles. In our case, the obstacles are the congested areas, that is, areas in the circuit in which we cannot route wires anymore.Since in NML the wire crossing is only possible when it involves only two wires, we define an area as congested when we have more than two wires crossing at the same point.

To build paths, the A* search relies on an evaluation function to estimate the total cost of the path through a node *N* to the goal. This function is depicted in Equation 1. In the evaluation function, *g(N)* is the cost so far to move from the source point to *N*. The factor *h(N)* is the estimated cost from *N* to goal. One has to pick the best heuristic in order to estimate the cost *h(N)* according to the nature of problem.

$$f(N) = g(N) + h(N) \tag{1}$$

When transversing the solution space, the algorithm will look for the candidate that has the lowest cost of *f(N)*. The heuristic we choose to the apply to the routing problem is the *Euclidean Distance* between two points. Considering a node *N* and a *goal*, Equation 2 depicts the calculation of this heuristic.

$$\begin{aligned} dx &= abs(N.x - goal.x) \\ dy &= abs(N.y - goal.y) \\ h(N) &= \sqrt{dx^2 + dy^2} \end{aligned} \tag{2}$$

In Fig. 6 we depict how A* makes intelligent choices at each step. The red block is the starting point, the gray blocks are not available, so the search has to go around them and arrive in the target, the green block. One can see that the algorithm goes from the position (4,2) to (5,3) instead of (4,3). Similarly, the algorithm goes from (5,3) to (6,2) instead of (6,3).

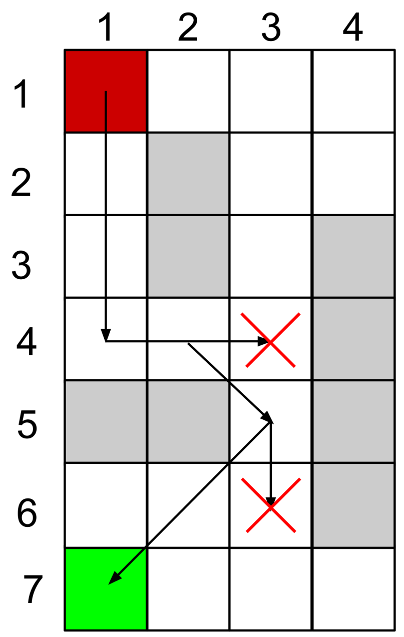


Fig. 6. A* algorithm pathfinding.

Our algorithm transverse the graph that represents the circuit from the inputs to the outputs layer by layer. In each layer, we apply the A* search to the nodes, connecting it to all its children in the next layer. At the end of the traversing we accomplish to build all the interconnections of the circuit, completing the *Routing* phase.

Each time a position is chosen to be part of a route, that is, an interconnection between two elements, that position is marked, so we can control how much wires are passing through a location. When a position has two wires passing through it, then we map it to a Crossover element. Once a position is mapped to a Crossover, we mark it as a blocked position, because it is impossible to implement a Crossover with more than two wires in NML.

In NML, we do not have problems with diagonal paths as the one depicted in Fig. 6. In fact, in some situations this kind of paths are quite helpful, since it acts as a ferromagnetic wire.

V. Results

In this section we focus on analysis and comparison of the proposed algorithm for circuit routing and also the impact of path balancing in NML circuits.

To this end, our algorithm has been implemented using the C++ programming language and all evaluations have been conducted on an Intel Core i5-7200U machine with 2.50 GHz (up to 3.10 GHz boost) and 8 GB of RAM memory.

A. Benchmarks

To perform our analysis, we selected some benchmark circuits [9], [23], [24], and also included some circuits generated using the ABC synthesis tool [18]. Our benchmarks are presented in Table I.

TABLE I
Information of the benchmarks.

Benchmark	Gates	Inputs	Outputs
c17	12	7	5
t	15	10	5
newtag	20	12	8
FA-AOIG	12	9	3
B1_r2	16	13	3
XOR5_r	37	32	5
XOR5_r1	31	26	5

B. Circuits routing comparison

Since previous works do not focus on the routing phase of circuit project, we do not have a fair way to compare the results of our work. Hence, we have to find a way to evaluate the performance of our algorithm. So, we work with the graphs before and after the balancement explained in Section IV-A. Table II depicts the comparison of execution times for each graph. We executed the algorithm 30 times for each circuit and then take the mean time of the executions.

TABLE II
Table of comparison of execution times for the balanced and unbalanced graphs.

	Running Time (seconds)	
Benchmark	Synchronized	Unsynchronized
c17	0.126713	0.73023
t	0.148908	0.084475
newtag	0.136516	0.084625
FA-AOIG	9.44082	0.100597
b1_r2	0.163079	9.66193
xor5_r	18.6761	0.183467
xor5_r1	0.268849	0.19861

One can notice that, generally speaking, the A* search takes more time to perform the routing of synchronized circuits. This makes sense since this set of circuits has a high number of nodes than the second one. However this is not a absolute rule. The unsychronized *b1_r2* takes more time to be fully routed than its synchronized version. This can occur when we face circuits with higher number of blocked cells, that is, *crossover* along the algorithm. In general, one can notice that the algorithm has shown a relatively small time to interconnect all the elements in the circuit. This suggests that A* is a good choice for routing NML circuits. To the best of our knowledge we are the first to apply the A* search to solve the *Routing* phase of NML circuits.

C. Graph elaboration impact

In [10], [11], the authors do not perform a preprocessing in the circuit graph in order to manage the maximum *fan-out* allowed. Once we choose to add this step to our approach we increase the number of nodes. Also, the synchronization effect inherent to NML circuits may lead to a huge increase in the number of nodes in the circuit, which lead to circuits that consume more area. In [15], the authors show that by relaxing the synchronization constraint, we can achieve less area circuits. Table III shows the new areas for the set of benchmarks we are working. The table depicts information regarding the graph before and after we perform the operation cited in Section IV-A.

The latency column indicates how long, int terms of clock zones, is the largest path in the graph, that is, the critical path. One can notice that the latency for both graphs are the same, this occurs because after we apply the operations of *fanout* management and path balancing, the critical path of the original circuit remains intact. Hence, the latency of the synchronized circuit is equal to the original one. It is worth to mention that this only occurs because we are performing a high level evaluation of only the nodes in the graph instead of the actual circuit.

In Table III, we calculate the fourth column for the unsynchronized circuits according to the method described in [15]. Just by looking to the number of nodes, one can notice that, generally speaking, unsynchronized circuits consumes less area than synchronized ones. When we look to the grid area, then we see that this really applies in the practice, we have reductions in the grid area varying from 20% to nearly 60%.

TABLE III
SYNCHRONIZATION COMPARISON

				Synchronized			Not synchronized			
Benchmark	**Gates**	**Inputs**	**Outputs**	**Grid area**	**Number of nodes**	**Latency**	**Grid area**	**Number of nodes**	**Latency**	**Throughput**
c17	12	7	5	40	23	6	32	12	6	0.5
t	15	10	5	40	25	5	32	15	5	0.5
newtag	20	12	8	70	27	10	70	20	10	0.333
FA-AOIG	12	9	3	56	26	10	24	12	10	0.2
B1_r2	16	13	3	50	23	6	16	16	6	0.5
XOR5_r	37	32	5	216	64	28	88	37	28	0.1
XOR5_r1	31	26	5	216	58	28	88	31	28	0.1

VI. CONCLUSION

NML is a novel technology that has shown promising results, but that are few tools to aid the design of circuits in this nanotechnology. The work in [12] is one of the few works in the literature regarding NML assisted by CAD tools.

In this work, we propose a way to perform the routing of NML circuits and, to the best of our knowledge, we are the first to utilize the A* algorithm in order to perform the interconnection of components. The chosen algorithm has presented a promising performance in terms of speed, and also it show some characteristics that are very suitable to design NML circuits. We also studied the impact of synchronization constraint on NML circuits, showing that is possible to make a trade between area and delay, providing the designer of the circuit more flexibility when building NML circuits.

As future work, we plan to investigate more about the routing phase, applying other algorithms, and also explore the placement.

ACKNOWLEDGMENTS

We would like to thank CAPES, CNPq, and FAPEMIG for the financial support.

REFERENCES

[1] R. K. Cavin, P. Lugli, and V. V. Zhirnov, "Science and engineering beyond moore's law," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1720–1749, 2012.

[2] T.-J. K. Liu and K. Kuhn, *CMOS and beyond: logic switches for terascale integrated circuits*. Cambridge University Press, 2015.

[3] C. S. Lent and P. D. Tougaw, "A device architecture for computing with quantum dots," *Proceedings of the IEEE*, vol. 85, no. 4, pp. 541–557, 1997.

[4] M. Niemier, G. H. Bernstein, G. Csaba, A. Dingler, X. Hu, S. Kurtz, S. Liu, J. Nahas, W. Porod, M. Siddiq *et al.*, "Nanomagnet logic: progress toward system-level integration," *Journal of Physics: Condensed Matter*, vol. 23, no. 49, p. 493202, 2011.

[5] R. Cowburn and M. Welland, "Room temperature magnetic quantum cellular automata," *Science*, vol. 287, no. 5457, pp. 1466–1468, 2000.

[6] A. Imre, G. Csaba, L. Ji, A. Orlov, G. Bernstein, and W. Porod, "Majority logic gate for magnetic quantum-dot cellular automata," *Science*, vol. 311, no. 5758, pp. 205–208, 2006.

[7] T. R. Soares, J. G. N. Rahmeier, V. C. De Lima, L. Lascasas, L. G. C. Melo, and O. P. V. Neto, "Nmlsim: a nanomagnetic logic (nml) circuit designer and simulation tool," *Journal of Computational Electronics*, vol. 17, no. 3, pp. 1370–1381, 2018.

[8] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[9] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," in *Proceedings of IEEE Int'l Symposium Circuits and Systems (ISCAS 85)*. IEEE Press, Piscataway, N.J., 1985, pp. 677–692.

[10] G. Fontes, P. A. R. Silva, J. A. M. Nacif, O. P. V. Neto, and R. Ferreira, "Placement and routing by overlapping and merging qca gates," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 1–5.

[11] M. Walter, R. Wille, D. Große, F. S. Torres, and R. Drechsler, "An exact method for design exploration of quantum-dot cellular automata," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 2018, pp. 503–508.

[12] 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, 2017.

[13] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 2, pp. 109–125, 1981.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[15] F. S. Torres, P. A. Silva, G. Fontes, J. A. Nacif, R. S. Ferreira, O. P. V. Neto, J. Chaves, and R. Drechsler, "Exploration of the synchronization constraint in quantum-dot cellular automata," in *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE, 2018, pp. 642–648.

[16] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014, pp. 1–6.

[17] L. Amarù, E. Testa, M. Couceiro, O. Zografos, G. De Micheli, and M. Soeken, "Majority logic synthesis," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, 2018, p. 79.

[18] A. Mishchenko *et al.*, "Abc: A system for sequential synthesis and verification," *URL http://www. eecs. berkeley. edu/alanmi/abc*, p. 17, 2007.

[19] M. T. Niemier and P. M. Kogge, "Problems in designing with qcas: Layout= timing," *International Journal of Circuit Theory and Applications*, vol. 29, no. 1, pp. 49–62, 2001.

[20] S. K. Lim, R. Ravichandran, and M. Niemier, "Partitioning and placement for buildable qca circuits," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 1, no. 1, pp. 50–72, 2005.

[21] A. Bundy and L. Wallen, "Breadth-first search," in *Catalogue of Artificial Intelligence Tools*. Springer, 1984, pp. 13–13.

[22] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[23] S. Yang, "Logic synthesis and optimization benchmarks," Tech. Rep., Dec. 1989.

[24] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Circuits and Systems, 1989., IEEE International Symposium on*, May 1989, pp. 1929–1934 vol.3.