FUNCODE: Effective Device-to-System Analysis of Field Coupled Nanocomputing Circuit Designs

FUNCODE: Effective Device-to-System Analysis of Field Coupled Nanocomputing Circuit Designs

U. Garlando, *Member, IEEE,* F. Riente, *Member, IEEE,* and M. Graziano, *Member, IEEE*

***Abstract*—Many beyond-CMOS technologies, based on different switching mechanisms, are arising. Field-coupled technologies are the most promising as they can guarantee an extremely low-power consumption and combine logic and memory into the same device. However, circuit-level explorations, like layout verification and analysis of the circuit performance, considering the constraints of the target technology, cannot be done using existing tools. Here, we propose a methodology to take on this challenge. We present FUNCODE (FUNction & COnnection DEtection), an algorithm that can detect element connections, functions and errors of custom-layouts and generate its corresponding VHDL netlist. It is proposed for in-plane and perpendicular Nano Magnetic Logic as a case study. FUNCODE netlists, which take into account the physical behavior of the technology, were verified using circuits with increasing complexity, from 6 up to 1400 gates with a number of layout elements varying from 200 to 2.3e6.**

***Keywords*—Emerging technologies, beyond CMOS, system analysis, circuit design, algorithm**

I. Introduction

THE continued scaling of CMOS transistor feature sizes is approaching not only the physical limit [1], but it also faces economic and technological challenges [2]. Many emerging devices are arising based on different switching mechanisms [3], ferroelectric FETs [4] and RRAM [5] are just one example. Among them, field-coupled technologies are the most promising [6] due to their extremely low power consumption, the ability to combine logic and memory into the same device [7] and the possibility to monolithically integrate such functional layers within the same device. These characteristics perfectly fit the Logic-in-Memory principle that can take great advantages from these technologies. Researchers are working on the device level [8] [9] [10], making technological progresses towards more reliable thin films, the possibility to support 3D integration [11] and compatibility with CMOS [12] [13]. However, the exploration of architectural solutions using these devices to find suitable applications [14] [15] is important, even at the early stage of development. Micromagnetic simulations cannot be used for this kind of analysis. A high-level design approach is mandatory for this purpose. The study of an emerging technology starts from the device. However, the single device, or the small circuit with few elements, is not enough to determine the real potential and drawbacks of the technology. The system-level analysis gives another point of view, which makes it possible to discover the benefits and disadvantages of emerging technologies [16] [17]. Furthermore, the analysis should be performed taking into account all the technological properties of the devices. It is important to study the effect of device parameters at the architectural level. Combining all these aspects it is possible to derive the features of a technology and also give feedback to the technologist to improve it. We think that system-level explorations are as important as low-level studies to identify killer applications for a particular technology.

We propose a solution that enables quick architectural investigations, using the high-level description language VHDL (Very high speed integrated Circuits Hardware Description Language), without ignoring the physical constraints introduced by the technology. For this reason, we propose FUNCODE (FUNction & COnnection DEtection), a netlist generation algorithm that can detect element connections, functions and errors of custom-made layouts. Here it is proposed for two emerging technologies, in-plane Nano Magnetic Logic (iNML) [18][19] and perpendicular Nano Magnetic Logic (pNML) [20] [21]. However, the algorithm can be extended to new interesting devices if their element interactions are known and their connections can be defined. FUNCODE was introduced in the custom layout editor MagCAD and is compatible with the ToPoliNano framework. It was tested on a variety of circuits, with increasing complexity. The remaining part of this paper is organized as follows. Section II provides the technological background of investigated technologies (iNML and pNML). Section III presents the methodology adopted and the contexts it was applied to. In section IV the details of the FUNCODE algorithm are described. Section V reports the execution time considering different circuits as benchmarks. Finally, conclusions are given in section VI.

U. Garlando, F. Riente and M. Graziare are with the Department of Electronics and Telecommunication Engineering, Politecnico di Torino, 10129 Torino, Italia. Corresponding author: F.Riente (fabrizio.riente@polito.it).

II. Background

In this section, an overview of the computational paradigm behind NML technology is provided. The information propagation mechanism is at the basis of the algorithm proposed. Differences between the iNML and pNML implementations are highlighted to clarify how the algorithm adapts to their behavior according to the signal flow.

A. in-plane Nano Magnetic Logic

In this technology, single domain nanomagnets are used as a basic computational cell. The shape anisotropy is exploited to encode the binary information. Rectangular shaped nanomagnets are usually preferred with a dimension of (50x100x20)nm or (60x90x20)nm [18]. They show only two stable configurations, which represent the logic 0 and logic 1 (Fig.1.A). Thus, the magnetization vector lies parallel to the plane where

the magnets are placed [22]. The magneto-dynamic interaction among neighboring devices makes it possible to propagate the digital information through the circuit. Magnets arranged in a row try to reach the minimum energy configuration, i.e. they are antiferromagnetic (AF) coupled and align themselves in an antiparallel way (Fig. 1.B). On the other hand, nanomagnets aligned vertically are coupled ferromagnetically (F) (Fig. 1.C). As described in [23], the coupling among neighboring cells

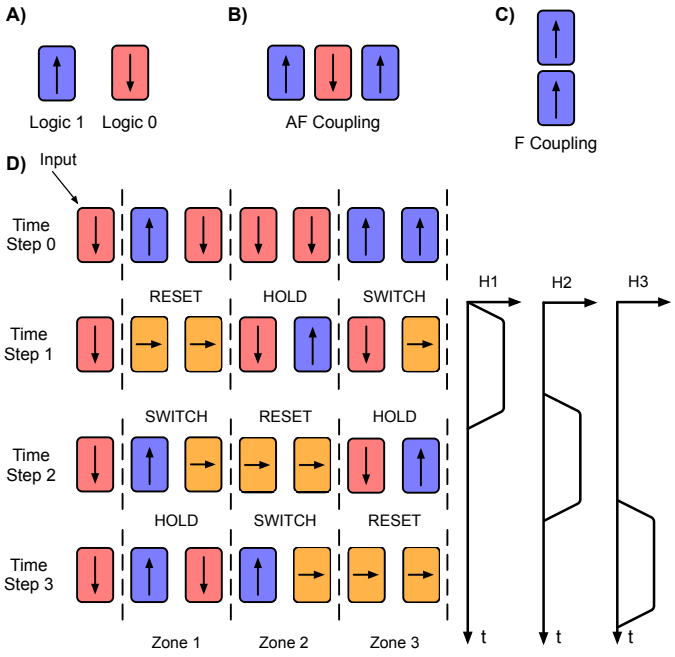

Fig. 1. A) iNML basic cells; B) Chain of nanomagnets antiferromagnetically coupled; C) Vertically aligned nanomagnets ferromagnetically coupled; D) iNML three phase clock system;

is not enough to obtain the switching of the nanomagnets. The energy barrier introduced by the shape anisotropy is too high to be overcome by the dipole-dipole interaction. Hence, magnets need to be forced into a metastable state to propagate the information correctly. An external field generated by a current wire buried inside the substrate is usually used to force the magnets into an unstable state, which is called the reset state. This external agent is called the clock and rotate the magnetization vector along the short axis by 90° [24]. Once the external field is released, the magnets re-align themselves according to the dipole-dipole interaction. Unfortunately, due to the thermal noise the number of magnets that can be cascaded is limited to five or six [18]. In the literature, several clocking mechanisms have been proposed to overcome this limitation. The most common is a three-phase clock scheme, where three partially overlapped clock signals are alternatively applied as depicted in Fig. 1.D. The picture summarizes the topology of common iNML circuit that in this particular case is a magnetic wire. The layout is divided in slices, named clock zones, and by applying in sequence the three clock signals the digital information propagates towards the output. Time step 0 shows the initial configuration of the magnets. At time step 1, the clock signal H1 is applied to the zone 1, forcing the magnets belonging to that zone into the metastable state (RESET). At time step 2, the clock signal is released from zone 1, consequently the magnets from that zone go into the SWITCH state, while zone 2 is forced into the reset state. In a similar way, at time step 3, zone 1 is retains its magnetization (HOLD state), zone 2 is switches, while zone 3 moves into the reset state.

Logic operations are achieved by iNML gates that are obtained by properly arranging the nanomagnets in the layout or changing their geometry. The basic gates are majority voters and inverters: the former is achieved by surrounding a central element with three magnets (Fig. 3.A). The latter, is obtained by chaining an odd number of magnets in the clock zone. Finally *AND* and *OR* gates are obtained with magnets with a slanted edge [25], in this way a preferred direction is set, and the logic function is defined.

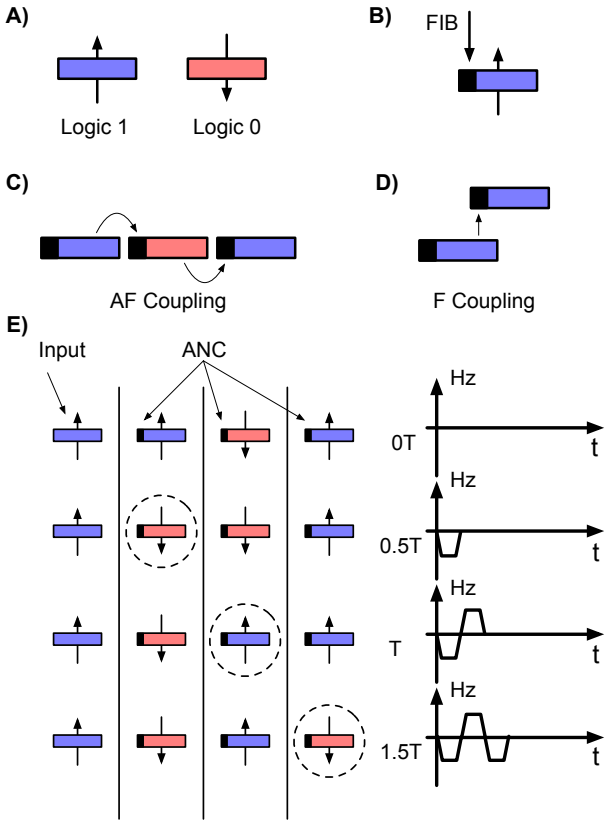

Fig. 2. A) pNML basic cells; B) Artificial nucleation center definition using FIB irradiation; C) Chain of nanomagnets antiferromagnetically coupled; D) Vertically aligned nanomagnets ferromagnetically coupled can transfer the information to adjacent layers; E) pNML clock system example;

B. perpendicular Nano Magnetic Logic

In pNML technology, the elementary cell is characterized by a perpendicular magnetic anisotropy obtained with a multilayer stack of Co/Pt or Co/Ni [26]. The thickness and the

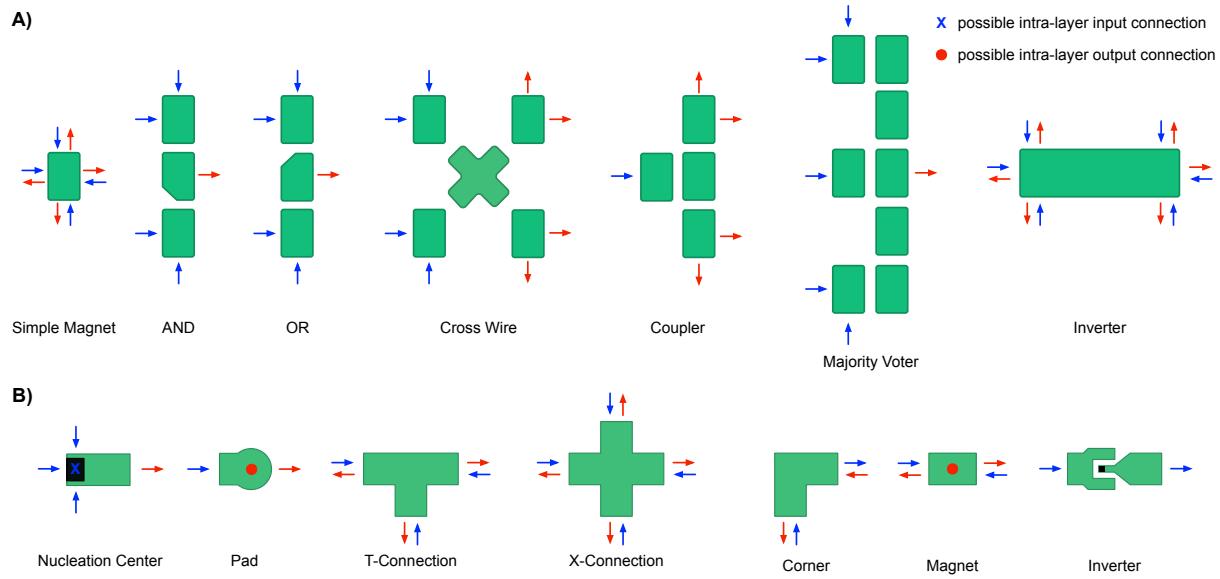

Fig. 3. A) iNML elements and corresponding available connections. Inputs are blue, while outputs are the red arrows. B) pNML elements and relative connections. Blue crosses show input coming from another layer, while red circles are output interconnections towards adjacent planes.

number of layers define the magnetic properties of the device [27]. Even this implementation shows two stable configurations, which encode the binary values. In contrast to the iNML implementation, here the magnetization vector points out-of-plane and the magnet's behavior does not depend on its shape. The magnetization pointing upwards represents logic 1, whereas logic 0 is represented by the downwards magnetization vector (Fig. 2.A). To control the switching behavior of the nanomagnets one side of the magnet is Focus Ion Beam (FIB) irradiated during the fabrication process (Fig. 2.B). The irradiated spot shows a locally reduced anisotropy and determines the region where the domain wall nucleates. It is named the artificial nucleation center (ANC) and defines the propagation direction of the information. As can be observed from Fig. 2.C, magnets lying on the same physical plane are coupled antiferromagnetically (AF). The ferromagnetic coupling is exploited to transfer the digital information from one physical plane to another (Fig. 2.D). Moreover, pNML makes it possible to design monolithically integrated 3D circuits as experimentally demonstrated in [11] [12]. However, even in this technology, the coupling field is not strong enough to reverse the magnetization of neighboring cells. In this case, a single, global and perpendicular clock field is needed to lower the energy barrier and switch the magnets. The working principle is depicted in Fig. 2.E. At time step 0, there is no clock field applied and the input magnet is magnetized upwards. During the first half clock cycle, a negative field is applied and the magnet close to the input flips its magnetization (AF coupling). When the positive field is applied (1T) the second magnet switches. The example depicted in Fig. 2.E demonstrates the relevance of the ANC to define the signal flow. The basic computational cells are the inverter and the N-input minority voter (also called negated majority). The former can be simply achieved by cascading two nanomagnets, the latter can be obtained by surrounding the ANC by an odd number of inputs. The 3D integrability offered by the pNML technology offers more flexibility in the minority gate design [11]. Inputs can be arranged on the same physical plane of the output, or in different planes which exploits both AF and F coupling [28].

III. Methodology

In this section, the methodolody adopted for the VHDL generation algorithm is described. The proposed solution is used to extract a VHDL netlist from a custom layout designed using MagCAD [29]. MagCAD is a custom circuit design tool and the user can select a technology and design a new circuit simply by placing the building block of the technology in the drawing area. The user has complete freedom of choice during the design phase, thus errors could occur. This approach makes it possible to quickly investigate architectural solutions using these emerging technologies. Moreover, the whole algorithm has been designed in a general way so that it may be easily extended to new devices. The effort for the introduction of a new technology requires the development of a software plugin. The plugin requires the definition of:

- the building blocks of the technology, their graphical representation and their compact model
- the connectivity that each building block supports
- how to resolve signal propagation ambiguity inside the circuit if any

The software plugin is implemented starting from the base classes available in MagCAD. The plugin interacts with FUNCODE and is completely developed in C++. Instead, the compact model of every block can be implemented in VHDL. This language is preferable in order to exploit existing VHDL

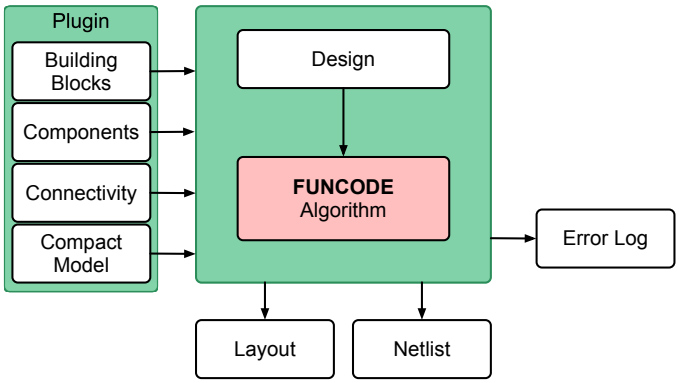

Fig. 4. Design methodology in which the FUNCODE algorithm has been introduced

simulators. The algorithm, with these information, determines the function implemented by groups of building blocks. Indeed, according to the surrounding elements and the signal direction, specific logic functions can be implemented.

First, the connectivity of the custom layout is verified according to technology-dependent rules. Fig. 3 reports the connections allowed for every building block of both iNML and pNML technology. For example, in Fig. 3.A the central magnet of the *Coupler* and the *Majority Voter* implement a different function according to the neighboring elements. Moreover, many building blocks do not have a predefined connection. Thus, the final connectivity can be determined with an overview of the whole layout by looking at how the digital information moves from input to output.

As an example, in the iNML wire described in Fig. 1.D, the clock zone sequence can be helpful for determining the signal propagation. A similar approach can be applied to pNML circuits. The signal flows from ANC to the opposite side of the magnetic nanowire.

After defining the connectivity of the building blocks, the basic gates are identified before writing the final VHDL netlist. Fig. 4 summarizes the design flow in which the proposed solution has been adopted. It is possible to use previous designed circuits, stored in a user library, as building blocks. In this case too, the connections are defined by the algorithm and a hierarchical netlist is provided to the user. The generated netlist could be simulated using a standard VHDL simulator, like ModelSim [30]. In the following sections, a detailed description of the main steps is presented.

IV. FUNCODE Algorithm

The proposed algorithm, FUNCODE (FUNction & COnnection DEtection), can be used for circuits based on both the technologies presented in section II-A and II-B. The general flow of the algorithm is summarized in Fig. 5. It starts from the inputs of the design, correctly connects each element of the circuit, determines the function implemented by an element based on the neighboring cells and then writes the final VHDL netlist.

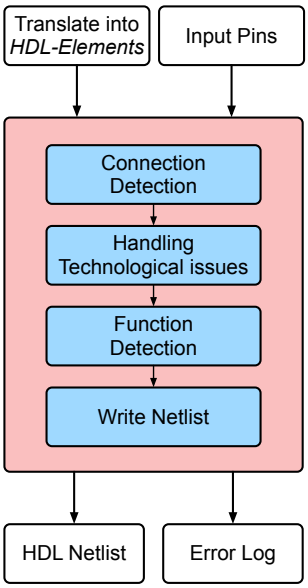

Fig. 5. FUNCODE flow chart

Firstly, all the elements of the layout are translated into their corresponding *HDL-Element*. This data structure and the list of circuit inputs are taken as input by the algorithm. The *HDL-Element* stores the information about the available connections. The connectable cells are set according to the source element. As an example, the *Simple Magnet* in iNML technology has a total of eight possible connections. Each side of the magnet can be both input or output (Fig. 3.A). On the contrary, the pNML Magnet has four connections available, which means that only two sides can be connected (Fig. 3.B). However, in both cases it is not possible to determine in advance which is the input and therefore the output.

Fig. 3 shows examples of connections for the elements of both iNML and pNML technology. From this figure it is possible to observe that some elements have pre-defined connectivity.

Each time an element in the layout is translated, an *HDL-Element* is allocated in the memory. As the circuit gets bigger, the matrices become incredibly large and sparse. Therefore, the position of the *HDL-Element* is inserted into two special table structures, which henceforth are called sparse matrices, (Fig. 6.C). In the case of multi-layer (3D) layouts, a vector of these sparse matrices is used, where the layer number is used as the index in the vector. The sparse matrices are implemented as vectors and a binary search is used to insert and read the data. This approach has been preferred in order to limit the required memory to just the actual number of elements in the layout. Fig. 6.B and Fig. 6.C show the *HDL-Elements* translation, comparing the matrix and sparse matrix representations respectively. Even if the circuit reported in Fig. 6.A is quite simple, it is possible to observe that for large circuits a lot of memory can be saved. Usually the number of building blocks enclosed within the bounding box of the circuit is much smaller than the total number of cells in the grid.

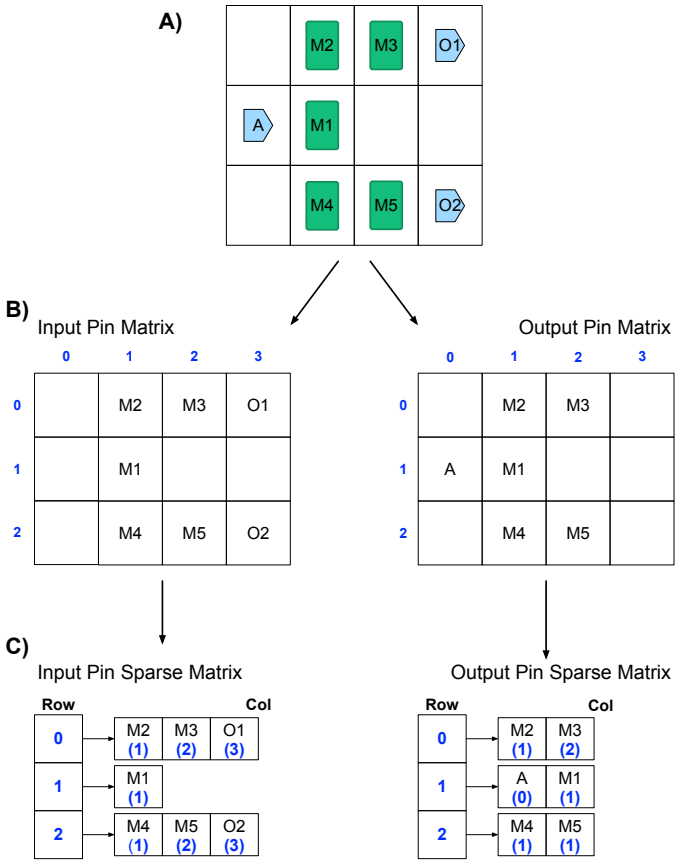

Fig. 6. Example of how to circuit is translated in memory. A) Example of iNML coupler; B) Example of translation using two matrices; C) Example of translation using two vectors.

Furthermore, some elements are not in both the input/output data structures. As an example, input pins are only available in the output sparse matrix, while output ones only in the input (Fig. 6.B). This distinction is necessary due to the fact that from the netlist point of view, inputs can provide an output connection, while outputs can receive an input connection from a neighboring cell. The vector representations are used for fast look-up during the element connections: the structure can be accessed with the coordinates of the element in the layout and the output is available in logarithmic time.
When this setup phase is completed, the main algorithm can start.

A. Connection Detection

The first step of the algorithm is responsible for the detecting the real connections among elements and therefore how the information propagates inside the circuit. The pseudocode is summarized in algorithm 1. Only some elements have their inputs and outputs pre-determined, while the others have their direction defined as they are linked to their neighboring cells. Since each element has different connection, at line 2 the

Algorithm 1 Connection detection pseudocode

Precondition: $Inputs$ and $Outputs$ are the two sparse matrices, $start$ is the list of the input pins of the circuit.

```
1:  for each element ∈ start do
2:      connectable ← element.GetConnectableCells()
3:      for each cell ∈ connectable do
4:          if inputs.contains(cell) then
5:              input ← inputs[cell]
6:              if cell.canConnect(input) then
7:                  CreateConnection()
8:                  input.SetEffectiveDirection()
9:                  start.insert(outputs[cell])
10:             else
11:                 return ConnectionError
12:             end if
13:         else
14:             return ConnectionMissing
15:         end if
16:     end for
17: end for
```

connectable cells are extracted. The list of connectable cells, with all the directions available for output interconnections, is used to access the input sparse matrix. Considering the layout in Fig. 6.A, pin "A" is the only element of *start*. This pin, according to its orientation, can be only be connected to its right. The input sparse matrix is accessed at $(1, 1)$ and "M1" is returned. The function at line 6 is used to determine if it is possible to create a connection between the current pin and the one extracted from the sparse matrix. The presence of the cell in the sparse matrix is not enough to determine if a connection is possible. The input connection of the target pin could already be occupied. Moreover, the orientation of the element in the layout provides information about its connectable sides. Indeed, the connection cannot be achieved with an element on a non-connectable side. When it is not possible to create a new connection, an error is generated. The error is used to provide a report message to the user. On the contrary, if the connection is available, it is formed. Each *HDL-Element* has the name of its output signal. In order to form a connection, the output name is inserted in the vector of input of the target cell. This information is useful when writing the VHDL netlist. In particular, the signal name is used in the *port map* of the corresponding cell. Once the new connection is defined, an element-specific function is called to define the propagation direction (line 8). These operations are repeated for all the possible outputs of the current cell, and each new connected element is added to the *start* list. When an output pin is reached, nothing is added to the *start* list, thus nothing is present in the corresponding position in the output sparse matrix. Therefore, the loop ends when all the elements have been visited.
As an example, consider Fig. 7.A. The *X-Connection* has connectable pins in every direction. As introduced in section III, the number of pins is doubled, i.e. each side could be both input or output. During the execution of the algorithm, the left input comes from an input pin of the circuit. When that connection is defined (Fig. 7.B), the redundant pins need to be removed. The one connected is the only available input and

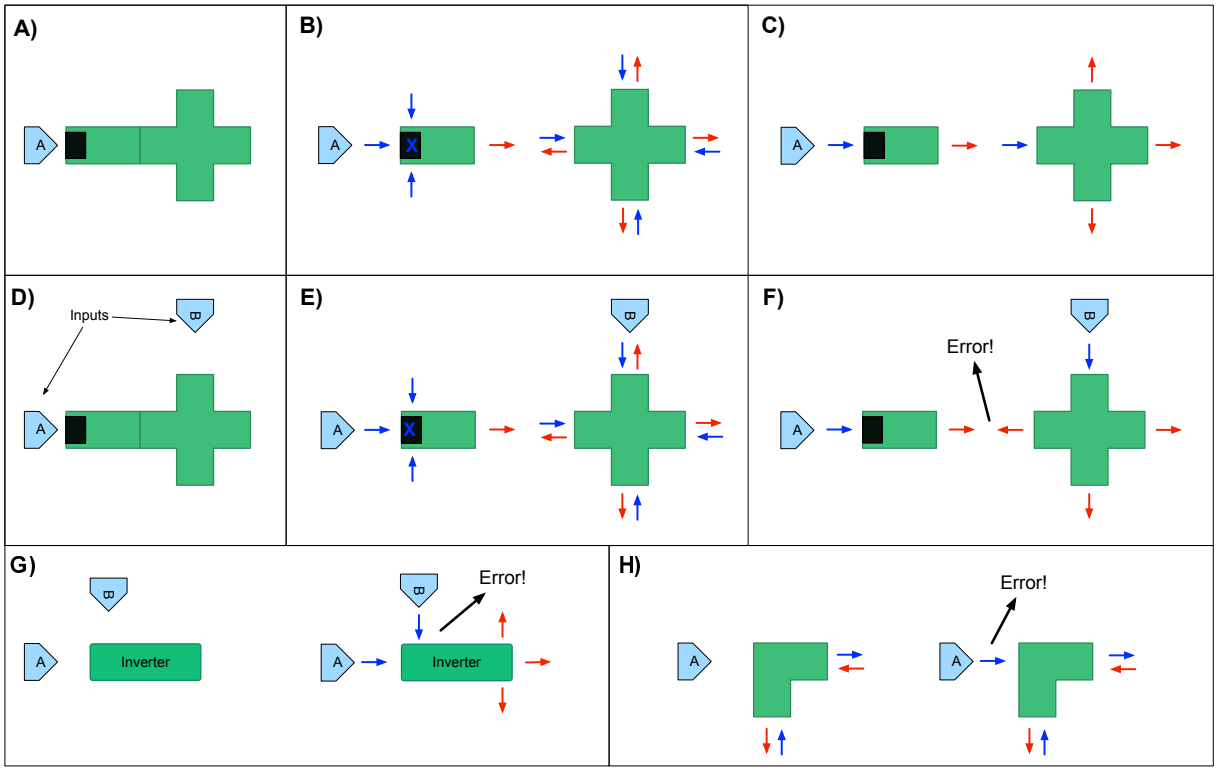

Fig. 7. Elements connectivity examples. A) shows a portion of a pNML circuit with two elements and one input pin. B) highlights the connection available for the elements and C) shows the correct connection detection. D) represents the same circuit with an extra input. E shows the connection available and F) is an example of output-output error. G) shows a portion of iNML circuit where an inverter has two neighboring pins. H) highlights the error due to multiple input connection to a single input element. I) is a pNML corner with a pin placed where there are no available connections, generating the error in L).

the others are set as outputs, as shown in Fig. 7.C.
Fig. 7.D shows an example of an incorrect layout. Two input pins are placed close to the *X-Connection* element. The first connected input sets the actual direction of the *Nucleation Center* cell. Subsequently, input "B" is connected defining the actual direction of the *X-Connection*. At this point, when it tries to connect the two elements, an output/output error occurs (Fig. 7.F). Another possible error is shown in Fig. 7.G: the first pin will be connected and since the inverter has only one possible input, the second will trigger an error. Finally, an input mismatch is shown in Fig. 7.H.
This part of the algorithm is common to both the technologies. Some elements need particular attention: the *Simple Magnet* in iNML and *Nucleation Center* in pNML. They are handled by the *Handling Technological Issues* blocks, which are described in sections IV-C and IV-D. Before entering into the details of this step, the function detection, still common to both the technologies, is described.

B. Function Detection

Once the connections have been defined for all the elements and no errors have been generated, no more entries are available in the *start* list. The execution moves to the next step, the function detection, whose implementation is summarized by pseudocode 2. The VHDL netlist is based on a library of

Algorithm 2 Function detection pseudocode

Precondition: $Elements$ is the list of all the HDL-Element of the layout.

```
1:  for each element ∈ Elements do
2:      if element.outputs ≠ connected then
3:          return OutputDisconnected
4:      end if
5:      if element.inputs ≠ connected then
6:          return InputDisconnected
7:      end if
8:      EvaluateFunctionality(element.connections)
9:      WriteVHDL()
10: end for
```

technological components. The library embeds the description of all the gates and building blocks supported. During the final phase, each layout element is associated to one of the library elements, depending on its function. Some elements have a one-to-one mapping in the library: an *AND* element in the layout is mapped into an *AND* gate in VHDL. A *Simple Magnet* in iNML could either be a *Majority Voter*, a *Coupler* or a *Simple Magnet* based on its interconnections. Each element is therefore analyzed. The first check is made on the output interconnections: if an output pin is not connected to any other element an error is generated. The same test is done on the input pins. If both tests fail, the function at

line 8 is executed. This function has specific implementation for each *HDL-Element*. The number of input and output connections is used to determine the function. Once detected, the correct library component is associated to the cell. For example, if a *Simple Magnet*, in iNML, has three inputs and one output, a *Majority Voter* is associated to the element and the corresponding VHDL netlist is written. On the contrary, if the same element has one input and two or three outputs, a *Coupler* is instantiated. Similarly, for the *Nucleation Center* in pNML, three inputs identify a *Majority Voter*, while one input identifies an inverter. Another example is the pNML *Pad*, but outputs are considered in this case. The components *Pad*, *T-Connection* or *X-Connection* are associated to one, two or three outputs respectively. Finally, during the VHDL writing step, the coupling among cells needs to be evaluated. The relative position between the interconnected elements is used and the inverters are inferred in the netlist in order to model the correct coupling.

The previous part of this section described the general algorithm adopted for connection and function detection. In the following, two sub-sections will describe the technology-dependent aspects of the proposed solution.

C. Handling iNML Technological Issues

In iNML technology, the *Simple Magnet* is the most critical element during the process of detecting the connection. When it is first connected, the number of surrounding elements is evaluated. If only one or two neighboring cells are available, the proper number of connections are created and the process continues to the next element. If a *Simple Magnet* has all the connectable positions occupied by other elements, it is marked as "ambiguous" and added to a specific list called $AmbiguousItems$. No connections are defined and the *start* list is not updated. After completing the analysis of the non-ambiguous elements, the $AmbiguousItems$ list is checked with the pseudocode presented in 3. In order to understand the connection of an ambiguous element, all the neighboring elements are analyzed. Each neighbor can be ambiguous, thus this connection is marked as undetermined, or non-ambiguous. In the case of non-ambiguous elements, a connection with a new element could be evaluated. If the new element has a fixed direction for input/output (it is an *AND* gate for example) or belongs to a different clock zone, the direction of the connection can be determined (line 12). Otherwise, this element becomes the new neighbor until the direction is fixed (line 9). When all the directions are completed, the if-then-else statement at line 19 is used to extract the actual number of connections. Since the resolution of an ambiguous pin could solve other connection uncertainty, the loop is interrupted as soon as a new function is identified. The resolved element is removed from the *AmbiguousItems* list and added to *start*. Thus, the normal connection algorithm 1 is executed with the updated *start* list. On the contrary, if it is impossible to determine the function of an element, the next one in the *AmbiguousItems* list is analyzed. If no ambiguous elements are resolved during the loop on the list, the number of iterations is increased and therefore the condition at line 29 becomes true.

Algorithm 3 Ambiguous element resolution

Precondition: $AmbiguousItems$ is the list of all the ambiguous elements found. $start$ is the list of the non-ambiguous elements.

```
1:  Iterations ← 0
2:  while iterations ≤ 2 do
3:      for each element ∈ AmbiguousItems do
4:          for each neighbor ∈ element.getConnectable() do
5:              while neighbor ≠ fixedDirection do
6:                  if neighbor.outputs > 1 then
7:                      neighbor ← Undetermined
8:                  else
9:                      neighbor ← neighbor.output
10:                 end if
11:             end while
12:             element.defineNeighborDirection()
13:         end for
14:
15:         inputs ← element.inputs
16:         outputs ← element.outputs
17:         undetermined ← element.undetermined
18:
19:         if inputs = 3 | (inputs = 2 & undetermined = 1) then
20:             element ← MajorityVoter
21:             start.append(element)
22:             AmbiguousItems.pop(element)
23:             break
24:         else if (inputs = 1 & undetermined < 2) | (outputs > 1
                    & undetermined = 1) then
25:             element ← Coupler
26:             start.append(element)
27:             AmbiguousItems.pop(element)
28:             break
29:         else if (inputs + undetermined) = 3 & outputs = 1 &
                    iteration ≠ 0 then
30:             element ← MajorityVoter
31:             start.append(element)
32:             AmbiguousItems.pop(element)
33:             break
34:         end if
35:     end for
36:     Iterations ← Iteration + 1
37: end while
```

This condition, lines 29-32, is used to "guess" the function of an element. The *Majority Voter* function is assigned to the first ambiguous element with exactly one output and a total of three connections, considering inputs and undetermined neighbors. Usually this situation is very uncommon and correct circuits are solved avoiding the need of two subsequent analyses of the *AmbiguousItems* list. If a guess is needed, there is probably an error in the layout. Indeed, this solution is able to detect errors. After guessing the function, the algorithm tries to connect all the neighboring cells as outputs, and conflicts appear in the circuit. Exceptions exist but are very uncommon: a symmetric circuit belonging to a single clock zone is correctly solved by the algorithm, but the function assignment is unpredictable. As mentioned, these kinds of circuits have been used as a corner case during the verification of the algorithm and without any practical function.

D. Handling pNML Technological Issues

In pNML technology the critical element is the *Nucleation Center*. Thanks to the fixed direction of all the pNML ele-

ments, in this technology the solution is simpler compared to iNML. During the connection generation, it is important to know the number of inputs for all the *Nucleation Centers* in the circuit. In the case of an even number of inputs, an error is present in the layout. To handle this issue, a *Set* data structure storing the ambiguous *Nucleation Center* has been introduced. When a *Nucleation Center* is connected, if the number of connections is even, its position is added to the *Set*. On the contrary, it is removed from the *Set* if an odd number of connections is present. When the $start$ list is empty, all the positions stored inside the *Set* are marked as error and added to the output log file.

V. Results

The presented algorithm was tested generating the netlist of several circuits with increasing complexity. Fig. 8 shows

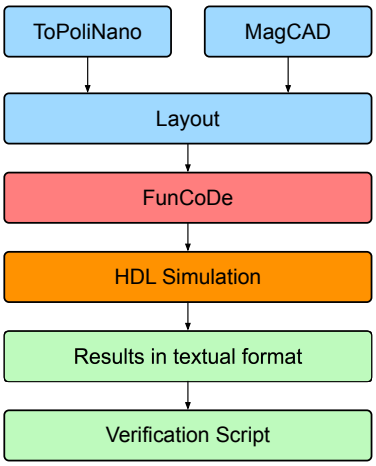

Fig. 8. Methodology adopted to verify the correct behavior of the proposed FUNCODE algorithm.

the flow adopted to test the algorithm. Different layouts were used, designed by hand with MagCAD [29] or generated with ToPoliNano [31] starting from a structural description. The FUNCODE algorithm was used to extract the VHDL netlists from the circuit layouts. The generated netlists are associated to compact models of the technology presented in [32] [29]. These models are based on physical experiments, when available or micromagnetic simulations [33] [34] [23]. The models represent an approximation of the physical behavior of the circuits, therefore there are simplifications.

This approach, which makes use of VHDL simulations, reduces drastically the simulation time and enables the exploration of large circuits. These kind of simulations are based on approximations and can give some insight on the performance of large architectures that would not be possible to explore with low level simulators. Moreover, with this approach, it possible not only to get information on the performance but also to verify the logical correctness of the design circuit. The main limitation of this approach is that the long-range integration among neighboring cells is not considered. Therefore, the user should have a good understanding of the technology when designing a circuit, since the tool gives complete freedom to the designer. The output waveforms are used to verify the logical correctness of the simulated circuits, considering the physical characteristics of the technological elements. All the simulations were performed using ModelSim from Menthor Graphics [30]. Fig. 9.A shows an example of circuit in iNML technology. It has three inputs (A, B and C) and two outputs (O1 and O2). It is composed of several *Simple Magnet* items and one inverter. Furthermore, two groups of magnets (the blue squares in the figure) implement the *Coupler* and the *Majority Voter* gates. Once the layout is processed by FUNCODE, it produces the VHDL nestlit, of which an excerpt is shown in Fig. 9.B. The highlighted cells report the automatically detected functions. Each cell label is named with its position, and a different library element is associated to it. For example *cell5_2* is associated to a *NML_cell* and the number of interconnections is specified using the *generic map* directive of VHDL. In the *port map*, the corresponding connections are made. Two issues can be noticed by observing the connections of the *Majority Voter* by name associations. First, a negated function is inserted to model the AF coupling of the input on the left. Second, the signal naming format. The signal *ls4_2o1* inverted is assigned to *CELL_IN(0)*. For each cell connection one signal is defined in VHDL. The name is constructed with the format ls + $cellposition$ + _o + $outputnumber$. The interconnections to the *Majority Voter* come from the cell on the left, above and below. For the *Coupler* the same type of cell is used but different interconnections are specified in the *generic map*. A different cell is instead used for the inverter. The simulation of the generated VHDL is shown in Fig. 9.C. The majority between A, B, C determines the O1 output, whereas the output O2 is the inverted value of input C. It is possible to observe that the output signals take into account the signal delays introduced by the clock scheme. The circles highlight the input-output correlation. The latency is due to the clock mechanism. Inputs have to travel through the circuit before reaching the outputs. Therefore, the generated VHDL takes into account the physical behavior in terms of timing introduced by the target technology.

The same approach was used for all circuits discussed in this section. Before starting the analysis of the results, the selected circuits are presented. The layouts for the iNML technology have been generated using ToPoliNano. We adopted two synthesis approaches to generate the circuits. The first is based on standard *AND*, *OR* and inverters gates. The second approach is based on majority gate synthesis. This approach makes larger circuits possible. The full adder, the Ripple Carry Adders (RCA) with different data parallelism and the array multiplier belong to first category. Furthermore, the c17, c432, c880 and c499 circuits from the ISCAS85 benchmark suit [35] were generated using the same strategy. The remaining circuits are based on the second approach, performed with SIS and ABC [36], two tools developed at University of California, Berkeley. The circuits based on the majority gate synthesis are part of the design of an Advance Encription Standard (AES) Substitution-BOX (S-BOX). The S-BOX is composed of the following sub-circuits: a 4-bit xor (xor4), a matrix multiplication circuit (affine_transformation), a Galois Field Isomorphic Mapping

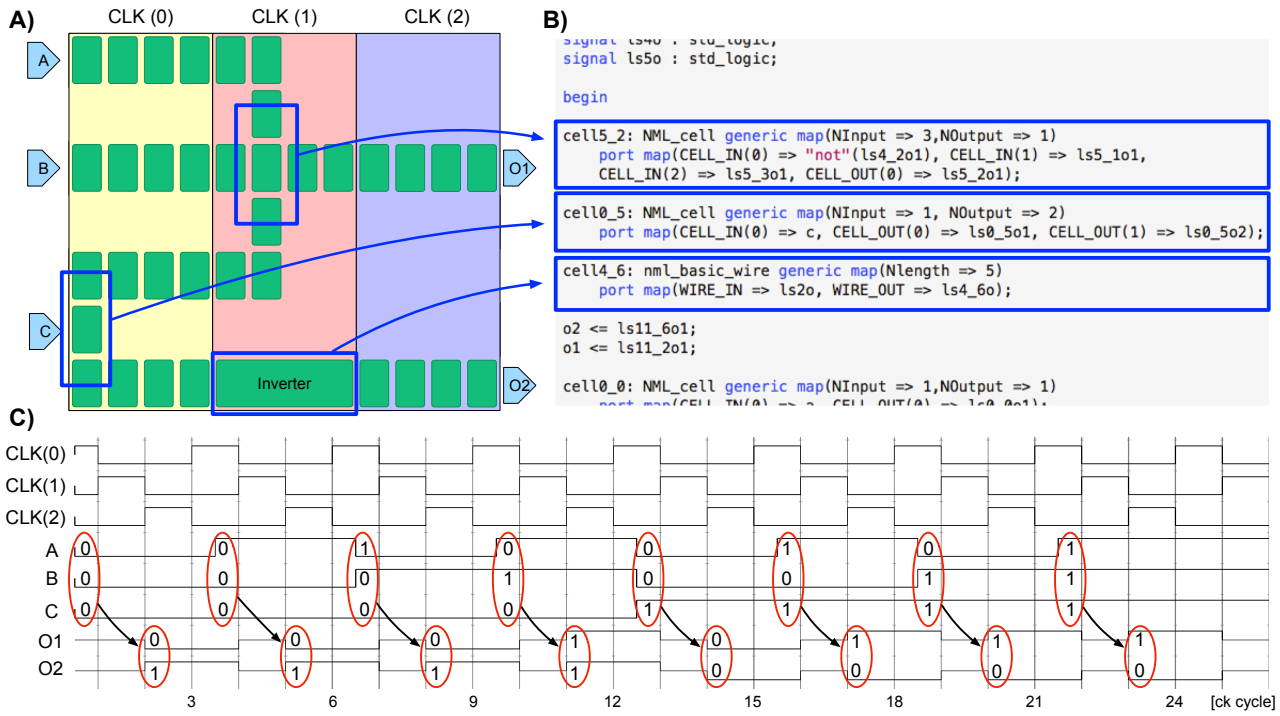

Fig. 9. A) Example of iNML layout; B) Excerpt of the generated VHDL that show the automatically detected functional blocks; C) Simulation of the sample circuit that take into account the latency introduced by the target technology.

circuit (GFMap), a 4-bit Galois Field multiplication (gfmul4), a Galois Field Inversion circuit (GFInv), a circuit performing square exponential computation plus constant multiplication (SquareEX) and finally the inversion of the Galois Field Isomorphic Mapping (GFmapinv).

The pNML circuits were manually designed using MagCAD. In pNML technology, the *Majority Voter* and the inverter are used to perform all the logic functions. Therefore, the number of gates in the same circuit is different with respect to the iNML version. The ripple carry adders with different input parallelism were used. The 32-bit version of the RCA has been presented in [37]. Furthermore, a decoder with two inputs and four outputs and a memory array 4x4, presented in [38], were used. Other circuits adopted are a finite state machine (FSM) with four states and a programmable logic array (PLA) with three inputs, four minterms and four maxterms, both presented in [37]. A circuit implementing the summed area table [28] (SAT) algorithm was analyzed. Finally, a hierarchical version of a 4-bit carry select adder (CSA) circuit was designed: the building blocks used are 2to1 multiplexers and 2-bit RCA. We performed VHDL simulations to verify the correct logic behavior of the generated netlists. Given the complexity of the benchmark circuits, the process was automated via a script. A behavioral code, or model in case of custom circuits, was simulated and the output was saved on a file. The extracted FUNCODE netlists were fed with the same input and all the produced output were written into file. The latency introduced in the new structures led to a larger file, with many more output lines. A script was used to extract the useful data in the simulation output and to compare these with the data obtained with the model. Two data are needed in order to run the script: circuit latency and number of clock cycles without changing the input vectors. The circuit latency is used to remove the firsts outputs, where no data is present since inputs are still loading the *pipeline*. The number of clock cycles is used to skip clock cycles where the outputs would have been equal.

Table I shows an example of the verification process for c432 circuit. Circuit c432 is a 27-channel interrupt controller. The input channels are grouped into three 9-bit buses (named A, B and C), where the bit position within each bus determines the interrupt request priority. A forth 9-bit input bus, named E, enables and disables interrupt requests within the respective bit positions. The seven outputs PA, PB, PC and Chan[3:0] specify which channels have acknowledged interrupt requests. Only the channel of highest priority in the requesting bus of highest priority is acknowledged. The input vectors are listed in the first column of Table I. Since the circuit is combinational the outputs of the reference description are immediately available without any latency. They are reported in the second column. The output file of the FUNCODE netlist was elaborated through the script and the resulting values are reported in the third column.

All the analyses reported in this section were performed on a Intel Core i-7 7700, equipped with 16GB of RAM, running CentOS 7 operating system.

The results show that the algorithm complexity is linear with

TABLE I. VERIFICATION EXAMPLE OF CIRCUIT C432. INPUTS ARE APPLIED TO BOTH THE MODEL AND THE FUNCODE GENERATED NETLIST, OUTPUTS ARE COLLECTED AND ANALYZED.

Inputs Vectors				Model outputs	FUNCODE netlist outputs
E[8;0]	A[8;0]	B[8;0]	C[8;0]	PA PB PC Chan[3;0]	
111111111	011111111	011111111	011111111	1 1 1 0000	1 1 1 0000
111111111	101111111	101111111	101111111	1 1 1 1111	1 1 1 1111
111111111	110111111	110111111	110111111	1 1 1 1110	1 1 1 1110
111111111	111011111	111011111	111011111	1 1 1 1101	1 1 1 1101
111111111	111101111	111101111	111101111	1 1 1 1100	1 1 1 1100
111111111	111110111	111110111	111110111	1 1 1 1011	1 1 1 1011
111111111	111111011	111111011	111111011	1 1 1 1010	1 1 1 1010
111111111	111111101	111111101	111111101	1 1 1 1001	1 1 1 1001
111111111	111111110	111111110	111111110	1 1 1 1000	1 1 1 1000
111111111	111111111	010011111	101111111	0 1 0 0110	0 1 0 0110
111111111	111111111	101111111	010000000	0 1 0 1111	0 1 0 1111
111111111	111111111	110110101	111111111	0 1 0 1110	0 1 0 1110
111111111	111111111	111010111	111111111	0 1 0 1101	0 1 0 1101
111111111	111111111	111100111	111111111	0 1 0 1100	0 1 0 1100
111111111	111111111	111110111	111111111	0 1 0 1011	0 1 0 1011
111111111	111111111	111111011	111111111	0 1 0 1010	0 1 0 1010
111111111	111111111	111111101	111111111	0 1 0 1001	0 1 0 1001
111111111	111111111	111111110	111111111	0 1 0 1000	0 1 0 1000
111111111	111111111	111111111	011111111	0 0 1 0000	0 0 1 0000
111111111	111111111	111111111	101111111	0 0 1 1111	0 0 1 1111
111111111	111111111	111111111	110111111	0 0 1 1110	0 0 1 1110
111111111	111111111	111111111	111011111	0 0 1 1101	0 0 1 1101
111111111	111111111	111111111	111101111	0 0 1 1100	0 0 1 1100
111111111	111111111	111111111	111110111	0 0 1 1011	0 0 1 1011
111111111	111111111	111111111	111111011	0 0 1 1010	0 0 1 1010
100000010	100000010	100000010	100000000	0 0 1 1001	0 0 1 1001
011111101	011111101	011111101	011111100	0 0 1 1000	0 0 1 1000
111111111	111011011	000100100	000100100	1 0 0 1101	1 0 0 1101
111111111	111101001	111111111	111111111	1 0 0 1100	1 0 0 1100
111111111	011111111	100000000	100000000	1 0 0 0000	1 0 0 0000
111111111	001111111	111111111	001111111	1 0 1 0111	1 0 1 0111
000000000	000000000	000000000	000000000	0 0 0 0000	0 0 0 0000

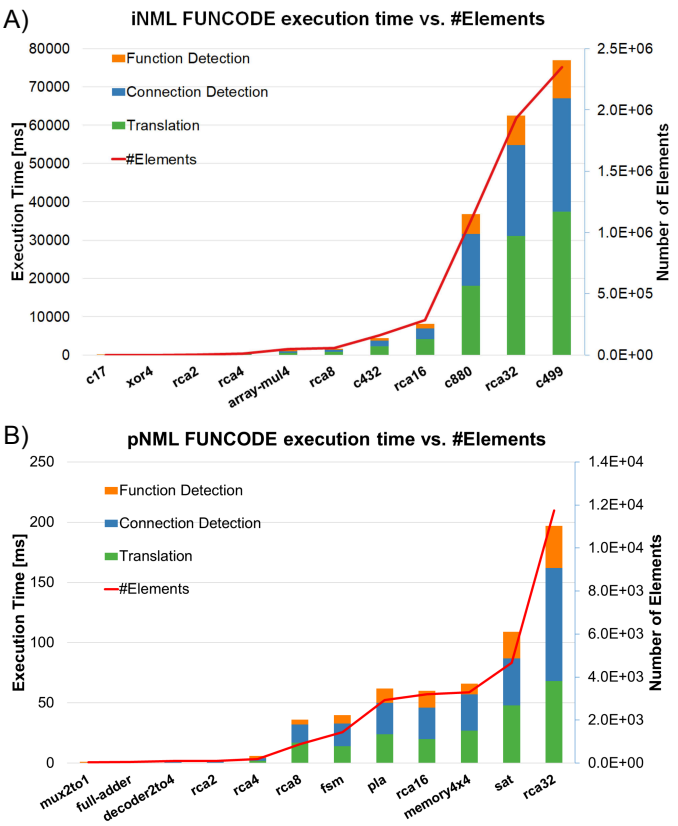

Fig. 10. A) iNML FUNCODE execution time as a function of the total number of elements; B) pNML FUNCODE execution time as a function of the total number of elements.

the number of elements in the circuit. Fig. 10 shows an analysis of the execution time for the different parts of the algorithm for the two technologies. The performance of the iNML version are reported in Fig. 10.A. The different circuits are listed on the x-axis. The line shows the number of elements of the circuit and is plotted on the secondary y-axis. The stacked bars show the time needed to execute the algorithm. The graph shows that the total execution time increases linearly with the number of elements. Furthermore, the different parts of the algorithm are associated to different portions of the stacked bars. The translation to the *HDL-Elements* is in green, while the connection detection is in blue. The connection time also includes the handling of the technological issues. The function detection and the VHDL netlist writing are represented by the orange part of the bars. The bars reveal that the translation time is the longest, and is almost half of the total time. In fact, during the translation, different data structures are allocated and different operations are performed on each element. Furthermore, the connection detection execution impacts the total time for approximately the 37%. Finally, the function detection is the shortest. Once all the connection have been defined, the function is assigned according to the connected elements. A similar analysis was performed for the pNML technology. Fig. 10.B shows the trend of the execution time over the number of elements. As expected, also for the pNML case, a linear complexity is achieved since the core of the algorithm is shared by both the technologies. Differently from iNML technology, the most time consuming portion of the algorithm is the connection detection, which is almost 45% on average. The translation time is about 38% of the total time and the function detection only impacts 18% of the time. The differences between the two technologies derive from the fact that the *Simple Magnet* in iNML technology, the most common item in the circuits, has a high connectivity. In fact, during translation a huge number of redundant connections are generated, and the ambiguous situations need to be resolved. This impacts on the final performance of the iNML version, which is slower than the pNML implementation. By contrast, in pNML technology the global clock mechanism increases the complexity of the interconnection detection: without the constraint of sequentiality given by the clock zones an additional degree of freedom is present.

Tables II and III show the results for iNML and pNML technology respectively. The tables are organized as follows. The first column is the circuit name. Circuit details, such as number of gates, inputs, outputs and elements are shown in the subsequent columns. In the fifth column the approach, flat or hierarchical, is presented. The last columns are used to show the time performance of the algorithm. As in Fig. 10, the times

TABLE II. GENERATION ALGORITHM TIMING REPORT USING DIFFERENT iNML ARCHITECTURES WITH AN INCREASING NUMBER OF ELEMENTS.

Circuit Name	#Gates	#Inputs/#Outputs	#Elements	Design Approach	Translation Elements [ms]	Connection Detection [ms]	Function Detection [ms]	Total Time [ms]
c17	6	5/2	233	flat	2	2	1	5
xor4	20	8/4	260	flat	5	3	1	9
squareXE	14	4/4	1688	flat	59	22	19	100
rca2	26	5/3	2687	flat	39	26	23	88
rca4	52	9/5	11987	flat	198	128	92	418
GFInv	34	4/4	16645	flat	291	195	129	615
GFmapinv	55	8/8	37145	flat	578	369	228	1175
array mul4	148	8/8	47972	flat	594	404	201	1199
GFmap	56	8/8	55864	flat	782	487	235	1504
rca8	104	17/9	57502	flat	815	525	324	1664
affine_transform	70	8/8	85206	flat	1341	883	496	2720
c432	160	36/7	161114	flat	2236	1500	634	4370
rca16	208	33/17	285874	flat	4209	2727	1202	8138
gfmolt4	199	8/4	379258	flat	5753	4035	1528	11334
c880	383	60/26	1093246	flat	18031	13574	5242	37027
rca32	416	65/33	1935112	flat	31135	23641	7747	62523
c499	202	41/32	2348735	flat	37431	29635	9921	76987

TABLE III. GENERATION ALGORITHM TIMING REPORT USING DIFFERENT pNML ARCHITECTURES WITH AN INCREASING NUMBER OF ELEMENTS.

Circuit Name	#Gates	#Inputs/#Outputs	#Elements	Design Approach	Translation Elements [ms]	Connection Detection [ms]	Function Detection [ms]	Total Time [ms]
mux2to1	13	2/1	36	flat	0	0	1	1
full-adder	8	3/2	55	flat	0	1	0	1
decoder2to4	21	2/4	94	flat	0	1	1	2
rca2	21	5/4	94	flat	0	1	1	2
mux2to1x3	49	7/3	162	hier	0	1	0	1
rca4	43	9/5	194	flat	2	2	2	6
carry select	149	9/5	838	hier	2	2	1	5
rca8	156	17/9	886	flat	17	15	4	36
fsm	116	5/12	1438	flat	14	19	7	40
pla	513	51/14	2939	flat	24	26	12	62
rca16	461	33/17	3199	flat	20	26	14	60
memory4x4	972	5/14	3294	flat	27	30	9	66
sat	1050	10/19	4662	flat	48	39	22	109
rca32	1391	65/33	11745	flat	68	94	35	197

are divided in translation, connection and function detection. The last column is the total time needed by the algorithm.

VI. CONCLUSION

With this paper, we have presented a general algorithm for the connection and function detection of custom layouts. FUNCODE was integrated in the latest version of MagCAD. We tested the algorithm with several circuits with an increasing complexity. This approach can simplify the investigation of device-aware architectural solutions. Here, iNML and pNML were considered as case studies. In the future, this algorithm will be extended to other promising technologies. The aim is to have a unified approach for their analysis, which takes into account the timing behavior and layout constraints. Researches that are interested in exploiting this approach on new devices could contact us.

ACKNOWLEDGMENT

The authors would like to thank Riccardo Chiola for his support in the development of the presented algorithm.

REFERENCES

[1] T. Chen, "Overcoming research challenges for cmos scaling: industry directions," in *2006 8th International Conference on Solid-State and Integrated Circuit Technology Proceedings*, Oct 2006, pp. 4–7.

[2] "International Technology Roadmap of Semiconductors," 2015, https://www.semiconductors.org/resources/2015-international-technology-roadmap-for-semiconductors-itrs/.

[3] D. E. Nikonov and I. A. Young, "Overview of beyond-cmos devices and a uniform methodology for their benchmarking," *Proceedings of the IEEE*, 2013.

[4] X. Yin, X. Chen, M. Niemier, and X. S. Hu, "Ferroelectric fets-based nonvolatile logic-in-memory circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 1, pp. 159–172, 2019.

[5] Z. Yang and L. Wei, "Logic circuit and memory design for in-memory computing applications using bipolar rrams," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1–5.

[6] R. L. Stamps, S. Breitkreutz, J. kerman, A. V. Chumak, Y. Otani, G. E. W. Bauer, J.-U. Thiele, M. Bowen, S. A. Majetich, M. Klui, and et al., "The 2014 magnetism roadmap," *Journal of Physics D: Applied Physics*, vol. 47, no. 33, p. 333001, Jul 2014.

[7] F. Riente, G. Ziemys, C. Mattersdorfer, S. Boche, G. Turvani, W. Raberg, S. Luber, and S. Breitkreutz-v. Gamm, "Controlled data storage for non-volatile memory cells embedded in nano magnetic logic," *AIP Advances*, vol. 7, no. 5, p. 055910, 2017.

[8] V. Jamshidi and M. Fazeli, "Pure magnetic logic circuits: A reliability analysis," *IEEE Transactions on Magnetics*, vol. 54, no. 10, pp. 1–10, Oct 2018.

[9] G. Ziemys, V. Ahrens, S. Mendisch, G. Csaba, and M. Becherer, "Speeding up nanomagnetic logic by dmi enhanced pt/co/ir films," *AIP Advances*, vol. 8, no. 5, p. 056310, 2018.

[10] M. Gonelli, S. Fin, G. Carlotti, H. Dey, G. Csaba, W. Porod, G. H.

Bernstein, and D. Bisero, "Robustness of majority gates based on nanomagnet logic," *Journal of Magnetism and Magnetic Materials*, vol. 460, pp. 432 – 437, 2018.

[11] I. Eichwald, S. Breitkreutz, J. Kiermaier, G. Csaba, D. Schmitt-Landsiedel, and M. Becherer, "Signal crossing in perpendicular nanomagnetic logic," *Journal of Applied Physics*, vol. 115, 2014.

[12] A. Papp, M. Niemier, A. Csurgay, M. Becherer, S. Breitkreutz, J. Kiermaier, I. Eichwald, X. Hu, X. Ju, W. Porod, and G. Csaba, "Threshold gate-based circuits from nanomagnetic logic," *Nanotechnology, IEEE Transactions on*, vol. 13, no. 5, pp. 990–996, Sept 2014.

[13] M. Becherer, G. Csaba, and G. iemys, "3d nanomagnetic logic: How far beyond cmos?" in *2017 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, Oct 2017, pp. 1–2.

[14] J. F. Chaves, M. A. Ribeiro, F. S. Torres, and O. P. Vilela Neto, "Enhancing fundamental energy limits of field-coupled nanocomputing circuits," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.

[15] J. Das, S. Alam, and S. Bhanja, "Addressing The Layout Constraint Problem when Cascading Logic Gates in Nanomagnetic Logic ," Aug. 2012.

[16] H. A. D. Nguyen, J. Yu, L. Xie, M. Taouil, S. Hamdioui, and D. Fey, "Memristive devices for computing: Beyond cmos and beyond von neumann," in *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2017, pp. 1–10.

[17] A. Chen, "Cooptimization of emerging devices and architectures for energy-efficient computing," in *2017 IEEE 12th International Conference on ASIC (ASICON)*, Oct 2017, pp. 136–139.

[18] M. Niemier and al., "Nanomagnet logic: progress toward system-level integration," *J. Phys.: Condens. Matter*, vol. 23, p. 34, Nov. 2011.

[19] 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 Trans. on Very Large Scale Integration (VLSI) Systems*, vol. PP, no. 99, pp. 1–10, 2016.

[20] M. Niemier, X. Ju, M. Becherer, G. Csaba, X. S. Hu, D. Schmitt-Landsiedel, P. Lugli, and W. Porod, "Systolic architectures and applications for nanomagnet logic," *Silicon Nanoelectronics Workshop (SNW)*, pp. 1,2, 2012.

[21] S. Breitkreutz, I. Eichwald, J. Kiermaier, A. Papp, G. Csaba, M. Niemier, W. Porod, D. Schmitt-Landsiedel, and M. Becherer, "1-bit full adder in perpendicular nanomagnetic logic using a novel 5-input majority gate," *EPJ Web of Conferences*, vol. 75, no. 05001, July 2014.

[22] A. Imre, L. Ji, G. Csaba, A.O. Orlov, G. Bernstein, and W. Porod, "Magnetic Logic Devices Based on Field-Coupled Nanomagnets," *2005 International Semiconductor Device Research Symposium*, p. 25, December 2005.

[23] E. Varga, M. T. Niemier, G. Csaba, G. H. Bernstein, and W. Porod, "Experimental realization of a nanomagnet full adderusing slanted-edge magnets," *IEEE Transactions on Magnetics*, vol. 49, no. 7, pp. 4452–4455, July 2013.

[24] 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, 2016.

[25] W. Porod, G. H. Bernstein, G. Csaba, S. X. Hu, J. Nahas, M. T. Niemier, and A. Orlov, *Nanomagnet Logic (NML)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 21–32.

[26] X. Ju, S. Wartenburg, J. Rezgani, M. Becherer, J. Kiermaier, S. Breitkreutz, D. Schmitt-Landsiedel, W. Porod, P. Lugli, and G. Csaba, "Nanomagnet logic from partially irradiated co/pt nanomagnets," *IEEE Transactions on Nanotechnology*, vol. 11, no. 1, pp. 97–104, Jan 2012.

[27] S. Breitkreutz, J. Kiermaier, C. Yilmaz, X. Ju, G. Csaba, D. Schmitt-Landsiedel, and M. Becherer, "Nanomagnetic logic: compact modeling of field-coupled computing devices for system investigations," *Journal on Computational Electronics*, 2011.

[28] F. Riente, D. Melis, and M. Vacca, "Exploring the 3-d integrability of perpendicular nanomagnet logic technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 7, pp. 1711–1719, July 2019.

[29] F. Riente, U. Garlando, G. Turvani, M. Vacca, M. Ruo Roch, and M. Graziano, "Magcad: Tool for the design of 3-d magnetic circuits," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 3, pp. 65–73, Dec 2017.

[30] "Mentor Graphics," http://www.modelsim.com.

[31] 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.

[32] G. Turvani, F. Riente, E. Plozner, D. Schmitt-Landsiedel, and S. B. v. Gamm, "A compact physical model for the simulation of pnml-based architectures," *AIP Advances*, vol. 7, no. 5, p. 056005, 2017.

[33] 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*, pp. n/a–n/a, 2016.

[34] S. Breitkreutz and al., "Experimental demonstration of a 1-bit full adder in perpendicular nanomagnetic logic," *Magnetics, IEEE Tran. on*, vol. 49, no. 7, pp. 4464–4467, July 2013.

[35] D. Bryan, "ISCAS '85 benchmark circuits and netlist format," in *International Symposium on Circuits and Systems*. Kyoto: IEEE, 1985.

[36] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Proceedings of the 22Nd International Conference on Computer Aided Verification*, ser. CAV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 24–40.

[37] U. Garlando, F. Riente, G. Turvani, A. Ferrara, G. Santoro, M. Vacca, and M. Graziano, "Architectural exploration of perpendicular nano magnetic logic based circuits," *Integration*, vol. 63, pp. 275 – 282, 2018.

[38] A. Ferrara, U. Garlando, L. Gnoli, G. Santoro, and M. Zamboni, "3d design of a pnml random access memory," in *2017 13th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, June 2017, pp. 5–8.

Umberto Garlando He received the B.S and M.S degree in electronic engineering, in 2013 and 2015 respectively, from Politecnico di Torino, where he is currently working towards e Ph.D. degree. His research activities mainly focus on developing Computer Aided Design tools for digital circuits based on emerging technologies, in particular magnetic and molecular devices.

Fabrizio Riente He received his M.Sc. Degree with honors (Magna Cum Laude) in Electronic Engineering in 2012 and the Ph.D. degree in 2016 from the Politecnico di Torino. He has been Postdoctoral Research Associate at the Technical University of Munich in 2016. He is currently Postdoctoral Research Associate at the Politecnico di Torino. His primary research interests are device modeling, circuit design for nano-computing, with particular interest on magnetic QCA. His interests cover also the development of EDA tool for beyond-CMOS technologies, with the main focus on the physical design.

Mariagrazia Graziano received the Dr.Eng. and Ph.D degrees in electronics engineering from the Politecnico di Torino in 1997 and 2001, respectively. Since 2002, she has been an Assistant Professor at the Politecnico di Torino. Since 2008, she has been adjunct Faculty at the University of Illinois at Chicago and since 2014 she is a Marie-Curie fellow at the London Centre for Nanoelectronics. She works on beyond CMOS devices, circuits and architectures.