Thinking Outside the Clock: Physical Design for Field-coupled Nanocomputing with Deep Reinforcement Learning

Simon Hofmann*, Marcel Walter*, Lorenzo Servadei*, and Robert Wille*†

*Chair for Design Automation, Technical University of Munich, Germany
†Software Competence Center Hagenberg GmbH, Austria
Email: {simon.t.hofmann, marcel.walter, lorenzo.servadei, robert.wille}@tum.de
https://www.cda.cit.tum.de/research/nanotech/

***Abstract*—Recent advances in atom-scale manufacturing are paving the way toward the emergence of *Field-coupled Nanocomputing* (FCN) as a viable real-world post-CMOS technology. Current FCN-specific solutions for placement and routing of logic functions are at risk of falling behind manufacturing capabilities. The problem lies in the fact that existing algorithms are either optimal in their result quality but do not scale; or are scalable, but produce results of sub-par quality limited to select clocking schemes. Furthermore, most existing approaches are tailored toward a concrete FCN implementation, limiting their applicability across the domain. To address these challenges, we propose a novel approach that utilizes deep reinforcement learning to learn the placement of logic elements and incorporate established routing strategies directly into the placement step. By relying only on abstract signal flow directions, this solution is technology-agnostic and therefore applicable to any FCN implementation, layout topology, or clocking scheme. The proposed approach is experimentally evaluated on a set of established benchmark functions common in the domain. While a state-of-the-art exact approach is limited to designing layouts for functions containing a maximum of around 40 gates, the proposed approach is able to generate solutions for all functions included in the considered benchmark sets, while reducing the layout area by an average of $59\,\%$ compared to the state-of-the-art heuristic. Furthermore, the proposed algorithm is made available to the scientific community as an open-source implementation.**

I. Introduction

The latest technological advancements in *Field-coupled Nanocomputing* (FCN, [1]), such as *Silicon Dangling Bonds* (SiDBs, [2]) and the multimillion-dollar investments made by enterprises such as *Quantum Silicon Inc.*, underscore the need for innovative approaches to FCN physical design in order to keep up with the rapid pace of progress.

However, cutting-edge solutions for layout mapping of logic functions are at risk of falling behind due to the rapid domain shifts. Optimal solutions for the placement and routing of a netlist onto a layout can only be obtained for small functions using *exact* approaches with exponential runtime behavior [3], [4]. To generate layouts for larger functions, *scalable* algorithms [5] are used to determine solutions. However, these solutions are typically of sub-par quality and are only applicable to a specific clocking scheme. Furthermore, a shift from Cartesian layouts for *Quantum-dot Cellular Automata* (QCA, [6]) to hexagonal ones for SiDBs [7] further emphasizes the need for novel scalable physical design automation approaches that are domain-independent.

To address these issues, rather than developing a solution that is specific to a particular clocking scheme and underlying cell technology, we propose an approach that relies only on abstract signal flow directions, which is, thereby, agnostic to both clocking and cell technology.

Inspired by recent developments in the field of machine learning-aided design automation [8], [9], we combined reinforcement learning with efficient path routing based on established algorithms such as *A^* Search* [10]. In this work, *Proximal Policy Optimization* (PPO) [11] is used to learn the placement of logic elements, which is further accelerated by incorporating action masks computed based on netlist structure and partial placements, ensuring valid and compact solutions. To minimize the occurrence of unpromising partial placements, several checks constantly ensure the early termination of sub-par solutions.

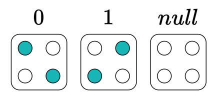
The remainder of this paper is structured as follows: Section II reviews technical background on selected FCN technologies and reinforcement learning. Section III reviews state-of-the-art design automation methods for QCA and SiDB. The reinforcement learning-based hybrid physical design algorithm proposed in Section IV is then experimentally evaluated based on common benchmark functions in Section V. Finally, Section VI concludes the paper.

The implementation is available as an open-source Python package[1] as part of the *Munich Nanotech Toolkit* (MNT).

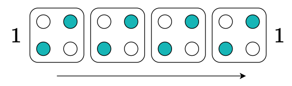
II. Background

FCN is a class of technologies that show promise as post-CMOS solutions to the growing need for computing power while addressing environmental concerns. These technologies use circuits that operate at the nanoscale without the need for electrical current flow [1]. Section II-A covers their preliminaries that are required for the comprehension of the remainder of this manuscript. Subsequently, Section II-B describes the reinforcement learning method used in our approach.

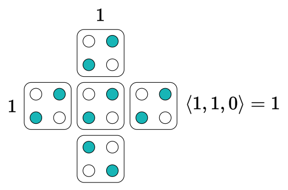
[1]`https://www.pypi.org/project/mnt.nanoplacer/`



(a) Polarization states of individual cells.

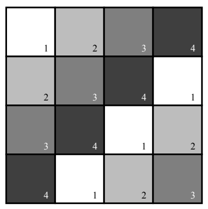


(b) Wire segment.

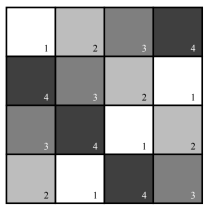


(c) Majority gate.

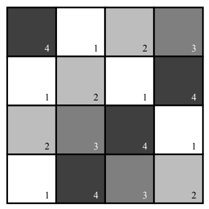
Fig. 1: Elementary QCA cells and compound structures.



(a) 2DDWave [20].

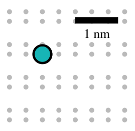


(b) USE [21].

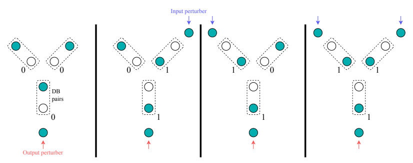


(c) RES [22].

Fig. 3: Common clocking schemes for FCN technologies.



(a) H-Si(100)-2×1 surface structure.



(b) Recreation of a binary-dot OR gate [12], adapted from [7].

Fig. 2: SiDBs on a H-Si(100)-2×1 lattice can implement logic gates.

A. *Field-coupled Nanocomputing*

First, Section II-A1 is concerned with QCA, arguably the most intensively researched FCN technology. Afterward, Section II-A2 presents an overview of the more recent fabrication breakthroughs achieved with SiDBs and Section II-A3 discusses the technological constraints imposed by these two technologies.

1) Quantum-dot Cellular Automata (QCA): The QCA technology operates by utilizing an elementary device called a *cell*, playing a role analogous to the transistor in traditional electronics. While a single cell can store a single bit of information in the form of a charge state, combining multiple cells allows for the creation of structures capable of computing any Boolean function.

Each QCA cell comprises four *quantum dots* arranged in a square frame on a substrate, as depicted in Fig. 1a. Binary values of 0 and 1 can be encoded using polarization in the form of electron configurations, based on the position of the charges. The polarization of cells placed in proximity can influence each other due to electrical fields, causing their polarization to align accordingly, which enables computation and propagation of information. The simplest configuration is a line of adjacent QCA cells that forms a binary wire segment, as shown in Fig. 1b. By placing a cell adjacent to three input cells, the majority-of-three (MAJ3) function can be implemented, as illustrated in Fig. 1c, of which AND and OR implementations can be derived by setting one input to constant 0 or 1, respectively. Furthermore, inverters can be created to achieve boolean completeness.

2) Silicon Dangling Bonds (SiDBs): Through a fabrication process that involves removing hydrogen atoms from a passivated silicon (H-Si(100)-2×1) surface [13] using a scanning tunneling microscope [2], SiDBs can be created that behave like atomically-sized, chemically identical quantum dots. An SiDB on a H-Si(100)-2×1 surface is illustrated in Fig. 2a. Recent breakthroughs in the domain allow for unparalleled control over the placement of the dots [14]–[18].

Arrangements of pairs of SiDBs yield a concept known as *Binary-dot Logic* (BDL) [12]; in contrast to elementary devices with four quantum dots, as in the QCA domain. Using the BDL concept, a fully functioning SiDB OR gate with a footprint of less than $30\,\mathrm{nm}^2$ was successfully manufactured [12], and standard gate libraries as well as initial design automation methods have been proposed [7], [19]. A recreation of this OR gate with varying inputs is depicted in Fig. 2b.

3) Technology Constraints: There are various limitations imposed by the respective technology that restrict the FCN circuit layouts that can be manufactured. The majority of FCN technologies are planar and possess restricted crossing capabilities, which poses a difficulty for wire routing. Additionally, to ensure signal synchronization, it is necessary to balance the lengths of wire segments throughout the layout [23]. A crucial criterion for FCN circuits is that they must be divided into uniform regions that are periodically activated and deactivated by external fields, in order to maintain signal stability and control the direction of information flow [24], [25].

This activation mechanism, referred to as *clocking*, plays a critical role in all FCN implementations. This is because both combinational and sequential circuits require clocking to ensure signal stability and regulate the direction of information flow [24], [25].

The default clocking system for FCN involves four consecutive clock signals, numbered from 1 to 4. This system supports a pipeline-like flow of information, transmitting signals from tiles under the control of clock 1 to those under clock 2, clock 3, and finally clock 4, before returning to clock 1 [24], [25]. However, this can present challenges for signal propagation and synchronization, requiring careful management to ensure that adjacent tiles are clocked consecutively, and that wire lengths are balanced throughout the circuit to prevent delay differences and subsequent desynchronization [26].

The distribution of clock signals to each tile is a widely discussed topic in the literature. There is a consensus that the signals can be transmitted through buried electrodes within the substrate of the circuit [25]. Several clocking schemes that tile a layout floorplan have been proposed in the literature [20]–[22], as illustrated in Fig. 3. Each of them offers a specific arrangement of regular clock zones to aid the physical design of FCN layouts.

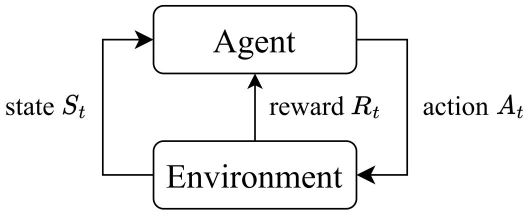


Fig. 4: The typical agent-environment interface.

B. Reinforcement Learning

Reinforcement learning [27] involves an agent exploring a dynamic environment, where the agent interacts with its surroundings and receives either positive or negative rewards depending on its actions.

At each time step t, the agent either has a full or partial observation of the state $S_t \in \mathcal{S}$ and the previously received reward $R_t \in \mathcal{R}$, which determines the following action $A_t \in \mathcal{A}$ it will take in the environment. After taking the most promising action, it receives the next reward R_{t+1} and updated state S_{t+1}, as shown in Fig. 4.

Therefore, the agent tries to learn a mapping between actions and states, which is called the policy π_t. In policy gradient methods, the policy is modeled by a set of neural network weights θ.

The reward function, which relies on the weights of the neural network θ, indicates how good an action was in a certain state:

$$J(\theta) = \sum_s d_\pi(s) \sum_a q_\pi(s,a) \nabla_\theta \pi(a|s,\theta), \tag{1}$$

with the stationary distribution $d_\pi(s)$ of the Markov chain for π_θ and the value function q_π.

Proximal Policy Optimization (PPO, [11]) builds upon the foundation of Monte-Carlo policy gradient methods as in *REINFORCE* [28] and controlling the difference between the old and the new policy, as introduced in *Trust Region Policy Optimization* (TRPO, [29]).

PPO keeps the old and new policy close to each other in the loss function J by clipping the ratio $r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$ to the range $[1-\epsilon, 1+\epsilon]$ with the clipping parameter ϵ:

$$J^{\text{CLIP}}(\theta) = \mathbb{E}[\min(r_t(\theta)\hat{A}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A})], \tag{2}$$

where $\hat{A}$ is the estimated advantage of the old policy.

An invalid action masking process [30] is used to prevent sampling of invalid actions during reinforcement learning to reduce the number of actions the agent has to explore. To compute the probability of an action, the unnormalized output, also called logit, is fed through a non-linear activation function, e. g., softmax or tanh. The output of these activation functions approach 0 by replacing the logits with a significantly large negative value (e. g., -1×10^8), therefore masking out the corresponding actions.

Given recent advancements in applying RL frameworks to hard combinational problems [31], PPO was the preferred choice for learning gate placements in FCN due to its ability to mask out actions, as well as its performance in other machine learning-aided chip design approaches [8], [9], [32].

III. Related Work: Design Automation for Field-coupled Nanocomputing

As previously discussed, FCN layouts possess distinct characteristics that distinguish them from traditional CMOS-based computing systems. Specifically, the physical design problems associated with FCN technologies are particularly challenging due to constraints such as planarity and signal balancing. As expected, placement and routing for FCN circuits are known to be $\mathcal{NP}$-complete [33]. This renders finding optimal solutions intractable, even for relatively small circuits.

More precisely, the following approaches have been proposed so far:

A. Exact Approaches

Exact physical design algorithms, e. g., [3], [4], are capable of obtaining optimal layouts from specifications with respect to a given cost metric, typically layout area. However, these algorithms suffer from performance issues stemming from the $\mathcal{NP}$-completeness of the task. This limits their applicability to relatively small instances.

B. Heuristic Approaches

A heuristic algorithm, called *ortho*, was recently proposed [5]. This algorithm is based on an approximation to orthogonal graph drawing and is capable of automatically designing layouts with hundreds of millions of tiles. To achieve this scalability, the algorithm prunes the search space aggressively.

Unlike previous FCN physical design algorithms that prioritize expensive pre-processing measures like node balancing and crossing reduction or substitution, *ortho* adopts a different approach based on the topological computation of relative positions through direction assignment. This approach eliminates the need for defining the layout area before placement, as the required size is automatically determined. However, the algorithm is restricted exclusively to the *2DDWave* clocking scheme. Utilizing a 45° turn [34], the generated layouts can be transformed into a hexagonal configuration to accommodate Y-shaped SiDB gates [7].

C. Semi-automated Approaches

To date, no fully automated and scalable design algorithms for FCN circuits using clocking schemes other than *2DDWave* have been developed. Heuristics known in the literature [35], [36] still require manual input from experts before and during the design process, based on each function to realize. Furthermore, the lack of open-sourced code for these heuristics renders a reproduction and verification of their claimed results impossible, unlike the exact [3], [4] and scalable [5] approaches, which are part of the openly accessible *fiction* [37] framework.

IV. Hybrid Approach for Placement & Routing

The general idea of the proposed hybrid placement and routing algorithm is as follows:[2] It first creates a topological ordering of the logic network and then places one of the gates

[2]Preliminary results of this method have been presented in [38].

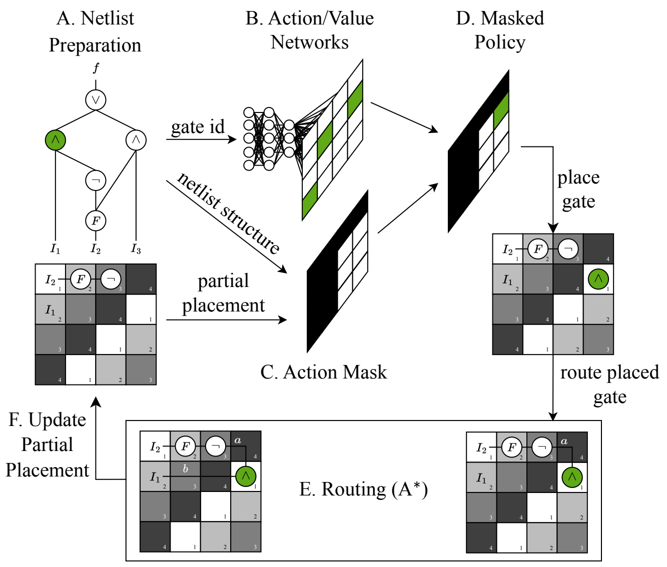


Fig. 5: Steps of the proposed algorithm annotated with letters that are matching the respective sub-section in Section IV.

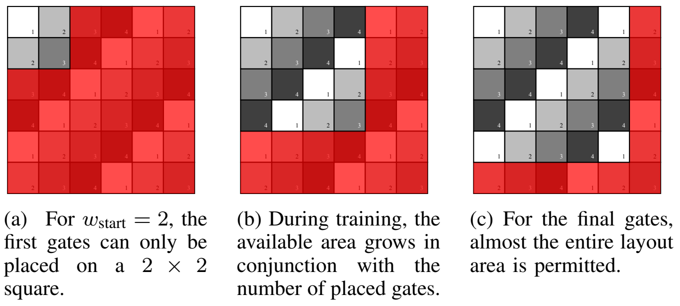
at each step, including routing it with its predecessor on the layout, regardless of the underlying clocking scheme. After the successful placement and routing of all gates, the algorithm outputs a representation of the physical design for the desired logic network, which can be directly realized with QCA or SiDB cells using gate libraries such as *QCA ONE* [39] or the Bestagon library [7].

The proposed hybrid placement and routing algorithm is described by means of Fig. 5 and is composed by the following steps:

- Section IV-A: An initial netlist preprocessing step.
- Section IV-B: Training action and value networks to output the most promising coordinates for gate placement.
- Section IV-C: Limiting the possible positions predicted by the action network based on the current gate to be placed with action masks, which can be obtained from the netlist and the partial placement.
- Section IV-D: Reducing the search space to valid coordinates only by combining the action mask with the output of the neural network.
- Section IV-E: Connecting the placed gate to its predecessors via A* path finding [10].
- Section IV-F: Receiving a reward and proceeding to place the next gate, while terminating unpromising or invalid partial placements early on based on further checks.

A. Netlist Preparation

At the beginning of a training cycle, a depth-first fanout substitution algorithm is applied to the input netlist to restrict the output degree of each node to a maximum of 2. Furthermore, a topological ordering of nodes is computed to guarantee a sequential placement, wherein parent nodes (predecessors) are allocated positions before placing their respective child node. In Fig. 5, a *current* node is highlighted in green and has both of its parent nodes already placed on the layout.



(a) For $w_{\text{start}} = 2$, the first gates can only be placed on a 2×2 square.

(b) During training, the available area grows in conjunction with the number of placed gates.

(c) For the final gates, almost the entire layout area is permitted.

Fig. 6: Action masks during a placement episode.

B. Action/Value Networks

The input dimension of both the action and value networks corresponds to the number of gates and comprises a one-hot encoded vector of the upcoming gate identification. Each of these networks is composed of two hidden layers, each layer having a size of 64. The output dimension of each network mirrors the number of coordinates present in the layout.

By employing a probability distribution, the network outputs predict the most desirable layout coordinates for placement of the current gate. The utilization of solely the gate identification as the observation leads to a comparatively low quantity of network weights. Consequently, swift updates can be made during training. As illustrated in Fig. 5, the three most probable positions are indicated by the color green.

For the smallest function in one of the considered benchmarks [36], namely the 2:1 Multiplexer with 9 gates and a layout size of $3 \times 5 = 15$ tiles, the number of weights is $9 \cdot 64 + 64 \cdot 64 + 64 \cdot 15 = 5632$. Even for the largest function (called *parity*) in another benchmark set [35], with 150 gates and a resulting layout size of $48 \times 48 = 2304$ tiles, the number of weights is only $150 \cdot 64 + 64 \cdot 64 + 64 \cdot 2304 = 161\,152$.

C. Action Masks

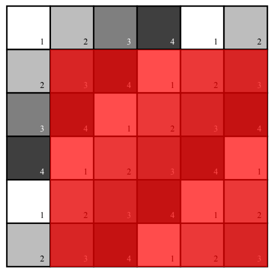
To limit the number of actions the agent has to explore, as well as to prevent the placement of gates on tiles that lead to invalid layouts, masks can be calculated based on the netlist structure, the current partial placement, the position of preceding and already placed gates, constraints imposed by the layout topology, information flow directions, and validity of the partial placement.

Depending on the type of gate to be placed, different action masks are generated, as described in the following.

1) Expanding Layout Size: On *2DDWave*-clocked layouts, information only flows to the east and south. Therefore, it is beneficial to start the placement in the top left corner and gradually expand the layout area, as the position of a placed gate determines the possible positions of all its dependent successors.

At first, a predefined quadratic layout area $w_{\text{start}} \times w_{\text{start}}$ is allowed, which expands linearly during training according to the current gate to be placed. The width w of the allowed square-sized area can be determined by

$$w = \left\lfloor w_{\text{start}} + g \cdot \frac{w_{\text{layout}} - w_{\text{start}}}{|G|} \right\rfloor, \tag{3}$$


(a) PIs are only allowed on the left and top border.

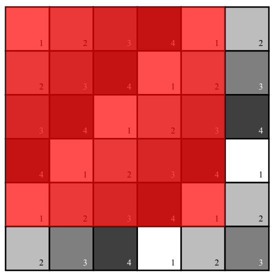
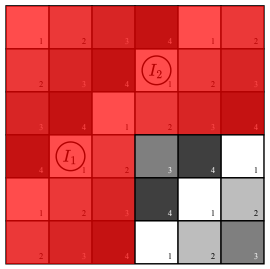

(b) POs are only allowed on the right and bottom border.

Fig. 7: Action masks for PIs and POs using *2DDWave*.


(a) Allowed positions for a gate with two predecessors.

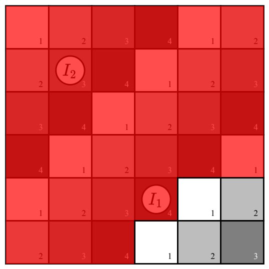
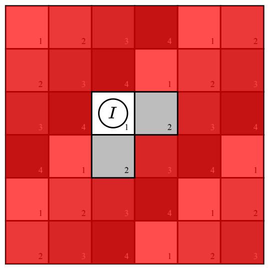

(b) Here, the coordinates of I_1 determine valid positions.

Fig. 9: Action masks for gates with two predecessors I_1 and I_2.


(a) On a *2DDWave*-clocked layout, only 2 positions are possible for a gate with a single predecessor.

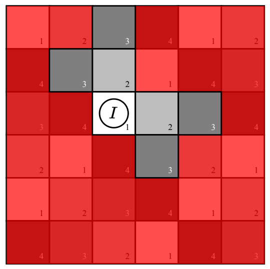
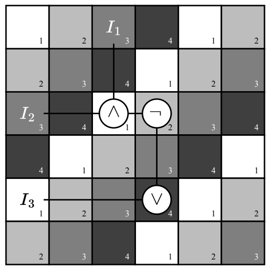

(b) In case of a more complex clocking scheme like *USE*, more tiles are allowed.

Fig. 8: Action masks for gates with a single predecessor I.


(a) Partial netlist placement.

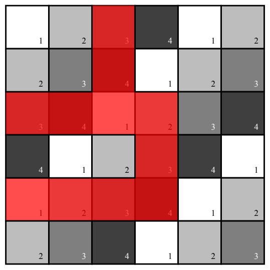

(b) Occupied tiles are masked.

Fig. 10: Action mask based on occupied tiles.

with g being the current gate id, $|G|$ the number of all gates and w_{layout} the width of the complete layout that is available. Setting w_{start} to 4 in our experiments led to the best results.

Fig. 6 illustrates the available area for placement at the beginning (Fig. 6a), midway (Fig. 6b) and close to the end (Fig. 6c) of a placement episode. In Fig. 6a, we used $w_{\text{start}} = 2$ to better show the growing available layout area.

2) Primary Inputs & Outputs: To enhance integrability of circuit layouts, primary inputs and outputs should be placed at the layout borders to make them accessible, which is achieved by masking out all non-border tiles. In the special case of the *2DDWave*-clocked layout, inputs are only placed at the left and top border, as shown in Fig. 7a, while the outputs are placed on the right and bottom border, as illustrated in Fig. 7b.

3) 1-Input Gates: Gates with only a single input, e. g., inverters or fanouts, can be placed one clock phase away from their predecessor in a *2DDWave*-clocked layout, as in Fig. 8a. In clocking schemes where signal flow in all directions is possible, such as *RES* or *USE*, gates can be placed up to two clock phases away as depicted in Fig. 8b. This prevents the occurrence of deadlocks, which can happen in *RES* or *USE* by placing multiple single input gates in a row, therefore creating a circle on the layout in some cases.

4) 2-Input Gates: Gates that have two inputs, such as AND, OR, XOR, etc., necessitate placement at specific locations that permit routing to adhere to the positioning of their predecessors and any previously routed wire connections.

In a *2DDWave*-clocked layout, the x coordinate of such gates must be set to the maximum x coordinate of its two predecessors at minimum, and the y coordinate must be set to the maximum y coordinate of the same predecessors. The positioning of the two predecessors, denoted by I_1 and I_2, influences the action masks exhibited in Fig. 9.

5) Layout Occupation: In FCN, a tile can only be occupied by a single gate or wire segments. Therefore, all occupied tiles are masked out. For the partial placement in Fig. 10a, Fig. 10b illustrates the resulting action mask.

6) Additive Mask Overlay: For each gate, multiple of the aforementioned conditions can be met simultaneously, leading to several action masks. A complete mask is determined by an additive overlay of all the individual masks. In Fig. 11, a gate with two inputs creates three masks:

1) expanding layout area mask as in Section IV-C1,
2) 2-input gate mask as in Section IV-C4, and
3) occupied tile mask as in Section IV-C5.

By combining these three masks, only one possible position remains, reducing the action space from $6 \times 6 = 36$ to a single tile. While this level of reduction is certainly a best-case scenario, it demonstrates the effectiveness of the proposed technique.

D. Masked Policy

In order to guarantee that the placement of gates is confined to valid locations, the action mask assigns a substantially negative value to the logits of coordinates deemed unsuitable for placement, thereby diminishing the probability of such

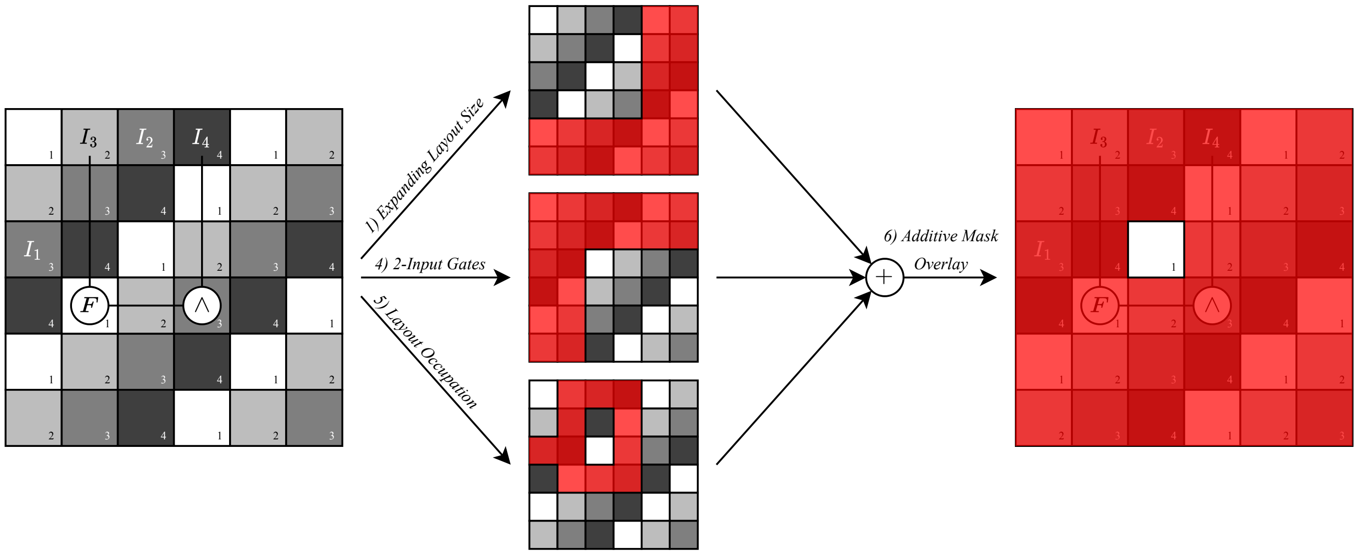
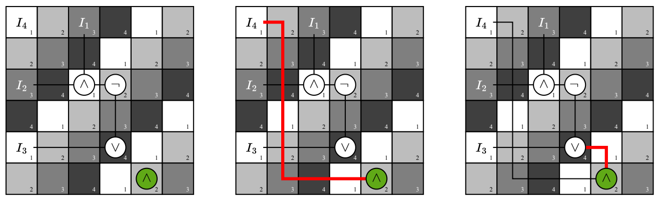
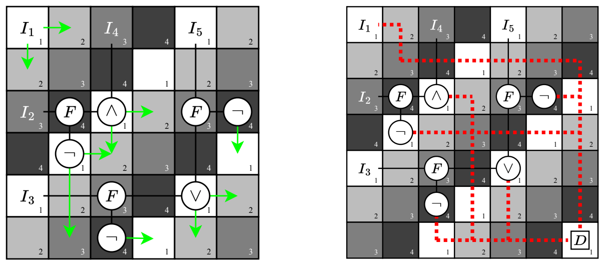


Fig. 11: For a gate with two inputs (I_1 and I_2), three different masks are calculated based on *1) Expanding Layout Size*, *4) 2-Input Gates*, and *5) Layout Occupation*. By *6) Additive Mask Overlay*, only a single position remains possible for placement, providing the RL agent with the best possible action space pruning.



(a) New gate is placed on the layout. (b) Routing the first predecessor. (c) Routing the second predecessor.

Fig. 12: The A* search is used twice to connect a placed gate with its two predecessors.



(a) If a gate is trapped, the current placement is terminated. (b) A drain tile D has to be reachable by every gate on the *2DDWave*-clocked layout.

Fig. 13: Several checks decide if the agent keeps on placing more gates onto the layout or if it restarts on an empty layout.

locations being selected by the agent. As demonstrated in Fig. 5, solely one of the three most probable coordinates satisfies the action mask restrictions, and this position is selected as a consequence of its augmented likelihood.

E. Wire Routing

After each placement step, the gate, depending on its type, must be connected to its predecessors without conflicting with existing wiring and without violating the information flow directions. A placed gate with a single predecessor can be either a 1-input gate, which is already connected to its predecessor during placement, or a primary output, which is then connected to its predecessor with A*. If a gate has two inputs, the predecessors are routed one after the other, with the second taking the first wiring into account.

In Fig. 12a, an AND gate was placed, which is colored green. First, A* connects the placed gate with I_4 in Fig. 12b before connecting it with its second predecessor, the other AND gate in Fig. 12c.

F. Update Partial Placement

At each time step t, the agent is exclusively granted a reward of $r_t = 1$ provided that the wire routing between the positioned gates and their predecessors is completed successfully, meaning it is free of conflicts. Otherwise, the reward is 0, and the agent may seek alternative positions as signified by the action mask. Only in the event that all viable positions have been evaluated without the possibility of a successful routing will the current placement be terminated.

Additionally, for *2DDWave*-clocked layouts, the reward is scaled according to the position of placed gates on the layout to favor placements closer to the top left corner, to maximize the number of possible positions for succeeding gates. In detail, the reward is scaled by the x and y-coordinate of the placed gate and the width w of the permitted layout area at the time, as described in Section IV-C1:

$$r_{t,scaled} = r_t \cdot (1 - \frac{x+y}{w^2}). \tag{4}$$

In case of a successful routing, the agent utilizes the updated partial placement to compute the next action mask for subsequent gates. Furthermore, the action and value networks are updated using PPO [11], depending on the batch size, which indicates the number of gates placed with the current policy π.

G. Premature Termination

The validity of partial placements is constantly monitored to prematurely terminate unpromising runs, thus, facilitating the initiation of a fresh layout. One trivial check is the action mask itself, which masks out all coordinates if no valid position can be determined. Two more sophisticated checks are illustrated in Fig. 13. For every dangling gate, i. e., a gate whose direct successors have not been placed yet, one algorithm checks if at least one of its adjacent tiles with the matching clock phase is still unoccupied. Additionally, if one of the tiles is occupied by a wire, it is checked if the wire can be crossed. In Fig. 13a, the check is successful, as every dangling gate has at least one free tile to connect with its successors.

Only if the first check was successful, a second algorithm temporarily adds another row and column to the layout and uses A* to test if every dangling gate is able to find a way to a virtual drain tile D. In Fig. 13b, every gate can be routed to D successfully, therefore, not terminating the current placement episode.

V. Experiments

In this section, we showcase the outcomes of the proposed hybrid physical design algorithm using a series of benchmark circuits commonly employed in the field [35], [36] for three different clocking schemes [20]–[22]. We also compare its efficacy to the state-of-the-art exact and heuristic approaches [3], [5] and validate layout correctness using formal verification [40]. Table I summarizes the obtained data, although we only present the achieved results for QCA to facilitate comparison with the *ortho* scalable approach for QCA, even though our proposed approach is technology-independent and, thus, also capable of producing SiDB layouts.

The exponential runtime behavior of the exact approach [3] limits its feasibility due to the $\mathcal{NP}$-completeness of the physical design problem [33]. Consequently, we applied a time-out limit of $24\,\mathrm{h}$, which was already exceeded by moderately-sized functions on every clocking scheme.

For the *2DDWave* clocking scheme, the state-of-the-art heuristic yields layouts for all functions in negligible runtime, but requires almost a factor of 6 more area, while for other clocking schemes, no approaches that are completely automatic and open-source are available. Impressively, across *all* functions in the two benchmark sets presented in Table I, the proposed hybrid approach yields a substantial area reduction on the *2DDWave* clocking scheme compared to the heuristic.

For other clocking schemes, our approach is able to find a solution for *all* functions, while the exact approach already receives a timeout for relatively small functions with ≈ 30 gates. The superiority of the *2DDWave* clocking scheme was anticipated, as previous research has shown that it imposes the least amount of overhead for combinational functions [4]. Results outlined in Table I demonstrate the superiority of the proposed approach over exact approaches when it comes to scalability and over heuristic methods concerning the occupied layout area, therefore, combing the best of both worlds.

VI. Conclusion

As *Field-coupled Nanocomputing* (FCN) becomes a reality, efficient methods for the automatic physical design are needed to match this promising emerging technology. In this work, we presented an approach that uses reinforcement learning for gate placement and established path finding algorithms for routing. The resulting hybrid algorithm for FCN physical design demonstrates remarkable proficiency in obtaining valid circuit layouts for logic functions of up to 150 gates, independent of the underlying clocking scheme or cell technology. While exact approaches are limited to designing layouts for functions containing a maximum of around 40 gates depending on the clocking scheme, the proposed approach is able to generate solutions for all functions included in the considered established benchmark sets, while reducing the obtained layout area by an average of $59\,\%$ compared to the state-of-the-art heuristic on *2DDWave*-clocked layouts.

The proposed algorithm is openly available as a Python package[3] as part of the *Munich Nanotech Toolkit* (MNT). Furthermore, the generated layouts have been included in the benchmark suite *MNT Bench* [41].[4]

References

[1] N. G. Anderson and S. Bhanja, Eds., *Field-Coupled Nanocomputing - Paradigms, Progress, and Perspectives*. Springer, 2014.

[2] R. Achal *et al.*, "Lithography for robust and editable atomic-scale silicon devices and memories," *Nat. Commun.*, vol. 9, no. 1, 2018.

[3] M. Walter *et al.*, "An Exact Method for Design Exploration of Quantum-dot Cellular Automata," in *DATE*, 2018, pp. 503–508.

[4] ——, "One-pass Synthesis for Field-coupled Nanocomputing Technologies," in *ASP-DAC*, 2021, pp. 574–580.

[5] ——, "Scalable Design for Field-Coupled Nanocomputing Circuits," in *ASP-DAC*, 2019, pp. 197–202.

[6] C. Lent *et al.*, "Quantum Cellular Automata: The Physics of Computing with Arrays of Quantum Dot Molecules," in *PhysComp*, 1994, pp. 5–13.

[7] M. Walter *et al.*, "Hexagons Are the Bestagons: Design Automation for Silicon Dangling Bond Logic," in *DAC*, 2022, pp. 739–744.

[8] A. Mirhoseini *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[9] R. Cheng *et al.*, "The Policy-gradient Placement and Generative Routing Neural Networks for Chip Design," *NIPS*, vol. 35, pp. 26 350–26 362, 2022.

[10] P. E. Hart *et al.*, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[11] J. Schulman *et al.*, "Proximal Policy Optimization Algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[12] T. Huff *et al.*, "Binary atomic silicon logic," *Nat. Electron.*, vol. 1, no. 12, pp. 636–643, 2018.

[13] M. B. Haider *et al.*, "Controlled Coupling and Occupation of Silicon Atomic Quantum Dots at Room Temperature," *Phys. Rev. Lett.*, vol. 102, p. 046805, Jan. 2009.

[14] T. Huff *et al.*, "Atomic White-Out: Enabling Atomic Circuitry through Mechanically Induced Bonding of Single Hydrogen Atoms to a Silicon Surface," *ACS nano*, vol. 11 9, pp. 8636–8642, 2017.

[15] J. L. Pitters *et al.*, "Charge Control of Surface Dangling Bonds Using Nanoscale Schottky Contacts," *ACS nano*, vol. 5, pp. 1984–9, Feb. 2011.

[16] R. A. Wolkow *et al.*, "Silicon Atomic Quantum Dots Enable Beyond-CMOS Electronics," in *Field-Coupled Nanocomputing*, 2013.

[3] `https://www.pypi.org/project/mnt.nanoplacer/`
[4] `https://www.cda.cit.tum.de/mntbench`

Table I: Comparative experimental evaluation of the state of the art against the proposed hybrid algorithm.

Benchmark Circuit [35], [36]				Exact Approach [3]			Heuristic Approach [5]			Proposed Approach		
Clocking Scheme	Name	I / O	$\|G\|$	$w \times h$	$= A$	t	$w \times h$	$= A$	t	$w \times h$	$= A$	t
2DDWave [20]	2:1 MUX	3 / 1	9	3 × 4	**12**	< 1	6 × 8	48	< 1	3 × 4	**12**	< 1
	XOR	2 / 1	9	3 × 6	**18**	< 1	5 × 8	40	< 1	3 × 6	**18**	< 1
	XNOR	2 / 1	11	3 × 6	**18**	< 1	6 × 9	54	< 1	3 × 6	**18**	< 1
	Half Adder	2 / 2	14	4 × 6	**24**	< 1	9 × 10	90	< 1	4 × 6	**24**	4
	Parity Gen.	3 / 1	18	4 × 8	**32**	< 1	9 × 14	126	< 1	8 × 8	64	2
	Parity Check.	4 / 1	26	6 × 8	**48**	2	12 × 20	240	< 1	9 × 9	81	14
	XOR5_R1	5 / 1	40	7 × 11	**77**	11	14 × 33	462	< 1	14 × 14	196	15
	cm82a	5 / 3	68	*timeout limit reached*			26 × 51	1326	< 1	25 × 25	**625**	210
	2bitAdderMaj	5 / 2	82	*timeout limit reached*			26 × 64	1664	< 1	29 × 29	**841**	180
	xor5Maj	5 / 1	102	*timeout limit reached*			30 × 79	2370	< 1	37 × 37	**1369**	595
	parity	16 / 1	150	*timeout limit reached*			48 × 120	5760	< 1	48 × 48	**2304**	951
USE [21]	2:1 MUX	3 / 1	9	3 × 5	**15**	< 1	*not applicable*			5 × 5	25	3
	XOR	2 / 1	9	4 × 5	**20**	< 1	*not applicable*			5 × 5	25	< 1
	XNOR	2 / 1	11	4 × 5	**20**	< 1	*not applicable*			6 × 6	36	< 1
	Half Adder	2 / 2	14	4 × 7	**28**	4	*not applicable*			7 × 7	49	19
	Parity Gen.	3 / 1	18	4 × 8	**32**	7	*not applicable*			9 × 9	81	26
	Parity Check.	4 / 1	26	*timeout limit reached*			*not applicable*			11 × 11	**121**	125
	XOR5_R1	5 / 1	40	*timeout limit reached*			*not applicable*			16 × 16	**256**	128
	cm82a	5 / 3	68	*timeout limit reached*			*not applicable*			35 × 35	**1225**	174
	2bitAdderMaj	5 / 2	82	*timeout limit reached*			*not applicable*			40 × 40	**1600**	966
	xor5Maj	5 / 1	102	*timeout limit reached*			*not applicable*			45 × 45	**2025**	2167
	parity	16 / 1	150	*timeout limit reached*			*not applicable*			70 × 70	**4900**	1157
RES [22]	2:1 MUX	3 / 1	9	3 × 5	**15**	< 1	*not applicable*			5 × 5	25	1
	XOR	2 / 1	9	4 × 4	**16**	< 1	*not applicable*			5 × 5	25	1
	XNOR	2 / 1	11	4 × 5	**20**	< 1	*not applicable*			6 × 6	36	6
	Half Adder	2 / 2	14	6 × 5	**30**	3	*not applicable*			8 × 8	64	6
	Parity Gen.	3 / 1	18	4 × 8	**32**	5	*not applicable*			11 × 11	121	27
	Parity Check.	4 / 1	26	7 × 9	**63**	2365	*not applicable*			15 × 15	225	78
	XOR5_R1	5 / 1	40	*timeout limit reached*			*not applicable*			20 × 20	**400**	155
	cm82a	5 / 3	68	*timeout limit reached*			*not applicable*			50 × 50	**2500**	516
	2bitAdderMaj	5 / 2	82	*timeout limit reached*			*not applicable*			65 × 65	**4225**	3750
	xor5Maj	5 / 1	102	*timeout limit reached*			*not applicable*			75 × 75	**5625**	3912
	parity	16 / 1	150	*timeout limit reached*			*not applicable*			110 × 110	**12100**	7498

I, O and $|G|$ are the number of inputs, outputs and total gates in the logic network, respectively; runtime values are in seconds; the timeout limit is 24 h; w, h and A are the width, height, and resulting area of the layout, respectively; numbers in bold indicate the approach with the least layout area for each clocking scheme.

[17] N. Pavliček *et al.*, "Tip-induced passivation of dangling bonds on hydrogenated Si(100)-2×1," *APL*, vol. 111, no. 5, p. 053104, 2017.

[18] M. Rashidi *et al.*, "Initiating and Monitoring the Evolution of Single Electrons Within Atom-Defined Structures," *PRL*, vol. 121, p. 166801, Oct. 2018.

[19] M. D. Vieira *et al.*, "Three-Input NPN Class Gate Library for Atomic Silicon Quantum Dots," *IEEE Design & Test*, 2022.

[20] V. Vankamamidi *et al.*, "Clocking and Cell Placement for QCA," in *IEEE-NANO*, vol. 1, 2006, pp. 343–346.

[21] C. Campos *et al.*, "USE: A Universal, Scalable and Efficient clocking scheme for QCA," *IEEE TCAD*, vol. 35, pp. 513–517, Feb. 2016.

[22] M. Goswami *et al.*, "An Efficient Clocking Scheme for Quantum-dot Cellular Automata," *Int. J. Electron. Lett.*, vol. 8, no. 1, pp. 83–96, 2020.

[23] F. Sill Torres *et al.*, "On the Impact of the Synchronization Constraint and Interconnections in Quantum-dot Cellular Automata," *MICPRO*, vol. 76, pp. 103–109, 2020.

[24] C. Lent and P. Tougaw, "A Device Architecture for Computing with Quantum Dots," *Proc. IEEE*, vol. 85, no. 4, pp. 541–557, 1997.

[25] K. Hennessy and C. S. Lent, "Clocking of Molecular Quantum-dot Cellular Automata," *J. Vac. Sci. Technol. B*, vol. 19, no. 5, pp. 1752–1755, 2001.

[26] F. Sill Torres *et al.*, "Synchronization of Clocked Field-Coupled Circuits," in *IEEE-NANO*, 2018.

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[28] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine learning*, vol. 8, no. 3–4, pp. 229–256, May 1992.

[29] J. Schulman *et al.*, "Trust Region Policy Optimization," in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 37, 2015, pp. 1889–1897.

[30] S. Huang and S. Ontañón, "A Closer Look at Invalid Action Masking in Policy Gradient Algorithms," *FLAIRS*, vol. 35, May 2022.

[31] N. Mazyavkina *et al.*, "Reinforcement Learning for Combinatorial Optimization: A Survey," *COR*, vol. 134, p. 105400, 2021.

[32] R. Cheng and J. Yan, "On Joint Learning for Solving Placement and Routing in Chip Design," *NIPS*, vol. 34, pp. 16 508–16 519, 2021.

[33] M. Walter *et al.*, "Placement and Routing for Tile-Based Field-Coupled Nanocomputing Circuits Is NP-Complete (Research Note)," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 3, Apr. 2019.

[34] S. Hofmann *et al.*, "Scalable Physical Design for Silicon Dangling Bond Logic: How a 45° Turn Prevents the Reinvention of the Wheel," in *IEEE-NANO*, 2023, pp. 872–877.

[35] G. Fontes *et al.*, "Placement and Routing by Overlapping and Merging QCA Gates," in *ISCAS*, 2018, pp. 1–5.

[36] A. Trindade *et al.*, "A Placement and Routing Algorithm for Quantum-dot Cellular Automata," in *SBCCI*, 2016, pp. 1–6.

[37] M. Walter *et al.*, "*fiction*: An Open Source Framework for the Design of Field-coupled Nanocomputing Circuits," 2019.

[38] S. Hofmann *et al.*, "Late Breaking Results From Hybrid Design Automation for Field-coupled Nanotechnologies," in *DAC*, 2023, pp. 1–2.

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

[40] M. Walter *et al.*, "Verification for Field-coupled Nanocomputing Circuits," in *DAC*, 2020, pp. 1–6.

[41] S. Hofmann *et al.*, "MNT Bench: Benchmarking Software and Layout Libraries for Field-coupled Nanocomputing," in *DATE*, 2024.