

A Hybrid GPU-CPU Framework for QCA Layout Synthesis via Parallel Simulated Annealing and Register-Level Bitboard Routing

Leanderson Leles¹, Ian Otoni¹, Nicolas Silva¹, Luis Viana¹, Ricardo Ferreira¹

¹Departamento de Informática – Universidade Federal Viçosa (UFV)
Viçosa – MG – Brazil

{leanderson.leles, ian.gomes1, nicolas.e.silva, luis.viana, ricardo}@ufv.br

Abstract. *Quantum-dot Cellular Automata (QCA) is a promising beyond-CMOS paradigm, but physical design automation remains a bottleneck: existing placement and routing (P&R) approaches rely on sequential, CPU-based search, limiting scalability. This paper presents, to the best of our knowledge, the first GPU-accelerated QCA P&R framework under the USE clocking scheme, combining a parallel Simulated Annealing guided by a novel $O(1)$ USE-aware heuristic with a register-resident bitboard-based BFS and a deterministic CPU-side A* routing pass. Across eleven benchmark circuits, our framework produces valid, fully-routed layouts in the large majority of runs, matching or improving state-of-the-art area results in less time.*

1. Introduction

For decades, the continuous scaling of Complementary Metal-Oxide-Semiconductor (CMOS) technology has been the primary driver of the electronics industry, faithfully adhering to the trajectory predicted by Moore’s Law [Moore 1998]. However, as transistor dimensions approach the atomic scale, fundamental physical limitations, such as quantum tunneling, severe power dissipation, and interconnect bottlenecks, have critically hindered further miniaturization [Keyes 2005]. Consequently, there is a need to explore viable beyond-CMOS computing paradigms. Among the emerging nanoscale alternatives, Quantum-dot Cellular Automata (QCA), first introduced by [Lent et al. 1993], stands out as a highly promising candidate. By representing binary information through the spatial configuration of localized electrons rather than traditional electric currents, QCA offers profound theoretical advantages, including ultra-low power consumption, exceptionally high integration density, and the potential for terahertz clock frequencies [Kim et al. 2005].

However, the practical viability of these QCA devices intrinsically depends on a complex clocking system, even for combinational circuits. In the QCA paradigm, the clock plays a fundamental dual role: it not only synchronizes logical operations but also acts as the driving force that provides the energy required for signal propagation, ensuring the correct directionality of data flow through the cells [Walus et al. 2003]. To mitigate the severe design challenges generated by this spatial dependency, [Campos et al. 2015] proposed the Universal, Scalable, and Efficient (USE) clocking scheme, establishing a standardized two-dimensional grid that simplifies synchronization in complex layouts.

Despite the advances provided by the USE clock, physical design automation (Electronic Design Automation - EDA) for QCA circuits remains a critical bot-

tleneck, particularly Placement and Routing (P&R), which is an NP-complete problem [Walter et al. 2019]. The existing approaches, such as the recent methodology [Li et al. 2024], treat the problem sequentially on CPUs and suffer drastically from severe scalability issues. Although their use of heuristic algorithms like Simulated Annealing helps escape local optima, the combinatorial search space expands exponentially. Consequently, CPU-based algorithms lead to low success rates and fail to converge to valid solutions for large-scale circuits.

To overcome these critical limitations, this paper proposes a novel, scalable hybrid GPU-CPU framework for QCA layout synthesis. By offloading the combinatorial exploration to the GPU while reserving the CPU for exact deterministic validation, our architecture achieves valid, routable layouts across a wide range of circuit sizes, reaching 90% or higher success rates for the majority of benchmarks tested, while parallelizing the search in GPU. The main contributions of this work are summarized as follows:

- **Hybrid GPU-CPU P&R Framework:** we introduce the first GPU-accelerated layout synthesis framework for QCA that supports non-balanced graphs, executing Multi-Start Simulated Annealing [Van Laarhoven and Aarts 1987] across hundreds of concurrent GPU instances, culminating in an exact A* routing validation on the CPU.
- **$O(1)$ Clocking Heuristic:** We formulate a relaxed, constant-time heuristic for evaluating spatial distance and placement viability under the USE clocking scheme, enabling ultra-fast area reduction with register memory latency during the high-temperature exploration phase.
- **Register-Based Bitboard Occupancy & Fast BFS:** We propose an evaluation mechanism that encodes the QCA occupancy grid as a compact bitboard, mapped directly into GPU physical registers via compile-time C++ template metaprogramming, eliminating memory access overhead during occupancy queries. We characterize this design choice through a controlled comparison against a shared-memory cooperative alternative, quantifying where and under what conditions the register-resident representation provides an advantage.
- **Coarse-to-Fine Evaluation Strategy:** We implement a dynamic cost-evaluation pipeline, combining an $O(1)$ heuristic with a parallel BFS validation pass during the low-temperature phase of the annealing process. All candidate layouts are re-scored exactly at a checkpoint stage, after which only the top-tier solutions advance to the second phase, which then operates on this smaller elite pool alone, balancing rapid probabilistic estimation with deterministic routability.

The remainder of this paper is organized as follows. Section 2 provides the fundamental background on QCA technology and the USE clocking scheme. Section 3 reviews previous P&R methodologies and discusses their computational limitations. Section 4 details our proposed hybrid GPU-CPU framework, including the $O(1)$ heuristic and the register-based bitboard BFS. Section 5 presents the experimental results and performance comparisons. Finally, Section 6 concludes the paper and outlines directions for future work.

2. Theoretical Background

2.1. QCA Fundamentals and Gates

Quantum-dot Cellular Automata (QCA) encode binary information through the spatial configuration of localized electric charges rather than electric current. A standard QCA cell consists of four quantum dots occupied by two mobile electrons that, due to mutual Coulombic repulsion, settle into one of two antipodal diagonal configurations, denoted $P = +1$ and $P = -1$, encoding logic 1 and 0 respectively [Tougaw and Lent 1994]. When a polarized cell is placed next to an unpolarized one, electrostatic repulsion forces the neighbor into the same minimum-energy configuration, propagating the signal without any conventional current; a chain of such cells forms a QCA wire [Orlov et al. 1997].

An inverter could be created by placing cells diagonally, so that Coulombic repulsion forces the neighboring cell into the opposite polarization [Orlov et al. 1997]. The majority gate is the basic primitive QCA gate [Tougaw and Lent 1994] as shown in Figure 1a. Fixing one input to a stable $P = -1$ or $P = +1$ turns the majority gate into a programmable AND or OR gate, and together with the inverter it forms a universal logic set [Tougaw and Lent 1994].

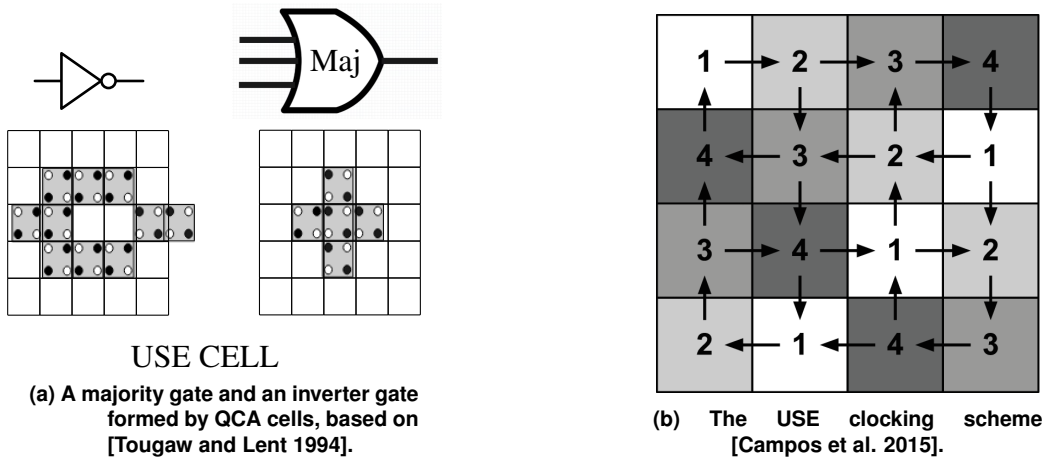


Figure 1. Overview of QCA logic gates and clocking scheme.

2.2. QCA Clocking and the USE Scheme

Because electrostatic signal propagation is inherently multidirectional, a temporal control mechanism is required to enforce a preferred direction of information flow [Walus et al. 2003]. The adiabatic switching scheme [Lent and Tougaw 1997] partitions the circuit into clock zones, each cycling through four phases: Switch, Hold, Release, and Relax.

Prior clocking schemes such as 2DDWave [Vankamamidi et al. 2008] enforce strictly unidirectional propagation, which complicates the feedback loops required by sequential circuits [Campos et al. 2015]. The Universal, Scalable, and Efficient (USE) scheme [Campos et al. 2015] addresses this by numbering clock zones 1–4 so that consecutively numbered zones are always adjacent, and by alternating the propagation direction of each row (Figure 1b); this bidirectional, cyclic structure creates feedback paths

without requiring any grid reconfiguration, and is the clocking scheme adopted throughout this work.

It should be noted that every logic gate or wire fits within exactly one USE cell. As shown in Figure 1a, a standard USE cell comprises a 5×5 grid of quantum dots, where, the majority gate utilizes fewer quantum dots than a standard inverter. Although Figure 1b shows a 4×4 USE layout capable of holding up to 16 logic gates, routing constraints typically limit this capacity. In practice, only a few gates can be placed, with the remaining cells acting as wires or left unoccupied.

3. Related Work

The placement and routing (P&R) problem in QCA circuits has been addressed by various strategies [Walter et al. 2019, Torres et al. 2020, Formigoni et al. 2021, Hofmann et al. 2024a, Hofmann et al. 2025]. Among the approaches that establish a foundational background for our research, [Trindade et al. 2016] proposed a greedy heuristic with backtracking specifically tailored for the USE clocking scheme. Their algorithm performs placement level-by-level, traversing the circuit from the outputs toward the inputs. However, a major limitation of this approach is its susceptibility to becoming trapped in local optima. Furthermore, the structural path balancing required by their method drastically increases the total number of dummy nodes, which exponentially expands the search space and hinders convergence.

Following this, [Fontes et al. 2018] introduced an alternative methodology based on a divide-and-conquer strategy over the USE grid. This approach partitions the logic circuit into smaller subgraphs and validates the overlapping of these partial solutions through an occupancy matrix. However, this heuristic still presents significant drawbacks. It heavily depends on manual user intervention to define the spacing between circuit levels (weight vectors) and is strictly limited to combinational circuits. Furthermore, without advanced pruning techniques, its sequential nature still struggles to scale autonomously for highly complex circuits due to the exponential growth of the search space.

More recently, [Li et al. 2024] proposed a hybrid optimization framework utilizing Simulated Annealing (SA) for intelligent placement and the A* algorithm for routing. A significant achievement of their work was the automatic calibration of the SA parameters according to the circuit scale. Nevertheless, this approach suffers drastically from the expansion of the combinatorial search space. As the circuit complexity and the underlying grid grow, their strictly CPU-based sequential execution struggles to converge, resulting in low success rates and prohibitive execution times for large-scale designs [Li et al. 2024].

Previous work has proposed GPU-accelerated placement for CGRA architectures [Canesche et al. 2020], which requires balanced internal paths but offers more routing resources compared to QCA constraints. In contrast, we propose the first hybrid GPU-CPU framework that offloads the combinatorial exploration of simulated annealing to the GPU, where thousands of threads concurrently evaluate placement configurations while a deterministic routing stage ensures physical realizability. The complete framework is detailed in Section 4.

4. Methodology

Figure 2 summarizes the proposed framework, which splits the work between CPU and GPU. In section 4.1 we explain how the CPU prepares the problem. Next, in sections 4.2–4.4 we describe how the GPU performs the massively parallel combinatorial exploration in two Simulated Annealing phases, separated by an exact checkpoint. Finally, in section 4.5 we show how CPU routes the best layout and validates it with a deterministic A* pass.

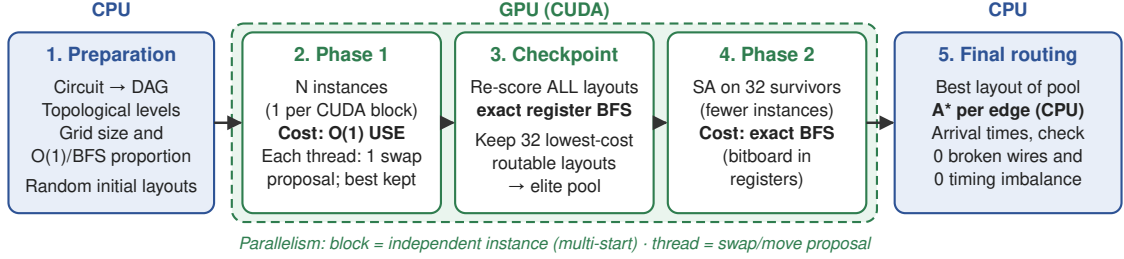


Figure 2. Overview of the proposed hybrid GPU-CPU framework. The CPU builds the DAG and topological levels (1); the GPU anneals many independent instances with the $O(1)$ heuristic (2), re-scores them exactly at a checkpoint (3) and refines the elite pool with the exact BFS cost (4); the CPU routes the best layout with A* and validates it (5).

4.1. Circuit Representation and Level Scheduling

Each input circuit is first modeled as a directed acyclic graph (DAG) in logic-level form, where each node corresponds to a logic gate and each directed edge encodes a data dependency between two gates. The circuit instances used in our experimental evaluation were obtained from the benchmark suites described in [Fontes et al. 2018] and [Hofmann et al. 2024b].

Once the DAG is built, we compute a *level* for every node through a standard Kahn-style topological traversal [Kahn 1962]: nodes with no predecessors (primary inputs) are assigned level 0, and every other node v receives $\text{level}(v) = \max_{u \in \text{pred}(v)} (\text{level}(u) + 1)$, i.e., the length of the longest path reaching v from any primary input.

Unlike prior USE-based approaches such as [Trindade et al. 2016], we deliberately avoid structural path balancing through buffer/dummy-cell insertion. Equalizing path lengths at the graph level requires committing, in advance, to how long each wire will be, information that is only truly known after placement. In practice this forces the insertion of a potentially large number of dummy cells, inflating the node count and, consequently, the required grid area, which sharply expands the combinatorial search space and degrades convergence for large circuits. Instead, we keep the DAG structurally unmodified and measure balance only after routing: once an edge is routed, we know its actual wire length, and can compute the real arrival time at each node from it. Any timing skew this reveals (when two inputs arrive at a node at different times) is fed back into the annealing cost function through the w_{balance} term, so placement itself learns to reduce imbalance through node positioning alone, without inflating the circuit. Figure 3 shows how edges skipping levels can end up spanning long physical distances once placed, a case the router must handle directly.

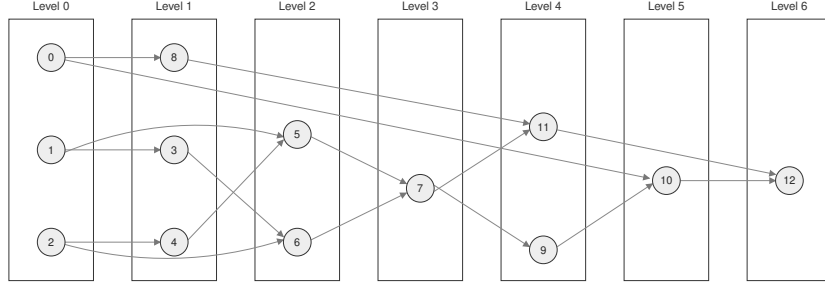


Figure 3. Example circuit organized by topological level. Nodes are grouped into columns according to their computed level; edges that connect non-adjacent levels illustrate long-range dependencies that increase routing complexity.

4.2. O(1) USE-Aware Distance Heuristic

To keep the high-temperature exploration phase of the annealing process free of any global and shared memory access, we replace exact pathfinding with a closed-form, constant-time estimate of the routed distance between two tiles under the USE clocking scheme. Let $z(r, c)$ denote the clock zone of tile (r, c) :

$$z(r, c) = \begin{cases} (r + c) \bmod 4 & \text{if } r \text{ is even} \\ (4 - ((r + c) \bmod 4)) \bmod 4 & \text{if } r \text{ is odd} \end{cases} \quad (1)$$

Given a source tile $A = (r_A, c_A)$ and a destination tile $B = (r_B, c_B)$, we compute the Manhattan distance $d_{man}(A, B) = |c_B - c_A| + |r_B - r_A|$ and the clock phase that would be reached after traversing it, $\phi = (z(A) + d_{man}(A, B)) \bmod 4$. The remaining phase misalignment with the destination's own zone is then corrected by

$$\Delta(A, B) = (z(B) - \phi + 4) \bmod 4, \quad (2)$$

yielding the final estimate

$$\hat{d}_{USE}(A, B) = d_{man}(A, B) + \Delta(A, B). \quad (3)$$

This formula is evaluated in $O(1)$ time and requires no global or shared memory access beyond the tile coordinates themselves, making it well suited for the thousands of concurrent delta-cost evaluations performed per annealing step.

It is important to stress that $\hat{d}_{USE}$ is a *heuristic*, not an exact routed distance: it assumes an unobstructed grid and only accounts for the geometric/clock relationship between A and B , ignoring tile occupancy entirely. Comparing it against ground-truth distances obtained from the register-resident BFS on representative circuit instances, we observe an agreement rate of approximately 90%. This means that, when estimating the distance between two tiles, the heuristic is correct approximately 90% of the time, but even when the predicted distance itself is correct, the underlying path may still be infeasible, since intervening tiles may already be occupied by other nodes. As circuit size and layout occupancy increase, this blind spot becomes more frequent, which is precisely why we complement this heuristic with the exact, occupancy-aware BFS validation described in the following sections.

4.3. Register-Resident Bitboard Occupancy

Both the $O(1)$ heuristic and the BFS-based validation are invoked hundreds of thousands of times per second during annealing, each requiring a query of which grid tiles are occupied. Representing occupancy as a plain per-tile boolean array would force every such query through global or shared memory, making occupancy checks the dominant cost of the search. We instead encode the entire grid as a *bitboard*: the $W \times H$ tile grid is flattened into a bit vector of $\lceil (W \cdot H)/64 \rceil$ 64-bit words, where bit i of word $\lfloor i/64 \rfloor$ indicates whether tile i is occupied. The number of words, `NUM_WORDS`, is a compile-time C++ template parameter rather than a runtime value, with a dispatch layer selecting the appropriate instantiation (up to `MAX_SUPPORTED_WORDS`= 9, i.e. grids up to 576 tiles). Because `NUM_WORDS` is known at compile time, the compiler keeps the entire bitboard resident in physical registers and fully unrolls all loops over it, eliminating memory traffic from occupancy queries almost entirely.

This representation also turns 2D grid movement into bitwise arithmetic: moving left or right is a one-bit shift, moving up or down corresponds to a bitwise shift equal to the grid’s width, both implemented as multi-word shifts that propagate carry bits across word boundaries. Each shift is masked against a precomputed adjacency mask encoding, for every tile, which neighbors lie in the next USE clock zone (the same $z(r, c)$ from Section 4.2), so only clock-valid moves survive. This lets the entire BFS frontier advance in one step using only `AND/OR/SHIFT/population-count` over `NUM_WORDS` words, regardless of frontier size. This same compactness makes it practical to replicate the full grid state across hundreds of concurrent annealing instances. The trade-off is a build-time ceiling: the dispatch layer picks the instantiation at run time for any square grid up to 576 tiles (24×24); larger grids require raising `MAX_SUPPORTED_WORDS` and recompiling (Section 6).

4.4. Hybrid Two-Phase Simulated Annealing

The annealing search exploits two levels of parallelism. *Across blocks*, each CUDA block owns one full instance (node-to-tile and tile-to-node mappings and bitboard occupancy state, Section 4.3) and anneals it from a different random placement; instances never communicate, so this multi-start level needs no inter-block synchronization and mitigates the risk of collapsing into a local optimum. *Within a block*, each thread proposes one move, exchanging a uniformly drawn node with a uniformly drawn grid tile (a swap if the tile is occupied, a relocation if empty), and evaluates its cost delta with either the $O(1)$ heuristic or the exact BFS-based cost, depending on the phase. A parallel tree reduction (`perform_reduction_and_accept_bitboard`) keeps the proposal with the smallest Δ ; only that move undergoes the Metropolis criterion (accepted if $\Delta \leq 0$, or with probability $e^{-\Delta/T}$) and is applied. Applying one move per step avoids write conflicts between threads, and register-resident occupancy queries avoid memory traffic.

The pipeline runs this scheme in two phases. In *Phase 1*, a large pool of instances is annealed from a high initial temperature down to a transition temperature, with cost deltas computed entirely through the $O(1)$ heuristic, allowing high-throughput exploration with zero memory-bound operations. Because the heuristic cannot detect unroutable configurations (Section 4.2), all resulting layouts are then re-scored exactly: a checkpoint kernel performs a full register-resident BFS routing pass over every instance, and only

the lowest-cost, actually-routable layouts are retained. *Phase 2* then continues annealing on this much smaller surviving pool, with every delta-cost evaluation now computed via the exact BFS-based router. Since this exact evaluation is markedly more expensive per move, the instance count is reduced by an order of magnitude and the per-temperature step count correspondingly increased, concentrating the costlier but exact search on the candidates most likely to converge to a valid, high-quality layout.

To isolate the impact of this design choice, we implement a second, otherwise identical variant in which the occupancy bitboard and BFS working set are instead placed in shared memory and processed cooperatively by the threads of a block, rather than kept resident in each thread’s own registers. This Shared variant reuses the exact same BFS algorithm, adjacency masks, and cost function described throughout this section, differing only in where the occupancy state physically resides and, as a consequence, in the number of threads that can be resident per block (Section 5). This consequence is not incidental: with the occupancy bitboard and BFS working set held in shared memory, the per-block footprint grows with the thread count, and a 64-thread configuration exceeds the shared-memory budget available per block on the target device. The Shared variant is therefore evaluated at 32 threads per block, its largest feasible and empirically best configuration, while the Register variant, whose bitboard consumes no shared memory, is evaluated at 64. Each variant is thus reported at its own optimum rather than at an artificially equalized thread count.

4.5. Deterministic A* Routing (CPU)

Once annealing converges, the lowest-cost layout among the 32 surviving instances (Section 4.4) is handed off to the CPU for a final, deterministic routing pass. Throughout the search, BFS is only ever queried for a *cost* (distance, crossings, routability) to guide candidate moves, never for an explicit path; once a layout is selected as final, however, an actual sequence of physical wire tiles must be produced for every edge. We therefore replace the bitboard BFS with a classical A* search per edge, run once, sequentially, on the CPU, in the same reverse topological order used during annealing.

For each edge (u, v) , A* searches for the shortest tile sequence from $\text{tile}(u)$ to $\text{tile}(v)$ respecting the USE clock zones (Section 4.2) and no intermediate tile is occupied by another node or already carries two overlapping wires, the same crossing limit assumed by the annealing cost function. The priority queue’s heuristic is exactly $\hat{d}_{USE}$, keeping the notion of distance consistent across the entire pipeline. Once every edge is routed (or marked broken, if infeasible), arrival times are recomputed from the real routed wire lengths, yielding physically-grounded values for wire length, crossings, and timing imbalance, the exact quantities that the GPU-side cost function could, until this point, only estimate. This final pass thus acts as an exact deterministic certificate that the layout found by the GPU-parallel search is not only cheap under the model’s cost function, but actually routable and physically realizable.

5. Results

To evaluate the proposed architecture, we performed approximately 50 independent runs per benchmark circuit, across eleven circuits ranging from 8 to 41 nodes and from 8 to 46 edges. All experiments ran on a single NVIDIA GB10 (Grace Blackwell, compute

capability 12.1) with CUDA 13.0 and driver 580.178.04. The host has a 20-core Arm (aarch64) CPU and 121 GiB of RAM, and runs the final A* pass sequentially. Phase 1 uses 512 instances (one per CUDA block) and Phase 2 uses 32. For each circuit, Table 1 also reports the grid dimensions used and the $O(1)$ /BFS-real step proportion applied during annealing (Section 4.4), which is adapted according to the observed difficulty of each instance rather than fixed globally. A run’s best-cost layout is considered *fully valid* when it achieves both zero broken wires and zero timing imbalance; the percentage of valid runs is reported in Table 1. We compare two BFS implementations under otherwise identical conditions: one storing the entire occupancy bitboard in physical registers (Section 4.3), and another placing the same structures in shared memory for cooperative processing within a thread block, each running with its architecturally appropriate thread count (64 and 32 threads per block, respectively). This section closes with a comparison against state-of-the-art placement and routing results for identically-sized circuits (Table 2).

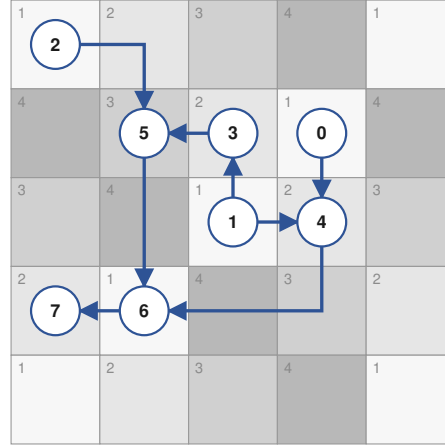


Figure 4. Final routed layout for mux2x1 (best-cost solution, cost = 115, 0 crossings). Background shading indicates the USE clock zone of each tile (numbered 1–4); every routed wire advances strictly from one zone to the next.

Figure 4 depicts a placement and routing solution generated by our algorithm. First, the user specifies the layout size, which in this example is a 5x5 USE grid. The input graph representing the circuit consists of 8 nodes. While basic grid placement and routing is already an NP-complete problem, QCA circuit design introduces further strict constraints. First, routing is unidirectional within each row and column: even rows route from left to right and odd rows from right to left, while even columns route from bottom to top and odd columns in the opposite direction. Second, wire crossings should be minimized; indeed, the solution shown in Figure 4 contains no crossings. Finally, all paths from the input nodes to the output nodes must be balanced and have identical lengths. For instance, input nodes 0 and 1 converge at node 4 after a single routing step. Input nodes 1 and 2 converge at node 5 after two routing steps, and ultimately, all input nodes converge at node 6 after six steps. Although initially targeting a 5x5 grid, the final layout effectively fits within a 4x4 area (16 tiles). As observed, due to the routing constraints, 8 tiles are occupied by logic nodes, 4 tiles are used for wiring, and the remaining 4 tiles are left unoccupied.

Table 1. Placement and routing results across the benchmark suite (64/32 thread configuration). Grid indicates the actual bounding box occupied by the best solution.

Circuit	Nodes / Edges	Prop. $O(1)$ /BFS	Variant	Grid	Valid (%)	Avg. Cost	Best Cost	Crossings (best)	Throughput (eval/s)
MUX2X1	8 / 8	60/40	Register	4×4	100.0	115	115	0	1.38×10^9
			Shared	4×4	100.0	115	115	0	5.30×10^8
XNOR	9 / 9	60/40	Register	6×6	90.0	698	580	3	6.27×10^8
			Shared	6×6	88.0	708	580	3	2.63×10^8
FULL_ADDER	11 / 14	60/40	Register	7×7	100.0	549	465	2	5.02×10^8
			Shared	7×7	100.0	555	400	0	2.25×10^8
PARITY_GEN	13 / 16	60/40	Register	7×7	78.0	1051	570	3	2.06×10^8
			Shared	7×7	84.0	1017	610	4	9.96×10^7
newtag	17 / 18	40/60	Register	8×8	100.0	263	250	0	2.03×10^8
			Shared	8×8	90.0	313	250	0	1.10×10^8
XOR5_R1	17 / 24	34/66	Register	10×10	90.0	1430	1120	7	3.51×10^7
			Shared	10×10	95.0	1401	1035	4	1.97×10^7
C17	18 / 20	60/40	Register	8×8	100.0	370	360	1	3.49×10^8
			Shared	8×8	98.0	437	340	0	1.72×10^8
PARITY_CHECK	19 / 24	34/66	Register	9×9	92.0	1334	890	5	7.62×10^7
			Shared	9×9	85.0	1358	1035	8	4.28×10^7
CLPL	25 / 28	34/66	Register	10×10	100.0	1453	1120	7	3.48×10^7
			Shared	10×10	95.0	1491	1175	9	1.98×10^7
XORMAJ	31 / 36	34/66	Register	10×10	85.0	1561	1155	6	6.40×10^7
			Shared	10×10	67.0	1834	1205	7	3.50×10^7
XorABC	41 / 46	25/75	Register	12×12	70.0	2579	2020	14	1.70×10^7
			Shared	12×12	20.0	3099	2130	17	1.09×10^7

The simulated annealing approach starts from random node positions and a cost function guides the search. Because all input-to-output paths must be balanced, a run does not always converge to a valid routing. For each circuit, Table 1 reports the **Grid** of the best solution; the percentage of runs that are **Valid**, i.e., whose best layout has zero broken wires and zero timing imbalance; the average and the lowest final cost over all runs (**Avg. Cost** and **Best Cost**); the number of wire **Crossings** in the best-cost layout (fewer is better); and the **Throughput** in cost evaluations per second. Crossings are only allowed when they are themselves valid, i.e., at most two wires overlap in a tile and the clocking and balance rules still hold; they are therefore accounted for in the cost function, but a layout with crossings is still valid.

The **Prop.** column gives the proportion of annealing iterations allocated to each cost function: a ratio of 60/40 means that the first 60% of the iterations use only the approximate $O(1)$ cost, and the remaining 40% use the exact BFS-based cost. The proportion was tuned empirically per circuit and follows a trend: grids up to 8×8 mostly use 60/40, 9×9 and 10×10 use 34/66, and XorABC (12×12) uses 25/75. This is consistent with the blind spot of the $O(1)$ heuristic, which ignores occupancy (Section 4.2) and becomes more frequent as layouts get denser, so larger instances benefit from more steps under the exact cost; an automatic selection rule is left for future work.

Across the eleven circuits, the Register variant reaches a higher throughput in every case, which is partly expected since it evaluates twice as many proposals per step, and a lower or equal best cost in eight of them. Its advantage in Valid rate tends to grow with circuit size: it is nil for the smallest circuits and reaches 85% vs. 67% on XORMAJ and 70% vs. 20% on XORABC (with exceptions on PARITY_GEN and XOR5_R1). We attribute this to the number of proposals per annealing step: each thread proposes a swap or move and the best of 64 (vs. 32) is kept, so more candidate moves are examined, which matters more as the search space grows. Since the register-resident bitboard is what allows 64 threads per block to fit on the device, memory placement and thread count are coupled in our design, and we report each variant at its own best configuration; separating the two effects is left for future work.

Finally, we compare our results against state-of-the-art QCA placement and routing approaches on identically-sized circuits (Table 2).

Table 2. Comparison with state-of-the-art placement and routing results. Our results are those of the Register variant of Table 1 (approximately 50 runs per circuit). “–” denotes a value not reported by the source; validity and crossing metrics are not reported by the compared works. The best grid (area) and time for each circuit are in bold (ties are all bold); times are compared only where the source reports them.

Circuit	Nodes / Edges	Prop. $O(1)$ /BFS	Our Grid	Our Time (s)	Our Valid (%)	Our Crossings	SOTA Grid	SOTA Time (s)	Source
C17	18 / 20	60/40	8×8	1.182	100.0	1	8×8 8×8	7.93 –	[Li et al. 2024] [Hofmann et al. 2024b]
MUX2X1	8 / 8	60/40	4×4	0.707	100.0	0	5×5 5×5	0.18 –	[Li et al. 2024] [Hofmann et al. 2024b]
PARITY_CHECK	19 / 24	34/66	9×9	3.074	92.0	5	14×14 11×11	302.07 –	[Li et al. 2024] [Hofmann et al. 2024b]
PARITY_GEN	13 / 16	60/40	7×7	2.144	78.0	3	8×8 9×9	10.4 –	[Li et al. 2024] [Hofmann et al. 2024b]
FULL_ADDER	11 / 14	60/40	7×7	1.211	100.0	2	8×8 15×15	13.09 –	[Li et al. 2024] [Hofmann et al. 2024b]
XNOR	9 / 9	60/40	6×6	0.917	90.0	3	4×5 6×6	0.05 –	[Li et al. 2024] [Hofmann et al. 2024b]
XorABC	41 / 46	25/75	12×12 (144 tiles)	10.647	70.0	14	13×13 272 tiles*	– –	[Hofmann et al. 2024b] [Fontes et al. 2018]
XORMAJ	31 / 36	34/66	10×10 (100 tiles)	3.581	85.0	6	195 tiles*	–	[Fontes et al. 2018]
newtag	17 / 18	40/60	8×8 (64 tiles)	1.364	100.0	0	13×13 80 tiles*	– –	[Hofmann et al. 2024b] [Fontes et al. 2018]
XOR5_R1	17 / 24	34/66	10×10	4.611	90.0	7	16×16	–	[Hofmann et al. 2024b]
CLPL	25 / 28	34/66	10×10 (100 tiles)	4.872	100.0	7	15×15 132 tiles*	– –	[Hofmann et al. 2024b] [Fontes et al. 2018]

* [Fontes et al. 2018] reports total occupied area as a single scalar value (tiles).

Table 2 shows that our layouts match or improve upon the state-of-the-art area for all circuits except XNOR, where [Li et al. 2024] obtains a smaller 4×5 grid. The most comparable approach is the CPU-based simulated annealing proposed in [Li et al. 2024]. For PARITY_CHECK, our solution is approximately $100\times$ faster than theirs; for PARITY_GEN and FULL_ADDER we obtain smaller grids (7×7 vs. 8×8) in less time. The

only circuits where we are slower are MUX2X1 and XNOR, which their method solves in under 0.2 s. Furthermore, [Li et al. 2024] reports no results for the larger circuits listed in the final rows of Table 2. Finally, compared with [Fontes et al. 2018], which also provides balanced solutions, our layouts for XORABC and XORMAJ occupy roughly half the area (144 vs. 272 and 100 vs. 195 tiles), and those for NEWTAG and CLPL occupy 20% and 24% less (64 vs. 80 and 100 vs. 132 tiles). The times of [Li et al. 2024] were taken from the original paper, measured on an Intel Core i5-10400 (2.90 GHz, 16 GB RAM, Ubuntu 20.04), whereas [Hofmann et al. 2024b] and [Fontes et al. 2018] report no times; runtime comparisons are therefore indicative only, given the different hardware platforms.

6. Conclusion and Future Work

This paper presented a GPU-accelerated framework for QCA placement and routing under the USE clocking scheme, combining a parallel, multi-start Simulated Annealing search on the GPU with an exact, deterministic A* routing pass on the CPU. The proposed $O(1)$ USE-aware distance heuristic drives high-throughput exploration during the high-temperature phase of the search, while a coarse-to-fine strategy re-scores candidate layouts with a register-resident, bitboard-based BFS to bridge the gap between fast probabilistic estimation and deterministic routability. Across eleven benchmark circuits ranging from 8 to 41 nodes, the resulting pipeline produces valid, routable layouts in the large majority of runs, with the register-resident BFS variant achieving higher throughput and generally lower-cost solutions than a shared-memory cooperative alternative under its thread configuration. A comparison against state-of-the-art placement and routing results further shows that our layouts match or improve upon reported area for several circuits, while running substantially faster than CPU-based simulated annealing on the larger circuits. Together, these results demonstrate that GPU parallelization is a viable and effective path toward scalable QCA physical design automation, addressing the scalability limitations that have constrained strictly sequential, CPU-based prior approaches [Trindade et al. 2016, Fontes et al. 2018, Li et al. 2024].

Several directions remain open for future work. First, our evaluation covers eleven combinational circuits with up to 41 nodes on grids up to 12×12 , and it assesses the components of the framework (the $O(1)$ heuristic, the two-phase scheme and the register-based BFS) only as part of the full pipeline. Ablation studies, other thread-per-block and block configurations, and an automatic choice of the $O(1)$ /BFS proportion would show which design choices matter most and whether the framework scales to circuits with hundreds or thousands of nodes. Second, the supported grid size is bounded at compile time by `MAX_SUPPORTED_WORDS`; a graph-partitioning strategy that recursively divides large circuits into smaller subgraphs, placing and routing each independently before reconciling interface constraints, is a promising divide-and-conquer path beyond this limit. Finally, a synthesis-normalized comparison with prior work remains an open problem, since node and edge counts of identically named benchmarks vary substantially across synthesis pipelines, precluding a fully controlled area or runtime comparison.

7. Acknowledgments

The authors thank UFV and the agencies CAPES, CNPq, and FAPEMIG APQ-01577-22, and UFV Lato Senu in Robotics for partially funding this work.

References

- Campos, C. A. T., Marciano, A. L., Neto, O. P. V., and Torres, F. S. (2015). Use: a universal, scalable, and efficient clocking scheme for qca. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 35(3):513–517.
- Canesche, M., Menezes, M., Carvalho, W., Torres, F. S., Jamieson, P., Nacif, J. A., and Ferreira, R. (2020). Traversal: A fast and adaptive graph-based placement and routing for cgras. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(8):1600–1612.
- Fontes, G., Silva, P. A. R., Nacif, J. A. M., Neto, O. P. V., and Ferreira, R. (2018). Placement and routing by overlapping and merging qca gates. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE.
- Formigoni, R. E., Ferreira, R. S., and Nacif, J. A. M. (2021). A survey on placement and routing for field-coupled nanocomputing. *Journal of Integrated Circuits and Systems*, 16(1):1–9.
- Hofmann, S., Walter, M., and Wille, R. (2024a). A* is born: Efficient and scalable physical design for field-coupled nanocomputing. In *2024 IEEE 24th International Conference on Nanotechnology (NANO)*, pages 80–85. IEEE.
- Hofmann, S., Walter, M., and Wille, R. (2024b). MNT Bench: Benchmarking software and layout libraries for field-coupled nanocomputing. In *2024 Design, Automation and Test in Europe (DATE)*.
- Hofmann, S., Walter, M., and Wille, R. (2025). Graph-oriented layout design for field-coupled nanocomputing via parallel multi-objective search space exploration. *IEEE Transactions on Circuits and Systems I: Regular Papers*.
- Kahn, A. B. (1962). Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562.
- Keyes, R. W. (2005). Physical limits of silicon transistors and circuits. *Reports on Progress in Physics*, 68(12):2701–2746.
- Kim, K., Wu, K., and Karri, R. (2005). Towards designing robust qca architectures in the presence of sneak noise paths. In *Design, Automation and Test in Europe*, pages 1214–1219. IEEE.
- Lent, C. S. and Tougaw, P. D. (1997). A device architecture for computing with quantum dots. *Proceedings of the IEEE*, 85(4):541–557.
- Lent, C. S., Tougaw, P. D., Porod, W., and Bernstein, G. H. (1993). Quantum cellular automata. *Nanotechnology*, 4(1):49–57.
- Li, G., Peng, F., Zhang, B., Li, Y., and Xie, G. (2024). A qca placement and routing algorithm based on the sa algorithm. *International Journal of Electronics*, 111(12):2143–2158.
- Moore, G. E. (1998). Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85.

- Orlov, A. O., Amlani, I., Bernstein, G. H., Lent, C. S., and Snider, G. L. (1997). Realization of a functional cell for quantum-dot cellular automata. *Science*, 277(5328):928–930.
- Torres, F. S., Silva, P. A., Fontes, G., Walter, M., Nacif, J. A. M., Ferreira, R. S., Neto, O. P. V., Chaves, J. F., Wille, R., Niemann, P., et al. (2020). On the impact of the synchronization constraint and interconnections in quantum-dot cellular automata. *Microprocessors and Microsystems*, 76:103109.
- Tougaw, P. D. and Lent, C. S. (1994). Logical devices implemented using quantum cellular automata. *Journal of Applied physics*, 75(3):1818–1825.
- Trindade, A., Ferreira, R., Nacif, J. A. M., Sales, D., and Neto, O. P. V. (2016). A placement and routing algorithm for quantum-dot cellular automata. In *2016 29th symposium on integrated circuits and systems design (SBCCI)*, pages 1–6. IEEE.
- Van Laarhoven, P. J. and Aarts, E. H. (1987). Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer.
- Vankamamidi, V., Ottavi, M., and Lombardi, F. (2008). Two-dimensional schemes for clocking/timing of qca circuits. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 27(1):34–44.
- Walter, M., Wille, R., Große, D., Torres, F. S., and Drechsler, R. (2019). Placement and routing for tile-based field-coupled nanocomputing circuits is np-complete (research note). *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 15(3):1–10.
- Walus, K., Jullien, G., and Dimitrov, V. (2003). Computer arithmetic structures for quantum cellular automata. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1435–1439. IEEE.